Diplomarbeit

Mengentransformationen für die mengen-orientierte Prototyping-Sprache ProSet

Melanie Everz

Inhaltsverzeichnis

1. Einführung

Programmtransformationen werden eingesetzt, um die Effizienz von Programmen, die auf einem hohen Niveau spezifiziert wurden, zu verbessern [Fea87, PS83].

1.1 Motivation

Die vorliegende Arbeit beschäftigt sich mit Mengentransformationen für die mengenorientierte Prototyping-Sprache ProSet [DFG+92]. ProSet ist eine Nachfolgesprache von SETL und zeichnet sich durch ein hohes expressives Niveau aus. Es werden Programmkonstrukte in Anlehnung an mathematische Notationen zur Verfügung gestellt. Eine kurze Beschreibung der Sprache ProSet befindet sich in Kapitel 2. Zunächst soll anhand eines Beispielprogramms die Thematik motiviert werden.

Das Programm in Abbildung 1.1 auf Seite 3 berechnet den Durchschnitt der Menge `B` mit der Menge der durch `get` eingelesenen Zahlen. Bei einer Betrachtung des Programms fällt auf, daß bei jeder Iteration in der mit `Anweisung 1` gekennzeichneten Zeile die Menge `A` neu berechnet wird. Es wird also bei jeder Iteration jedes Element aus Menge `B` mit jedem Element aus Menge `C` verglichen und, sofern es in beiden Mengen enthalten ist, in die Menge `A` eingefügt. Da die Menge `A` jedoch nur von den Mengen `B` und `C` abhängt, ist innerhalb einer Iteration eine Veränderung der Menge `A` nur in Zeile `Anweisung 2` möglich, da nur dort die Menge `C` durch Einfügen eines neuen Elementes `i` modifiziert werden kann. Die Menge `B` bleibt innerhalb der Schleife unverändert. Nach einer Analyse dieses Verhaltens ist es leicht einzusehen, daß die Laufzeit des Programms effizienter wäre, wenn nur das Element `i` zu der in der vorhergehenden Iteration berechneten Menge `A` hinzugefügt würde, sofern `i` auch in der Menge `B` enthalten ist. Es würde also ausreichen, die gesamte Menge `A` einmal zu berechnen und dann nur die zu Menge `C` neu hinzugefügten Elemente in `A` einzufügen, falls sie in der Menge `B` enthalten sind. Das Programm in Abbildung 1.2 auf Seite 3 berechnet ebenfalls den Durchschnitt der Mengen `B` und `C`.

In Zeile `Anweisung 3` des Programms in Abbildung 1.2 wird vor dem Eintritt in die Schleife die Menge `A` berechnet. Dabei wird die Hilfsvariable `pstt1` verwendet, da erst innerhalb der Schleife in Zeile `Anweisung 1` im Programm in Abbildung 1.1 die Menge `A` definiert wurde und daher `A` auch nur in Zeile `Anweisung 4` berechnet werden darf. In Zeile `Anweisung 4` wird `A` der Wert von `pstt1` zugewiesen. In Zeile `Anweisung 5` wird nun `i` zu `pstt1` hinzugefügt, unter der Bedingung, daß `i` in `B` enthalten ist. Hier ist wieder die Hilfsvariable `pstt1` notwendig, da nur in Zeile `Anweisung 4` der Wert von `A` modifiziert werden darf. Es läßt sich noch weiter feststellen, daß `i` nur dann in `C` bzw. in `pstt1` eingefügt werden muß, wenn es noch nicht in `C` enthalten war. Das Beispiel in Abbildung 1.3 auf Seite 4 wurde um diese Modifikation erweitert.

An diesem Beispiel wird deutlich, daß ProSet eine Breitbandsprache ist. Eine Breitbandsprache zeichnet sich durch die Eigenschaft aus, daß Algorithmen auf verschiedenen Niveaus beschrieben werden können.

Bei einem Vergleich der Programme in Abbildung 1.1 und 1.3 fällt zuerst auf, daß das Programm in Abbildung 1.1 leichter zu verstehen ist, als das Programm in Abbildung 1.3. Bei einer Betrachtung der Laufzeitkomplexität[1] erhält man beim Programm in Abbildung 1.1 eine obere Grenze von $O(I^2)$, wobei I die Anzahl der eingelesen Zahlen i (`i` $\neq$ `om`) ist (I entspricht somit auch der Anzahl der Schleifeniterationen). Für das Programm in Abbildung 1.3 erhält man eine Komplexität von $O(I)$. Diese Effizienzsteigerung wird erreicht, da in diesem Programm innerhalb der Schleife nicht über die Menge `C` iteriert wird, sondern nur das zu `C` neu hinzugefügte Element `i` betrachtet wird.

ProSet hat eine hohe Ausdruckskraft, die sich leider negativ auf die Laufzeit auswirkt. Ziel dieser Arbeit ist es nun, die oben beschriebene Transformation [PK82] während der Übersetzungsphase eines Programms automatisch durchzuführen. Dadurch erhält man ein effizienteres Programm, ohne auf ProSet's Ausdruckskraft verzichten zu müssen.

[1] Die Berechnungen verwenden das Kostenmaß aus [DF89, Abschnitt V.5.3]. Für die Zuweisung „`A := pstt1`“ werden entsprechend die Kosten $O(1)$ angenommen.

```
program example1;
begin
   B := {1 .. 100};
   get(i);
   C := {i};
   while (i /= om) do
      A := {x : x in B | x in C};  -- Anweisung 1
      get(i);
      if (i /= om) then
         C with := i;              -- Anweisung 2
      end if;
   end while;
end example1;
```

Abbildung 1.1: Ein Beispielprogramm, daß den Durchschnitt einer Menge mit der Menge, der von der Tastatur eingelesenen Zahlen, berechnet.

```
program example2;
begin
   B := {1 .. 100};
   get(i);
   C := {i};
   pstt1 := {x : x in B | x in C};  -- Anweisung 3
   while (i /= om) do
      A := pstt1;                   -- Anweisung 4
      get(i);
      if (i /= om) then
         if (i in B) then
            pstt1 with := i;        -- Anweisung 5
         end if;
         C with := i;
      end if;
   end while;
end example2;
```

Abbildung 1.2: Erste modifizierte Version des Beispielprogramms in Abbildung 1.1.

```
program example3;
begin
   B := {1 .. 100};
   get(i);
   C := {i};
   pstt1 := {x : x in B | x in C};
   while (i /= om) do
      A := pstt1;
      get(i);
      if (i /= om) then
         if (i notin C) then
            if (i in B) then
               pstt1 with := i;
            end if;
            C with := i;
         end if;
      end if;
   end while;
end example3;
```

Abbildung 1.3: Zweite modifizierte Version des Beispielprogramms in Abbildung 1.1.

1.2 Übersicht

Die vorliegende Arbeit gliedert sich in drei Teile. Im ersten Teil werden die theoretischen Grundlagen für das zu entwickelnde Transformationsprogramm vorgestellt. Zunächst folgt eine kurze Einführung in ProSet (Kapitel 2) und danach wird die für die Transformation verwendete Theorie vorgestellt (Kapitel 3).

Im darauffolgendem Teil werden die Anforderungen an das Transformationsprogramm (Kapitel 5), das bei der Implementierung verwendete Werkzeug (Kapitel 6) und die Implementierung (Kapitel 7 und 8) beschrieben. Im Anschluß werden die Ergebnisse einiger Laufzeittests angegeben (Kapitel 9).

Im letzten Teil befinden sich eine Zusammenfassung dieser Arbeit (Kapitel 11) und ein Ausblick auf mögliche Erweiterungen des Transformationsprogramms (Kapitel 12).

In Anhang A befinden sich die implementierten Transformationsmuster und in Anhang B wird das Einfügen eines neuen Transformationsmusters in das Programm ausführlich diskutiert. In Anhang C befindet sich der mit **noweb** [Ram92] erstellte literale Quellcode.

Teil I

Grundlagen

2. Die Prototyping-Sprache ProSet

In diesem Kapitel wird die Prototyping-Sprache ProSet kurz vorgestellt. Eine vollständige Beschreibung der Sprache befindet sich in [DFG+92]. Prototyping mit mengenorientierten Programmiersprachen wird ausführlich in [DF89] behandelt.

ProSet ist schwach getypt. An primitiven Datentypen werden ganze und reelle Zahlen, Zeichenketten, die booleschen Werte `TRUE` und `FALSE` sowie Atome zur Verfügung gestellt. Desweiteren gibt es die zusammengesetzten Typen Tupel und Mengen und die Datentypen höherer Ordnung Funktionen, Module und Instanzen. Ein weiterer Bestandteil der Sprache ist der undefinierte Wert `om`. Eine Ausnahmebehandlung ist in die Sprache integriert worden und darüber hinaus wurde der Sprachkern um Parallelität [Has93] und Persistenz [Dob92] erweitert. Es stehen ein Prä- und ein Makroprozessor zur Verfügung. In dieser Arbeit werden nur die Teile der Sprache vorgestellt, die im weiteren Verlauf dieser Arbeit benötigt werden.

2.1 Mengen

ProSet stellt für Mengen Operationen für die endliche Mengenlehre zur Verfügung. Die vordefinierten Operationen sind Vereinigung (`+`), Durchschnitt (`*`), Mengendifferenz (`-`), Kardinalität (`#`), Einfügen eines Elements (`with`), Löschen eines Elements (`less`), Test auf Enthaltensein (`in`, `notin`), Gleichheit (`=`, `/=`), Teilmengenbeziehung (`subset`) und die Potenzmengenoperationen `pow` und `npow` (`npow` liefert alle `n`-elementigen Mengen der Potenzmenge). Die unären Operatoren `arb` und `random` liefern ein beliebiges bzw. stochastisch ausgewähltes Element aus einer Menge. Dieses Element bleibt in der Menge enthalten. Die Operation `from` entfernt ein beliebiges Element aus einer Menge und die Variable `x` erhält diesen Wert („`x from A`“). Diese Operation ist äquivalent zu der Folge der Zuweisungen

```
x := arb A;
A less := x;
```

Von den Möglichkeiten in ProSet Mengen zu konstruieren, ist für diese Arbeit die deskriptive Beschreibung von Mengen die interessante. Der allgemeine Mengenausdruck

$$\{\texttt{e:}x_1\ \texttt{in}\ s_1\texttt{,}\ x_2\ \texttt{in}\ s_2\texttt{,}\ x_3\ \texttt{in}\ s_3\ \mid\ \texttt{C}\}$$

erzeugt die Menge aller Werte, die der Ausdruck `e` liefert, wobei mit x_1 über s_1, mit x_2 über s_2 und mit x_3 über s_3 iteriert wird und die Bedingung `C` erfüllt ist. Diese Bedingung kann auch entfallen. Zum Beispiel liefert der Mengenausdruck

```
{x:x in {1,2,3,4,5} | x < 3}
```

die Menge {`1,2`}. Bei diesem Beispiel wurde für die Grundmenge ein weiterer Mengenformer, die Aufzählung, verwendet.

Als letzte Möglichkeit Mengen deskriptiv zu definieren, stellt ProSet das Intervall zur Verfügung. Der Mengenformer {`i..k`} erzeugt die Menge, die alle Zahlen von `i` bis `k` (einschließlich `i` und `k`) enthält, wobei `i` und `k` ganze Zahlen sein müssen. Als weitere Möglichkeiten lassen sich in ProSet Intervalle angeben, die nur jedes `m`te Element von `i` bis `k` enthalten: {`i,i+m..k`}. Zum Beispiel liefert der Mengenformer {`1,1+3 .. 11`} die Menge {`1,4,7,10`}.

2.2 Tupel

Tupel sind, im Gegensatz zu Mengen, geordnete Folgen von Werten. Anders als in Mengen können Werte (auch der undefinierte Wert) mehrfach in einem Tupel auftreten und die Reihenfolge der Komponenten ist von Bedeutung. Für Tupel gibt es die Operationen Konkatenation (`+`), stochastische Auswahl einer Komponente (`random`), neue Komponenten an Tupel anhängen (`with`) und einen Ausschnitt eines Tupels modifizieren bzw. einer Variablen zuweisen. Weiterhin stellt die Sprache eine Operation, die die Länge eines Tupels liefert (`#`) und Operationen, die zwei Tupel auf Gleichheit bzw. Enthaltensein testen (`=`, `/=`, `in`, `notin`) zur Verfügung. Durch die Operationen `fromb` und `frome` erhält eine Variable den Wert der ersten bzw. letzten Komponente eines Tupels. Das entsprechende Element wird aus dem Tupel entfernt.

Tupel können, ähnlich wie Mengen, deskriptiv erzeugt werden. Dabei werden die Klammern { und } durch `[` und `]` ersetzt.

2.3 Abbildungen

Abbildungen sind kein eigener Datentyp in ProSet, können jedoch mit Mengen und Tupeln realisiert werden. Eine Abbildung ist eine Menge von Paaren (zweielementige Tupel) [`x,y`] mit `x` $\neq$ `om` und `y` $\neq$ `om`. Den Definitions- bzw. Wertebereich einer Abbildung liefern die Operationen `domain` bzw. `range`. Die Prädikate `is_map` und `is_smap` testen, ob es sich bei einer gegebenen Menge um eine Relation bzw. eine (mathematische) Abbildung, in der jedem Element des Definitionsbereichs genau ein Element des Wertebereichs zugewiesen wird, handelt. Als weitere Operation für Abbildungen gibt es `lessf`. Die Anweisung „`f lessf x`" entfernt aus `f` alle Paare [`x`,`y`] mit `x` im Definitionbereich von `f`. Um das eindeutige Bild bzw. die Bildmenge für einen Wert `x` aus dem Definitionsbereich einer Abbildung `f` bzw. Relation `f` zu erhalten oder zu modifizieren, wird `f(x)` bzw. `f{x}` benutzt.

2.4 Ein- und Ausgabe

Das Einlesen eines Objektes `x` von der Standardeingabe erfolgt mit `get(x)` und die Ausgabe eines Objektes `x` auf die Standardausgabe erfolgt mit `put(x)`. In ProSet ist auch die Ein- und Ausgabe in Dateien vorgesehen.

2.5 Kontrollstrukturen

Von den vorhandenen Kontrollstrukturen in ProSet sind für diese Arbeit hauptsächlich die Schleifen von Bedeutung. In der Motivation (Abschnitt 1.1) konnte die Effizienz des Programms in Abbildung 1.1 auf Seite 3 verbessert werden, indem innerhalb der Schleife der Programmcode optimiert wurde. In ProSet existieren `loop`-, `while`-, `repeat`-, `for`- und `whilefound`-Schleifen. Weiterhin gibt es in ProSet die folgenden Kontrollstrukturen: `if`-Anweisung, `case`-Anweisung, `quit`, `continue`, `pass` und `stop`. Im folgenden werden die `while`- und die `for`-Schleife exemplarisch durch Programmfragmente eingeführt. Für die anderen Kontrollstrukturen sei auf die Sprachbeschreibung verwiesen [DFG+92].

- Ein Beispiel für eine `while`-Schleife sieht folgendermaßen aus:

  ```
  get(i);           -- Anweisung 1
  while i < 10 do
     i := i + 1;    -- Anweisung 2
     put(i);        -- Anweisung 3
  end while;
  j := i;           -- Anweisung 4
  ```

  Solange `i` die Bedingung `i` < `10` erfüllt, werden die Zeilen `Anweisung 2` und `Anweisung 3` ausgeführt. Anschließend wird `Anweisung 4` ausgeführt. Gilt für das `i` aus Zeile `Anweisung 1` die Bedingung nicht, wird die Schleife nicht durchlaufen und es wird sofort `Anweisung 4` ausgeführt.

- Ein Beispiel für eine `for`-Schleife sieht folgendermaßen aus:

```
S := {9 .. 15};
T := {};
for x in S | x < 10 do
   T with := x;
end for;
```

Die `for`-Schleife iteriert über alle Elemente in `S`, die die Bedingung `x < 10` erfüllen, und fügt diese Elemente in `T` ein. `x` ist eine *gebundene* Variable, die lokal zur `for`-Schleife ist und innerhalb der Schleife nicht modifiziert werden darf.

2.6 Kurzform für Zuweisungen

Es ist in ProSet erlaubt, binäre Operatoren mit dem Zuweisungszeichen zu kombinieren. Die beiden folgenden Zuweisungen sind somit gleichwertig:

```
x := x * (1 + y);

x *:= 1 + y;
```

2.7 Ausnahmebehandlung

In ProSet besteht die Möglichkeit in Zuweisungen und Ausdrücken Fehler durch Ausnahmen abzufangen. Hier wird die Beschreibung der Ausnahmebehandlung auf ein kleines Beispiel beschränkt. Eine umfassende Darstellung der Ausnahmebehandlung befindet sich in [DFG+92]. In Zeile `Anweisung 1` des folgenden Beispiels wird -1 ausgegeben.

```
y := x/0 when illegal_operand use substitute;
put(y);                    -- Anweisung 1
...
handler substitute();
begin
   put("zero_divide"):
   return -1;
end substitute;
```

Die vordefinierte Ausnahme `illegal_operand` wird bei der Ausführung der Zuweisung „`y := x/0`“ ausgelöst. Diese Ausnahme wird im Beispiel durch den Handler `substitute` abgefangen. Anstelle eines Programmabruchs bei der Berechnung von „`x/0`“ wird der Handler `substitute` ausgeführt, der „`zero_divide`“ ausgibt und `y` den Wert `-1` zuweist.

3. Endliche Differentiation mengentheoretischer Ausdrücke

Die endliche Differentiation mengentheoretischer Ausdrücke (Finite Differencing of Computable Expressions) wurde von R. Paige und S. Koenig [PK82] entwickelt. Die Grundlagen der Theorie befinden sich auch in [DF89]. Bei der Übersetzung der Fachterminologie aus dem Englischen ins Deutsche werden die dort benutzten Begriffe verwendet.

Im nächsten Abschnitt wird zunächst die Theorie diskutiert (Abschnitt 3.1). Im nachfolgenden Abschnitt wird kurz auf das Programmtransformationssystem RAPTS eingegangen, welches von R. Paige auf der Grundlage der endlichen Differentiation implementiert wurde (Abschnitt 3.2). Im Anschluß wird beschrieben, welche Teile der Theorie in der Implementierung des Transformationsprogramms für ProSet verwendet wurden und welche Gründe dazu führten, nicht die gesamte Theorie zu implementieren (Abschnitt 3.3).

3.1 Endliche Differentiation

Im Beispiel aus Abschnitt 1.1 wurde die Berechnung von „`{x : x in B | x in C}`“ vor die Schleife bewegt und mit Hilfe der Hilfsvariablen `pstt1` und differentiellem Code die Invariante

```
pstt1 = {x : x in B | x in C}
```

aufrechterhalten [Pai86]. Eine Invariante ist eine Bedingung, die an bestimmten Punkten im Programm gilt. Mit differentiellem Code wird der in der Schleife hinzugefügte Code bezeichnet. Durch die obige Invariante wird sichergestellt, daß in `Anweisung 4` in Abbildung 1.2 auf Seite 3 die Menge `A` den Wert des Ausdrucks „`{x : x in B | x in C}`“ erhält.

In diesem Beispiel wurde das Konzept angewendet, das die Grundlage der endlichen Differentiation bildet. An der beschriebenen Vorgehensweise wird erkennbar, daß die

„endliche Differentiation für mengentheoretische Ausdrücke“ das Prinzip der „Reduktion der Stärke“ auf mengentheoretische Ausdrücke verallgemeinert. Die Neuberechnung eines Mengenausdrucks wird durch Einfüge- und Löschoperationen ersetzt. Dieses Konzept wird nun genauer vorgestellt.

3.1.1 Vorbemerkungen

Zunächst werden einige Begriffe benötigt, die die Verifikation eines Programms und den Vergleich der berechneten Funktionen zweier Programme ermöglichen.

Wir erweitern ProSet[1] um die Anweisung „assert(cond)“. Diese Anweisung läßt sich mit Hilfe des folgenden Makros realisieren:

```
macro assert <<Bedingung>>
   if not (Bedingung) then
      put("assertion failed");
      stop;
   end if;
endm assert;
```

Die Benutzung dieses Makros erfolgt mit der Anweisung „assert<<cond>>“. Die Syntax für Makrodefinitionen und -aufrufe ist in [DFG$^+$92] beschrieben.

Definition 3.1 Die Ausführung eines Programms heißt *regulär*, falls aus der Tatsache, daß die Bedingungen aller assert-Anweisungen erfüllt sind, folgt, daß das Programm normal terminiert. Ein Programm *terminiert normal*, falls es durch eine stop-Anweisung und nicht durch eine Ausnahme beendet wird[2]. Ein Programm heißt *regulär*, falls alle seine möglichen Ausführungen regulär sind.

Definition 3.2 Der *Definitionsbereich* eines regulären Programms P ist die Menge aller Eingabewerte, die zu regulären Ausführungen von P führen.

Um nun die Korrektheit von Programmtransformationen überprüfen zu können, werden die Austrittspunkte eines Programms P mit assert-Anweisungen versehen, die die Werte aller Variablen überprüfen.

[1] Wir beschreiben die Theorie wie sie in [PK82] definiert wurde, verwenden jedoch für die Beispiele ProSet anstelle von SETL. Im folgenden wird nur der Sprachkern von ProSet verwendet, der im Prinzip mit dem Sprachkern von SETL übereinstimmt (abgesehen von syntaktischen Unterschieden).

[2] Die letzte Anweisung eines Programms ist implizit eine stop-Anweisung.

Definition 3.3 Eine Programmtransformation T *erhält* die Regularität eines Programms P, wenn das transformierte Programm $P' = T(P)$ regulär ist und die `assert`-Anweisungen an den Austrittspunkten des Programms P durch T nicht verändert werden.

Definition 3.4 Wenn T die Regularität erhält und zusätzlich der Definitionsbereich des regulären Programms P im Definitionsbereich von P' enthalten ist, heißt T *semantiktreu* für das reguläre Programm P.

Diese Begriffe lassen sich auch auf Programm-Regionen mit jeweils genau einem Eintritts- und Austrittspunkt anwenden. Diese Programm-Regionen werden als *Blöcke* bezeichnet.

$f(x_1, \ldots, x_n)$ sei ein applikativer Ausdruck. Das heißt, f verhält sich wie eine mathematische Abbildung und löst insbesondere keine Seiteneffekte aus. Für den Wert eines applikativen Ausdrucks $f(x_1, \ldots, x_n)$ wird die neue[3] Variable $V = f(x_1, \ldots, x_n)$ verwendet. V wird als die mit f assoziierte *virtuelle* Variable bezeichnet. V entspricht der Variablen `pstt1` aus dem Einführungsbeispiel.

Definition 3.5 V ist *am Ausgang* eines Programmpunktes p *verfügbar*, wenn V den Wert hat, den f hätte, falls f unmittelbar nach der Anweisung p ausgeführt würde.

Definition 3.6 V heißt *verfügbar am Eingang zu* p, wenn V am Ausgang aller Vorgänger von p verfügbar ist.

Definition 3.7 Wenn V am Eingang von p verfügbar ist und f in p ausgewertet wird, dann ist das Vorkommen von f in p *redundant* und f kann durch V ersetzt werden.

Definition 3.8 Ein Ausdruck $f(x_1, \ldots, x_n)$ heißt *wohldefiniert* in einem Programmpunkt p, falls für jede reguläre Ausführung, die durch p führt, die Werte von $x_1, \ldots, x_n$ im Definitionsbereich von f liegen.

Definition 3.9 In einem Programm werden zwei Arten von Variablenvorkommen unterschieden: *Benutzungen* und *Definitionen*. Eine Benutzung einer Variablen v in einem Programmpunkt p ist ein Vorkommen von v in p, wobei der Wert von v nicht modifiziert wird. Bei einer Definition einer Variablen v in einem Programmpunkt p wird der Wert von v in p modifiziert.

Definition 3.10 Eine Definition d einer Variablen v *erreicht* einen Programmpunkt p, wenn es einen Ausführungspfad von d nach p gibt, der außer d keine weiteren Definitionen von v enthält.

[3] Dies ist ein Unterschied zu der Theorie, wie sie in [PK82] dargestellt wurde. Siehe hierzu auch Abschnitt 11.1.1.

Definition 3.11 Eine Benutzung u einer Variablen v *lebt* in einem Programmpunkt p, wenn es einen Ausführungspfad von p nach u gibt, der frei von Definitionen von v ist.

Definition 3.12 Bei den Transformationen werden **achieve**-Anweisungen verwendet

achieve $V = f(x_1, \ldots, x_n)$;

die zur Zuweisung

$V := f(x_1, \ldots, x_n)$;

gleichwertig sind. Sie werden mitunter als *Initialisierung der virtuellen Variablen V* bezeichnet.

3.1.2 Die Ableitung und die Kettenregel

Gegeben sei ein applikativer Ausdruck $f(x_1, \ldots, x_n)$ und V sei die zu f gehörige virtuelle Variable. Weiterhin sei ein Block B gegeben, in dem sich Variablen x_i, von denen f abhängt, ändern können.

Definition 3.13 Es sei $V = f(x_1, \ldots, x_n)$ ein applikativer Ausdruck, der von den Variablen $x_1, \ldots, x_n$ abhängt und dx_i sei eine Definition der Variablen x_i. Das Paar $[B_1, B_2]$ von Blöcken B_1 und B_2 heißt *die Ableitung von V bezüglich* dx_i, falls

1. in den Blöcken B_1 und B_2 nur V und zu den Blöcken B_1 und B_2 lokale Variablen modifiziert werden und
2. der Block

   achieve $V = f(x_1, \ldots, x_n)$;
   B_1
   dx_i
   B_2
   `assert` $V = f(x_1, \ldots, x_n)$;

   semantik-treu ist für dx_i und nur redundante Benutzungen des applikativen Ausdrucks $f(x_1, \ldots, x_n)$ enthält.

Die Ableitung beschreibt also, wie sich der Wert von V durch die Modifikation von x_i ändert. Durch „**achieve** $V = f(x_1, \ldots, x_n)$“ ist V am Eingang von B_1 verfügbar und durch „`assert` $V = f(x_1, \ldots, x_n)$“ wird sichergestellt, daß V am Ausgang von B_2 verfügbar ist. Wenn keine Benutzung von V in B_1 oder B_2 am Eingang zu B_1 lebt, kann auf die **achieve**-Anweisung in Definition 3.13 verzichtet werden. In diesem Fall wird $[B_1, B_2]$ als eine *starke* Ableitung bezeichnet. Weiterhin wird B_1 als *Vorableitung von V bezüglich dx_i* bezeichnet, da sich die Vorkommen von x_i auf den (alten) Wert von x_i vor der Definition dx_i beziehen und B_2 als *Nachableitung von V bezüglich dx_i*, da sich die Vorkommen von x_i auf den (neuen) Wert von x_i nach der Definition dx_i beziehen. Entsprechend wird B_1 als $\partial^- V\langle dx_i\rangle$ und B_2 als $\partial^+ V\langle dx_i\rangle$ geschrieben.

Es bleibt noch anzumerken, daß die Ableitung nicht eindeutig bestimmt ist und daß die Vor- bzw. Nachableitung der leere Block sein kann.

Beispiel: „$V = \{x : x \text{ in } A \mid x \bmod 2 = 0\}$"[4] soll bezüglich „`A with := i`" differenziert werden. Es gilt

$$\partial^- V\langle \texttt{A with := i}\rangle \equiv \quad \begin{array}{l}\texttt{if i mod 2 = 0 then}\\ \quad\texttt{V with := i;}\\ \texttt{end if;}\end{array}$$

da das Element `i` in `V` enthalten ist, wenn es die Bedingung „`x mod 2 = 0`" erfüllt. $\partial^+ V\langle \texttt{A with := i}\rangle$ ist in diesem Beispiel der leere Block.

Weitere Beispiele befinden sich in [DF89].

Definition 3.14 Sei $V = f(x_1, \ldots, x_n)$ ein wohldefinierter applikativer Ausdruck in einem Block B eines regulären Programms P. V heißt *differenzierbar bezüglich B*,[5] falls gilt:

1. Keine Benutzung von V in B lebt in P.
2. Ist f am Eingang von B nicht wohldefiniert, beginnt B mit Anweisungen die f auswertbar machen und dann auswerten.

Definition 3.15 Sei V differenzierbar bezüglich B, dann wird das *Differential $\partial V\langle B\rangle$ von V bezüglich B* folgendermaßen definiert:

1. Ersetze jede Definition dx_i von Variablen x_i, von denen V abhängt, durch den Block:

   $\partial^- V\langle dx_i\rangle$
   dx_i
   $\partial^+ V\langle dx_i\rangle$

---

[4] „$V = \{x : x \text{ in } A \mid x \bmod 2 = 0\}$" ist die Invariante, die aufrecht erhalten werden soll und keine Anweisung.

[5] Siehe Abschnitt 11.1.1.

2. Ersetze jede Benutzung des applikativen Ausdrucks $f(x_1, \ldots, x_n)$ in dem durch Schritt 1 entstandenen Block durch die Variable V.

Anhand des Beispiels aus Abschnitt 1.1 läßt sich diese Definition nachvollziehen. Die Korrektkeit folgt aus

Satz 3.1 Es sei B ein Block in einem regulären Programm P und $V = f(x_1, \ldots, x_n)$ sei ein applikativer Ausdruck, der bezüglich B differenzierbar ist. Dann gilt:

- Gibt es eine Benutzung von V in $\partial V \langle B \rangle$, die am Eingang zu $\partial V \langle B \rangle$ lebt, dann ist der Block

  achieve $V = f(x_1, \ldots, x_n)$;
  $\partial V \langle B \rangle$

  semantik-treu für B.

- Lebt keine Benutzung von V am Eingang von $\partial V \langle B \rangle$, so ist $\partial V \langle B \rangle$ semantik-treu für B.

- V ist am Ausgang von $\partial V \langle B \rangle$ verfügbar.

Beweise dieses Satzes befinden sich in [PK82] und [DF89]. Die endliche Differentiation wurde bisher für einzelne Blöcke und einfache (nicht geschachtelte) applikative Ausdrücke vorgestellt. Im folgenden wird die Theorie auf Folgen von Blöcken und verschachtelte applikative Ausdrücke angewendet. Die Beweise der folgenden Sätze und Folgerungen befinden sich wieder in [PK82] und [DF89].

Folgerung 3.1 Die Transformation ∂V ist ein linearer Operator bezüglich sequentieller Blöcke, es gilt also

$$\partial V \langle B_1 B_2 \rangle = \partial V \langle B_1 \rangle \, \partial V \langle B_2 \rangle$$

Seien nun $f(x)$ und $g(y)$ einfache applikative Ausdrücke, so daß $V_1 = f(x)$ bezüglich dx differenzierbar ist und $V_2 = g(V_1)$ differenzierbar ist bezüglich Modifikationen von V_1 im Block $\partial V_1 \langle dx \rangle$. Dann zeigt sich, daß sich das Differential des verschachtelten applikativen Ausdrucks $g(f(x))$ bezüglich dx folgendermaßen ergibt:

$$\begin{aligned} \partial V_2 \langle \partial V_1 \langle dx \rangle \rangle = \quad & \partial V_2 \langle \partial^- V_1 \langle dx \rangle \rangle \\ & \partial^- V_2 \langle dx \rangle \\ & dx \\ & \partial^+ V_2 \langle dx \rangle \\ & \partial V_2 \langle \partial^+ V_1 \langle dx \rangle \rangle \end{aligned}$$

Allgemein läßt sich diese Beobachtung wie folgt formulieren:

Definition 3.16 Die Ausdrücke $f_1, \ldots, f_n$ heißen von *innen nach außen geordnet*, falls gilt: hängt f_i von f_j ab, so muß $i < j$ sein.

Definition 3.17 Es seien n von innen nach außen geordnete applikative Ausdrücke $V_1 = f_1, \ldots, V_n = f_n$ und ein Block B in einem regulären Programm P gegeben. V_1 sei differenzierbar bezüglich B, V_2 sei differenzierbar bezüglich $\partial V_1 \langle B \rangle$, ..., V_n sei differenzierbar bezüglich $\partial V_{n-1} \langle \ldots \langle \partial V_1 \langle B \rangle \rangle \ldots \rangle$. Dann formen $V_n, \ldots, V_1$ eine *differenzierbare Kette* und das *erweiterte Differential* dieser Kette bezüglich B ist wie folgt definiert:

$$\partial V_n, V_{n-1}, \ldots, V_1 \langle B \rangle = \partial V_n, V_{n-1}, \ldots, V_2 \langle \partial V_1 \langle B \rangle \rangle$$

Satz 3.2 Sei nun $V_n = f_n, \ldots, V_1 = f_1$ eine differenzierbare Kette von n applikativen Ausdrücken, die bezüglich eines Blocks B in einem regulären Programm P differenzierbar sind. Sei S eine Menge von Indizes $i = 1, \ldots, n$ für die Benutzungen von V_i in $B' = \partial V_n, \ldots, V_1 \langle B \rangle$ am Eingang zu B' leben. Dann ist der Block

achieve $\forall i \in S : V_i = f_i$;
$\partial V_n, \ldots, V_1 \langle B \rangle$

semantik-treu für B und $V_1, \ldots, V_n$ sind am Ausgang von B' verfügbar.

Folgerung 3.2 Das erweiterte Differential ist ein linearer Operator bezüglich sequentieller Blöcke:

$$\partial V_n, \ldots, V_1 \langle B_1 B_2 \rangle = \partial V_n, \ldots, V_1 \langle B_1 \rangle \, \partial V_n, \ldots, V_1 \langle B_2 \rangle$$

In [PK82, DF89] befindet sich ein Formalismus, der die Berechnung des erweiterten Differentials auf die Berechnung des (einfachen) Differentials zurückführt. In diesen Artikeln und im Kapitel 9 befinden sich Beispiele für das erweiterte Differential.

3.1.3 Profitabilität

Nachdem nun ein wesentlicher Teil der Theorie vorgestellt wurde, ist es an der Zeit, sich mit der Profitabilität der endlichen Differentiation zu beschäftigen.

Endliche Differentiation lohnt sich nur dann, wenn sichergestellt ist, daß der differenzierte Code effizienter ist als der ursprüngliche Code. Für die Berechnung der Kosten wird in dieser Arbeit das heuristische Kostenmaß aus [PK82, DF89] benutzt. Dieses Kostenmaß verwendet für die einzelnen Operationen die kleinsten Kosten, die erfahrungsgemäß durch geeignete Datentypen und Algorithmen erreicht werden können. Auf eine vollständige Auflistung der Kostentabelle wird hier verzichtet. Für das weiter unten aufgeführte Beispiel reichen die folgenden zwei Kosten aus:

- Einfache Zuweisungen wie „`S with := x`“ verursachen Kosten der Ordnung $O(1)$.
- Für „`{x : x in S | k(x)}`“ gilt die Ordnung $O(\sharp\mathtt{S} \times \mathit{Kosten}(\mathtt{k}))$.

Bei diesem Kostenmaß wird von effizienten Hashtabellen-Implementierungen für Mengen ausgegangen.

Definition 3.18 Ein Ausdruck V heißt bezüglich eines Blocks B *profitabel differenzierbar*, wenn die Ausführung von B im Sinne des oben zitierten Kostenmaßes teurer ist als die Ausführung von $\partial V \langle B \rangle$.

Um die Profitabilität der Transformation sicherzustellen, wird die endliche Differentiation in [PK82] daher nur auf solche Ausdrücke $f(x_1, \ldots, x_n)$ in Blöcke B beschränkt, für die gilt

- jeder differenzierte Block für f ist profitabel differenzierbar, und
- in B ist die Anzahl der Definitionen von Variablen x_i relativ klein (im Sinne von R. Paige) zur Anzahl der Benutzungen von f.

Bei diesen Beschränkungen handelt es sich also um pragmatische Aspekte. Anhand von zwei Beispielen werden diese Beschränkungen im folgenden erläutert.

Beispiel 3.1 Der folgende Block wird bezüglich „`{x : x in A | x mod 2 = 0}`“ differenziert. Als virtuelle Variable wird `V` verwendet.

```
A := {};
get(i);
while i /= om do
   A with := i;
   get(i);
end while;
put({x : x in A | x mod 2 = 0});
```

Die Herleitung des differenzierten Blocks befindet sich in [PK82] und [DF89]. Der differenzierte Block lautet:

```
V := {};
A := {};
get(i);
while i /= om do
   if i mod 2 = 0 then
      V with := i;
```

```
    end if;
    A with := i;
    get(i);
  end while;
  put(V);
```

Bei der Berechnung der Kosten für dieses Programm werden für jede Zuweisung, für die Ein- und Ausgabeoperationen und für die Berechnung von „`i mod 2 = 0`“ Kosten der Ordnung $O(1)$ angenommen. Weiterhin sei I die Anzahl der eingelesenen `i` (`i /= om`). Im ursprünglichen Block befinden sich insgesamt $3 \times I + 2$ Operationen mit Kosten der Ordnung $O(1)$, während sich im differenzierten Block $3 \times I + 4$ Operationen mit Kosten der Ordnung $O(1)$ befinden.

In diesem Beispiel wurde also durch Anwendung der endlichen Differentiation eine Erhöhung der Kosten verursacht, obwohl bei der Berechnung der Kosten im differenzierten Block davon ausgegangen wurde, daß die Anweisung „`V with := i`“ nie erreicht wird.

Der Block erfüllt die zweite Bedingung der obigen Auflistung (Anzahl der Definitionen relativ klein zur Anzahl der Benutzungen) nicht. Es gibt I Modifikationen in der Schleife und nur eine Benutzung des differenzierten Ausdrucks. Diese Beobachtung führt zu der Überlegung, die endliche Differentiation nur innerhalb von Schleifen durchzuführen, da hier das Verhältnis von Definitionen zu Benutzungen im allgemeinen kleiner ist, als außerhalb von Schleifen. Diese Feststellung entspricht auch den Vorstellungen von R. Paige:

> „... Paige mainly treated differencing as a loop optimization, in which efficient set theoretic derivativs and the loop boundedness requirement promised speedup under the standard assumption that code is executed more frequently inside loops than outside.“[PK82, Seite 418f]

Beispiel 3.2 Die Notwendigkeit der ersten Bedingung (jeder differenzierte Block für f ist profitabel differenzierbar) wird an der Ableitung des applikativen Ausdrucks

$$\{\texttt{[x,y] : [x,y] in G | y in Q}\}$$

bezüglich „`Q with := z`“ belegt. Als virtuelle Variable wird `V` verwendet. Es ist leicht einzusehen, daß bei dieser Modifikation in `V` alle Tupel `[x,z]` eingefügt werden müssen, die die Eigenschaft haben, daß `x` im Definitionsbereich von `G` liegt und, daß das Tupel `[x,z]` in `G` enthalten ist. Die Vorableitung von `V` bezüglich „`Q with := z`“ sieht dann wie folgt aus:

$$\partial^- V \, \langle \texttt{Q with := x} \rangle \equiv \begin{array}{l} \texttt{for u in \{x : x in domain G | z in G\{x\}\} do} \\ \quad \texttt{V\{u\} with := z;} \\ \texttt{end if;} \end{array}$$

Die Nachableitung ist der leere Block. Diese Ableitung enthält wieder einen Mengenformer. Die Berechnung der dazugehörigen Menge kann in Abhängigkeit von `G` und `Q` teurer sein, als die Berechnung der ursprünglichen Menge (z.B. wenn `Q` sehr viel kleiner ist als `G`). Die Bedingung, das `V` profitabel differenzierbar ist, ist bei dieser Ableitung also nicht unbedingt erfüllt. Es kann jedoch versucht werden, den neuen Mengenformer zu differenzieren und dann die Kosten mit dem ursprünglichen Block zu vergleichen. Falls auch dann nicht profitabel differenziert werden kann, sollte in diesem Fall auf eine Differentiation verzichtet werden. Diese Ableitung befindet sich im Anhang des Artikels [PK82]. Dort und in [PK80] befinden sich noch weitere Beispiele dieser Art.

Eine weitere Verringerung der Kosten läßt sich erreichen, wenn nur auf *strikten* Operationen gearbeiten wird. Somit kann zum Beispiel bei der Operation

```
S with := x;
```

vorausgesetzt werden, daß `x` nicht schon in `S` enthalten ist. Dies läßt sich erreichen, indem die Zuweisung „`S with := x`“ durch

```
if x notin S then
   S with := x;
end if;
```

ersetzt wird. Dies ist vor allem bei komplexeren Ableitungen von Vorteil. In der obigen Ableitung von `V` bezüglich „`Q with := z`“ müßte die Menge in der Ableitung nur dann berechnet werden, wenn `z` noch nicht in `Q` enthalten ist.

3.1.4 Vertikale und horizontale Verschmelzung von Schleifen

Die **achieve**-Anweisungen müssen noch erläutert werden. Durch die endliche Differentiation kann vor dem (erweiterten) Differential eine Sequenz von **achieve**-Anweisungen entstehen. In diesen **achieve**-Anweisungen erhalten die virtuellen Variablen V_i die Werte der applikativen Ausdrücke $f_i(x_1, \ldots, x_n)$. Hier müssen also die applikativen Ausdrücke ausgewertet werden. Unter bestimmten Voraussetzungen lassen sich dabei weitere Kosten einsparen. Dieses Vorgehen wird in diesem Abschnitt kurz beschrieben. Eine ausführlichere Beschreibung des zugrundliegenden Konzepts befindet sich wieder in [PK82] und [DF89].

Gegeben seien die beiden Mengen

```
A := {x : x in S | k(x)};
B := {x : x in A | m(x)};
```

Da für die Berechnung der Menge `B` über `A` iteriert werden muß, ist es sinnvoll die Berechnung der Mengen `A` und `B` mit Hilfe der Vorableitung zu *verschmelzen*. Als Ergebnis erhält man:

```
A := {};
B := {};
for x in S do
   if k(x) then
      if m(x) then
         B with := x;
      end if;
      A with := x;
   end if;
end for;
```

Durch diese *vertikale Verschmelzung* der (impliziten) Schleife, braucht nicht über die Menge `A` iteriert zu werden. Dieses Prinzip läßt sich auch auf Mengen anwenden, die die gleiche Grundmenge verwenden.

```
A := {x : x in S | k(x)};
B := {x : x in S | m(x)};
```

Dabei ergibt sich:

```
A := {};
B := {};
for x in S do
   if k(x) then A with := x; end if;
   if m(x) then B with := x; end if;
end for;
```

In diesem Fall wird von einer *horizontalen Verschmelzung* gesprochen. Es ist jedoch nicht immer sinnvoll, eine Verschmelzung von Schleifen durchzuführen. In [PK82, DF89] befinden sich Beispiele (u.a. die Berechnung des Zentrums eines freien Baumes), in denen das Resultat der Verschmelzung ineffizienter wäre, als der ursprüngliche Code.

In [PK82] werden Heuristiken angegeben, die beschreiben, unter welchen Bedingungen eine Verschmelzung erfolgen kann. Eine dieser Heuristiken ist folgende:

> „[...] For each elementary expression $f(x_1, \ldots, x_n)$ we only allow f to be initialized differentially (or by separate expansion) with respect to certain of its parameters, called "expandable" parameters, for which the technique is most likely to be profitable.“[PK82, Seite 438]

Es ist nicht bekannt, ob es einen Algorithmus gibt, der entscheidet, ob eine Verschmelzung durchgeführt werden kann oder nicht. Ein Vorteil der vertikalen Verschmelzung ist, daß Abhängigkeiten innerhalb der virtuellen Variablen, die durch die endliche Differentiation entstanden sind, aufgelöst werden. Durch eine Datenflußanalyse können dann Teile des differenzierten Codes eleminiert werden, die außerhalb des differenzierten Codes nicht benötigt werden. Dies wird im Beispiel zur Berechnung des Zentrums einen freien Baumes in [PK82, Seite 429ff] und [DF89, Seite 174ff] deutlich.

Die **achieve**-Anweisungen dienen dazu, die Programmblöcke, in denen eine Verschmelzung möglich ist, zu lokalisieren.

3.2 Das Programmtransformationssystem RAPTS

R. Paige hat auf der Grundlage der endlichen Differentiation das Programmtransformationssystem RAPTS (Rutgers Abstract Program Transformation System) für die Sprache SQ+ entwickelt [Läu91]. SQ+ ist eine abstrakte funktionale Sprache, die auf der endlichen Mengenlehre basiert. In [Läu91] wird sie als eine Teilmenge der Sprache SETL bezeichnet, die um Fixpunkt-Operationen erweitert wurde. Die endliche Differentiation ist die zweite von drei Phasen des RAPTS-Compilers. Sie wird auf Ausdrücke in iterativer Fixpunkt-Form angewendet [CP88]. Der Compiler erzeugt C-Code. Die Basis der endlichen Differentiation in RAPTS ist eine Sammlung von Ableitungsregeln, die für jede Modifikation eines applikativen Ausdrucks die Vor- und Nachableitung enthält. In [Pai84] wird RAPTS ausführlich beschrieben.

3.3 Anwendung der endlichen Differentiation im ProSet-Compiler

In den vorherigen Abschnitten wurde die endliche Differentiation, wie sie von R. Paige und S. Koenig entwickelt wurde, vorgestellt. In diesem Abschnitt geht es nun darum, wie die endliche Differentiation in den ProSet-Compiler integriert werden kann.

Zuerst muß der Begriff der regulären Ausführung erweitert werden. In ProSet besteht die Möglichkeit den Programmablauf durch Ausnahmen zu steuern. Eine Ausnahme kann Auswirkungen auf einen applikativen Ausdruck haben (Seiteneffekte), so daß die entsprechende Invariante nicht aufrechterhalten werden kann. Der Begriff der regulären Ausführung wird dahingehend erweitert, daß während der Ausführung keine Ausnahme ausgelöst werden darf:

Definition 3.19 Die Ausführung eines Programms heißt ProSet-*regulär*, falls aus den Tatsachen, daß die Bedingungen aller `assert`-Anweisungen erfüllt sind und während der Ausführung keine Ausnahme ausgelöst wird, folgt, daß das Programm normal terminiert.

Ein Programm heißt ProSet-*regulär*, falls alle seine möglichen Ausführungen ProSet-regulär sind.

Weiterhin wird die endliche Differentiation auf Schleifen beschränkt. Es wurde bereits festgestellt, daß die effektivste Einsatzmöglichkeit der endlichen Differentiation Schleifenoptimierung ist. Im Transformationsprogramm für ProSet wird auf die Verschmelzung von Schleifen verzichtet, da hierfür kein Algorithmus bekannt ist. Die Umwandlung einer impliziten Schleife in eine explizite Schleife geschieht in einer späteren Phase des aktuellen ProSet-Compilers, so daß im Transformationsprogramm für die `achieve`-Anweisung die Zuweisung verwendet werden kann.

R. Paige gibt in [PK82] für die applikativen Ausdrücke Ableitungen für die Zuweisung der leeren Menge („`x := {}`") an. In ProSet entstehen dabei Probleme mit der Eigenschaft, daß ein applikativer Ausdruck wohldefiniert sein muß, um transformiert werden zu können (Definition 3.8 auf Seite 15). Im Beispiel in Abbildung 3.1 kann die virtuelle Variable `V` nicht vor der `repeat`-Schleife durch „`V := {x : x in A | x in B}`" initialisiert werden, da `A` an dieser Stelle keine Menge sein muß. Die virtuelle Variable kann also erst nach der Zuweisung der leeren Menge an `A` berechnet werden. Für die Ableitung der Definition „`B with:= i`" wird jedoch bereits die virtuelle Variable benötigt. Dieses Problem tritt auf, da ProSet schwach getypt ist und daher keine Möglichkeit besteht, zur Übersetzungszeit zu überprüfen, ob ein Ausdruck wohldefiniert ist. Bei den anderen Definitionen, die R. Paige für die applikativen Ausdrücke betrachtet, wird der Typ der Variablen nicht durch die Definition verändert (z.B. „`B with:= i`" oder „`f(x) := y`"). Im Transformationsprogramm muß deshalb auf die Differentiation der Zuweisung der leeren Menge verzichtet werden.

```
program beispiel;
begin
   get(A);    -- Typ von A zur Uebersetzungszeit unbekannt
   B := {1, 1+3 .. 50};
   C := {1 .. 100};
   repeat
      i from C;
      B with:= i;
      A := {}; -- Typ von A ist hier Menge
      put({x : x in A | x in B});
   until (C = {});
end beispiel;
```

Abbildung 3.1: Ein ProSet-Programm, das nicht transformiert werden kann.

Nach Definition 3.14 auf Seite 17 müßte überprüft werden, ob die virtuelle Variable

`V` bezüglich eines Blocks differenzierbar ist. Auf die Überprüfung kann verzichtet werden, da für die applikativen Ausdrücke neue Variablen verwendet werden (siehe Abschnitt 11.1) und der applikative Ausdruck $f(x_1, \ldots, x_n)$ am Eingang zur Schleife wohldefiniert sein muß. Andernfalls würde für eine freie Variable dieses Ausdrucks innerhalb der Schleife eine Zuweisung verwendet, die den alten Wert dieser Variablen nicht berücksichtigt. In diesem Fall wäre eine Transformation nicht möglich.

Eine Umwandlung von Operationen in strikte Operationen (Seite 22) kann nicht durchgeführt werden. Im Programm in Abbildung 3.2 würde es zu einem Laufzeitfehler kommen, da `A` ein Tupel ist, und somit Werte mehrfach vorkommen dürfen. In Abbildung 3.3 befindet sich das transformierte Programm, bei dem die Operation „`A with := x`“ in eine strikte Operation umgewandelt wurde.

```
program beispiel;
begin
   A := [1,2,3,4,5];
   B := {1,2,5};
   C := {1 .. 20};
   whilefound x in C | x < 6 do
      V := {i : i in A | i notin B};
      A with := x;
      C less := x;
   end whilefound;
   put(A); -- [1,2,3,4,5,5,1,2,3,4]
end beispiel;
```

Abbildung 3.2: Ein ProSet-Programm.

Anders als in RAPTS wird das Transformationsprogramm auf die endliche Differentiation beschränkt. ProSet bietet keine expliziten Fixpunktausdrücke, wie sie aus SQ+ [CP88] bekannt sind, an. Es besteht daher für das Transformationsprogramm nicht die Möglichkeit, auf Fixpunktausdrücken zu arbeiten. Das Transformationsprogramm soll in den ProSet-Compiler integriert werden und muß für die nachfolgende Phase ein ProSet-Programm zur Verfügung stellen.

Vergleichbar mit RAPTS wird das Transformationsprogramm auf einer Sammlung von Ableitungsregeln aufgebaut. Dies scheint aufgrund der Theorie die effektivste Vorgehensweise zu sein. Zum einen hängt die Differentiation eines applikativen Ausdrucks entscheidend vom Ausdruck und der darauf stattfindenden Modifikation ab. Zum anderen ermöglicht dieses Vorgehen, das Transformationsprogramm beliebig um neue applikative Ausdrücke zu erweitern. Im nächsten Kapitel wird die Implementierung dieses

```
program beispiel ;
begin
   A := [ 1 , 2 , 3 , 4 , 5 ] ;
   B := { 1 , 2 , 5 } ;
   C := { 1 .. 20 } ;
   pstt1 := { i : i in A | i notin B };
   whilefound x in C | x < 6 do
      V := pstt1;
      if (x notin A) then
         if (x notin B) then
            pstt1 with := x;
         end if;
         A with := x;
      end if;
      C less := x;
   end whilefound;
   put(A);  -- [1,2,3,4,5]
end beispiel;
```

Abbildung 3.3: Ein ProSet-Programm, das nicht semantik-treu ist zum Programm in Abbildung 3.2.

Programms beschrieben. Anstelle des Ausdrucks „Sammlung der Ableitungsregeln“ werden dabei die kürzeren Begriffe *Muster* oder *Transformationsmuster* verwendet.

4. Zusammenfassung zu Teil I

In Kapitel 2 wurde ein kurzer Einblick in die Prototyping-Sprache ProSet gegeben. Es wurden kurz die für diese Arbeit wichtigen Datentypen und Kontrollstrukturen vorgestellt. In Kapitel 3 wurde die endliche Differentiation beschrieben und die Anwendung der endlichen Differentiation im ProSet-Compiler diskutiert.

Teil II

Die Implementierung

5. Anforderungen an das Transformationsprogramm

Die Arbeitsweise des Transformationsprogramms wurde bereits im letzten Abschnitt dargelegt, in diesem Kapitel werden die Anforderungen präzisiert. Das Transformationsprogramm soll ein ProSet-Programm einlesen, es entsprechend der endlichen Differentiation transformieren und wieder ein ProSet-Programm ausgeben. Beim Transformieren gelten die Einschränkungen aus Abschnitt 3.3. Für die Transformation eines ProSet-Programms muß ein Übersetzer konstruiert werden. Dieser Übersetzer soll auf Transformationsmustern basieren. Es ist wünschenswert, daß neue Muster (relativ) einfach in das Transformationsprogramm eingefügt werden können. Da das Transformationsprogramm ein ProSet-Programm erzeugt, sollte die Formatierung des eingegebenen Programms weitgehend erhalten bleiben, damit nichttransformierte Ausdrücke leicht zu erkennen sind und eventuell als neue Muster in das Transformationsprogramm eingebaut werden können.

In diesem Teil der Arbeit wird nun die Implementierung des Übersetzers beschrieben. Zunächst wird das für den Übersetzer verwendete Werkzeug vorgestellt (Kapitel 6). Die Beschreibung der Implementierung gliedert sich in zwei Teile. Zuerst wird die Implementierung allgemein beschrieben (Kapitel 7). Dann wird diskutiert, wie das Transformationsprogramm um die Erkennung und Transformation eines neuen applikativen Ausdrucks erweitert werden kann (Kapitel 8). Abschließend werden die Ergebnisse einiger Laufzeittests angegeben (Kapitel 9).

Verweise, die mit „C.“ beginnen, beziehen sich auf den literalen Quellcode in Anhang C und beinhalten die Beschreibung des entsprechenden Programmcodes.

6. Der GMD Werkzeugkasten für den Übersetzerbau

Die Implementierung des ProSet-Compilers wurde mit dem Übersetzerbauwerkzeugkasten *Eli* [GHL+92] entwickelt. Für das Transformationsprogramm wird jedoch der „GMD Werkzeugkasten für den Übersetzerbau“ [GE90a, Gro91d], der auch unter dem Namen *Cocktail* bekannt ist, verwendet. *Cocktail* wird verwendet, da dieser Werkzeugkasten, anders als *Eli*, ein Werkzeug enthält, das die Mustererkennung auf Bäumen unterstützt. Die Mustererkennung wird im Transformationsprogramm für das Erkennen der applikativen Ausdrücke (Muster) in zu transformierenden Programmen benötigt.

Der Werkzeugkasten enthält unter anderem die folgenden Werkzeuge:

rex	ein Scanner-Generator (siehe Abschnitt 6.1)
lalr	ein LALR(1)[ASU86] Parser-Generator (siehe Abschnitt 6.2)
cg, *rpp*	Präprozessoren für die Erzeugung von Scanner und Parser (Abschnitt 6.3)
ast	ein Generator für abstrakte Syntaxbäume (siehe Abschnitt 6.4.1)
ag	ein Generator für Attributauswerter (siehe Abschnitt 6.4.2)
puma	ein Werkzeug für die Mustererkennung und -transformation auf Bäumen (siehe Abschnitt 6.5)
Reuse	eine Bibliothek von Funktionen (siehe Abschnitt 6.6)

Die beschriebenen Werkzeuge erzeugen C- oder Modula-2 Code. Für die Implementierung des Transformationsprogramms wurde die Sprache C verwendet.

Die einzelnen Werkzeuge werden nun kurz vorgestellt.

6.1 Der Scanner-Generator *rex*

Der Scanner wird bei der lexikalischen Analyse eines Programms eingesetzt. Er muß daher die Token der Sprache (Bezeichner, Schlüsselwörter usw.), spezifiziert durch reguläre Ausdrücke, kennen [ASU86]. Wird ein Token erkannt, werden die bei diesem Token angegebenen Aktionen durchgeführt und die interne Darstellung an den Parser weitergeleitet.

Der Aufruf

`rex -cd` *file*

verwendet *file* als Eingabedatei und erzeugt die Dateien `Scanner.c` und `Scanner.h`. Die erzeugten Dateien enthalten die Funktionen „`BeginFile`", „`EndFile`" und „`GetToken`". „`BeginFile`" öffnet die Eingabedatei, aus der die Funktion „`GetToken`" das nächste Token erkennt und die für das Token angegebenen Aktionen ausführt.

Weitere Informationen zum Scanner befinden sich in [Gro92d].

6.2 Der LALR(1) Parser-Generator *lalr*

Der Parser wird für die syntaktische Analyse eines Programms verwendet und baut den abstrakten Syntaxbaum auf. Er benötigt die kontextfreie Grammatik der Sprache in EBNF Notation.

Der Aufruf

`lalr -c -d` *file*

erzeugt aus der Datei *file* die Dateien `Parser.c` und `Parser.h`. Der erzeugte Parser wird durch die Funktion „`Parser()`" gestartet, die durch Aufruf der Scanner-Funktion „`GetToken`" die interne Darstellung der eingelesenen Token erhält. Der erzeugte Parser enthält auch eine automatische Fehlerkorrektur.

Ausführlichere Beschreibungen des Parsers befinden sich in den Artikeln [GV92] und [Gro88a].

6.3 Die Präprozessoren *cg* und *rpp*

Damit die Angabe der Token und ihrer internen Darstellung im Scanner und im Parser erleichert wird, gibt es die Präprozessoren *cg* und *rpp* [Gro92b]. Die konkrete Grammatik wird hierbei nicht in EBNF Notation spezifiziert, sondern in der Spezifikationssprache, die von den Werkzeugen *ast* und *ag* verwendet wird (Abschnitt 6.4).

Der Aufruf

`cg -cxzj` *file*

erzeugt aus einer Grammatik-Spezifikation in der im Abschnitt 6.4 erläuterten Notation eine Grammatik-Spezifikation in EBNF Notation. Die Ausgabe wird in die Datei

`Parser.lalr` geschrieben. Mit der Datei `Parser.lalr` wird der Parser-Generator aufgerufen. Weiterhin liefert dieser Präprozessor die regulären Ausdrücke und Aktionen für die Scanner-Spezifikation und schreibt diese in die Datei `Scanner.rpp`.

Der Aufruf

```
rpp Scanner.rpp < Scanner.scan > Scanner.rex
```

schreibt in die Datei `Scanner.rex` die Eingabe für den Scanner Generator, falls in der Datei `Scanner.scan` die restliche Scanner-Spezifikation enthalten ist. Dieser Rest umfaßt die Beschreibung der benutzerdefinierten Token (zum Beispiel Bezeichner, Kommentare) und ihre Aktionen.

6.4 Die Spezifikationssprache der Werkzeuge *ast* und *ag*

Die Spezifikationssprache für *ast* und *ag* wird verwendet, um konkrete und abstrakte Grammatiken [WG85, Seite 17 und 86ff] zu spezifizieren. Die abstrakte Grammatik kann eine Vereinfachung der konkreten Grammatik sein. Zum Beispiel können die Schlüsselwörter fehlen oder auch mehrere Nichtterminals der konkreten Grammatik zu einem Nichtterminal der abstrakten Grammatik zusammengefaßt sein. In der semantischen Analyse und der Code-Erzeugung wird mit der abstrakten Grammatik gearbeitet.

Die Nichtterminals und Terminals der konkreten Grammatik (Knotentypen) werden durch die Angabe der Zeichen „=“ und „:“ unterschieden.

Beispiel:

```
IF = .
Ident : .
```

„`IF`“ wird hier als Nichtterminal und „`Ident`“ als Terminal definiert. Die Unterscheidung zwischen Nichtterminals und Terminals ist bei der Angabe der abstrakten Grammatik nicht notwendig und es wird bei allen Regeln das „=“ verwendet. Die Namen aller Knotentypen müssen paarweise verschieden sein. Eine Regel wird durch „`.`“ beendet. Die Knotentypen auf der rechten Seite einer Regel (Nachfolger im Baum) müssen ebenfalls verschieden sein. Dies kann durch Angabe einer eindeutigen Marke erreicht werden. Im folgenden Beispiel sind „`then`“ und „`else`“ Marken:

Beispiel:

```
if = Expr then:Stats else:Stats.
```

Es kommt häufig vor, daß für einen Knotentyp mehrere Alternativen angegeben werden müssen. Diese müssen mit Hilfe eines Erweiterungsmechanismus in eine Regel gefaßt werden:

Beispiel:

```
Stats    = <
  while  = Expr Stats .
  repeat = Stats Expr .
> .
```

„`Stats`“ wird als Basistyp bezeichnet und die zwischen „`<`“ und „`>`“ angegeben Knotentypen als Subtypen. Die Angabe der Namen der Subtypen (hier „`while`“ und „`repeat`“) ist optional. Sie werden jedoch benötigt, wenn explizit auf einen Subtyp zugegriffen werden soll. Die Namen der Subtypen müssen ebenfalls paarweise verschieden sein.

Weiterhin können für einen Knotentyp Attribute angegeben werden. Diese Attribute werden durch einen eindeutigen Namen und einen Typ charakterisiert. Die Beschreibung der Attribute wird in eckigen Klammern angegeben („`[`“ und „`]`“). An Typen kann jeder gültige C-Typ (auch Benutzerdefinierte) und Zeiger auf Knotentypen verwendet werden. Die Attribute können auch Eigenschaften (`INPUT`, `OUTPUT`, `THREAD`, ...) erhalten, die Aussagen über ihre Erzeugung oder die Dauer ihrer Existenz machen. Durch ein `THREAD` Attribut (Kettenattribut) kann die Auswertungsreihenfolge der Nachfolger vorgegeben werden [Kas, Abschnitt 3.3]. Zum Beispiel kann damit erreicht werden, daß ein Baum in Pre-, In- oder Postorder durchlaufen wird. Die Berechnung der Attribute wird in geschweiften Klammerpaaren „{“ und „}“ angegeben. Es existieren verschiedene Möglichkeiten, Attributberechnungen durchzuführen. Sie werden in [Gro91a] beschrieben. Die Beschreibung der Knotentypen und Attribute kann in mehrere Module aufgeteilt werden. Mit dem Befehl

```
SELECT Module1 Module2
```

werden die Module `Module1` und `Module2` aus den vorhandenen Moduln einer Datei ausgewählt. Eine vollständige Beschreibung der Spezifikation wird in [Gro92a] und [Gro91a] gegeben.

6.4.1 Der Generator für abstrakte Syntaxbäume *ast*

Ein abstrakter Syntaxbaum wird bei der Übersetzung eines Programms verwendet. Der Generator *ast* erwartet eine Eingabedatei in der oben spezifizierten Sprache. *ast* erzeugt aus der angegebenen (abstrakten) Grammatik einen abstrakten Datentyp, mit dem der abstrakte Syntaxbaum aufgebaut wird.

Der Aufruf

`ast -cdim` *file*

erzeugt die Dateien `Tree.c` und `Tree.h`. In diesen Dateien werden die Knotentypen als C-Strukturen dargestellt. Für jeden Knotentyp wird eine Prozedur „`m<Knotentyp>`" erzeugt, die zur Baumkonstruktion während der syntaktischen Analyse verwendet wird. Weiterhin existiert der Typ „`tTree`", der ein Zeiger auf die Baumstruktur ist. Zu diesem Typ gehört die Variable „`TreeRoot`", die als Wurzel eines Baumes dienen kann, und die Konstante „`NoTree`". Der Generator *ast* ist ausführlich in [Gro92a] beschrieben.

6.4.2 Der Generator für Attributauswerter *ag*

Attributauswerter werden ebenfalls bei der Übersetzung eines Programms verwendet. Sie werden unter anderem für die semantische Analyse eingesetzt. Ein erzeugter Attributauswerter arbeitet auf dem abstrakten Syntaxbaum. Der Generator *ag* erwartet das Schlüsselwort „`EVAL`". Es besteht die Möglichkeit mit dem Befehl „`PROPERTY`" allen Attributen eines Moduls eine Eigenschaft zuzuweisen.

Der Aufruf

`ag -cDI` *file*

erzeugt die Dateien `Eval.c` und `Eval.h`, falls hinter dem Schlüsselwort „`EVAL`" kein Name angegeben wurde. Andernfalls wird der dort angegebene Name als Basisname der Ausgabedateien verwendet. Durch den Aufruf der Funktion „`Eval(tTree t)`" in einem C-Programm wird der Attributauswerter „`Eval`" gestartet, der entsprechend den spezifizierten Regeln die Attribute auswertet. Eine ausführliche Beschreibung dieses Generators befindet sich in [Gro91a].

6.5 Das Transformationswerkzeug *puma*

puma erzeugt Funktionen, die Muster im abstrakten Syntaxbaum erkennen und gegebenenfalls transformieren. Die Eingabedatei beginnt mit dem Schlüsselwort „`TRAFO`" und mit der Angabe des Namens der zu erzeugenden Datei und dem Namen des zugrundeliegenden abstrakten Syntaxbaumes:

```
TRAFO Name1 TREE Treename
```

Dann werden die Routinen, die die Mustererkennung auf Bäumen durchführen, angegeben. Es gibt verschiedene Arten von Routinen. Für die Implementierung des Transformationsprogramms wurden Prozeduren (`PROCEDURE`) und Funktionen (`FUNCTION`) verwendet. Anhand der Prozedur in Abbildung 6.1 soll die Arbeitsweise von *puma* dargestellt werden. Die erste Zeile stellt den Routinenkopf dar. Hinter dem Schlüsselwort

```
PROCEDURE getscope(scope:Scope,now:xPHM,number:int) (* Zeile 1 *)

   _,1xPHM,5                                        (* Zeile 2 *)
   ? {scope->names = idphmlist(scope->names,Id);    (* Zeile 3 *)
     getscope(scope,now);}; .                       (* Zeile 4 *)

   _,2xPHM(xId(_,Id),xBody:xBody(N:xPHM,env)),3     (* Zeile 5 *)
   ? {scope->names = idphmlist(scope->names,Id);    (* Zeile 6 *)
     getscope(env,N);}; .                           (* Zeile 7 *)

   _,_,_                                            (* Zeile 8 *)
   ? nextscope(); .                                 (* Zeile 9 *)
```

Abbildung 6.1: Eine Prozedur für das Werkzeug *puma*.

„`PROCEDURE`" wird der Name der Routine, gefolgt von den formalen Parametern, angegeben. Für die Parameter können alle gültigen C-Typen, Knoten- und Baumtypen verwendet werden. Der Name eines Parameters ist seinem Typ vorangestellt. Name und Typ werden durch „`:`" voneinander getrennt. Die Zeilen 2 bis 4, 5 bis 6 und 8 bis 9 stellen jeweils Regeln dar. Die aktuellen Parameter werden zuerst mit dem Pattern[1] der ersten Regel (`Zeile 2`) verglichen. Ein Vergleich trifft zu, wenn ein Parameter mit dem beim entsprechenden Parameter im Pattern angegebenen Typ oder Wert übereinstimmt. Ein „_" wird benutzt, wenn für die Regel der entsprechende Parameter nicht verwendet wird (*Wildcard*). Im obigen Beispiel paßt „`scope`" immer, da an der ersten Stelle in den Pattern ein „_" angegeben wurde. Der zweite Parameter paßt, wenn „`now`" den Typ „`1xPHM`" hat. Beim dritten Parameter muß „`number`" den Wert „`5`" haben. Stimmen alle Parameter mit den Parametern im Pattern überein, werden die Anweisungen hinter dem Fragezeichen ausgeführt und die Prozedur beendet. Diese Anweisungen können C-Anweisungen, Aufrufe von *puma*-Funktionen oder Modifikationen der aktuellen Parameter sein. Die Anweisungen sollten in geschweifte Klammern eingeschlossen werden, wenn C-Code verwendet wird. Stimmt ein aktueller Parameter nicht mit dem entsprechenden Parameter des Pattern überein, werden die Parameter mit dem Pattern der nächsten Regel verglichen.

Bei der zweiten Regel wurde beim zweiten Parameter des Pattern nicht nur der Typ „`2xPHM`" angegeben, sondern in Klammern auch die Nachfolger und Attribute des Knotentyps „`2xPHM`". Hierdurch kann das Pattern genauer spezifiziert werden, da auch für die Nachfolger und Attribute bestimmte Knotentypen und Werte vorausgesetzt werden

[1] Der Begriff „Pattern" wird verwendet, um eine deutliche Trennung zwischen Transformationsmuster und Grammatikmuster (Pattern), zu haben.

können (z.B. „`xId`"). Weiterhin ist es dadurch möglich, in den Anweisungen auf die Werte der Nachfolger und Attribute zuzugreifen (z.B. „`Id`"). Die Anweisung der dritten Regel dieses Beispiels wird immer ausgeführt, da bei dieser Regel kein spezielles Pattern spezifiziert wurde. Wird keine Regel ausgewählt, passiert nichts.

Funktionen arbeiten auf die gleiche Weise, sie liefern zusätzlich einen Wert an die aufrufende Routine und signalisieren einen Laufzeitfehler, wenn keine Regel ausgewählt wird.

Eine ausführliche Beschreibung des Werkzeugs *puma* befindet sich in [Gro91b].

6.6 Die *Cocktail*-Bibliothek

In dieser Bibliothek befinden sich Funktionen und Typen, die häufig im Übersetzerbau verwendet werden. Es sind zum Beispiel Funktionen für die Verwaltung von Zeichenketten und Bezeichner enthalten. Zum Beispiel werden Bezeichner im Typ `tIdent` gespeichert. Die vollständige Auflistung der verfügbaren Funktionen befindet sich in [Gro92c].

6.7 Anmerkungen zu *Cocktail*

Die Dokumentation zu *Cocktail* ist sehr umfangreich, enthält aber nur wenige Beispiele. Zu den bereits erwähnten Berichten gibt es noch die Berichte [Gro89, GE90b, Gro91c, Gro91e, Gro88b, Vie89]. Die Beschreibungen der Werkzeuge sind jedoch sehr knapp und gelegentlich auch fehlerhaft. Häufig mußte ausprobiert werden, wie die Werkzeuge arbeiten. Besonders schwierig zu verstehen waren die Beschreibungen des Attributauswerter-Generators *ag* und des Transformationswerkzeugs *puma*. Hier gab es Fehler in den Beschreibungen. Bei *puma* ergaben sich Probleme mit den Marken (Anhang B) und durch Kommentare konnte falscher C-Code erzeugt werden. Ein Einführungsbeispiel fehlt in *Cocktail*. Bei der Implementierung des Parsers war es sehr aufwendig, die Abbildung von der konkreten zur abstrakten Grammatik anzugegeben. Der Attributauswerter-Generator *ag* hat den Vorteil, daß mehrere Attributauswerter erzeugt werden können. Es kann für jedes Attribut ein eigener Attributauswerter erzeugt werden. Dadurch werden die Attributauswerter übersichtlich. *Cocktail* bietet keine Unterstützung bei der Erzeugung und Verwaltung einer Symboltabelle.

Das Werkzeug *puma* bietet eine reichhaltige Unterstützung bei der Erzeugung von Pattern. *puma* gibt eine Fehlermeldung aus, falls ein unerreichbares Pattern in einer *puma*-Routine angegeben wurde. Ein unerreichbares Pattern ist ein Pattern, das in einem bereits vorher definiertem Pattern enthalten ist. Bei einem Pattern über einen Baum- oder Knotentyp wird die Grammatikstruktur des Pattern überprüft. Es ist nachteilig, daß bei jedem Knotentyp alle Attribute angegeben werden müssen. Wird also bei einem Knotentyp ein Attribut hinzugefügt oder gelöscht, müssen alle Pattern, die die rechte Seite dieses Knotentyps beinhalten, überarbeitet werden. Die Bibliotheksfunktion

„`MakeIdent(char *string, cardinal length)`“ trägt jede Zeichenkette `string` in die Bezeichnertabelle ein, ohne zu überprüfen, ob diese Zeichenkette bereits in der Tabelle enthalten ist. Dies erschwerte die Erzeugung der virtuellen Variablen. Es stellt sich allgemein die Frage, inwieweit die vorhandenen Funktionen effizient implementiert sind.

7. Die Implementierung im Überblick

Das Diagramm in Abbildung 7.1 auf Seite 45 gibt einen groben Überblick über den Aufbau der Implementierung des Transformationsprogramms `pstt` und die dabei verwendeten *Cocktail*-Werkzeuge.

In der Datei `pstt.c` wird das Hauptprogramm und in der Datei `global.h` werden globale Variablen des Transformationsprogramms spezifiziert. Die Datei `pstt.c` wird in Abschnitt 7.2 vorgestellt. Die Datei `global.h` befindet sich in Abschnitt C.42.17.

Aus den Dateien `trans.scan` und `trans.pars` werden mit den Werkzeugen *cg*, *rpp*, *rex* und *lalr* der Scanner und der Parser erzeugt. In der Datei `trans.scan` sind die Terminals (außer Schlüsselwörter) spezifiziert und in der Datei `trans.pars` befindet sich die konkrete Grammatik. Die konkrete (und die abstrakte) Grammatik wurden von der aktuellen Version des ProSet-Compilers übernommen. Die Übertragung der Grammatiken von der *Eli*-Spezifikation in die *Cocktail*-Spezifikation wird in Abschnitt 7.7 beschrieben. Die Erzeugung eines Scanners und eines Parsers mit *Cocktail* wurde in den Abschnitten 6.1, 6.2 und 6.3 angesprochen. Die Dateien `trans.scan` und `trans.pars` befinden sich in Abschnitt C.3.

In der Datei `trans.cg` wird die abstrakte Grammatik spezifiziert. Durch den Aufruf des Werkzeugs *ast* werden aus der Datei `trans.cg` die Dateien `Tree.h` und `Tree.c` erzeugt, in denen die C-Struktur des abstrakten Syntaxbaumes definiert ist und die Datei `Tree.TS`, die die interne Darstellung des Baums beinhaltet, die für das Werkzeug *puma* benötigt wird. Die gesamten *puma*-Routinen befinden sich in der Datei `trans.puma`. Diese werden durch den Aufruf des Werkzeugs *puma* in C-Funktionen übersetzt. Die Werkzeuge *ast* und *puma* wurden in den Abschnitten 6.4.1 und 6.5 beschrieben. Die abstrakte Grammatik befindet sich in Abschnitt C.4. Die *puma*-Funktionen sind im Inhaltsverzeichnis an den Abschnittsüberschriften erkennbar. Die *Cocktail*-Bibliothek wurde in Abschnitt 6.6 angesprochen. `MyLib` umfaßt Dateien, in denen C-Typen und Funktionen spezifiziert wurden, die für das Transformationsprogramm verwendet werden. In `MyLib` ist die Datei `mydtypes.h` enthalten, in der C-Typen definiert werden. In den Dateien `myscope.h` und `myscope.c` befinden sich Funktionen für eine Prozedurnamenliste (siehe

Abschnitt C.6). Weiterhin gibt es die Dateien `mytrans.h` und `mytrans.c`, in denen die Ableitungen (siehe Abschnitt C.11) spezifiziert sind und die Dateien `myprint.h` und `myprint.c`, in denen die Funktionen zur Ausgabe des transformierten Programms (Abschnitt C.19) definiert sind.

In der Datei `trans.cg` befinden sich die Module für die Attributauswerter, die mit dem Werkzeug *ag* erzeugt werden. Der Attributauswerter `EnvAtt` wird durch den unten angegebenen Aufruf aus dieser Datei erzeugt und in `EnvAtt.h` und `EnvAtt.c` gespeichert. Durch den UNIX-Befehl [Sta93]

```
echo EVAL EnvAtt
     SELECT AbstractSyntax EnvOutAtt
     PROPERTY OUTPUT FOR EnvOutAtt | cat - trans.cg | ag - cDI
```

wird der Inhalt der Datei `trans.cg` hinter der Zeichenkette

```
EVAL EnvAtt SELECT AbstractSyntax EnvOutAtt PROPERTY OUTPUT FOR EnvOutAtt
```

angefügt und als Eingabe für das Werkzeug *ag* verwendet. Dadurch wird von *ag* der Attributauswerter `EnvAtt` erzeugt (`EVAL EnvAtt`), der die Module `AbstractSyntax` und `EnvOutAtt` verwendet (`SELECT AbstractSyntax EnvOutAtt`) und allen Attributen des Moduls `EnvOutAtt` die Eigenschaft `OUTPUT` zuweist (`PROPERTY OUTPUT FOR EnvOutAtt`). Diese Angaben werden im `Makefile` gemacht, da sie für jeden Attributauswerter verschieden sind. Wären sie in der Datei `trans.cg` enthalten, müßte für jeden Attributauswerter eine eigene Datei geschrieben werden. Diese Dateien würden jedoch zum größten Teil übereinstimmen. Die Erzeugung der übrigen Attributauswerter erfolgt entsprechend mit ähnlichen Aufrufen. Diese Aufrufe stehen im Makefile in Abschnitt C.2. Das Werkzeug *ag* wurde in Abschnitt 6.4.2 beschrieben. Die einzelnen Attributauswerter werden in den nachfolgenden Abschnitten beschrieben und sind im Inhaltsverzeichnis ebenfalls an den Abschnittsüberschriften erkennbar.

Nachdem alle Dateien erzeugt wurden, werden sie übersetzt und zum ausführbaren Transformationsprogramm **pstt** gebunden.

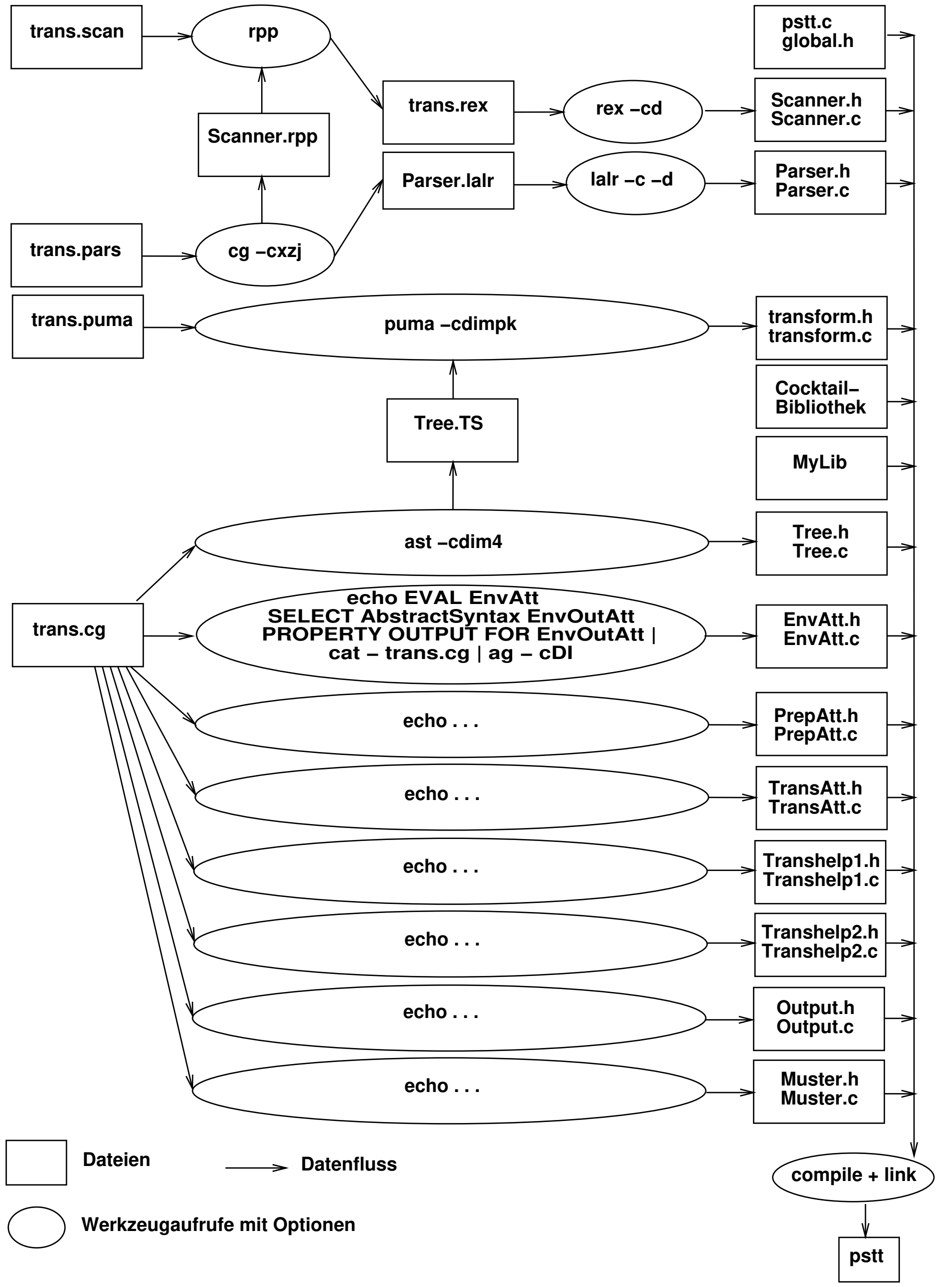

Abbildung 7.1: Der Aufbau und die Generierung des Transformationsprogramms im Überblick. Für die Attributauswerter `PrepAtt`, `TransAtt`, `Transhelp1`, `Transhelp2`, `Output` und `Muster` sind die Aufrufe im `Makefile` angegeben.

7.1 Funktionalität und Implementierungsbeschränkungen des Transformationsprogramms

In Kapitel 5 wurden bereits die Anforderungen an das Transformationsprogramm beschrieben. In diesem Abschnitt sollen nun die Funktionalität und die Implementierungsbeschränkungen erläutert werden.

Das Transformationsprogramm implementiert das erweiterte Differential (differenzierbare Ketten) mit der Einschränkung, daß nur applikative Ausdrücke, die in Anhang A angegeben sind (Muster), differenziert werden können. Für die Muster wurden nur Ableitungen angegebenen, die profitabel differenzierbar sind. Es wird nur transformiert, wenn für die freien Variablen dieser Muster innerhalb eines Blocks nur solche Operationen verwendet werden, für die eine Ableitung vorhanden ist. Für alle virtuellen Variablen V_i wird die Initialisierung (Satz 3.2 auf Seite 19) vor einer Schleife benötigt, da innerhalb der Schleife nur Benutzungen und Modifikationen mit Benutzungen (z.B. „`V := V with x`") von V_i vorkommen, das heißt, daß es für alle V_i Benutzungen in der differenzierten Schleife gibt, die am Eingang zur Schleife leben. Auf eine Verschmelzung der Initialisierungen wird verzichtet (siehe Abschnitte 3.1.4 und 3.3).

In Abschnitt 3.3 wurde bereits erläutert, daß auf eine Überprüfung der Eigenschaft differenzierbar zu sein, verzichtet werden kann und, daß die endliche Differentiation nur auf Schleifen angewendet wird (siehe Abschnitt 3.3). Dabei wird die Schleifenbedingung bzw. der Schleifeniterator und jede Anweisung innerhalb der Schleife nach Mustern durchsucht. Bei den Schleifen kann die `loop`-Schleife jedoch nicht betrachtet werden, da sie nur durch `quit`-, `return`- und `stop`-Anweisungen verlassen werden kann, und somit nicht unbedingt einen Block (Seite 15) darstellt. Aus diesem Grunde kann auch in anderen Schleifen nicht transformiert werden, wenn dort `quit`-, `return`- oder `stop`-Anweisungen verwendet werden.

Weiterhin dürfen keine Eingabefunktionen (Abschnitt 2.4) und Prozeduraufrufe in der Schleife verwendet werden, da Seiteneffekte auf benötigte Variablen nicht ausgeschlossen werden können. Um bei Prozeduren Seiteneffekte ausschließen zu können, wird eine Datenflußanalyse benötigt, die bisher für ProSet nicht implementiert wurde. Weiterhin dürfen vorläufig keine Parallelität und keine Ausnahmebehandlung verwendet werden. Ausnahmen dürfen aufgrund der Definition einer ProSet-regulären Ausführung ohnehin nicht ausgelöst werden (siehe Abschnitt 3.19).

Zuweisungen an Tupel oder Abbildungen sollten innerhalb der Schleifen nur über Bezeichner erfolgen, andernfalls müßten für jede Modifikation vier Muster (für Bezeichner, Zeichenketten, ganze und reelle Zahlen) definiert werden. Anstelle der Zuweisung

```
G(3) := 7;
```

sollte der Block

```
x := 3;
y := 7;
G(x) := y;
```

verwendet werden. Innerhalb von Schleifen ist eine Zuweisung in der ersten Form wahrscheinlicher. Bei einer Erweiterung des Transformationsprogramms könnte dieser Schritt als ein Vorbereitungsschritt für das Transformieren eingebaut werden.

7.2 Das Hauptprogramm `pstt.c`

Das Übersetzen eines ProSet-Programms in ein differenziertes ProSet-Programm geschieht durch den folgenden Ausschnitt des Hauptprogramms (siehe Abschnitt C.1 auf Seite 139):

```
BeginFile(inputfile);
Parser();
EnvAtt(TreeRoot);
mygettree(TreeRoot);
PrepAtt(TreeRoot);
TransAtt(TreeRoot);
Output(TreeRoot);
```

Durch die ersten beiden Anweisungen wird die Eingabedatei geöffnet und der abstrakte Syntaxbaum aufgebaut. Hierzu befinden sich in den Abschnitten 6.1, 6.2, 6.3 und 7.7 nähere Informationen. Die Funktionen `EnvAtt`, `mygettree` und `PrepAtt` erzeugen die Prozedurnamentabelle und fügen einen Zeiger auf den gültigen Prozedurnamenbereich an den Knotentyp, der die Schleifen repräsentiert, an. Eine genauere Beschreibung dieser Vorbereitungen wird in Abschnitt 7.3 gegeben. Die Funktion `TransAtt` durchsucht eine Schleife nach einem differenzierbaren applikativen Ausdruck. Wurde ein solcher Ausdruck gefunden, werden die Bedingungen aus Abschnitt 7.1 überprüft. Besteht danach noch die Möglichkeit, den Ausdruck zu differenzieren, wird der applikative Ausdruck im abstrakten Syntaxbaum durch die entsprechende virtuelle Variable ersetzt. Die Arbeitsweise dieser Funktion wird in Abschnitt 7.4 näher erläutert. Die Funktion `Output` gibt den abstrakten Syntaxbaum aus und fügt dabei die Initialisierung der virtuellen Variablen und die Ableitungen der Definitionen in die Ausgabedatei ein. Die Ausgabe wird genauer in Abschnitt 7.5 beschrieben.

Bei einem Durchlauf durch diese Anweisungen kann in jeder Schleife nur ein applikativer Ausdruck differenziert werden. Diese Vorgehensweise stellt keine Einschränkung des Transformationsprogramms dar, da das Hauptprogramm iterativ konstruiert wurde. Es werden diese Anweisungen (= ein Schritt) ausgeführt, solange im vorhergehenden Schritt eine Transformation durchgeführt wurde. Die vollständige Datei befindet sich in Abschnitt C.1.

7.3 Die Prozedurnamentabelle

Der Attributauswerter `EnvAtt` deklariert für jeden Programmblock ein Attribut vom Typ `Scope`, das einen Zeiger auf die Prozedurnamentabelle (C-Strukturen) darstellt. Der Attributauswerter `EnvAtt` befindet sich in Abschnitt C.6. Ein neuer Sichtbarkeitsbereich beginnt am Knotentyp `xProgBody`. `xProgBody` ist das Nichtterminal für Programmanweisungsblöcke. In der Struktur `Scope` werden für jeden Sichtbarkeitsbereich die dort definierten Prozedur-, Handler- und Modulnamen eingetragen und ein Zeiger auf den übergeordneten Sichtbarkeitsbereich gesetzt. Diese Struktur ist in Abschnitt C.6.1 genau spezifiziert.

Durch die *puma*-Funktion `mygettree` wird die Prozedurnamentabelle aufgebaut (siehe Abschnitt C.7). Diese Funktion initialisiert die Prozedurnamentabelle und ruft eine weitere *puma*-Funktion auf, die den abstrakten Syntaxbaum durchläuft und die Namen in den dort gültigen Sichtbarkeitsbereich einfügt.

Die Attributauswerter, die durch *ag* erzeugt werden, können nur auf Attribute eines Knotentyps und die Attribute seiner direkten Nachfolger zugreifen. Der Attributauswerter `TransAtt` (siehe Abschnitt 7.4) arbeitet auf Schleifen und benötigt die Bezeichner aus der Prozedurnamentabelle. Daher erzeugt der Attributauswerter `PrepAtt` (siehe Abschnitt C.8) ein Attribut vom Typ `Scope` an den Nichtterminals für Schleifen (`xLoops`). Dieses Attribut zeigt auf den gültigen Sichtbarkeitsbereich der Schleife. Dies geschieht über eine globale Variable vom Typ `Scope`, die am Knotentyp `xProgBody` einen Zeiger auf den gültigen Sichtbarkeitsbereich erhält. Alle in diesem Sichtbarkeitsbereich definierten Schleifen erhalten den Wert der globalen Variablen.

7.4 Die Transformation

Der Attributauswerter `TransAtt` (Abschnitt C.9) ruft für die Schleifenbedingung bzw. den Schleifeniterator einer Schleife die *puma*-Funktion `Pattern` auf. Diese Funktion enthält die Pattern der in Anhang A angegebenen Ausdrücke und Anweisungen, die nach diesen Ausdrücken durchsucht werden. Die Funktion `Pattern` überprüft, ob die zu untersuchende Bedingung bzw. der zu untersuchende Iterator in den applikativen Ausdrücken vorkommt. Befindet sich dieser Ausdruck nicht unter den vorhandenen applikativen Ausdrücken, wird die Funktion `getcode` (Abschnitt C.9.3) mit allen Anweisungen innerhalb der Schleife aufgerufen. Die *puma*-Funktion `getcode` ruft dann solange die Funktion `Pattern` mit einer Anweisung auf, bis in dieser Anweisung ein differenzierbarer Ausdruck gefunden wurde. Für diesen applikativen Ausdruck werden die gebundenen und freien Variablen in einer Variablen vom Typ `DSET` (Abschnitt C.9.2.1) gespeichert (Abschnitt C.13). An den Schleifenknoten `xLoops` wird im Attribut `transatt` die Kennung des von `Pattern` erkannten Musters gespeichert (Abschnitt C.9.2.1). Anhand der Kennung können die nachfolgenden Funktionen erkennen, welcher applikative Ausdruck erkannt wurde. Die Funktion `Pattern` wird genauer in Abschnitt 7.6 beschrieben. Wird ein differenzierbarer Ausdruck gefunden, überprüfen die Funktionen `CanIDelete`, `Transhelp1` und `Transhelp2`, ob keine der in Abschnitt 7.1 beschriebenen Anweisungen in der Schleife verwendet wird. Der Attributauswerter `Transhelp1` (Abschnitt C.16) durchsucht die Anweisungen nach Prozedur- und Handleraufrufen. Dabei werden alle Bezeichner mit den sichtbaren Bezeichnern in der Prozedurnamentabelle verglichen. Der Attributauswerter `Transhelp2` (Abschnitt C.17) ruft für alle Definitionen von freien Variablen des erkannten Ausdrucks die Funktion `Pattern` auf, die dann für diese Zuweisung feststellt, ob es sich um eine Operation handelt, für die eine Ableitung vorhanden ist. Die *puma*-Funktion `CanIDelete` (Abschnitt C.15) überprüft, ob andere Anweisungen aus Abschnitt 7.1 verwendet werden. Sobald eine dieser Funktionen feststellt, daß innerhalb der Schleife nicht transformiert werden kann, werden die nachfolgenden Funktionen nicht mehr aufgerufen und der Attributauswerter `TransAtt` betrachtet die nächste Schleife. Gibt es jedoch einen differenzierbaren Ausdruck, ersetzt die Funktion `Delete` (Abschnitt C.18) den gefundenen Ausdruck durch die neu generierte virtuelle Variable. Diese Variable wird bereits beim Speichern der Variablen des gefundenen Ausdrucks erzeugt.

7.5 Die Ausgabe des differenzierten Programms

Der Attributauswerter `Output` (Abschnitt C.19) schreibt den modifizierten abstrakten Syntaxbaum mit Schlüsselwörtern in die Ausgabedatei. Der abstrakte Syntaxbaum wurde bisher nur durch die Ersetzung der differenzierbaren Ausdrücke durch die jeweiligen virtuellen Variablen modifiziert. Diese Vorgehensweise ermöglicht es, die Initialisierungen der virtuellen Variablen und die Ableitungen in ProSet-Code und nicht mit Hilfe der abstrakten Grammatik als Baum, anzugeben. Für die Ausgabe der Terminals werden Funktionen aus der *Cocktail*-Bibliothek (Abschnitt 6.6) und aus `MyLib` verwendet.

Für den Attributauswerter `Output` wird ein `THREAD`-Attribut verwendet (Abschnitt 6.4.1). Dieses Attribut wird benutzt, um aus dem abstrakten Syntaxbaum ein syntaktisch korrektes ProSet-Programm zu erzeugen. Erkennt der Attributauswerter eine Schleife, ist am Attribut `transatt` erkennbar, welcher differenzierbare Ausdruck in der Schleife vorkommt. Für diesen Ausdruck wird dann die Funktion `inittrans` (Abschnitt C.20) aufgerufen, die die Initialisierung der virtuellen Variablen in die Ausgabedatei schreibt. Für jede Anweisung innerhalb der Schleife, wird die Funktion `Statement` (siehe Abschnitt C.19.1) aufgerufen, die für die Anweisungen, bei denen keine Ausnahme angegeben wurde, die Funktion `Pattern` aufruft. Die Funktion `Pattern` gibt dann für Anweisungen, in denen eine freie Variable des zu differenzierenden Ausdrucks definiert wird, die Vorableitung aus. Anhand des Rückgabewertes der Funktion `Pattern` erkennt der Attributauswerter `Output`, ob die Funktion `Pattern` nach Ausgabe der Modifikationen der freien Variablen für eine Nachableitung erneut aufgerufen werden muß. Die Rückgabewerte der Funktion `Pattern` werden in Abschnitt C.19.1 beschrieben.

Für die Ausgabe des restlichen Programms, werden die Funktionen `putmykey`, `putmyid`, `putmyint`, `putmyfloat`, `putmystr` und `putmyend` (Abschnitt C.40) verwendet. Vor und nach der Ausgabe der Initialisierungen und der Ableitungen können line-Direktiven (Abschnitte 7.8 undC.40.1) und neue Kommentare ausgegeben werden (Abschnitt 7.8).

7.6 Die *puma*-Funktion `Pattern`

In der Funktion `Pattern` sind alle applikativen Ausdrücke und Modifikationen ihrer freien Variablen, für die eine Ableitung existiert, angegeben. Die Funktion `Pattern` wird an allen Stellen des Transformationsprogramms verwendet, an denen die applikativen Ausdrücke (Muster) oder die Modifikationen der freien Variablen eines Musters (Modifikationsmuster) benötigt werden. Dies hat den Vorteil, daß jedes (Modifikations-)Muster nur einmal spezifiziert werden muß. Die Funktion `Pattern` befindet sich vollständig in Abschnitt C.10. Der Routinenkopf der Funktion `Pattern` wird in Abbildung 7.2 dargestellt.

```
FUNCTION Pattern(REF pat:Tree, REF variable:DSET,
                 REF transatt:CODE, funcflag:FUNCT) int
```

Abbildung 7.2: Der Funktionskopf der *puma*-Funktion `Pattern`.

In `pat` wird ein Ausdruck oder eine Zuweisung übergeben, die auf das Vorkommen eines applikativen Ausdrucks oder Modifikation einer freien Variablen überprüft werden soll. `variable` wird für die freien und gebundenen Variablen und Konstanten des Musters verwendet. In `transatt` befindet sich die Kennung des Musters und `funcflag` wird weiter unten erläutert.

In der Funktion `Pattern` werden zunächst die Pattern der Muster aus Anhang A und die Modifikationsmuster angegeben. Im Anschluß sind die Anweisungen und Ausdrücke angegeben, in denen Teilausdrücke nach applikativen Ausdrücken durchsucht werden. Über dieses Prinzip werden die differenzierbaren Ketten (Kettenmuster) gebildet.

Die Pattern lassen sich also in vier getrennte Bereiche einteilen:

- Transformationsmuster
- Modifikationsmuster
- Kettenmuster

- das default Muster „_,_,_,_“

Das default Muster wird für den Fall benötigt, daß keine angegebene Regel ausgewählt wurde. Die Funktion `Pattern` wird an verschiedenen Stellen des Transformationsvorgangs verwendet. Der Kontext, in dem die Funktion aufgerufen wird, ist am Parameter `funcflag` vom Typ `FUNCT` (Abschnitt C.12.4) zu erkennen. `funcflag` kann einen der Werte `code`, `helpstmt`, `delete`, `preder` oder `postder` annehmen. Bei den einzelnen Werten werden von der Funktion `Pattern` die folgenden Operationen durchgeführt:

code: Diese Operation wird bei Transformationsmustern verwendet, um diese zu erkennen und die dazugehörigen Variablen zu speichern, und bei Kettenmustern, um die Funktion `Pattern` für Teilausdrücke aufzurufen. Wird das default Muster ausgewählt, wird zurückgegeben, daß kein Muster gepaßt hat.

helpstmt: Dieser Wert wird vom Attributauswerter `Transhelp2` verwendet (siehe Abschnitt 7.4), um sicherzustellen, daß die freien Variablen des zu transformierenden Ausdrucks nur in Zuweisungen modifiziert werden, für die eine Ableitung vorhanden ist. Wird eine Variable in einer Zuweisung modifiziert, die für dieses Muster und diese Variable nicht vorhanden ist oder wird erst das default Muster ausgewählt, kann nicht transformiert werden.

delete: Das erkannte Muster wird durch die virtuelle Variable ersetzt. Diese Operation arbeitet auf Transformationsmustern und Kettenmustern. Bei Kettenmustern wird die Funktion `Pattern` wieder mit Teilausdrücken aufgerufen.

preder: Gibt für die Modifikationsmuster die entsprechende Vorableitung aus.

postder: Gibt für die Modifikationsmuster die entsprechende Nachableitung aus.

7.7 Pragmatische Aspekte

In diesem Abschnitt wird beschrieben, wie die konkrete und die abstrakte Grammatik für das Transformationsprogramms entstanden sind. Dieser Abschnitt ist sehr ausführlich gehalten, um bei zukünftigen Änderungen der konkreten Grammatik von ProSet die konkrete Grammatik des Transformationsprogramms anzupassen.

Es wurde bereits erwähnt, daß der aktuelle ProSet-Compiler mit *Eli* erzeugt wurde. Es war also naheliegend, die *Eli*-Grammatik zu verwenden und sie nur soweit syntaktisch zu modifizieren, daß sie von den *Cocktail*-Werkzeugen verarbeitet werden kann. Dieses Vorgehen hat den Vorteil, daß beide Grammatiken leicht konsistent gehalten werden können. In diesem Abschnitt werden nun die Unterschiede zwischen der *Eli*-Grammatik und der *Cocktail*-Grammatik beschrieben.

In *Eli* werden die benutzerdefinierten Token für die lexikalische Analyse in der Datei `pst.gla` spezifiziert. Durch die folgende Anweisung wird ein Bezeichner spezifiziert:

```
id:     $[a-zA-Z_][a-zA-Z_0-9]* [mkidn]
```

Die entsprechende Spezifikation befindet sich in der *Cocktail* Datei `trans.scan`:

```
DEFINE
 digit = { 0-9 } .
 letter = { a-z A-Z _} .

#STD#   letter ( letter | digit )*
        : { Attribute.id.Ident = MakeIdent(TokenPtr,TokenLength);
            Attribute.id.Length = TokenLength;
            return id;}
```

Ein wesentlicher Unterschied in den Spezifikationen ist die Ausführung der Aktionen nach dem Erkennen des Token. Bei *Eli* muß für Bezeichner nur die Funktion `mkidn` aufgerufen werden, während bei *Cocktail* die Berechnung der Attribute und der internen Darstellung explizit angegeben werden muß.

Interessanter ist jedoch der Aufbau der Grammatiken. In *Eli* wird die konkrete Grammatik in der Datei `pst.con` spezifiziert und in *Cocktail* in der Datei `trans.pars`. Die Unterschiede sollen anhand des Nichtterminals `xPHMDefn` dargestellt werden. Dieses Nichtterminal ist das Startsymbol für Prozedur-, Handler- und Moduldeklarationen. Zuerst wird ein Teil der *Eli*-Spezifikation angegeben:

```
xPHMDefn ::=   'module' xId xModImport xModExport xProgBody 'end'
               xId ';' .
xPHMDefn ::=   'handler' xId xRdParamList xImplAsso ';' xProgBody
               'end' xId ';' .
```

Die *Cocktail*-Spezifikation sieht wie folgt aus:

```
xPHMDefn = <
  new2xPHMDefn = 'module' xId1:xId xModImport xModExport xProgBody
                 'end' xId2:xId ';' .
  new3xPHMDefn = 'handler' xId1:xId xRdParamList xImplAsso S1:';'
                 xProgBody 'end' xId2:xId S2:';' .
>.

id : [Ident: tIdent] [Length]{ Ident := NoIdent; Length := 0; } .
```

Der gravierendste Unterschied besteht darin, daß in *Eli* für ein Nichtterminal mehrere Regeln angegeben werden können und in *Cocktail* nur eine, bei der jedoch mehrere Alternativen angegeben werden können. In *Cocktail* werden für das Transformationswerkzeug *puma* eindeutige Grammatiknamen benötigt, damit die Pattern genau spezifiziert werden können. Für *Cocktail* werden also alle Regeln eines Nichtterminals zu einer Regel zusammengefaßt, wobei die *Eli*-Regeln als Alternativen (Abschnitt 6.4) definiert werden und durch „`new*xPHMDefn`" eindeutige Namen erhalten. „`*`" wird in diesem Kapitel als Wildcard verwendet. In der *Cocktail*-Grammatik wird anstelle des „`*`" eine ganze Zahl angegeben. Der Basistyp bekommt den Namen der *Eli*-Regel. Weiterhin müssen auch alle Nachfolger, die mehrfach auf der rechten Seite einer Grammatikregel vorkommen, einen eindeutigen Namen erhalten. In der *Cocktail* Datei müssen auch die Attribute der Terminals, die in der Datei `trans.scan` berechnet werden sollen, angegeben und initialisiert werden.

Auch die abstrakten Grammatiken unterscheiden sich in einem Punkt sehr deutlich voneinander. Zuerst wird wieder die *Eli*-Grammatik angegeben.

```
xProgDefn ::= 'program' xId ';' xProgBody 'end' xId ';'
```

Als nächstes folgt die *Cocktail*-Grammatik.

```
xProgDefn = [pos1:tPosition] xId1:xId [pos2:tPosition]
            xProgBody [pos3:tPosition] xId2:xId
            [pos4:tPosition] .
```

In der abstrakten Grammatik für *Cocktail* dürfen keine Schlüsselwörter vorkommen. Die Grammatik wird durch die Anwendung des Werkzeugs *ast* in C-Strukturen für den abstrakten Syntaxbaum übersetzt, wobei für das Schlüsselwort „`'program'`" zum Beispiel die Konstante „`k'program'`" erzeugt würde, die keine gültige C-Konstante wäre. Für die Ausgabe des abstrakten Syntaxbaumes sind die Positionen der Schlüsselwörter von Bedeutung, daher werden anstelle der Schlüsselwörter Attribute vom Typ `tPosition` verwendet. Die abstrakte Grammatik wird in *Eli* in der Datei `rules.lido` und in *Cocktail* in der Datei `trans.cg` spezifiziert.

Das größte Problem bei der Übertragung der Grammatiken von *Eli* nach *Cocktail* ist die Abbildung zwischen der konkreten und der abstrakten Grammatik. In *Eli* erfolgt die Abbildung der konkreten auf die abstrakte Grammatik durch die Datei `pst.sym`. Die Nichtterminals auf der rechten Seite der folgenden Regel werden in der abstrakten Grammatik auf das Nichtterminal der linken Seite abgebildet.

```
xExpr ::= xcBinExpr xcPrimary xcMulTerm xcPowTerm xcOrTerm.
```

Für *Cocktail* muß diese Abbildung explizit in der Datei `trans.pars` angegeben werden. *Cocktail* erzeugt lediglich Funktionen, die einen Knoten im abstrakten Syntaxbaum erzeugen (Abschnitt 6.4.1). Dadurch werden nur die Anzahl und die Typen der Parameter festgelegt. Der Aufbau des abstrakten Syntaxbaumes erfolgt über ein Attribut vom Typ `tTree`.

```
DECLARE xProgDefn = [Tree : tTree] .

RULE
xProgDefn = { Tree := mxProgDefn('program':Position,
                        mxId(xId1:Tree), S1:Position,
                        xProgBody:Tree, 'end':Position,
                        mxId(xId2:Tree), S2:Position);
            } .
```

7.8 Die Benutzung des Transformationsprogramms

Der Aufruf des Transformationsprogramms erfolgt durch

`pstt [-number` *anzahl* | `-n` *anzahl*`] [-NL] [-tm]` *file*

Durch die Option „`-number` *anzahl*" oder „`-n` *anzahl*" wird angegeben, wieviele Transformationsdurchläufe höchstens durchgeführt werden sollen. Dabei muß aus Implementierungsgründen *anzahl* < 999 sein. Das Transformationsprogramm führt jedoch nur soviele Durchläufe durch, bis nicht mehr transformiert werden kann. Die Ausgabe jedes Durchlaufs wird in eine Datei geschrieben, die hinter der Endung „`.tps`" die Nummer des Durchlaufes angefügt bekommt. Der letzte Durchlauf befindet sich jedoch immer in der Datei mit der Endung „`.tps`".

Wird die Option „`-NL`" gesetzt, werden keine line-Direktiven ausgegeben.

Die Option „`-tm`" kann beim Einfügen eines neuen Musters in das Transformationsprogramm verwendet werden, und wird im Anhang B erläutert.

Wichtig ist, daß es in der Verantwortung des Benutzers liegt, daß sein Programm ProSet-regulär ist, also nicht durch eine Ausnahme abgebrochen wird (siehe Abschnitt 3.3).

8. Das Einfügen eines neuen Musters

In diesem Kapitel wird das Einfügen eines neuen Musters in das Transformationsprogramm skizziert. Eine ausführliche Anleitung befindet sich in Anhang B. Hier soll nur ein kurzer Einblick in die Aufgaben, die beim Einfügen auftreten können, gegeben werden. In Abbildung 8.1 sind die verschiedenen Aufgaben in einem Flußgraphen dargestellt. Diese Abbildung wird im folgenden erläutert.

Zunächst muß für einen neuen Mengenausdruck festgestellt werden, ob für Modifikationen freier Variablen dieses Ausdrucks profitabel differenzierbare Ableitungen angegeben werden können. Existieren solche Ableitungen, muß der C-Aufzählungstyp `CODE` um eine Kennung für diesen Mengenausdruck erweitert werden. Die Konstanten- und Variablennamen dieses Mengenausdrucks müssen in einer Variablen der C-Union `Dset` gespeichert werden. `Dset` könnte möglicherweise schon eine passende Komponente für diese Daten enthalten. Ist keine passende Komponente vorhanden, muß `Dset` erweitert werden. Für diese neue Komponente müssen dann die C-Funktionen `get_Neues_Muster` und `check_Neues_Muster` geschrieben werden. Die erste Funktion speichert die Konstanten- und Variablennamen des Mengenausdrucks in einer Variable vom Typ `Dset` und die zweite Funktion vergleicht gegebene Variablen- und Konstantennamen mit den Namen in einer Variablen des Typ `Dset`. Danach kann der neue Mengenausdruck in die *puma*-Funktion `Pattern` eingefügt werden. Nun müssen die Ableitungsfunktionen geschrieben werden und die Modifikationsmuster in die *puma*-Funktion `Pattern` eingefügt werden. Als nächstes muß die C-Funktion `testident` erweitert werden. Diese Funktion überprüft, ob ein gegebener Variablenname unter den Variablennamen des Mengenausdrucks vorkommt. Dann muß die C-Funktion `inittrans`, die die Initialisierungsfunktion für die virtuelle Variable aufruft, erweitert werden. Als letzter Schritt muß die Initialisierungsfunktion für diesen Mengenausdruck geschrieben werden. Bei den Ableitungsfunktionen und der Initialisierungsfunktion handelt es sich um C-Funktionen.

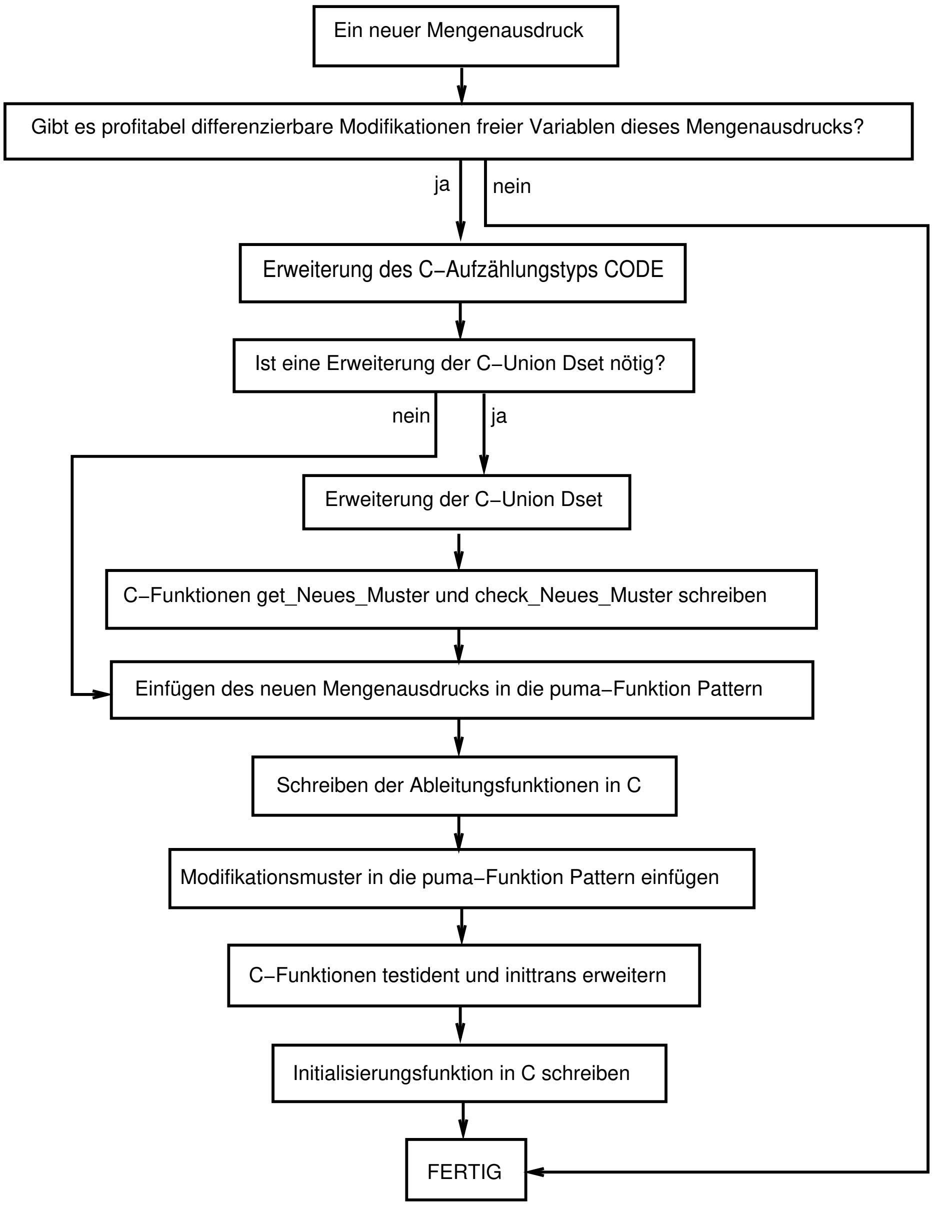

Abbildung 8.1: Übersicht über die Aufgaben beim Einfügen eines neuen Musters.

9. Laufzeitvergleiche

In diesem Kapitel werden Ergebnisse von Laufzeittests beschrieben, mit denen die Effizienz der implementierten Transformationen exemplarisch untersucht wurden. Die Laufzeitmessungen wurden auf einer SparcStation-20 (50 MHz, 137 MIPS) durchgeführt.

Für die Zeitmessung wurde die UNIX-Funktion `time` [Sta93] verwendet. `time` liefert die Realzeit und die Zeit, die ein Programm im User- und System-Mode verbracht hat. Unter Realzeit wird die tatsächliche Zeit verstanden, die seit dem Start des Programms vergangen ist. Sie beinhaltet die Wartezeiten auf Ein-/Ausgabe, Rechenzeit und andere Ereignisse. Bei den Laufzeittests wurde die Realzeit verwendet, da es bei den anderen Zeiten leicht zu Meßungenauigkeiten kommt [SL94]. Die Tests wurden nachts durchgeführt, damit die Realzeit nicht durch andere Ereignisse beeinflußt wird.

Es wurden fünf Testreihen durchgeführt. In den ersten vier Testreihen wird der Mengenformer „{`[F(j),j] : j in G`}" betrachtet. Die Testprogramme unterscheiden sich nur durch die verwendete Definition („`G with:= x`" und „`F(x) := x`") und die Mächtigkeit der Mengen `F` und `G`. In der fünften Testreihe wird anhand eines verschachtelten Mengenformers untersucht, wie sich die Laufzeit eines Programms bei mehrfachen Transformationsschritten entwickelt. Auf der x-Achse der Diagramme wird die Anzahl der Schleifeniterationen dargestellt und auf der y-Achse die Realzeit in Sekunden. Bei den Diagrammen sollen nur die Unterschiede der Laufzeiten deutlich werden. Die genauen Werte befinden sich in den Tabellen.

9.1 Die erste Testreihe

Das erste Testprogramm ist in Abbildung 9.1 angegeben. Mit dem Kostenmaß aus Abschnitt 3.1.3 ergeben sich Kosten der Ordnung $O(n^2)$. Für das transformierte Programm in Abbildung 9.2 auf Seite 60 ergeben sich Kosten der Ordnung $O(n)$. Diese Effizienzsteigerung spiegelt sich auch in den in Abbildung 9.3 auf Seite 61 dargestellten Ergebnissen der Laufzeittests wider. Werden die Ergebnisse der Tests in Tabelle 9.1 auf Seite 62 genau betrachtet, ist zu erkennen, daß das transformierte Programm bei null und einer Schleifeniterationen langsamer ist, als das Original. Dies liegt an der Initialisierung der virtuellen Variablen `pstt1` vor der Schleife.

9.2 Die zweite Testreihe

Das zweite Testprogramm ist in Abbildung 9.4 auf Seite 62 angegeben. Es unterscheidet sich vom ersten Testprogramm durch die Mächtigkeiten der Mengen `F` und `G`. Auch bei den Laufzeittests für diese Testreihe (Abbildung 9.6 auf Seite 64) ist die Effizienzsteigerung deutlich sichtbar. In Tabelle 9.2 auf Seite 65 wird ebenfalls die Verschlechterung der Laufzeit bei null und einer Schleifeniterationen deutlich.

9.3 Die dritte Testreihe

Das dritte Testprogramm ist in Abbildung 9.7 auf Seite 65 angegeben. Im Unterschied zum ersten Testprogramm wird hier die Anweisung „`F(x) := x`“ anstelle der Anweisung „`G with:= x`“ betrachtet. Das Ergebnis der Laufzeittests befindet sich in Abbildung 9.9 auf Seite 67 und in Tabelle 9.3 auf Seite 68.

9.4 Die vierte Testreihe

Das vierte Testprogramm ist in Abbildung 9.10 auf Seite 68 angegeben. Es unterscheidet sich vom dritten Testprogramm durch die Mächtigkeiten der Mengen `F` und `G`. Das Ergebnis der Laufzeittests befindet sich in Abbildung 9.12 auf Seite 70 und in Tabelle 9.4 auf Seite 71.

9.5 Die fünfte Testreihe

Das fünfte Testprogramm ist in Abbildung 9.13 auf Seite 71 angegeben. In dieser Testreihe wird untersucht, wie sich mehrfache Transformationsschritte (differenzierbare Ketten) auf die Laufzeit auswirken. Der Mengenformer

```
{k : k in {j : j in F | G(j) in H} | #Vor{k} = 1}
```

enthält die drei Mengenformer

```
{j : j in F | G(j) in H}
{[j,#F{j}] : j in domain F} --Initialisierung von V(k) = #Vor{k}
{j : j in F | G(j) = H}
```

In Abbildung 9.14 auf Seite 72 ist das Programm angegeben, das sich nach einem Transformationsschritt ergibt. In Abbildung 9.15 auf Seite 73 ist das Programm nach zwei

Transformationsschritten und in Abbildung 9.16 auf Seite 74 ist das endgültige Programm angegeben. Es sind keine weiteren Transformationsmöglichkeiten vorhanden. Anhand der Ergebnisse der Laufzeittests in Abbildung 9.17 auf Seite 75 sind wieder deutliche Effizienzsteigerungen erkennbar. Das Programm, das sich nach drei Transformationsschritten ergibt, ist am effizientesten. In Tabelle 9.5 auf Seite 76 ist jedoch wieder erkennbar, daß dieses Programm für null oder eine Schleifeniteration langsamer ist, als das Originalprogramm und die Programme mit ein oder zwei Transformationsschritten. Auch die Programme mit ein oder zwei Transformationsschritten sind für null oder eine Schleifeniteration langsamer, als das Originalprogramm.

9.6 Anmerkungen

Die Laufzeitvergleiche haben gezeigt, daß die Verwendung dieses Transformationsprogramms zu wesentlich effizienteren Programmen führt. In den ersten vier Testreihen wurden unabhängig von der betrachteten Definition und der Mächtigkeit der Mengen `F` und `G` wesentliche Effizienzsteigerungen erreicht. Die fünfte Testreihe zeigt, daß durch mehrfache Transformationsschritte weitere Effizienzverbesserungen erzielt werden. In allen Testreihen zeigt sich jedoch auch, daß es durch die Initialisierung der virtuellen Variablen zu Laufzeitverschlechterungen bei wenig durchzuführenden Schleifeniterationen kommt.

```
program test1;
begin
   H := {1 .. 100};
   F := {[2*i,3*i] : i in H};
   get(N);
   C := {1 .. N};
   G := {1, 1+3 .. 250};
   while C /= {} do
      V := {[F(j),j] : j in G};
      x from C;
      G with := x;
   end while;
end test1;
```

Abbildung 9.1: Das erste Testprogramm.

```
program test1;
begin
   H := {1 .. 100};
   F := {[2*i, 3*i] : i in H};
   get(N);
   C := {1 .. N};
   G := {1, 1+3 .. 250};
   pstt1 := {[F(j),j] : j in G};
   while C /= {} do
      V := pstt1;
      x from C;
      pstt1 with := [F(x),x];
      G with := x;
   end while;
end test1;
```

Abbildung 9.2: Das transformierte Programm zum Programm in Abbildung 9.1.

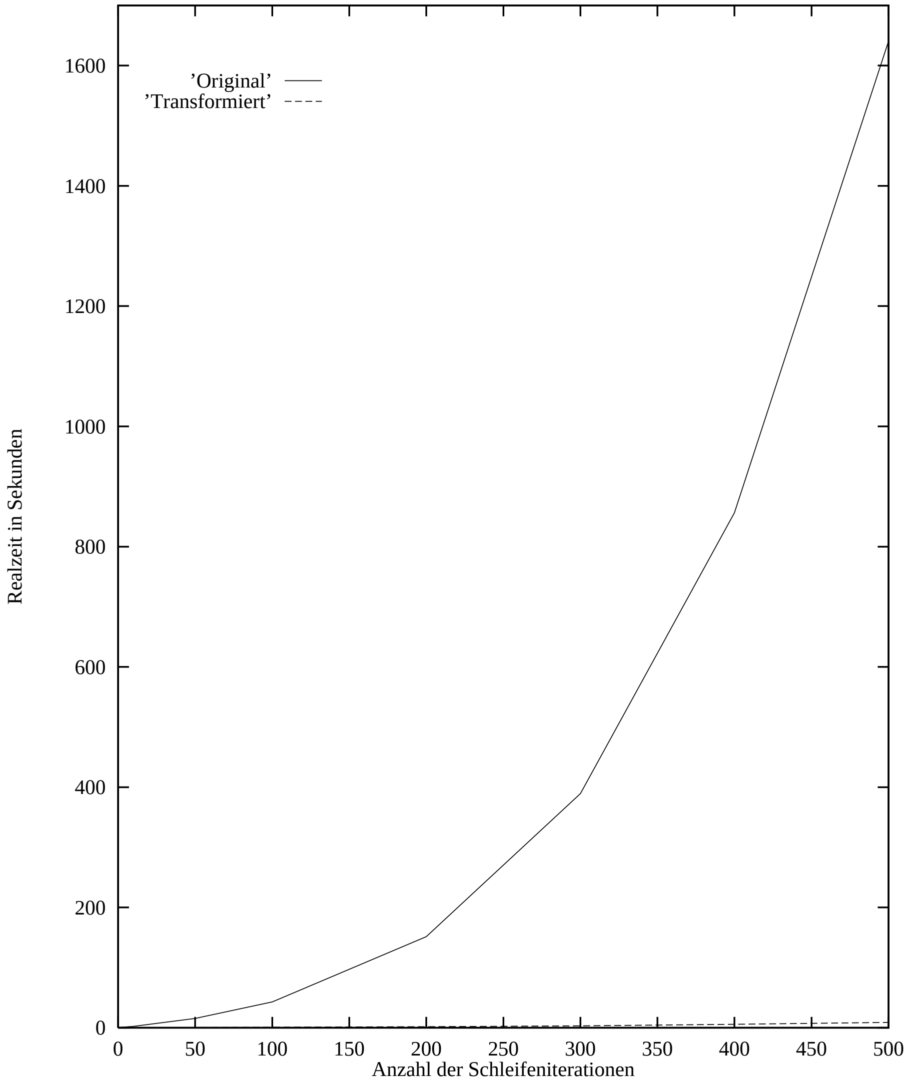

Abbildung 9.3: Ergebnisse der Laufzeittests für die Programme in Abbildung 9.1 und Abbildung 9.2. 'Original' besitzt eine Laufzeit von $O(n^2)$ und 'Transformiert' eine Laufzeit von $O(n)$.

Anzahl der Schleifeniterationen	Realzeit in Sekunden	
	Original	Transformiert
0	0.28	0.44
1	0.44	0.46
10	2.30	0.42
50	15.50	0.60
100	42.94	0.90
200	151.50	1.60
300	389.42	3.12
400	856.26	5.60
500	1640.76	8.86
600	2785.90	13.00
700	4417.70	17.96
800	6547.90	23.50
900	9376.60	31.02
1000	12939.80	38.16

Tabelle 9.1: Ergebnisse der Laufzeittests für die Programme in Abbildung 9.1 und Abbildung 9.2.

```
program test2;
begin
   H := {1 .. 1000};
   F := {[2*i,3*i] : i in H};
   get(N);
   C := {1 .. N};
   G := {1, 1+3 .. 2500};
   while C /= {} do
      V := {[F(j),j] : j in G};
      x from C;
      G with := x;
   end while;
end test2;
```

Abbildung 9.4: Das zweite Testprogramm.

```
program test2;
begin
   H := {1 .. 1000};
   F := {[2*i,3*i] : i in H};
   get(N);
   C := {1 .. N};
   G := {1, 1+3 .. 2500};
   pstt1 := {[F(j),j] : j in G};
   while C /= {} do
      V := pstt1;
      x from C;
      pstt1 with := [F(x),x];
      G with := x;
   end while;
end test2;
```

Abbildung 9.5: Das transformierte Programm zum Programm in Abbildung 9.4.

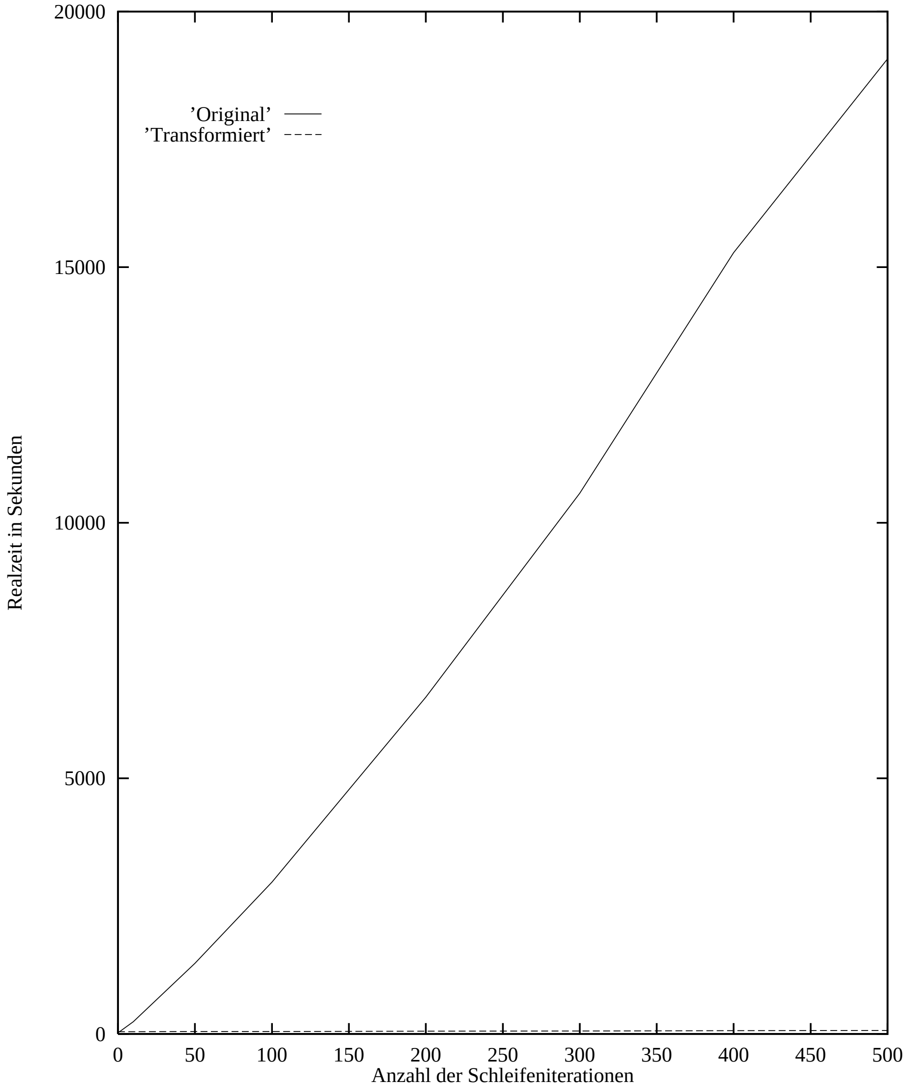

Abbildung 9.6: Ergebnisse der Laufzeittests für die Programme in Abbildung 9.4 und Abbildung 9.5.

Anzahl der Schleifeniterationen	Realzeit in Sekunden	
	Original	Transformiert
0	25.00	43.44
1	43.78	44.76
10	238.46	43.78
50	1381.96	46.40
100	2969.38	47.98
200	6586.50	52.66
300	10579.20	57.86
400	15283.40	63.12
500	19075.96	68.12

Tabelle 9.2: Ergebnisse der Laufzeittests für die Programme in Abbildung 9.4 und Abbildung 9.5.

```
program test3;
begin
   H := {1 .. 100};
   F := {[2*i,3*i] : i in H};
   get(N);
   C := {1 .. N};
   G := {1, 1+3 .. 250};
   while C /= {} do
      V := {[F(j),j] : j in G};
      x from C;
      F(x) := x;
   end while;
end test3;
```

Abbildung 9.7: Das dritte Testprogramm.

```
program test3;
begin
   H := {1 .. 100};
   F := {[2*i,3*i] : i in H};
   get(N);
   C := {1 .. N};
   G := {1, 1+3 .. 250};
   pstt1 := {[F(j),j] : j in G};
   while C /= {} do
      V := pstt1;
      x from C;
      if (x in G) then
         pstt1 less := [F(x),x];
      end if;
      F(x) := x;
      if (x in G) then
         pstt1 with := [F(x),x];
      end if;
   end while;
end test3;
```

Abbildung 9.8: Das transformierte Programm zum Programm in Abbildung 9.7.

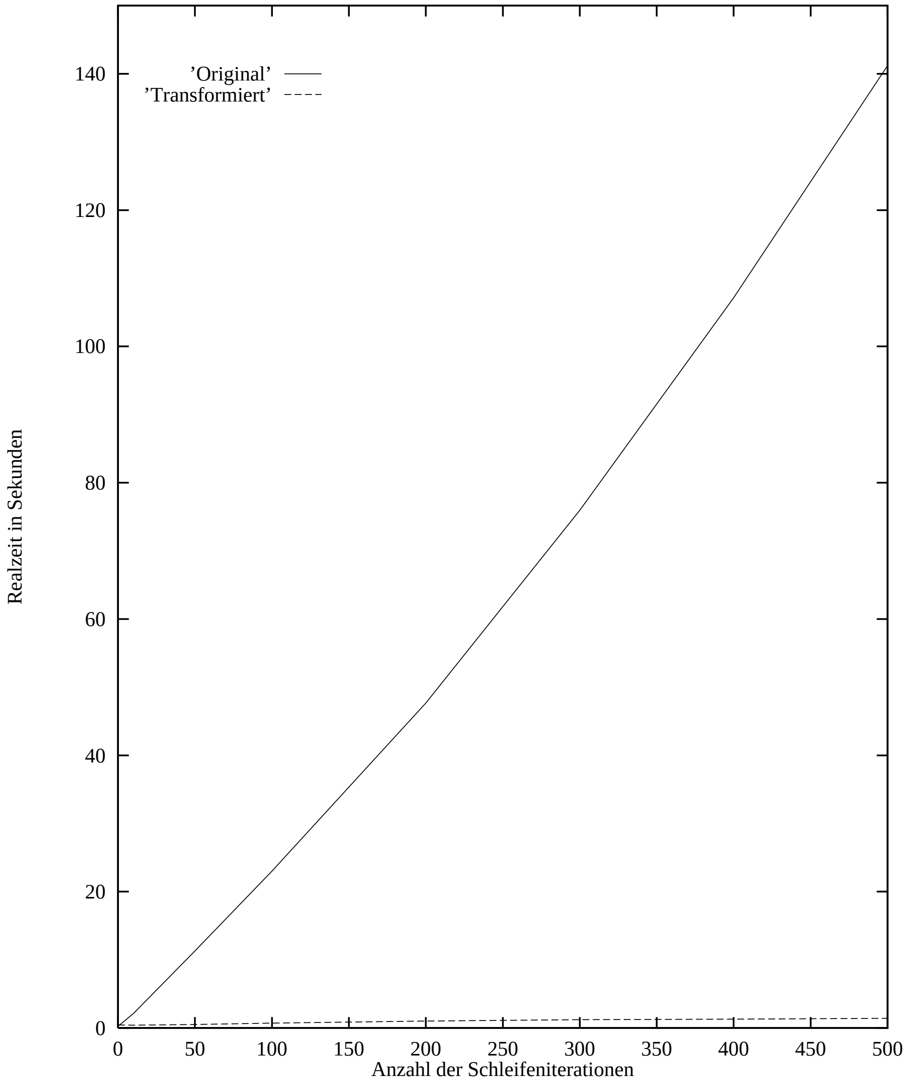

Abbildung 9.9: Ergebnisse der Laufzeittests für die Programme in Abbildung 9.7 und Abbildung 9.8.

Anzahl der Schleifeniterationen	Realzeit in Sekunden Original	Transformiert
0	0.28	0.50
1	0.44	0.44
10	2.10	0.40
50	11.3	0.50
100	23.02	0.70
200	47.68	1.00
300	75.96	1.20
400	107.12	1.30
500	141.20	1.42
600	179.20	1.64
700	219.60	1.80
800	262.80	2.00
900	310.70	2.24
1000	360.40	2.50

Tabelle 9.3: Ergebnisse der Laufzeittests für die Programme in Abbildung 9.7 und Abbildung 9.8.

```
program test4;
begin
   H := {1 .. 1000};
   F := {[2*i,3*i] : i in H};
   get(N);
   C := {1 .. N};
   G := {1, 1+3 .. 2500};
   while C /= {} do
      V := {[F(j),j] : j in G};
      x from C;
      F(x) := x;
   end while;
end test4;
```

Abbildung 9.10: Das vierte Testprogramm.

```
program test4;
begin
   H := {1 .. 1000};
   F := {[2*i,3*i] : i in H};
   get(N);
   C := {1 .. N};
   G := {1, 1+3 .. 2500};
   pstt1 := {[F(j),j] : j in G};
   while C /= {} do
      V := pstt1;
      x from C;
      if (x in G) then
         pstt1 less := [F(x),x];
      end if;
      F(x) := x;
      if (x in G) then
         pstt1 with := [F(x),x];
      end if;
   end while;
end test4;
```

Abbildung 9.11: Das transformierte Programm zum Programm in Abbildung 9.10.

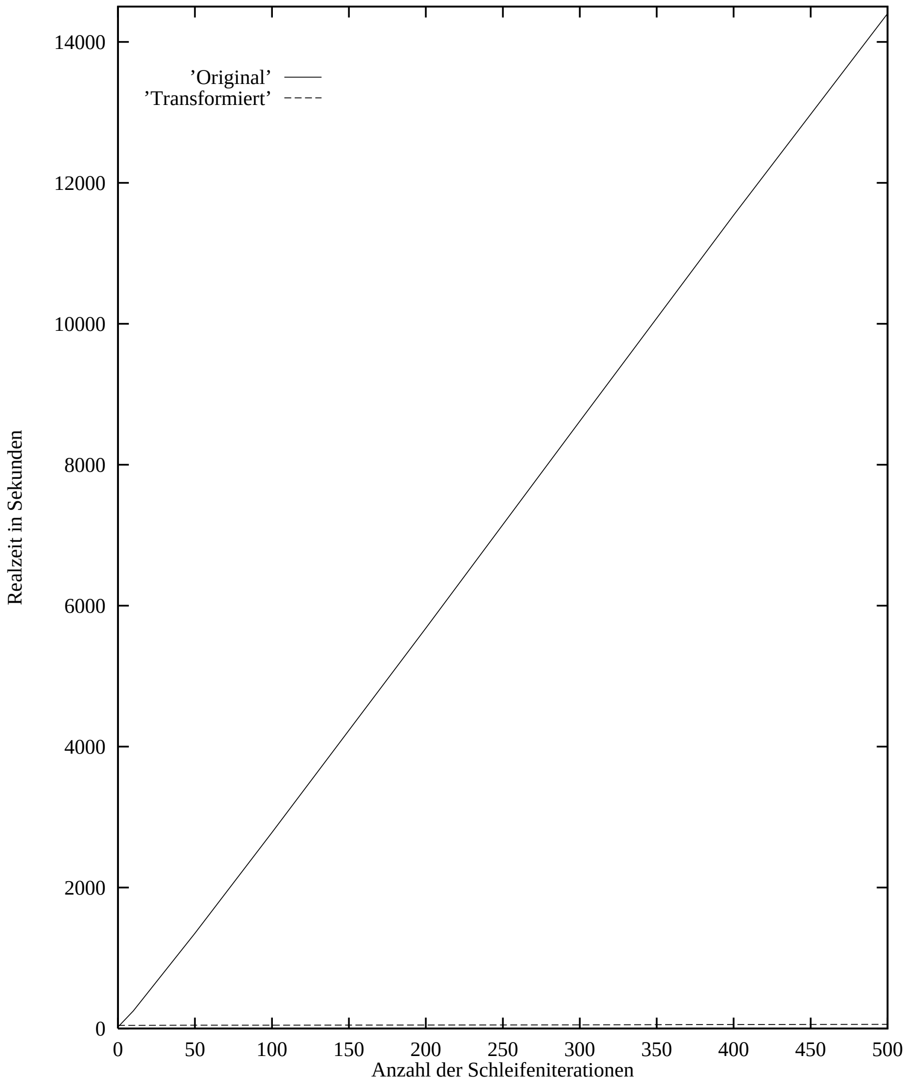

Abbildung 9.12: Ergebnisse der Laufzeittests für die Programme in Abbildung 9.10 und Abbildung 9.11.

Anzahl der Schleifeniterationen	Realzeit in Sekunden	
	Original	Transformiert
0	24.88	43.40
1	43.70	44.16
10	249.44	43.88
50	1350.10	45.76
100	2780.64	46.74
200	5680.12	49.36
300	8617.60	52.12
400	11539.76	56.08
500	14403.34	58.74

Tabelle 9.4: Ergebnisse der Laufzeittests für die Programme in Abbildung 9.10 und Abbildung 9.11.

```
program test5;
begin
   H := {1 .. 150};
   G := {[j, 2*j-j/2] : j in H};
   A := {1 .. 150};
   B := {1 .. 100};
   Vor := {[i, j] : i in A, j in B |
           (i = 2*j+1) or (j mod i = 0) or (i = j)};
   F := {1, 1+7 .. 111};
   get(N);
   C := {1 .. N};
   while C /= {} do
      Blaetter := {k : k in {j : j in F | G(j) in H}
                     | #Vor{k} = 1};
      x from C;
      F with := x;
   end while;
end test5;
```

Abbildung 9.13: Das fünfte Testprogramm.

```
program test5;
begin
   H := {1 .. 150};
   G := {[j, 2*j-j/2] : j in H};
   A := {1 .. 150};
   B := {1 .. 100};
   Vor := {[i, j] : i in A, j in B |
           (i = 2*j+1) or (j mod i = 0) or (i = j)};
   F := {1, 1+7 .. 111};
   get(N);
   C := {1 .. N};
   pstt1 := {j : j in F | G(j) in H};
   while C /= {} do
      Blaetter := {k : k in pstt1| #Vor{k} = 1};
      x from C;
      if G(x) in H then
         pstt1 with := x;
      end if;
      F with := x;
   end while;
end test5;
```

Abbildung 9.14: Das einmal transformierte Programm zum Programm in Abbildung 9.13.

```
program test5;
begin
   H := {1 .. 150};
   G := {[j, 2*j-j/2] : j in H};
   A := {1 .. 150};
   B := {1 .. 100};
   Vor := {[i, j] : i in A, j in B |
           (i = 2*j+1) or (j mod i = 0) or (i = j)};
   F := {1, 1+7 .. 111};
   get(N);
   C := {1 .. N};
   pstt1 := {j : j in F | G(j) in H};
   pstt2 := { [k,#Vor{k}] : k in domain Vor};
   while C /= {} do
      Blaetter := {k : k in pstt1 | pstt2(k)  = 1};
      x from C;
      if G(x) in H then
         pstt1 with := x;
      end if;
      F with := x;
   end while;
end test5;
```

Abbildung 9.15: Das zweimal transformierte Programm zum Programm in Abbildung 9.13.

```
program test5;
begin
   H := {1 .. 150};
   G := {[j, 2*j-j/2] : j in H};
   A := {1 .. 150};
   B := {1 .. 100};
   Vor := {[i, j] : i in A, j in B |
           (i = 2*j+1) or (j mod i = 0) or (i = j)};
   F := {1, 1+7 .. 111};
   get(N);
   C := {1 .. N};
   pstt1 := {j : j in F | G(j) in H};
   pstt2 := { [k,#Vor{k}] : k in domain Vor};
   pstt3 := { k : k in pstt1 | pstt2(k) = 1};
   while C /= {} do
      Blaetter := pstt3 ;
      x from C;
      if G(x) in H then
         if (pstt2(x) = 1) then
             pstt3 with := x;
         end if;
         pstt1 with := x;
       end if;
      F with := x;
   end while;
end test5;
```

Abbildung 9.16: Das dreimal transformierte Programm zum Programm in Abbildung 9.13.

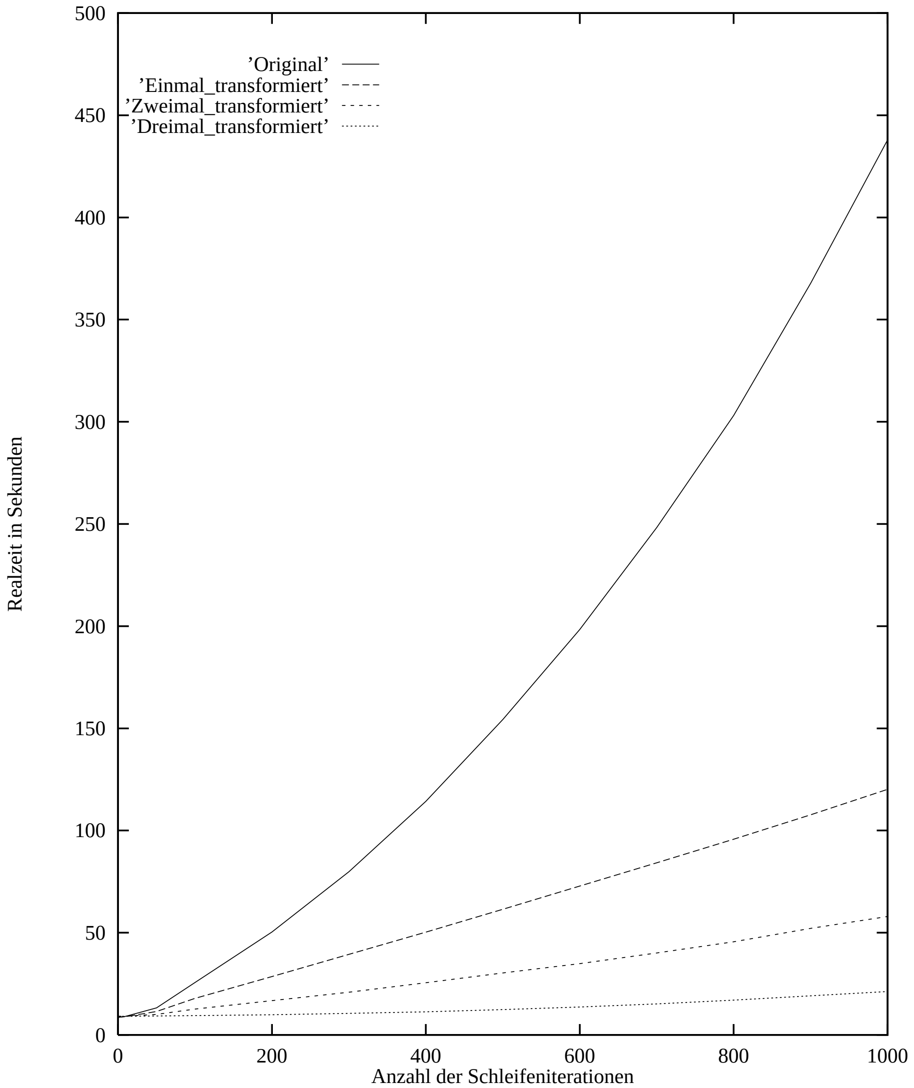

Abbildung 9.17: Ergebnisse der Laufzeittests für die fünfte Testreihe.

Anzahl der Schleifen-iterationen	Realzeit in Sekunden			
	Original	Einmal-transformiert	Zweimal-transformiert	Dreimal-transformiert
0	7.14	8.86	9.38	9.38
1	8.62	8.64	9.12	9.12
10	9.12	8.94	9.22	9.12
50	13.22	11.42	10.04	9.26
100	25.60	17.82	12.60	9.44
200	50.32	28.56	16.73	9.83
300	79.84	39.40	20.88	10.50
400	114.26	50.28	25.50	11.30
500	154.34	61.48	30.32	12.34
600	198.40	72.82	34.86	13.62
700	248.26	84.18	40.04	15.16
800	303.08	95.76	45.52	17.00
900	367.76	107.72	52.08	19.08
1000	437.96	120.16	57.92	21.18

Tabelle 9.5: Ergebnisse der Laufzeittests für die fünfte Testreihe.

10. Zusammenfassung zu Teil II

In diesem Teil der Arbeit wurde die Implementierung des Transformationsprogramms beschrieben. Zunächst wurden die Anforderungen spezifiziert und der verwendete Werkzeugkasten vorgestellt. Danach wurden die Implementierung und das Einfügen eines neuen Musters skizziert. Abschließend wurden die Ergebnisse einiger Laufzeittests vorgestellt.

Teil III

Erfahrungen und Ausblick

11. Erfahrungen

In dieser Arbeit wurde ein Transformationsprogramm für ProSet entworfen und implementiert. Das Transformationsprogramm basiert auf der endlichen Differentiation. In Kapitel 9 wurde an einigen ProSet-Programmen gezeigt, daß die Laufzeit dieser Programme wesentlich geringer ist, falls bei ihrer Übersetzung das Transformationsprogramm benutzt wird. In diesem Kapitel werden nun die Erfahrungen, die bei der Implementierung dieses Transformationsprogramms gesammelt wurden, diskutiert. Zunächst werden mögliche Verständnisschwierigkeiten der theoretischen Grundlage aufgezeigt, die durch die Präsentation der endlichen Differentiation durch R. Paige entstehen können (Abschnitt 11.1.1). Dann wird die Mächtigkeit der endlichen Differentiation (Abschnitt 11.1.2) und die Mächtigkeit des Transformationsprogramms (Abschnitt 11.2) diskutiert.

11.1 Erfahrungen mit der endlichen Differentiation

11.1.1 Anmerkungen zur Präsentation der endlichen Differentiation

Bei der Vorstellung der endlichen Differentiation schreibt R. Paige:

> „We sometimes use the mathematical function notation $C = f(x_1, \ldots, x_n)$ to uniquely associate a text expression f involving n distinct free variables $x_1, \ldots, x_n$ with a variable C (which we call the virtual variable associated with f)."[PK82, Seite 409]

> „In [...] reduction in strength (which we call finite differencing) is viewed as an extension of code motion whereby the major cost of evaluating an expression $E = f(x_1, \ldots, x_n)$ is moved outside a program region R despite modifications to its parameters $x_1, \ldots, x_n$ occuring within R. The basic idea of this technique can be expresses as follows: by making E available on entry to R (by evaluating f and storing its value in E immediately prior to R), and keeping E available within R (by appropriately modifying E each time one of the parameters $x_1, \ldots, x_n$ is modified),..."[PK82, Seite 410]

Diese Zitate können so interpretiert werden, daß im Beispiel in Abbildung 11.1 die Zuweisung „`G := {i : i in B | i in D}`“ als der Ausdruck „$E = f(x_1, \ldots, x_n)$“ im zweiten Zitat angesehen wird. „$E = f(x_1, \ldots, x_n)$“ ist in einigen Programmiersprachen auch eine Zuweisung (z.B. in C) und im ersten Zitat wird beschrieben, daß gelegentlich eine virtuelle Variable für einen applikativen Ausdruck verwendet wird. Bei dieser Interpretation würde das Programm in Abbildung 11.1 in das Programm in Abbildung 11.2 transformiert werden. Das Programm in Abbildung 11.1 gibt für `G` den Wert `om` aus, da `G` erst in der Schleife definiert wird, diese aber nie durchlaufen wird. Das Programm in Abbildung 11.2 gibt jedoch die Menge, die den Wert `2` enthält, aus. Diese Transformation ist somit nicht semantik-treu.

```
program beispiel1;
begin
   A := {};
   B := {1, 2};
   D := {2};
   while A /= {} do
      G := { i : i in B | i in D };
      x from A;
      D with := x;
   end while;
   put ( G );  -- Ausgabe om
end beispiel1;
```

Abbildung 11.1: Ein ProSet-Programm.

Ein ebenso falsches Ergebnis liefert die Transformation des Programms in Abbildung 11.3 in das Programm in Abbildung 11.4. Diese Beispiele unterscheiden sich nur in der Mächtigkeit der Menge `A`. Im Programm in Abbildung 11.3 wird die `while`-Schleife genau einmal durchlaufen. Im nichttransformierten Programm wird die Menge {`2`} ausgegeben, während im transformierten Programm die Menge {`1,2`} ausgegeben wird. Dieser Unterschied entsteht, da im transformierten Programm die Variable `G` modifiziert wird, wenn in die Menge `D` der Wert von `x` eingefügt wird.

An diesen Beispielen wird erkennbar, daß die Variable `G` nur dort modifiziert werden darf, wo sie im ursprünglichen Programm den Wert von „`{i : i in B | i in D}`“ erhält. Es wird also eine Variable zur Aufrechterhaltung der Invarianten „$V = \{i : i \text{ in } B \mid i \text{ in } D\}$“ bei der Modifikation von `D` benötigt. Aus diesem Grunde wurde in Abschnitt 3.1.1 die Theorie so neu formuliert, daß für jeden applikativen Ausdruck immer eine virtuelle Variable verwendet wird.

```
program beispiel1;
begin
   A := {};
   B := {1, 2};
   D := {2};
   G := { i : i in B | i in D };
   while A /= {} do
      x from A;
      if x in B then
         G with := x;
      end if;
      D with := x;
   end while;
   put ( G );  -- Ausgabe {2}
end beispiel1;
```

Abbildung 11.2: Ein transformiertes ProSet-Programm, das nicht semantik-treu zum Programm in Abbildung 11.1 ist.

Bei der Definition der Differenzierbarkeit (Definition 3.14 auf Seite 17) ist die erste Bedingung (Keine Benutzung von V in B lebt in P) immer gegeben. Für eine neue Variable im transformierten Programm gibt es im nichttransformierten Programm keine Benutzungen und keine Definitionen. Diese erste Bedingung würde nur dann benötigt, wenn die Ableitung für eine im ursprünglichen Programm vorhandene Variable betrachtet wird, wie es in den beiden obigen Beispielen geschehen ist. Daher könnte auch diese Definition dazu führen, daß Transformationen durchgeführt werden, die nicht semantik-treu sind. Bei den in [PK82] angegebenen Beispielprogrammen wird niemals einer Variablen ein zu differenzierender applikativer Ausdruck zugewiesen. Aus der Präsentation der Theorie wird daher nicht eindeutig ersichtlich, daß auch für applikative Ausdrücke, die einer Variablen zugewiesen werden, (neue) virtuelle Variablen benötigt werden.

```
program beispiel2;
begin
   A := {1};
   B := {1, 2};
   D := {2};
   while A /= {} do
      G := { i : i in B | i in D };
      x from A;
      D with := x;
   end while;
   put ( G );  -- Ausgabe {2}
end beispiel2;
```

Abbildung 11.3: Ein weiteres ProSet-Programm.

```
program beispiel2;
begin
   A := {1};
   B := {1, 2};
   D := {2};
   G := { i : i in B | i in D };
   while A /= {} do
      x from A;
      if x in B then
         G with := x;
      end if;
      D with := x;
   end while;
   put ( G );  -- Ausgabe {1,2}
end beispiel2;
```

Abbildung 11.4: Ein weiteres transformiertes ProSet-Programm, das nicht semantik-treu zum Programm in Abbildung 11.3 ist.

11.1.2 Anmerkungen zur Mächtigkeit der Theorie

Die endliche Differentiation wird von R. Paige nur auf Mengen angewendet. ProSet stellt auch Tupelformer (Abschnitt 2.2) zur Verfügung, die ebenfalls die Laufzeit eines Programms erhöhen und optimiert werden sollten. Weiterhin werden bei der endlichen Differentiation für die freien Variablen eines applikativen Ausdrucks, die eine Menge als Wert haben, nur für die folgenden Modifikationen Ableitungen angegeben.

```
A := {};
A with := x;
A less := x;
A := A with x;
A := A less x;
```

Kleine Änderungen in einem applikativen Ausdruck können jedoch auch durch die Anweisungen „`x from A`" oder „`A with := [x,y]`" entstehen, auf die die endliche Differentiation in der vorliegenden Form nicht anwendbar ist. Bei der ersten Anweisung erhält `x` seinen Wert erst durch die Zuweisung, so daß sein Wert bei einer Vorableitung nicht verfügbar ist. Da jedoch bei den wenigsten Ableitungen eine Nachableitung verwendet wird, könnte die Vorableitung von „`A less := x`" als Nachableitung von „`x from A`" verwendet werden. Die Semantik-Treue müßte jedoch noch bewiesen werden. In [PK82] werden bei den vorhandenen Ableitungen Annahmen über Spezialfälle gemacht, die die weitere Transformation der Ableitungen gewährleisten. Als Beispiel lassen sich dazu die Ableitungen des Musters in Abschnitt A.8 anführen. Dieses Muster entspricht in [PK82] dem Muster M2. R. Paige setzt dabei voraus, daß `V(q)` für `q` außerhalb des Definitionsbereiches von `V` den Wert `0` hat und, daß durch die Zuweisung von `0` an `V(q)` der Wert `q` aus dem Definitionsbereich von `V` entfernt wird. Bei einem anderen Muster wird vorausgesetzt, daß bestimmte Variablen in einem applikativen Ausdruck ganze Zahlen sind. Für ProSet müssen diese Spezialfälle jedoch in der Ableitung berücksichtigt werden, was dazu führt, daß auf diesen Ableitungen keine weiteren Transformationen mehr stattfinden können, und sie somit das Ende einer differenzierbaren Kette bilden.

Ein ähnliches Problem ergibt sich zum Beipsiel beim Muster in Abschnitt A.4. Bei der Zuweisung „`V{F(x)} := V{F(x)} with x`" muß sichergestellt sein, daß `F(x)` im Definitionsbereich von `V` liegt. Diese Überprüfung findet für das entsprechende Muster C1 in [PK82] nicht statt.

Abschließend sei dazu die folgende Anmerkung von R. Paige zur Mächtigkeit der endlichen Differentiation zitiert:

> „However, a significant problem remains concerning just how generally applicable our approach really is."[Pai84, Seite 338]

11.2 Anmerkungen zum Transformationsprogramm

Die implementierten Muster wurden zum größten Teil dem Artikel [PK80] entnommen. Bisher wurden kaum Erfahrungen mit der Sprache PROSET gesammelt, so daß nicht klar ist, ob die implementierten Muster den Anforderungen an PROSET entsprechen. R. Paige gibt für den applikativen Ausdruck „`{x : x in A | x mod 2 = 0}`" kein Muster an. Er betrachtet lediglich das Muster „`{x : x in A | B(x)}`" (A.9). Ein Ausdruck der ersten Form wird im Transformationsprogramm aber nicht von einem Muster der zweiten Form erkannt. Für das Transformationsprogramm wurde also ein weiteres Muster (A.26) implementiert, das diesen Ausdruck erkennt. Durch den Aufbau des Transformationsprogramms lassen sich jedoch relativ leicht neue Muster einfügen. Weiterhin gibt es in [PK80] Muster, die nicht in das Transformationsprogramm eingebaut werden konnten, da bei ihnen Voraussetzungen an die Typen freier Variablen gemacht werden. Da PROSET schwach getypt ist, können solche Voraussetzungen im allgemeinen nicht überprüft werden.

12. Ausblick

In diesem Kapitel wird beschrieben, an welchen Punkten in ProSet weitere Möglichkeiten für Optimierungen bestehen (Abschnitt 12.1) und wie das Transformationsprogramm erweitert werden könnte (Abschnitt 12.2). Zum Schluß werden dann andere Ansätze für Mengentransformationen diskutiert (Abschnitt 12.3).

12.1 Bessere Abstimmung zwischen dem ProSet-Compiler und dem Transformationsprogramm

In Abschnitt 3.1.3 wurde bei den Kosten vorausgesetzt, daß keine Kopieroperationen vorkommen, und daß die Mengen in einer effizienten Hashtabelle gespeichert werden. Die Annahme, daß keine Kopieroperationen verwendet werden, ist jedoch eine Annahme, die im allgemeinen nicht erfüllt ist. In [Sch75] werden Kopieroptimierungen für SETL beschrieben. Bei diesen Optimierungen werden Fälle untersucht, in denen das vollständige Kopieren eines Objektes vermieden werden kann. Diese und andere Optimierungen (z.B. die Wahl von effizienten Darstellungen für SETL-Strukturen) sind im SETL-Compiler integriert. In [Fea87] und [PS83] wird dieser SETL-Compiler kurz beschrieben. Durch ein Laufzeitsystem, das effiziente Hashtabellen für Mengen verwendet und Optimierungen, ähnlich den Optimierungen im SETL-Compiler, durchführt, sind weitere Effizienzsteigerungen zu erwarten.

Durch die Einführung einer strengen Typisierung in ProSet könnten weitere applikative Ausdrücke transformiert werden (siehe Abschnitt 11.1.1). Weiterhin können durch eine strenge Typisierung Mengenoperationen durch strikte Operationen ersetzt (siehe Abschnitt 3.3) und damit weitere Effizienzsteigerungen erreicht werden (siehe Seite 22).

12.2 Mögliche Erweiterungen für das implementierte Transformationsprogramm

In ProSet-Programmen werden auch häufig Tupelformer verwendet. Es scheint also sinnvoll zu sein, die Muster im Transformationsprogramm um Tupelformer zu erweitern. Da Tupel ähnlich wie Mengen erzeugt werden können, könnte eine Erweiterung der endlichen Differentiation möglich sein. Bei Tupeln muß berücksichtigt werden, daß die Reihenfolge der Komponenten relevant ist, daß Werte mehrfach auftreten können, und daß Komponenten den Wert `om` haben können. In Abschnitt 11.1.2 wurde bereits erwähnt, daß es in ProSet Anweisungen gibt, die einen applikativen Ausdruck nur geringfügig modifizieren (z.B. „`x from A`"), auf die die endliche Differentiation in der vorliegenden Form jedoch nicht anwendbar ist. Das Transformationsprogramm könnte mit Hilfe anderer Theorien, die vielleicht weniger effizient sind als die endliche Differentiation, aber dafür andere Programmkonstrukte transformieren können, erweitert werden. In Abschnitt 12.3 werden einige Ansätze für Mengentransformationen vorgestellt.

Es wäre interessant, das Transformationsprogramm in ein halbautomatisches System umzuwandeln. In Abschnitt 11.1.2 wurde beschrieben, daß es bei R. Paige Muster gibt, bei denen Voraussetzungen an die Typen der Variablen der applikativen Ausdrücke gestellt werden. Bei einem halbautomatischen System könnte bei einem solchen Muster dem Benutzer interaktiv die Möglichkeit gegeben werden, Transformationsdirektiven in sein Programm einzubauen, die Bereiche kennzeichnen, in denen transformiert werden kann oder nicht. Aufgrund der Transformationsdirektiven könnte dann bei einem weiteren Transformationsdurchlauf vom System entschieden werden, ob transformiert werden soll, ob also die Voraussetzungen als erfüllt angesehen werden können. Die folgenden Kommentare könnten zum Beispiel als Transformationsdirektiven verwendet werden.

```
-- do transformation:
-- end do transformation
-- no transformation:
-- end no transformation
```

Falls für ProSet eine Kontroll- und Datenflußanalyse verfügbar wäre, könnte unerreichbarer Code, der durch die endliche Differentiation eingefügt wird, eleminiert werden. Zur Zeit können in einer Schleife keine Transformationen durchgeführt werden, wenn in der Schleife eine Prozedur aufgerufen wird. Durch eine Kontroll- und Datenflußanalyse kann diese Voraussetzung abgeschwächt werden, da überprüft werden kann, ob Prozeduren Seiteneffekte auf Variablen des applikativen Ausdrucks haben.

12.3 Andere Ansätze für Mengentransformationen

In diesem Abschnitt werden Ansätze für Mengentransformationen von M. Sharir, A. Fong und ein kombinierter Ansatz aus den in dieser Arbeit beschriebenen Ansätzen diskutiert.

12.3.1 Formale Differenzbildung

M. Sharir hat in [Sha82] einen algebraischen Ansatz für Mengentransformationen präsentiert. Dieser theoretische Ansatz verwendet die symmetrische Differenz $\triangle$ bei geringen Modifikationen einer Menge S. Das Zufügen einer (kleinen) Menge DS zur Menge S, die zu S disjunkt ist, läßt sich zum Beispiel durch „$S = S \triangle DS$" darstellen. Für die Transformationen werden Eigenschaften der symmetrischen Differenz angewendet. In [DSW82] und [DF89] wird mit diesem Ansatz ein Algorithmus zur Speicherbereinigung hergeleitet. Dieser Ansatz wurde von M. Sharir nicht implementiert.

12.3.2 Induktiv berechenbare Konstrukte in expressiv hohen Programmiersprachen

A. Fong und J. Ullman haben bereits 1976 eine Theorie für Transformationen auf Booleschen Mengenoperationen veröffentlicht [FU76]. Diese Theorie wurde später auf Mengenformer erweitert [Fon79], jedoch nicht in dem Umfang wie es in [PK82] geschieht. A. Fong und J. Ullman definieren für jede Variable V in einer Schleife, die eine Menge zugewiesen bekommt, die Variablen $\Delta^-(V,p)$ und $\Delta^+(V,p)$, wobei p ein Punkt in der Schleife ist. $\Delta^-(V,p)$ ist die Menge aller Elemente von V, die, seitdem der Kontrollfluß das letzten Mal den Punkt p erreicht hat, aus V entfernt und nicht wieder eingefügt wurden. $\Delta^+(V,p)$ ist die Menge aller Elemente, die, seitdem der Kontrollfluß das letzten Mal den Punkt p erreicht hat, zu V hinzugefügt und nicht wieder entfernt wurden. Bei einer Definition von V kann dann der neue Wert von V aus dem alten Wert von V und den Variablen $\Delta^-(V,p)$ und $\Delta^+(V,p)$ berechnet werden. Über eine Implementierung dieses Ansatzes wird nichts berichtet.

12.3.3 Eine Idee für einen kombinierten Ansatz

In diesem Abschnitt wird als Abschluß dieser Arbeit eine Idee für einen weiteren Ansatz für Mengentransformationen vorgestellt. Diese Idee entstand bei der Entwicklung und Implementierung des Transformationsprogramms. Sie ist durch die in dieser Arbeit beschriebenen Ansätze geprägt. In [PK82] und [Sha82] wurde bereits vermutet, daß eine Kombination der verschiedenen Ansätze zu einer konzeptionellen und pragmatischen Verbesserung führen könnte.

Die hier vorgestellte Idee wird, wie die endliche Differentiation, auf beliebige Mengenformer angewendet. Wie beim Ansatz von A. Fong (siehe Abschnitt 12.3.2) werden Hilfsvariablen $\Delta^{-}A$ und $\Delta^{+}A$ verwendet. Der Beweis der Korrektheit der Transformationen kann mit Hilfe der Mengenlehre erfolgen. M. Sharir verwendet bei der formalen Differenzbildung ebenfalls die Mengenlehre (siehe Abschnitt 12.3.1).

Diese Idee wird im folgenden kurz informell vorgestellt. Im Gegensatz zu A. Fong werden nicht für Variablen V, die eine Menge zugewiesen bekommen, die Hilfsvariablen $\Delta^{-}V$ und $\Delta^{+}V$ definiert, sondern für die freien Variablen x_i eines Mengenformers. Für den Mengenformer „`{i : i in A | i in B}`“ werden zum Beispiel die Variablen $\Delta^{-}A$, $\Delta^{+}A$, $\Delta^{-}B$ und $\Delta^{+}B$ definiert. In diesen Variablen werden die Elemente, die innerhalb einer Iteration zu `A` oder `B` zugefügt bzw. entfernt werden, gespeichert. In einem Iterationsschritt wird dann der Mengenformer nur für die kleineren Mengen $\Delta^{-}A$, $\Delta^{+}A$, $\Delta^{-}B$ und $\Delta^{+}B$ berechnet, die dann entsprechend zur bereits berechneten Menge hinzugefügt bzw. entfernt werden.

Im Gegensatz zur endlichen Differentiation hängt diese Transformation nicht von der Struktur des Mengenformers ab. Bei dieser Transformation wird der Aufbau des Mengenformers für die Hilfsvariablen nicht betrachtet. Da zusätzlich auch Modifikationen der Form „`x from A`“ und „`A +:= B`“ betrachtet werden können, könnte diese Idee eine weitere Anwendbarkeit im Vergleich zur endlichen Differentiation haben. Es müßte noch untersucht werden, ob diese Idee auch effizient auf Tupelformer angewendet werden kann.

13. Zusammenfassung

Diese Arbeit beschäftigte sich mit Mengentransformationen für die mengen-orientierte Prototyping-Sprache ProSet. Im Rahmen dieser Aufgabe wurde zunächst ProSet vorgestellt und für das implementierte Transformationsprogramm die Theorie und Realisierung präsentiert. Die Effizienz dieses Ansatzes konnte durch Laufzeitvergleiche belegt werden. Aufgrund der Erfahrungen, die mit dem Transformationsprogramm gesammelt wurden, konnten im letzten Teil der Arbeit mögliche Verbesserungen des Ansatzes und Änderungen der Sprache ProSet diskutiert werden.

Literaturverzeichnis

[ASU86] A.V. Aho, R. Sethi, und J.D. Ullman. *Compilers, Principles, Techniques, and Tools.* Addison-Wesley Publishing Company, 1986.

[CP88] J. Cai und R. Paige. Program Derivation by Fixed Point Computation. In *Science of Computer Programming II*, Seiten 197–261. North-Holland, 1988.

[DF89] E.-E. Doberkat und D. Fox. *Software Prototyping mit SETL.* B.G. Teubner Stuttgart, 1989.

[DFG+92] E.-E. Doberkat, W. Franke, U. Gutenbeil, W. Hasselbring, U. Lammers, und C. Pahl. ProSet — Prototyping with Sets: Language Definition. Informatik-Bericht 2/92, Universität - Gesamthochschule - Essen, April 1992.

[Dob92] E.-E. Doberkat. Integrating persistence into a set-oriented prototyping language. *Structured Programming*, 13(3):137–153, März 1992.

[DSW82] R.B.K. Dewar, M. Sharir, und E. Weixelbaum. Transformational Derivation of a Garbage Collection Algorithm. *ACM Transactions on Programming Languages and Systems*, 4(4):650–667, 1982.

[Fea87] M.S. Feather. A Survey and Classification of some Program Transformation Approaches and Techniques. In Meertens, L.G.L.T., Hrsg., *program specification and transformation*, Seiten 165–195. North-Holland, 1987.

[Fon79] A.C. Fong. Inductively Computable Constructs in Very High Level Languages. In *Conference Record of the 6th ACM Symposium on Principles of Programming Languages*, 1979.

[FU76] A.C. Fong und J.D. Ullman. Induction Variables in Very High Level Languages. In *Conference Record of the 3th ACM Symposium on Principles of Programming Languages*, 1976.

[GE90a] J. Grosch und H. Emmelmann. A Tool Box for Compiler Construction. Compiler Generation Report No. 20, GMD Forschungsstelle an der Universität Karlsruhe, Januar 1990.

[GE90b] J. Grosch und H. Emmelmann. Werkzeuge für den Übersetzerbau. Compiler Generation Report No. 21, GMD Forschungsstelle an der Universität Karlsruhe, Februar 1990.

[GHL+92] R. Gray, V. Heuring, S. Levi, A. Sloane, und W. Waite. Eli: A Complete, Flexible Compiler Construction System. *Communications of the ACM*, 35(2), Februar 1992.

[Gro88a] J. Grosch. Lalr — A Generator for Efficient Parsers. Compiler Generation Report No. 10, GMD Forschungsstelle an der Universität Karlsruhe, Oktober 1988.

[Gro88b] J. Grosch. Selected Examples of Scanner Specifications. Compiler Generation Report No. 7, GMD Forschungsstelle an der Universität Karlsruhe, März 1988.

[Gro89] J. Grosch. Tool Support for Data Structures. Compiler Generation Report No. 17, GMD Forschungsstelle an der Universität Karlsruhe, November 1989.

[Gro91a] J. Grosch. Ag — An Attribute Evaluator Generator. Compiler Generation Report No. 16, GMD Forschungsstelle an der Universität Karlsruhe, September 1991.

[Gro91b] J. Grosch. Puma — A Generator for the Transformation of Attributed Trees. Compiler Generation Report No. 26, GMD Forschungsstelle an der Universität Karlsruhe, November 1991.

[Gro91c] J. Grosch. Specification of a MiniLAX — Interpreter. Compiler Generation Report No. 22, GMD Forschungsstelle an der Universität Karlsruhe, November 1991.

[Gro91d] J. Grosch. Toolbox Introduction. Compiler Generation Report No. 25, GMD Forschungsstelle an der Universität Karlsruhe, Mai 1991.

[Gro91e] J. Grosch. Transformation of Attributed Trees Using Pattern Matching. Compiler Generation Report No. 27, GMD Forschungsstelle an der Universität Karlsruhe, August 1991.

[Gro92a] J. Grosch. Ast — A Generator for Abstract Syntax Trees. Compiler Generation Report No. 15, GMD Forschungsstelle an der Universität Karlsruhe, August 1992.

[Gro92b] J. Grosch. Preprocessors. Compiler Generation Report No. 24, GMD Forschungsstelle an der Universität Karlsruhe, August 1992.

[Gro92c] J. Grosch. Reusable Software — A Collection of C–Modules. Compiler Generation Report No. 30, GMD Forschungsstelle an der Universität Karlsruhe, August 1992.

[Gro92d] J. Grosch. Rex — A Scanner Generator. Compiler Generation Report No. 5, GMD Forschungsstelle an der Universität Karlsruhe, Juli 1992.

[GV92] J. Grosch und B. Vielsack. The Parser Generators Lalr and Ell. Compiler Generation Report No. 8, GMD Forschungsstelle an der Universität Karlsruhe, Juli 1992.

[Has93] W. Hasselbring. Prototyping parallel algorithms with ProSet-Linda. In J. Volkert, Hrsg., *Parallel Computation (Proc. Second International ACPC Conference)*, Band 734 von *Lecture Notes in Computer Science*, Seiten 135–150, Gmunden, Austria, Oktober 1993. Springer-Verlag.

[Kas] U. Kastens. *LIDO — A Language for Attribute Grammar Specification.* University of Paderborn, D-4790 Paderborn.

[Läu91] K. Läufer. Comparing three approaches to transformational programming. Technischer Bericht, Courant Institute of Mathematical Sciences, New York University, 1991.

[Pai84] R. Paige. Supercompilers — Extended Abstract. In Pepper, P., Hrsg., *Program Transformation and Programming Environment*, Seiten 331–340. Springer-Verlag, 1984.

[Pai86] R. Paige. Programming with Invariants. *IEEE Software*, Seiten 56–69, 1986.

[PK80] R. Paige und S. Koenig. Finite Differencing of Computable Expressions. Technischer Bericht LCSR-TR-8, Dep. of Computer Science, Rutgers Univ., New Brunswick, N.J., August 1980.

[PK82] R. Paige und S. Koenig. Finite Differencing of Computable Expressions. *ACM Transactions on Programming Languages and Systems*, 4(3):402–454, 1982.

[PS83] H. Partsch und R. Steinbrüggen. Program Transformation Systems. *Computing Surveys*, 15(3):199–236, 1983.

[Ram92] N. Ramsey. Literate-Programming Tools Need Not Be Complex. *noweb*, August 1992.

[Sch75] J.T. Schwartz. Optimization of Very High Level languages — I. *Computer Languages*, 1:161–194, 1975.

[Sha82] M. Sharir. Some Observations Concerning Formal Differentiation of Set Theoretic Expressions. *ACM Transactions on Programming Languages and Systems*, 4(2):196–225, 1982.

[SL94] M. Schumann und A. Lobinger. Zeitspiel. *iX*, Seiten 168–173, November 1994.

[Sta93] S. Stapelberg. *UNIX SYSTEM V.4 für Einsteiger und Fortgeschrittene*. Addison-Wesley, 1993.

[Vie89] B. Vielsack. Spezifikation und Implementierung der Transformation attributierter Bäume. Diplomarbeit, Universität Karlsruhe, Fakultät für Informatik, Juni 1989.

[WG85] W.M. Waite und G. Goos. *Compiler Construction*. Springer-Verlag, 1985.

Index

Teil IV

Anhänge

A. Muster und ihre Beweise

In diesem Anhang werden die implementierten Muster aufgeführt. Diese Muster wurden zum größten Teil [PK80] entnommen und an ProSet angepaßt. Es konnten jedoch nicht alle angegebenen Muster für ProSet verwendet werden, da ProSet schwach getypt ist und zum Beispiel bei der Zuweisung „`V := F + G`“ nicht entscheidbar ist, ob es sich bei `F` und `G` um Tupel oder Mengen handelt. Bei einigen dort angegebenen Mustern enthalten die Ableitungen neue Mengenformer. Diese Ableitungen werden für ProSet nicht verwendet, da es nicht sicher ist, ob sie immer profitabel differenzierbar sind.

Für jedes Muster ist eine Tabelle angegeben, in der zuerst die Kennung und die Initialisierung aufgeführt sind und dann die profitabel differenzierbaren Modifikationen und ihre Vor- und Nachableitungen (In [PK80] sind alle Ableitungen aufgeführt). In den Abschnittsüberschriften werden die applikativen Ausdrücken und die virtuellen Variablen angegeben.

Für den Beweis der Semantik-Treue der endlichen Differentiation muß Definition 3.13 in Abschnitt 3.1.2 überprüft werden. Für jede Definition dx_i muß der Block

> **achieve** $V = f(x_1, \ldots, x_n)$;
> $\partial^- V \ \langle dx_i \rangle$
> dx_i
> $\partial^+ V \ \langle dx_i \rangle$
> **assert** $V = f(x_1, \ldots, x_n)$;

semantik-treu sein. Bei einigen Ableitungen wird die Semantik-Treue exemplarisch überprüft.

A.1 V = {j : j in F | j in G}

Kennung	Initialisierung
`A1`	$V := \{j : j \text{ in } F \mid j \text{ in } G\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

F with := x;	if x in G then
	  V with := x;
	end if;	

F less := x;	if x in G then
	  V less := x;
	end if;	

G with := x;	if x in F then
	  V with := x;
	end if;	

G less := x;	if x in F then
	  V less := x;
	end if;	

virtuelle Variable: V
freie Variablen: F und G
gebundene Variable: j

Wird in F durch die Modifikation „F with := x“ ein neues Element x eingefügt, ist dieses Element nach der Modifikation in der Menge „{j : j in F | j in G}“ enthalten, falls x auch in G enthalten ist. Hierdurch folgt die Semantik-Treue der Ableitung bezüglich „F with := x“.

Die Beweise der anderen Ableitungen folgen diesem Prinzip.

A.2 V = {j : j in F | j notin G}

Kennung	Initialisierung
`A2`	$V := \{j : j$ in $F \mid j$ notin $G\}$;

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

F with := x;	if x notin G then
	V with := x;
	end if;	

F less := x;	if x notin G then
	V less := x;
	end if;	

G with := x;	if x in F then
	V less := x;
	end if;	

G less := x;	if x in F then
	V with := x;
	end if;	

virtuelle Variable: V
freie Variablen: F und G
gebundene Variable: j

Dieses Muster unterscheidet sich vom vorgehenden Muster nur in der Bedingung durch das „notin“ anstelle des „in“. Die Beweise der Semantik-Treue erfolgen nach dem gleichen Prinzip, wie beim vorhergehenden Muster.

A.3 V = {j : j in F | G(j) = H}

Bei diesem Muster wird vorausgesetzt, daß H ein konstanter Integerwert ist.

Kennung	Initialisierung
A3	$V := \{j : j \text{ in } F \mid G(j) = H\}$;

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

F with := x;	if $G(x) = H$ then
	    V with := x;
	end if;	

F less := x;	if $G(x) = H$ then
	    V less := x;
	end if;	

$G(x) := G(x) + const$;	if $const$ /= 0 then
	    if x in F then
	        if $G(x) = H$ then
	            V less := x;
	        else if $G(x) = H - const$ then
	            V with := x;
	        end if;
	    end if;
	end if;	

$G(x) := G(x) - const$;	if $const$ /= 0 then
	    if x in F then
	        if $G(x) = H$ then
	            V less := x;
	        else if $G(x) = H + const$ then
	            V with := x;
	        end if;
	    end if;
	end if;	

virtuelle Variable: V
freie Variablen: F und G
konstanter Integerwert: H
gebundene Variable: j

Wird der Wert von $G(x)$ durch die Definition „$G(x) := G(x) + const$“ modifiziert, ergibt sich für die Menge „$\{j : j \text{ in } F \mid G(j) = H\}$“ die folgende Veränderung. Ist x in F enthalten und galt vor der Definition „$G(x) = H$“, ist x nach dieser Definition nicht mehr in der Menge enthalten. Galt jedoch vor der Definition „$G(x) = H$ - $const$“ und x ist in F enthalten, gilt nach der Definition „$G(x) = H$“ und x ist muß in die Menge eingefügt werden.

A.4 V = {[F(j), j] : j in G}

Kennung	Initialisierung	
`A4`	$V := \{[F(j), j] : j \text{ in } G\};$	
dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$
G with := x;	V with := $[F(x),x]$;	
G less := x;	V less := $[F(x),x]$;	

$F(x)$:= om;	if x in G then
	V less := $[F(x),x]$;
	end if;
		if x in G then
		V with := $[F(x),x]$;
		end if;

$F(x)$:= y;	if x in G then
	V less := $[F(x),x]$;
	end if;
		if x in G then
		V with := $[F(x),x]$;
		end if;

virtuelle Variable: V
freie Variablen: F und G
gebundene Variable: j

In [PK80] wird die Ableitung „$V\{F(x)\}$:= $V\{F(x)\}$ with x" verwendet. In ProSet gibt es dabei einen Laufzeitfehler, wenn $F(x)$ den Wert „om" annimmt. Die Verwendung von „V with := $[F(x),x]$" führt dazu, daß keine weiteren Transformationen auf V durchgeführt werden können. Ein weiterer Unterschied zu [PK80] ist, daß im Falle „$F(x)$:= om" das Tupel $[F(x),x]$ in der Menge V enthalten bleibt und daher eine Nachableitung verwendet werden muß.

A.5 V{q} = {j : j in G | F(j) = q}

Bei diesem Muster wird vorausgesetzt, daß q im Wertebereich von F liegt. In diesem Fall stimmt V mit der berechneten Menge in Abschnitt A.4 überein.

Kennung	Initialisierung
`A5`	$V := \{[F(j), j] : j \text{ in } G\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$
G with := x;	V with := [$F(x)$,x];	
G less := x;	V less := [$F(x)$,x];	

$F(x)$:= om;	if x in G then
	V less := [$F(x)$,x];
	end if;
		if x in G then
		V with := [$F(x)$,x];
		end if;

$F(x)$:= y;	if x in G then
	V less := [$F(x)$,x];
	end if;
		if x in G then
		V with := [$F(x)$,x];
		end if;

virtuelle Variable: V
freie Variablen: q, F und G
gebundene Variable: j

A.6 V = #F

Kennung	Initialisierung
`A6`	$V := \#F;$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$
F with := x;	if x notin F then	
	V+:= 1;	
	end if;	end if;
F less := x;	if x in F then	
	V−:= 1;	

virtuelle Variable: V
freie Variable: F

A.7 V ={[j, #F{j}] : j in domain F}

Kennung	Initialisierung
`A7`	$V := \{[j, \#F\{j\}] : j \text{ in domain } F\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if x notin V then
	$V(x) := 1;$
	else $V(x) := V(x) + 1;$
	end if;	

$F\{x\} := F\{x\}$ less y;	$V(x) := V(x) - 1;$	

virtuelle Variable: V
freie Variablen: F
gebundene Variable: j

A.8 V(q) = #F{q}

Es wird vorausgesetzt, daß j im Definitionsbereich von F liegt. V stimmt dann mit der Menge in Abschnitt A.7 überein.

Kennung	Initialisierung
`A8`	$V := \{[j, \#F\{j\}] : j \text{ in domain } F\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if x notin V then
	$V(x) := 1;$
	else $V(x) := V(x) + 1;$
	end if;	

$F\{x\} := F\{x\}$ less y;	$V(x) := V(x) - 1;$	

virtuelle Variable: V
freie Variablen: q und F

A.9 V = {j : j in F | G(j)}

Kennung	Initialisierung	
A9	$V := \{j : j \text{ in } F \mid G(j)\};$	
dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

F with := x;	if $G(x)$ then
	    V with := x;
	end if;	

F less := x;	if $G(x)$ then
	    V less := x;
	end if;	

$G(x)$:= true;	if not $G(x)$ then
	    if x in F then
	        V with := x;
	    end if;
	end if;	

$G(x)$:= false;	if $G(x)$ then
	    if x in F then
	        V less := x;
	    end if;
	end if;	

virtuelle Variable: V
freie Variablen: F und G
gebundene Variable: j

A.10 V = {j : j in F}

Kennung	Initialisierung	
A10	$V := \{j : j \text{ in } F\};$	
dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$
F with := x;	V with := x;	
F less := x;	V less := x;	

virtuelle Variable: V
freie Variablen: F
gebundene Variable: j

A.11 V = {j : j in F | G(j) /= H}

Bei diesem Muster wird vorausgesetzt, daß H ein konstanter Integerwert ist.

Kennung	Initialisierung
`A11`	$V := \{j : j \text{ in } F \mid G(j)/=H\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

F with := x;	if $G(x)$ /= H then
	V with := x;
	end if;	

F less := x;	if $G(x)$ /= H then
	V less := x;
	end if;	

$G(x) := G(x) + const$;	if $const$ /= 0 then
	if x in F then
	if $G(x) = H$ then
	V with := x;
	else if $G(x) = H - const$ then
	V less := x;
	end if;
	end if;
	end if;	

$G(x) := G(x) - const$;	if $const$ /= 0 then
	if x in F then
	if $G(x) = H$ then
	V with := x;
	else if $G(x) = H + const$ then
	V less := x;
	end if;
	end if;
	end if;	

virtuelle Variable: V
freie Variablen: F und G
konstanter Integerwert: H
gebundene Variable: j

A.12 V = {[[F(j), k], j] : [k, j] in G}

Kennung	Initialisierung
`A12`	$V := \{[[F(j), k], j] : [k, j] \text{ in } G\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$
$G\{x\} := G\{x\}$ with y;	V with $:= [[F(y), x], y];$	
$G\{x\} := G\{x\}$ less y;	V less $:= [[F(y), x], y];$	

virtuelle Variable: V
freie Variablen: F und G
gebundene Variable: j und k

A.13 V = {[j, k] : [j, k] in F | k in G}

Kennung	Initialisierung
`A13`	$V := \{[j, k] : [j, k] \text{ in } F \mid k \text{ in } G\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if y in G then
	    V with $:= [x, y];$
	end if;	

$F\{x\} := F\{x\}$ less y;	if y in G then
	    V less $:= [x, y];$
	end if;	

virtuelle Variable: V
freie Variablen: F und G
gebundene Variable: j und k

A.14 $V\{q\} = \{k : k \text{ in } F\ \{q\} \mid q \text{ in } G\}$

Es wird vorausgesetzt, daß q im Definitionsbereich von F liegt. Dann stimmt die Menge mit der Menge in Abschnitt A.13 überein.

Kennung	Initialisierung
A14	$V := \{[j,k] : [j,k] \text{ in } F \mid k \text{ in } G\};$

dx_i	$\partial^- V\ \langle dx_i \rangle$	$\partial^+ V\ \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if y in G then
	V with $:= [x,y]$;
	end if;	

$F\{x\} := F\{x\}$ less y;	if y in G then
	V less $:= [x,y]$;
	end if;	

virtuelle Variable: V
freie Variablen: q, F und G
gebundene Variable: k

A.15 $V\{q\} = \{k : k \text{ in } G \mid k \text{ in } F\{q\}\}$

Es wird vorausgesetzt, daß q im Definitionsbereich von F liegt. Dann stimmt die Menge mit der Menge in Abschnitt A.13 überein.

Kennung	Initialisierung
A15	$V := \{[F,j] : [F,j] \text{ in } G \mid j \text{ in } H\};$

dx_i	$\partial^- V\ \langle dx_i \rangle$	$\partial^+ V\ \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if y in G then
	V with $:= [x,y]$;
	end if;	

$F\{x\} := F\{x\}$ less y;	if y in G then
	V less $:= [x,y]$;
	end if;	

virtuelle Variable: V
freie Variablen: q, F und G
gebundene Variable: k

A.16 V = {[j,k] : [j,k] in F | k notin G}

Kennung	Initialisierung
A16	$V := \{[j,k] : [j,k]$ in $F \mid k$ notin $G\}$;

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if y notin G then
	    $V\{x\} := V\{x\}$ with y;
	end if;	

$F\{x\} := F\{x\}$ less y;	if y notin G then
	    $V\{x\} := V\{x\}$ less y;
	end if;	

virtuelle Variable: V
freie Variablen: F und G
gebundene Variable: j und k

A.17 V{q} = {k : k in F{q} | k notin G}

Es wird vorausgesetzt, daß q im Definitionsbereich von F liegt. Dann stimmt die Menge mit der Menge in Abschnitt A.16 überein.

Kennung	Initialisierung
A17	$V := \{[j,k] : [j,k]$ in $F \mid k$ notin $G\}$;

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if y notin G then
	    $V\{x\} := V\{x\}$ with y;
	end if;	

$F\{x\} := F\{x\}$ less y;	if y notin G then
	    $V\{x\} := V\{x\}$ less y;
	end if;	

virtuelle Variable: V
freie Variablen: q, F und G
gebundene Variable: k

A.18 V = {j : j in F | G(j) in H}

Kennung	Initialisierung	
`A18`	$V := \{j : j \text{ in } F \mid G(j) \text{ in } H\};$	
dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

F with $:= x;$	if $G(x)$ in H then
	    V with $:= x;$
	end if;	

F less $:= x;$	if $G(x)$ in H then
	    V less $:= x;$
	end if;	

$G(x) :=$ om;	if x in F then
	    $\partial^- V \langle F$ less $:= x; \rangle$
	end if;
		if x in F then
		    $\partial^- V \langle F$ with $:= x; \rangle$
		end if;

$G(x) := y;$	if x in F then
	    $\partial^- V \langle F$ less $:= x; \rangle$
	end if;
		if x in F then
		    $\partial^- V \langle F$ with $:= x; \rangle$
		end if;

virtuelle Variable: V
freie Variablen: F und G und H
gebundene Variable: j

A.19 V = {j : j in F | G(j) notin H}

Kennung	Initialisierung
`A19`	$V := \{j : j \text{ in } F \mid G(j) \text{ notin } H\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

F with := x;	if $G(x)$ notin H then
	    V with := x;
	end if;	

F less := x;	if $G(x)$ notin H then
	    V less := x;
	end if;	

$G(x)$:= om;	if x in F then
	    $\partial^- V \langle F$ less := x; $\rangle$
	end if;
		if x in F then
		    $\partial^- V \langle F$ with := x; $\rangle$
		end if;

$G(x) := y$;	if x in F then
	    $\partial^- V \langle F$ less := x; $\rangle$
	end if;
		if x in F then
		    $\partial^- V \langle F$ with := x; $\rangle$
		end if;

virtuelle Variable: V
freie Variablen: F und G und H
gebundene Variable: j

A.20 V = {[j,k] : [j,k] in F | G(k) in H}

Kennung	Initialisierung
`A20`	$V := \{[j,k] : [j,k] \text{ in } F \mid G(k) \text{ in } H\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if $G(y)$ in H then
	    V with := $[x, y]$;
	end if;	

$F\{x\} := F\{x\}$ less y;	if $G(y)$ in H then
	    V less := $[x, y]$;
	end if;	

virtuelle Variable: V
freie Variablen: F und G und H
gebundene Variable: j und k

A.21 V{q} = {k : k in F {q}| G(k) in H}

Es wird vorausgesetzt, daß q im Definitionsbereich von F liegt. Dann stimmt die Menge mit der Menge in Abschnitt A.20 überein.

Kennung	Initialisierung
`A21`	$V := \{[j,k] : [j,k] \text{ in } F \mid G(k) \text{ in } H\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if $G(y)$ in H then
	  V with $:= [x,y]$;
	end if;	

$F\{x\} := F\{x\}$ less y;	if $G(y)$ in H then
	  V less $:= [x,y]$;
	end if;	

virtuelle Variable: V
freie Variablen: q, F, G und H
gebundene Variable: k

A.22 V = {[j,k] : [j,k] in F | G(k) notin H}

Kennung	Initialisierung
`A22`	$V := \{[j,k] : [j,k] \text{ in } F \mid G(k) \text{ notin } H\};$

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

$F\{x\} := F\{x\}$ with y;	if $G(y)$ notin H then
	  V with $:= [x,y]$;
	end if;	

$F\{x\} := F\{x\}$ less y;	if $G(y)$ notin H then
	  V less $:= [x,y]$;
	end if;	

virtuelle Variable: V
freie Variablen: F und G und H
gebundene Variable: j und k

A.23 V{q} = {k : k in F{q} | G(k) notin H}

Es wird vorausgesetzt, daß q im Definitionsbereich von F liegt. Dann stimmt die Menge mit der Menge in Abschnitt A.22 überein.

Kennung	Initialisierung
`A23`	$V := \{[j,k] : [j,k] \text{ in } F \mid G(k) \text{ notin } H\}$;

dx_i	$\partial^- V\ \langle dx_i \rangle$	$\partial^+ V\ \langle dx_i \rangle$

$F\{x\}$:= $F\{x\}$ with y;	if $G(y)$ notin H then
	V with := $[x,y]$;
	end if;	

$F\{x\}$:= $F\{x\}$ less y;	if $G(y)$ notin H then
	V less := $[x,y]$;
	end if;	

virtuelle Variable: V
freie Variablen: q, F, G und H
gebundene Variable: k

A.24 V = {[j, k] : k in F, j in G{k}}

Kennung	Initialisierung
`A24`	$V := \{[j,k] : k \text{ in } F, j \text{ in } G\{k\}\}$;

dx_i	$\partial^- V\ \langle dx_i \rangle$	$\partial^+ V\ \langle dx_i \rangle$

F with := x;	for y in $G\{x\}$ do
	V with := $[y,x]$;
	end for;	

F less := x;	for y in $G\{x\}$ do
	V less := $[y,x]$;
	end for;	

$G\{x\}$:= $G\{x\}$ with y;	if x in F then
	V with := $[y,x]$;
	end if;	

$G\{x\}$:= $G\{x\}$ less y;	if x in F then
	V less := $[y,x]$;
	end if;	

virtuelle Variable: V
freie Variablen: F und G
gebundene Variable: j und k

A.25 $V\{q\} = \{k : k \text{ in } F, q \text{ in } G\{k\}\}$

Es wird vorausgesetzt, Daß q im Definitionsbereich von G liegt. Die Menge stimmt dann mit der Menge in Abschnitt A.24 überein.

Kennung	Initialisierung
A25	$V := \{[j,k] : k \text{ in } F,\ j \text{ in } G\{k\}\};$

dx_i	$\partial^- V\ \langle dx_i \rangle$	$\partial^+ V\ \langle dx_i \rangle$

F with := x;	for y in $G\{x\}$ do
	  V with := $[y,x]$;
	end for;	

F less := x;	for y in $G\{x\}$ do
	  V less := $[y,x]$;
	end for;	

$G\{x\}$:= $G\{x\}$ with y;	if x in F then
	  V with := $[y,x]$;
	end if;	

$G\{x\}$:= $G\{x\}$ less y;	if x in F then
	  V less := $[y,x]$;
	end if;	

virtuelle Variable: V
freie Variablen: q, F und G
gebundene Variable: k

A.26 V = $\{$k : k in F $|$ k op1 *intlit1* op2 *intlit2*$\}$

Durch dieses Muster wird zum Beispiel der Ausdruck `{x : x in A | x mod 2 = 0}` erkannt. `op1` und `op2` stehen für binäre Operatoren[1] und *intlit1* und *intlit2* für konstante ganze Zahlen.

Kennung	Initialisierung	
`A26`	$V := \{k : k$ in $F \mid k$ binop1 $intlit1$ binop2 $intlit2\}$;	
dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$

F with := x;	if (x op1 intlit1 op2 intlit2) then
	V with := x;
	end if;	

F less := x;	if (x op1 intlit1 op2 intlit2) then
	V less := x;
	end if;	

virtuelle Variable: V
freie Variablen: F
gebundene Variable: k
Konstanten : $intlit1$ und $intlit2$
Operatoren: $op1$ und $op2$

[1] Unter binären Operatoren werden hier alle zweistelligen Operatoren verstanden, wie zum Beispiel arithmetische, Boolesche und relationale Operatoren.

B. Das Einfügen eines neuen Musters in das Transformationsprogramm

In diesem Kapitel wird anhand des applikativen Ausdrucks in Abbildung B.1 das Einfügen eines Musters in das Transformationsprogramm erläutert. Dieses Einfügen erfolgt manuell.

```
{ j : j in F | G(j) = 100 };
    -- 100 steht fuer eine beliebige ganzzahlige Konstante
```

Abbildung B.1: Ein applikativer Ausdruck.

Das Einfügen gliedert sich in neun Schritte. Im ersten Schritt werden die Ableitungen für den applikativen Ausdruck erzeugt. In den Schritten 2 bis 4 und 8 werden einige Strukturen und Funktionen für den applikativen Ausdruck modifiziert bzw. neu geschrieben. In den Schritten 5 und 7 wird das Muster des applikativen Ausdrucks mit den dazugehörigen Modifikationsmuster in die Funktion `Pattern` eingebaut. In den Schritten 6 und 9 werden die Ableitungsfunktionen und die Initialisierungsfunktionen geschrieben.

Schritt 1: Zunächst ist zu überlegen, welche Ableitungen für diesen Ausdruck profitabel differenzierbar sind. An freien Variablen existieren F und G. Bei F kann es sich um eine Menge oder um ein Tupel und bei G um eine Abbildung oder ebenfalls ein Tupel handeln. Für folgende Definitionen der freien Variablen können Ableitungen angegeben werden:

```
F := {};
F := [];
F with := x;
F less := x;
```

```
G(x) := G(x) + const1;
        -- const1 ist eine beliebige konstante ganze Zahl
G(x) := G(x) - const1;
        -- const2 ist eine beliebige konstante ganze Zahl
```

Zur Erinnerung: Die Ableitung berechnet den neuen Wert der virtuellen Variablen in Abhängigkeit vom alten Wert. Es können also nur für solche Definitionen freier Variablen Ableitungen angegeben werden, die den alten Wert berücksichtigen. Für den obigen applikativen Ausdruck ergeben sich die Vor- und Nachableitungen in Abbildung B.2, die alle profitabel differenzierbar sind. In der Abbildung wurde die Bezeichnung H anstelle der „100" verwendet. H soll eine beliebige ganze Zahl sein. Bei der Implementierung werden jedoch keine Ableitungen für die Zuweisung des leeren Tupels und der leeren Menge angegeben (siehe Abschnitt 3.3).

dx_i	$\partial^- V \langle dx_i \rangle$	$\partial^+ V \langle dx_i \rangle$
F := {};	V := {};	
F := [];	V := {};	

F with := x;	if $G(x) = H$ then
	    V with := x;
	end if;	

F less := x;	if $G(x) = H$ then
	    V less := x;
	end if;	

$G(x) := G(x) + const$;	if x in F then
	    if $G(x) = H$ then
	        V less := x;
	    elseif $G(x) = H$ - $const$ then
	        V with := x;
	    end if;
	end if;	

$G(x) := G(x)$ - $const$;	if x in F then
	    if $G(x) = H$ then
	        V less := x;
	    elseif $G(x) = H + const$ then
	        V with := x;
	    end if;
	end if;	

Abbildung B.2: Die Ableitungen für den applikativen Ausdruck in Abbildung B.1. H und *const* seien beliebige ganzzahlige Konstanten.

Schritt 2: Als nächstes muß für den applikativen Ausdruck die Kennung im C-Typ `CODE` in der Datei `mydtypes.h` eingefügt werden:

```
typedef enum { noset, A1, A2, Neues_Muster } CODE;
```

`A1` und `A2` seien die Kennung von bereits implementierten Mustern. Der applikative Ausdruck in Abbildung B.1 erhält die Kennung `Neues_Muster`. `noset` wird im Transformationsprogramm verwendet, wenn in einer Schleife kein differenzierbarer Ausdruck vorhanden ist, oder die Voraussetzungen zum Transformieren nicht erüllt sind (siehe Abschnitt 7.4). Der vollständige Aufzählungstyp ist im Abschnitt C.9.2.1 angegeben.

Schritt 3: Ebenfalls in der Datei `mydtypes.h` muß die C-Struktur `Dset` erweitert werden, falls für die Variablen des Ausdrucks in Abbildung B.1 keine passende Komponente vorhanden ist. Diese Struktur nimmt die freien und gebundenen Variablen und Konstanten des applikativen Ausdrucks auf. Die Variablen und Konstanten werden für die Ausgabe benötigt. Diese Struktur wird in Abbildung B.3 dargestellt.

In `newlsidename` und `newlength` wird der Name der virtuellen Variablen und die Länge des Namens gespeichert. In der Union `A` werden nun die Variablen, die speziell für ein Muster benötigt werden, gespeichert. In dieser Struktur sollten alle im Muster verwendeten Bezeichner, Zeichenketten und ganze und reelle Zahlen gespeichert werden. Bezeichner erhalten in der Struktur den Typ `tIdent`, Zeichenketten den Typ `tStringRef`, ganze Zahlen den C-Typ `int` und reelle Zahlen den C-Typ `float`. Für das Muster `Neues_Muster` werden drei Variablen vom Typ `tIdent` für j, F und G benötigt und eine Variable vom Typ `int` für die Konstante `100`. Die Typen `tIdent` und `tStringRef` sind in der *Cocktail*-Bibliothek (Abschnitt 6.6) definiert. Es wird angenommen, daß keine passende Struktur vorhanden ist,daß also die neue Struktur `Neues_Muster` in `A` eingefügt werden muß (Abbildung B.3).

Bei der Wahl der Namen der Komponenten sollte darauf geachtet werden, daß sie so gewählt werden, daß bei der Ausgabe der Ableitungen auch die richtigen Variablen verwendet werden. In den Typen wurden daher Namen wie `iterator`, `r1sidename` und `r2sidename` verwendet, die Auskunft über die Position im applikativen Ausdruck geben sollen (`r1sidename` = erste freie Variable im applikativen Ausdruck). Auch am Namen der Struktur sollte man erkennen können, zu welchem Muster sie gehört. Konnte ein Typ wiederverwendet werden, sollte bei diesem Typ im Kommentar angegeben werden, für welche Kennungen er verwendet wurde. Die vollständige Struktur `Dset` ist in Abschnitt C.9.2.1 angegeben.

Schritt 4: Wird die Struktur `Dset` erweitert, müssen für diese Erweiterung zwei C-Funktionen geschrieben werden, die Daten in eine Variable der Struktur `Dset` schreiben und Daten mit den Daten in einer Variablen dieser Struktur vergleichen. Bei diesen Funktionen handelt es sich um `get_Neues_Muster_variable` und

```
typedef struct { tIdent iterator;
                 tIdent r1sidename;
                 tIdent r2sidename;
               } tA1; /* A2 A4 */

typedef struct { tIdent iterator;
                 tIdent rsidename;
               } tA8;

typedef struct { tIdent iterator;
                 tIdent r1sidename;
                 tIdent r2sidename;
                 int integer;
               } tNeues_Muster;

typedef struct { tIdent newlsidename;
                 int newlength;
                 union {
                         tA1 a1; /* A2 A4 */
                         tA8 a8;
                         tNeues_Muster neu;
                       } A;
               } Dset;
```

Abbildung B.3: Die C-Datenstruktur `Dset` nach dem Einfügen der neuen Struktur `tNeues_Muster`.

`check_Neues_Muster_variable`. Die Verwendung der Funktionen wird im nächsten Punkt erläutert. Die Funktionen in den Abschnitten C.13 und C.14 können bei der Erstellung dieser Funktionen als Vorlage dienen.

Die Funktion `get_Neues_Muster_variable` in Abbildung B.4 erhält für jede Komponente der Union `A` einen Parameter. Für `newlsidename` und `newlength` werden keine Parameter benötigt. Alle Parameter erhalten immer den Typ `tTree` (Abschnitt 6.4.1). In der Funktion werden die lokalen Variablen `variable` vom Typ `DSET` (Zeiger auf `Dset`) und `newqual` vom Typ `tTree` definiert. Die ersten vier Anweisungen können ebenfalls aus den vorhandenen Funktionen `get*variable`[1] übernommen werden. Durch die erste Anweisung wird Speicherplatz für `variable` allokiert (Abschnitt C.13.12). Die zweite Anweisung liefert den eindeutigen Namen für die virtuelle Variable (Abschnitt C.10.1). Die nächsten beiden Anweisungen

[1] „*“ wird als Wildcard für Kennungen von vorhandenen Mustern verwendet

```
DSET get_Neues_Muster_variable(tTree iterator, tTree r1sidename,
                   tTree r2sidename, tTree integer)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable();
   variable->newlsidename =
        (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength =
        (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   variable->A.neu.iterator =
        (((((iterator->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.neu.r1sidename =
        (((((r1sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.neu.r2sidename =
        (((((r2sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.neu.integer = (integer->intlit).Integer;
   return variable;
}

extern DSET get_Neues_Muster_variable(tTree, tTree, tTree, tTree);
```

Abbildung B.4: Eine C-Funktion zum Füllen der Struktur `Dset`.

weisen `variable` den Namen der virtuellen Variablen und dessen Länge zu. Die nächsten Anweisungen beziehen sich speziell auf die neue Komponente der Struktur `Dset`. Für Bezeichner und ganze Zahlen erfolgt der Zugriff immer auf die in Abbildung B.4 beschriebene Art. Es muß nur der Parametername ausgetauscht werden. Für Zeichenketten und reelle Zahlen erfolgt der Zugriff folgendermaßen:

```
variable->A.*.zeichenkette = (zeichenkette->str).Str;
variable->A.*.reell = (reell->floatlit).Float;
```

wobei `zeichenkette` und `reell` die Parameter- bzw. Komponentennamen sein sollen. Diese Funktion muß in der Datei `mytrans.c` und der `extern`-Verweis in der Datei `mytrans.h` definiert werden.

Die Funktionen für Vergleiche befinden sich in der Datei `myset.c` bzw. `myset.h`. Diese Funktionen benötigen einen Parameter mehr als die obige Funktion. Dieser Parameter muß vom Typ `DSET` sein (Abbildung B.5). Die Funktionen bestehen aus einer `if`-Anweisung, in der die Parameter vom Typ `tTree` mit den (richti-

```
int check_Neues_Muster_variable(tTree iterator, tTree r1sidename,
                    tTree r2sidename, tTree integer, DSET variable)
{
   if((((((iterator->new2xQualId).xId)->xId).id)->id).Ident
         == variable->A.neu.iterator
      && (((((r1sidename->new2xQualId).xId)->xId).id)->id).Ident
         == variable->A.neu.r1sidename
      && (((((r2sidename->new2xQualId).xId)->xId).id)->id).Ident
         == variable->A.neu.r2sidename
      && (integer->intlit).Integer == variable->A.neu.integer)
      return 1;
   else return 0;
}

extern int check_Neues_Muster_variable(tTree, tTree, tTree,
                                       tTree, DSET);
```

Abbildung B.5: Eine C-Funktion zum Vergleichen von Parametern mit den Komponenten in der Struktur `Dset`.

gen) Komponenten von `variable` verglichen werden. Stimmen alle Komponenten überein, wird der Wert `1`, andernfalls der Wert `0` zurückgegeben.

Schritt 5: Nach diesen Vorbereitungen kann nun der applikative Ausdruck in die *puma*-Funktion `Pattern` in der Datei `trans.puma` eingefügt werden. Diese Funktion arbeitet auf der abstrakten Grammatik. Zunächst wird also die Grammatikstruktur des Musters benötigt. Diesem Zweck dient die Option „`-tm`“ im Transformationsprogramm. Der Aufruf „`pstt -tm` *file*“ gibt für das Progamm in der Datei *file* alle Knoten und Blätter des abstrakten Syntaxbaumes in einem Preorder-Durchlauf aus. Da dies durch den Attributauswerter `Muster` (Abschnitt C.41) erfolgt, wird in der Datei *file* ein vollständiges ProSet-Programm erwartet. Das folgende Programm bietet sich dazu an:

```
program neu;
begin
   E := {j : j in F | (G(j) = 100)};
end neu;
```

Bei diesem Programm wird rechts vom Gleichheitszeichen eine ganzzahlige Konstante benötigt. Die Verwendung der Variablen H anstelle der Konstanten würde einen Bezeichner im abstrakten Syntaxbaum erzeugen. Die Ausgabe dieses Aufrufs ist in Abbildung B.6 angegeben. Das applikative Muster wird durch den Teil, der zwischen den Kommentaren `/* von hier */ /* bis hier */` liegt, dargestellt.

Wird für einen weiteren applikativen Ausdruck der gleiche Programmaufbau verwendet, beginnt der applikative Ausdruck mit dem ersten Vorkommen eines Subtypen von `xExpr` (hier `new36xExpr`) und endet vor `new2xExplAsso`.

Für ein vollständiges Pattern werden auch die bei den verwendeten Knotentypen definierten Attribute benötigt. Um die Attribute und ihre Positionen im abstrakten Syntaxbaum zu erhalten, gibt es bei *puma* die Option `-r`. Diese Option gibt alle Knotentypen im abstrakten Syntaxbaum mit ihren Attributen und Nachfolgern aus. Der Aufruf lautet „`puma -r trans.puma`“. In Abbildung B.7 ist ein Teil dieser Ausgabe aufgeführt. Die Attribute werden für das Pattern nicht benötigt. Es ist also ausreichend, ihre Position durch ein „_“ zu markieren.

Die Funktion `Pattern` ersetzt den applikativen Ausdruck durch die virtuelle Variable. Es wird also eine Marke („label:“) für `xExpr` benötigt (siehe Abbildung B.8). Weiterhin benötigt die Funktion Marken, um auf die gebundenen und die freien Variablen und die Konstanten des Ausdrucks zugreifen zu können. *Cocktail* gibt eine Fehlermeldung aus, wenn ein Bezeichner eine Marke erhält, daher erhalten die Knotentypen `new2xQualId` die Marken. Bei `Pattern` handelt es sich um eine Funktion mit vier Parametern, wobei die letzten drei Parameter für die Erkennung des Musters nicht erforderlich sind. Für eine vollständige Regel werden also noch „_“ für die drei letzten Parameter und der Rückgabewert benötigt. Das vollständige Pattern inklusive des Rückgabewertes ist in Abbildung B.8 auf Seite 134 angegeben.

Für diese Regel fehlt noch der Anweisungsteil. Dieser Teil besteht aus einer `switch`-Anweisung über den Parameter `funcflag`. Im Abschnitt 7.6 wurden die einzelnen Komponenten des Typs und die in den einzelnen `case`-Zweigen durchgeführten Operationen bereits erläutert. Hier wird nur noch der Aufbau der Anweisung beschrieben. Die Anweisung ist in Abbildung B.9 auf Seite 135 angegeben. Auffällig ist bei der `switch`-Anweisung, daß bei den `case`-Zweigen anstelle des „`:`“ ein „`\:`“ vorkommt. Dies liegt daran, daß bei *puma* das „`:`“ bereits bei Marken verwendet wird.

code Im applikativen Ausdruck kommt dreimal ein `j` vor. Damit das neue Pattern auch wirklich den erwarteten applikativen Ausdruck darstellt, muß überprüft werden, ob die entsprechenden Bezeichner übereinstimmen. Ist dies der Fall, können die Variablen und die Konstante in `variable` gespeichert werden und `transatt` kann die Kennung des applikativen Ausdrucks zugewiesen werden. Als letztes wird der Rückgabewert auf `1` gesetzt, damit bei den Kettenmustern erkennbar ist, daß ein Muster gefunden wurde.

helpstmt In diesem `case`-Zweig werden die Variablen von applikativen Ausdrücken angegeben, für die das Pattern nicht definiert ist. Wird eine dieser Variablen in diesem Pattern auf der linken Seite verwendet, wird `transatt` auf `noset` gesetzt. Dieser `case`-Zweig kann bei allen Transformationsmustern von applikativen Ausdrücken übernommen werden.

delete Stimmt dieses Muster mit dem zu differenzierenden Ausdruck überein, gilt also „`transatt == Neues_Muster`", müssen die Bezeichner dieses Musters mit den gespeicherten Variablen des Ausdrucks verglichen werden, um sicherzustellen, daß dieser Ausdruck auch wirklich der zu transformierende Ausdruck ist, und nicht nur den gleichen Aufbau hat. Handelt es sich um den korrekten Ausdruck, wird der applikative Ausdruck im Baum durch die virtuelle Variable ersetzt. Der in Abbildung B.8 angegebene Ausdruck kann für alle Transformationsmuster applikativer Ausdrücke verwendet werden, die einen Bezeichner als virtuelle Variable haben. Für applikative Ausdrücke, deren Variable die Struktur `V(j)` haben, befindet sich auf Seite 195 der neue Ausdruck. Für andere Typen von Variablen sollte der Ausdruck mit Hilfe der Option „`-tm`" erzeugbar sein.

default In diesem `case`-Zweig wird nur der Wert `0` zurückgegeben.

Schritt 6: Für die Definitionen müssen die Ableitungsfunktionen in die Datei `mytrans.c` und die entsprechenden `extern`-Verweise in die Datei `mytrans.h` geschrieben werden. In Abbildung B.10 auf Seite 136 ist die Ableitungsfunktion für die Definition „`F with:= x`" angegeben. Die Ableitungsfunktionen benötigen `variable` und alle benutzerdefinierten Token der Definition, die nicht in `variable` vorkommen, als Parameter. Für die Bezeichner wird der Typ `tTree` verwendet. Die erste Anweisung in der Ableitungsfunktion ist optional. Die letzte Anweisung ermöglicht die Ausgabe von line-Direktiven. Die Ableitung wird nun durch `fprintf`-Anweisungen in die Ausgabedatei geschrieben. Ganze und reelle Zahlen können mit den entsprechenden Formatparametern in der `fprintf`-Anweisung ausgegeben werden. Für Bezeichner und Zeichenketten gibt es die Bibliotheks-Funktionen `WriteIdent` und `WriteString` und für die Variablen vom Typ `tTree`, die in diesen Fällen den Typ `new2xQualId` haben, gibt es die Funktion `WriteQualId`, die in Abschnitt C.12.3 definiert ist.

Schritt 7: Für diesen applikativen Ausdruck müssen nun noch die Modifikationsmuster, also die Muster für die Modifikationen freier Variablen apllikativer Ausdrücke, implementiert werden. Hier kommen drei verschiedene Varianten vor.

1. Gibt es ein Modifikationsmuster für diesen applikativen Ausdruck, das noch nicht implementiert ist, erhält man die Grammatikstruktur, wie es im ersten Schritt beschrieben wurde. Die Grammatikstruktur beginnt in diesem Fall mit einem Subtyp von `xStmt` und endet vor `new2xExplAsso`. Alle benutzerdefinierten Token erhalten dabei eine Marke. Für Bezeichner wird diese Marke an den Knotentyp `new2xQualId` gehängt (siehe Abbildung B.11 auf Seite 137).

Für Modifikationsmuster werden nur die `case`-Zweige `helpstmt`, `preder` und `postder` benötigt. Im `case`-Zweig `helpstmt` muß zunächst für alle bereits definierten Kennungen die `if`-Abfrage

```
if(transatt == Altes_Muster)
{
   transatt = noset;
}
```

eingefügt werden, da dieses Modifikationsmuster für die bereits implementierten Transformationsmuster nicht definiert ist. Ist dieses Modifikationsmuster nicht für alle freien Variablen eines Transformationsmusters definiert, wird auch dafür eine `if`-Abfrage eingebaut:

```
else if(transatt == Neues_Muster
        && Ident == variable->A.neu.r1sidename)
{
   transatt = noset;
}
```

In `variable`“ sind alle freien Variablen des Transformationsmusters gespeichert. Am Ende dieses `case`-Zweiges wird der Rückgabewert der Funktion `Pattern` auf `0` gesetzt und ein „`break`“ eingefügt (Abbildung B.11).

Im `case`-Zweig `preder` werden line-Direktiven ausgegeben und die Ableitungsfunktionen aufgerufen. `result` wird auf `1` gesetzt, wenn für diese Definition keine Nachableitung existiert, andernfalls auf `2`.

```
if(transatt == Neues_Muster)
{
   if(Ident == variable->A.neu.r2sidename)
   {
      lineprint();fprintf(ofile,"\n");
      insert_Neues_Muster_plus(int1, int2, variable);
      result = 1;
   }
}
```

Die Abfrage „`if(transatt == Neues_Muster)`“ wird benötigt, da bei späteren Erweiterungen das gleiche Modifikationsmuster auch für andere Transformationsmuster definiert sein kann. Dieser `case`-Zweig wird mit

```
else
{
   result = 0;
   break;
}
```

beendet. Im `case`-Zweig `postder` wird die Nachableitung ausgegeben:

```
if (transatt == A4)
{
   insertA4ypostdef(qual2, variable);
   result = 0;
}
```

Abschließend befindet sich in dieser `switch`-Anweisung

```
default \: result = 0;
```

2. Bei Modifikationsmustern, die nicht für dieses Transformationsmuster definiert sind, muß die `if`-Anweisung im `case`-Zweig `helpstmt` um den Zweig

   ```
   else if(transatt == Neues_Muster)
   {
      transatt = noset;
   }
   ```

   erweitert werden. Ist ein Modifikationsmuster nur für eine bestimmte Variable nicht definiert, kann der Zweig auch die folgende Struktur haben.

   ```
   else if(transatt == Neues_Muster
           && Ident == variable->A.neu.r2sidename)
   {
      transatt = noset;
   }
   ```

3. Ein implementiertes Modifikationsmuster ist für diesen applikativen Ausdruck definiert. In diesem Fall wird die `if`-Anweisung im `case`-Zweig `preder` erweitert.

   ```
   else if(transatt == Neues_Muster)
   {
      if(Ident == variable->A.neu.r2sidename)
      {
         lineprint();fprintf(ofile,"\n");
         insert_Neues_Muster_plus(int1, int2, variable);
         result = 1;
      }
   }
   ```

   Das zu dieser Anweisung gehörende Modifikationsmuster ist für die freie Variable `variable->A.neu.r1sidename` definiert. Die Anweisungen

   ```
   lineprint(); fprintf(ofile,"\n");
   ```

   fügen in die Ausgabedatei eine line-Direktive ein, und in der Funktion

   `insert_Neues_Muster_plus`

ist die Ableitung für diese Modifikation und diese Variable definiert.

Beispiele zu diesem Punkt befinden sich ab Seite 231.

Schritt 8: Für den Attributauswerter `Transhelp2` muß die C-Funktion `testident` erweitert werden. Diese Funktion überprüft, ob eine Variable unter den freien Variablen des zu transformierenden Ausdrucks vorkommt. Diese Funktion wird um einen `case`-Zweig erweitert, die den gegebenen Bezeichner mit den freien Variablen des Ausdrucks vergleicht, und bei einer Übereinstimmung `rside` zurückgibt. Andernfalls wird `noside` zurückgegeben. `rside` und `noside` sind Werte im Aufzählunstyp `ISET`. Die Funktion `testident` befindet sich im Abschnitt C.17.1 auf Seite 368.

Schritt 9: Es fehlt noch die Initialisierung der virtuellen Variablen. In der Funktion `inittrans` wird der `case`-Zweig

```
case Neues_Muster: init_Neues_Muster(variable); break;
```

eingefügt. Die Initialisierungsfunktion „`init_Neues_Muster`“ wird auf die gleiche Weise erzeugt, wie die Ableitungsfunktionen, nur hier reicht `variable` als Parameter aus. Die Initialisierungsfunktionen befinden sich ebenfalls in der Datei `mytrans.c`. In Abbildung B.12 ist die Initialisierungsfunktion für den applikativen Ausdruck in Abbildung B.1 angegeben.

Das hier eingefügte Muster entspricht dem Muster `A3` aus Abschnitt A.3.

```
xInitChain(xProgDefn(xId(id(neu)),
  xProgBody(
   new2xDecls,
   new3xStmts(
    new2xStmts(new28xStmt(new5xLValue(new2xQualId(xId(id(E)))),

/* von hier */
      new36xExpr(new3xFormer(new38xExpr(new2xQualId(Id(id(j)))),
        new1xIterator(
         new2xSimpleIts(
          new1xSimpleIt(new5xLValue(new2xQualId(xId(id(F)))),
           new50xExpr(
            new44xExpr(new38xExpr(new2xQualId(xId(id(G)))),
             new1xSelector(
              new2xExprList(
               new38xExpr(new2xQualId(xId(id(j))))))),
            new5xBinOp,
            new13xExpr(intlit(100))))),
         new50xExpr(
          new44xExpr(new38xExpr(new2xQualId(xId(id(G)))),
           new1xSelector(
            new2xExprList(
             new38xExpr(new2xQualId(xId(id(j))))))),
          new5xBinOp,
          new13xExpr(intlit(100))))))),
/* bis hier */

   new2xExplAsso)),
 new5xPHMDefn),
xId(id(neu))))
```

Abbildung B.6: Die Ausgabe des Aufrufs „`pstt -tm` *file*“.

```
xId(id:id,helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut)
new36xExpr(helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut,pos1,xFormer:xFormer,pos2)
new38xExpr(helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut,xQualId:xQualId)
new44xExpr(helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut,xExpr:xExpr,xSelector:xSelector)
new50xExpr(helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut,Expr1:xExpr,xBinOp:xBinOp,Expr2:xExpr)
new1xIterator(helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut,xSimpleIts:xSimpleIts,pos,xExpr:xExpr)
new2xSimpleIts(helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut,xSimpleIt:xSimpleIt)
new1xSimpleIt(helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut,xLValue:xLValue,pos,xExpr:xExpr)
new5xLValue(helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut,xQualId:xQualId)
new2xQualId(helpIn,helpOut,help2In,help2Out,writeIn,writeOut,
    musterIn,musterOut,xId:xId)
id(Pos,Ident,Length,helpIn,helpOut,help2In,help2Out,writeIn,
    writeOut,musterIn,musterOut)
intlit(Pos,Integer,Length,helpIn,helpOut,help2In,help2Out,
    writeIn,writeOut,musterIn,musterOut)
```

Abbildung B.7: Ein Teil der Ausgabe des Aufrufs „`puma -r trans.puma`“.

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        new44xExpr(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
          new1xSelector(_,_,_,_,_,_,_,_,_,
            new2xExprList(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)),_)),
        new5xBinOp,
       new13xExpr(_,_,_,_,_,_,_,_,intlit:intlit)))),_),_,_,_
RETURN result;
```

Abbildung B.8: Das vollständige Pattern des applikativen Ausdrucks in Abbildung B.1. Dieses Pattern wird in der *puma*-Funktion `Pattern` verwendet.

```
{ switch(funcflag)
  {
     case code \:
          if(EqualQualId(qual1, qual2) == 1
             && EqualQualId(qual1, qual5) == 1)
          {
             variable = get_Neues_Muster_variable(qual1,
                                  qual3, qual4, intlit);
             transatt = Neues_Muster;
             result = 1;
          }
          else
          {
             variable = NULL;
             transatt = noset;
          }
          break;
     case helpstmt \: transatt = noset;
          result = 0;
          break;
    case delete \:
         if(transatt == Neues_Muster)
         {
            result = check_Neues_Muster_variable(qual1, qual3,
                                    qual4, intlit, variable);
            if(result == 1)
            {
               expr = mnew38xExpr(mnew2xQualId(mxId(
                           mid(NoPosition,
                           variable->newlsidename,
                           variable->newlength))));
            }
         }
         else result = 0;
         break;
     default \: result = 0;
  }};.
```

Abbildung B.9: Die Anweisungen zum Pattern in Abbildung B.8.

```
void insert_Neues_Muster_plus(tTree const1, tTree const2,
                              DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (%d /= 0) then\n   if ( %d in ",
           (const2->intlit).Integer, (const1->intlit).Integer);
   WriteIdent(ofile, variable->A.neu.r1sidename);
   fprintf(ofile, " ) then\n   if ( ");
   WriteIdent(ofile, variable->A.neu.r2sidename);
   fprintf(ofile, "(%d) = %d ) then\n     ",
           (const1->intlit).Integer, variable->A.neu.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := %d;\n elseif ( ",
           (const1->intlit).Integer);
   WriteIdent(ofile, variable->A.neu.r2sidename);
   fprintf(ofile, "(%d) = %d - %d) then\n    ",
           (const1->intlit).Integer, variable->A.neu.integer,
           (const2->intlit).Integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := %d ;\nend if; \nend if;",
           (const1->intlit).Integer);
   linedirflag = 1;
}
extern void insert_Neues_Muster_plus(tTree const1, tTree const2,
                                                     DSET variable);
```

Abbildung B.10: Eine Ableitungsfunktion in C-Code.

```
new29xStmt(_,_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,
      qual1:new2xQualId(_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new13xExpr(_,_,_,_,_,_,_,_,int1:intlit)),_)),
  new31xBinOp,_,
  new13xExpr(_,_,_,_,_,_,_,_,int2:intlit)),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if(transatt == A1)
               transatt = noset;
            else if(transatt == A2)
               transatt = noset;
            else if(transatt == A3
                    && Ident == variable->A.a3.r1sidename)
               transatt = noset;
            result = 0;
            break;
       case preder \:
            if(transatt == A3)
              if(Ident == variable->A.a3.r2sidename)
              {
                 lineprint();fprintf(ofile,"\n");
                 insertA3plus(int1, int2, variable);
                 result = 1;
              }
              else
                 result = 0;
            else
              result = 0;
            break;
        default \: result = 0;
    }
  };.
```

Abbildung B.11: Ein Modifikationsmuster. Ein Teil der *puma*-Funktion `Pattern`.

```
void init_Neues_Muster(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := { ");
   WriteIdent(ofile, (variable->A.neu).iterator);
   fprintf(ofile, " : ");
   WriteIdent(ofile, (variable->A.neu).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.neu).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.neu).r2sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.neu).iterator);
   fprintf(ofile, ") = %d};",(variable->A.neu).integer);
}

extern void init_Neues_Muster(DSET);
```

Abbildung B.12: Eine Initialisierungsfunktion in C-Code.

C. Literaler Quellcode

Die Implementierung und die Dokumentation ist in einer **noweb**-Datei [Ram92] beschrieben. **noweb** ist ein literate programming tool, das es ermöglicht Quellcode und beschreibenden Text in einem Dokument zu verschachteln. Die Idee ist, Codestücke in einer Reihenfolge anordnen zu können, die die Erklärung des Programms erleichtert. Eine **noweb**-Datei ist eine Folge von *Chunks*. Ein Chunk kann Code oder Dokumentation enthalten. Ein Code-Chunk beginnt mit

$\langle$*Chunk Name*$\rangle \equiv$

in einer eigenen Zeile. Chunks für die Dokumentation sind anonym. Ein Chunk endet mit dem Beginn eines neuen Chunks oder dem Ende der Datei. Mehrere Code-Chunks können denselben Namen haben. Diese werden zusammengefügt und erzeugen einen einzelnen Code-Chunk. Ein Code-Chunk kann in einen anderen Code-Chunk eingefügt werden, indem das $\equiv$ hinter $\langle$*Chunk Name*$\rangle$ weggelassen wird. das Programm `notangle` extrahiert den Quellcode aus dem Dokument und das Programm `noweave` erzeugt eine LaTeX Datei mit Quellcode und Dokumentation.

Das extrahieren des Quellcodes erfolgt auf die folgende Weise:

- `notangle -RMakefile -t8 pstt.nw | cpif Makefile`
- `make`

C.1 Das Hauptprogramm

Im Hauptprogramm werden die gesetzten Optionen abgefragt, die Eingabe- und Ausgabedatei geöffnet und der Übersetzer gestartet.

$\langle$*pstt.c*$\rangle \equiv$

```
#include "Parser.h"
#include "Tree.h"
#include "EnvAtt.h"
#include "PrepAtt.h"
```

```
#include "Output.h"
#include "Muster.h"
#include "transform.h"
#include <string.h>
#include <stdlib.h>
```

In `linedir` wird gespeichert, ob line-Direktiven erzeugt werden sollen (`linedir = 0`) oder nicht (`= 1`). `anzahl` und `TRANSFLAG` werden für die Anzahl der Transformationsdurchläufe benötigt. Wurde die Option `-number` (`-n`) nicht gesetzt, wird das Transformationsprogramm solange durchlaufen, wie im vorhergehenden Schritt eine Transformation durchgeführt wurde, höchstens jedoch `anzahl` (= 999) Schritte. Ob eine Transformation durchgeführt wurde, ist an `TRANSFLAG` erkennbar. `TRANSFLAG` wird in der Funktion `inittrans` der Wert `1` zugewiesen (an dieser Stelle steht fest, daß eine Transformation durchgeführt wurde), welcher am Ende eines Transformationsdurchlaufs im Hauptprogramm abgefragt wird. `linedirflag` wird verwendet, um in der Funktion `putmybetween` entscheiden zu können, ob line-Direktiven ausgegeben werden sollen. `*ifile` wird für die line-Direktiven benötigt. `*ofile` wird bei der Funktion `fopen`, `fclose`, `WriteIdent`, `WriteQualId` und `fprintf` verwendet, um in die Ausgabedatei zu schreiben.

⟨*pstt.c*⟩+≡
```
int linedir = 0;
int linedirflag = 0;
int TRANSFLAG = 0;
char *ifile;
FILE *ofile;

main(int argc, char *argv[])
{
   int i = 1, /* Laufvariable */
      anzahl = 999,
      fileflag = 0, /* wurde ein Dateiname angegeben? */
      musterflag = 0, /* Soll nur der "AST" ausgegeben werden? */
      numberflag = 0, /* wieviel Transformationsschritte waren noetig */
      n = 0; /* wird fuer extension benoetigt, und gibt an, die wievielte
                temporaere Datei erzeugt wird */
   char *helpname, *newstr, /* werden benoetigt, um herauszufinden, ob ein
                               Basename uebergeben wurde */
      *helpfile, /* Hilft die (temporaeren) Ausgabedateien zusammenzubasteln.
                    Beinhaltet den Namen der endgueltigen Ausgabedatei. */
      *inputfile, /* Die gerade aktuelle Eingabedatei */
      *outputfile,  /* Die gerade aktuelle Ausgabedatei */
      *extension, /* hier wird n in einen String umgewandelt, und an den
                     Dateinamen gehaengt */
      *str; /* benoetigen wir im system-Befehl fuer die Ausgabedatei */
   size_t length; /* Die Laenge des Dateinamens */
```

Nun werden die Optionen eingelesen.

Hinter der Option `-number` bzw. `-n` wird die (maximale) Anzahl der Transformationsdurchläufe angegeben.

⟨*pstt.c*⟩+≡
```
while(i < argc)
{
   if(argv[i][0] == '-')
   {
      if(strcmp(argv[i], "-number") == 0 ||
         strcmp(argv[i], "-n") == 0)
      {
         i++;
         numberflag = 1;
         anzahl = atoi(argv[i]);  /* MAXIMALE Anzahl der Durchlaeufe */
         if(anzahl > 999)
         {
            ErrorMessageI(0, 1, NoPosition, 7,
                          "Maximal 999 Transformationsteps are allowed");
         }
      }
```

Bei der Option `-NL` werden keine line-Direktiven ausgegeben.

⟨*pstt.c*⟩+≡
```
      else if(strcmp(argv[i], "-NL") == 0)
      {
         linedir = 1;  /* es werden keine line-Direktiven ausgegeben */
      }
```

Bei der folgenden Option werden die linken Seiten der Grammatikregeln des erzeugten AST in Preorder-Durchlauf ausgegeben.

Die Option `-tm` ist für den Fall gedacht, daß neue Transformationsmuster in das Programm eingebaut werden sollen.

⟨*pstt.c*⟩+≡
```
      else if(strcmp(argv[i], "-tm") == 0)
      {
         musterflag = 1; /*  es wird nur der Preorder-Durchlauf durch den
                             AST ausgegeben  */
      }
      else
      {
         ErrorMessageI(0, 1, NoPosition, 7, "wrong option");
      }
   }
   else
   {
```

Hier wird nun der Dateiname eingelesen. Enthält der Dateiname eine Endung, wird für die Ausgabedatei die Endung durch ".tps" ersetzt. Andernfalls wird ".tps" hinter den Dateinamen gehängt. Der eingelesene Dateiname darf natürlich keine Endung haben, die mit ".tps" beginnt.

$\langle$*pstt.c*$\rangle$+$\equiv$
```
            if(fileflag == 1)
            {
               ErrorMessageI(0, 1, NoPosition, 7, "only one inputfile expected");
            }
            fileflag = 1;
            if(strncmp(strrchr(argv[i], '.'), ".tps", 4) == 0)
            {
               Message("it is not allowed to use an .tps file", 2, NoPosition);
               exit(-1);
            }
            if((ifile = (char *)malloc(strlen(argv[i]) + 1)) == NULL)
            {
               Message("not enough memory", 2, NoPosition);
               exit(-1);
            }
            ifile = strcpy(ifile, argv[i]);
            if((inputfile = (char *)malloc(strlen(argv[i]) + 1)) == NULL)
            {
               Message("not enough memory", 2, NoPosition);
               exit(-1);
            }
            inputfile = strcpy(inputfile, argv[i]);
```

Ist das Einlesen der Optionen beendet, ohne das eine Eingabedatei angegeben wurde, wird eine Fehlermeldung ausgegeben.

$\langle$*pstt.c*$\rangle$+$\equiv$
```
         }
         i++;
      }
      if(fileflag == 0)
      {
         ErrorMessageI(0, 1, NoPosition, 7, "filename expected");
      }
```

Jetzt werden noch `helpfile` und `extension` bearbeitet, die für `outputfile` benötigt werden. In `helpfile` wird `filename.tps` gespeichert und in `extension` wird die Nummer der Iteration als Zeichenkette gespeichert.

⟨*pstt.c*⟩+≡
```
length = strlen(inputfile) - strlen(strrchr(inputfile, '.'));
if((helpfile = (char *)malloc(length + 5)) == NULL)
{
   Message("not enough memory", 2, NoPosition);
   exit(-1);
}
helpfile = strncpy(helpfile, inputfile, length);
helpfile = strcat(helpfile, ".tps");
if((outputfile = (char *)malloc(length + 8)) == NULL)
{
   Message("not enough memory", 2, NoPosition);
   exit(-1);
}
if((extension = (char *)malloc(4)) == NULL)
{
   Message("not enough memory", 2, NoPosition);
   exit(-1);
}
```

Hier beginnt nun die eigentliche Arbeit des Transformationsprogramms. Die `while`-Schleife wird benötigt, da bei mehrfachen Durchläufen des Transformationsprogramms jede Phase des Übersetzers erneut durchlaufen werden muß (es muß jedesmal ein neuer AST aufgebaut werden).

`outputfile` besteht aus `helpfile` und `extension`.

⟨*pstt.c*⟩+≡
```
while(anzahl > 0)
{
   n++;
   sprintf(extension, "%d", n);
   outputfile = strcpy(outputfile, helpfile);
   outputfile = strcat(outputfile, extension);
```

Nun kann die (temporäre) Ausgabedatei geöffnet werden.

⟨*pstt.c*⟩+≡
```
   if((ofile = fopen(outputfile, "w")) == NULL)
   {
      ErrorMessageI(0, 1, NoPosition, 7, "cannot open outputfile");
   }
```

Hier beginnt der Übersetzer. Die Eingabedatei wird geöffnet und der `Parser` gestartet. Die Funktion `BeginFile` wird vom `Scanner` zur Verfügung gestellt.

Wurde die Option `-tm` angegeben, also `musterflag == 1`, wird der Attributauswerter `Muster` (C.41) aufgerufen und die `while`-Schleife verlassen, da weitere Durchläufe aufgrund der Ausgabe von `Muster` nicht möglich und nicht sinnvoll sind.

⟨*pstt.c*⟩+≡
```
        BeginFile(inputfile);
        Parser();
        if (musterflag == 1)
        {
           Muster(TreeRoot);
        }
```

Andernfalls wird die Prozedurnamentabelle erzeugt (`EnvAtt` (C.6) und `mygettree` (C.7)), transformiert (`PrepAtt`(C.8)) und die Ausgabe erzeugt (`Output`(C.19)).

⟨*pstt.c*⟩+≡
```
        else
        {
           EnvAtt(TreeRoot);
           mygettree(TreeRoot);
           PrepAtt(TreeRoot);
           TransAtt(TreeRoot);
           fprintf(stderr, "%d.pst-Durchlauf schreibt in %s\n", n, outputfile);
           Output(TreeRoot);
        }
```

Nun wird noch die Eingabedatei geschlossen (`fclose`), `inputfile` auf die neue Eingabedatei gesetzt, `anzahl` dekrementiert und vor Beendigung des Transformationsprogramms ein paar Kommentare auf `stderr` geschrieben. Bei `anzahl == 0` wird kein Kommentar ausgegeben, da in diesem Fall nicht sicher ist, ob wirklich alle Transformationsmöglichkeiten ausgeschöpft wurden. Das Programm wurde in diesem Fall beendet, da der Benutzer die Anzahl der Transformationsdurchläufe festgelegt hat oder bereits 999 Durchläufe stattgefunden haben, und nicht, weil keine Transformationsmöglichkeiten mehr vorhanden waren. Der Kommentar wird ebenfalls nicht ausgegeben, wenn die Option `-tm` gesetzt war. Die letzte temporäre Datei wird dann noch nach `''$\ast$''.tps` kopiert.

Jetzt ist ein Transformationsdurchlauf beendet und `TRANSFLAG` wird wieder auf `0` gesetzt.

⟨*pstt.c*⟩+≡

```
        fclose(ofile);
        inputfile = strcpy(inputfile, outputfile);
        anzahl--;
        if(TRANSFLAG == 0 || anzahl == 0)
        {
           fprintf(stderr, "kopiere %s nach %s\n", outputfile, helpfile);
            if(anzahl != 0 && musterflag == 0)
            {
              if(numberflag == 1)
              {
                 fprintf(stderr,
                         "Es werden %d Transformationsschritte benoetigt\n", n);
              }
              else
              {
                 fprintf(stderr,
                         "Es werden %d Transformationsschritte benoetigt\n",n-1);
              }
           }
           if((str = (char *)malloc(strlen(outputfile)+strlen(helpfile)+10))
                   == NULL)
           {
              Message("not enough memory", 2, NoPosition);
              exit(-1);
           }
           strcpy(str, "cp ");
           strcat(str, outputfile);
           strcat(str, " ");
           strcat(str, helpfile);
           system(str);
           break;
        }
        else
        {
```

```
            TRANSFLAG = 0;
         }
      }
```

Nun kann das Programm korrekt verlassen werden. Die `if`-Abfrage für den Rückgabewert wird für die Benutzungsoberfläche von ProSet benötigt.

$\langle pstt.c \rangle {+}{\equiv}$

```
      if(n <= 200)
      {
         exit(n);
      }
      else
      {
         exit(201);
      }
   }
```

C.2 Das Makefile

⟨*Makefile*⟩≡

```
HOME    = /usr/local/ls10
LIB     = $(HOME)/lib
INCDIR  = $(LIB)/include
REUSE   = reuse
CFLAGS  = -I$(INCDIR)
CC      = gcc -g

MYFILE  = pstt.nw

NOWFILES = trans.scan trans.pars trans.cg myprint.c myprint.h \
        pstt.c trans.puma debugenv.h debugenv.c myscope.c myscope.h \
        mydtypes.h global.h myset.h myset.c mytrans.c mytrans.h

SOURCES = Scanner.h Scanner.c Parser.h Parser.c Tree.h Tree.c \
        PrepAtt.h PrepAtt.c TransAtt.h TransAtt.c \
        Output.h Output.c transform.h transform.c EnvAtt.h EnvAtt.c Muster.h \
        Muster.c Transhelp1.c  Transhelp1.h Transhelp2.h Transhelp2.c

OBJS    = Scanner.o Parser.o Tree.o myprint.o transform.o \
        PrepAtt.o TransAtt.o Output.o myscope.o EnvAtt.o debugenv.o \
        Muster.o myset.o Transhelp1.o Transhelp2.o mytrans.o

pstt:   $(NOWFILES) $(OBJS) pstt.c
        $(CC) -ansi -pedantic $(OBJS) pstt.c -o pstt -I $(REUSE) \
        -L$(REUSE) -lreuse

Makefile:       pstt.nw
        notangle -RMakefile -t8 pstt.nw | cpif Makefile

trans.scan:     pstt.nw
        notangle -Rtrans.scan pstt.nw | cpif trans.scan

trans.pars:     pstt.nw
        notangle -Rtrans.pars pstt.nw | cpif trans.pars

trans.cg:       pstt.nw
        notangle -Rtrans.cg pstt.nw | cpif trans.cg

myprint.c:      pstt.nw
        notangle -Rmyprint.c pstt.nw | cpif myprint.c

myprint.h:      pstt.nw
        notangle -Rmyprint.h pstt.nw | cpif myprint.h
```

```
pstt.c: pstt.nw
	notangle -Rpstt.c pstt.nw | cpif pstt.c

trans.puma:	pstt.nw
	notangle -Rtrans.puma pstt.nw | cpif trans.puma

debugenv.h:	pstt.nw
	notangle -Rdebugenv.h pstt.nw | cpif debugenv.h

debugenv.c:	pstt.nw
	notangle -Rdebugenv.c pstt.nw | cpif debugenv.c

myscope.c:	pstt.nw
	notangle -Rmyscope.c pstt.nw | cpif myscope.c

myscope.h:	pstt.nw
	notangle -Rmyscope.h pstt.nw | cpif myscope.h

mytrans.c:	pstt.nw
	notangle -Rmytrans.c pstt.nw | cpif mytrans.c

mytrans.h:	pstt.nw
	notangle -Rmytrans.h pstt.nw | cpif mytrans.h

myset.c:	pstt.nw
	notangle -Rmyset.c pstt.nw | cpif myset.c

myset.h:	pstt.nw
	notangle -Rmyset.h pstt.nw | cpif myset.h

mydtypes.h:	pstt.nw
	notangle -Rmydtypes.h pstt.nw | cpif mydtypes.h

global.h:	pstt.nw
	notangle -Rglobal.h pstt.nw | cpif global.h

Scanner.rpp Parser.lalr:	trans.pars
	cg -cxzj trans.pars;

trans.rex:	trans.scan Scanner.rpp
	rpp < trans.scan > trans.rex;

Scanner.h Scanner.c:	trans.rex global.h
	rex -cd trans.rex;

Parser.h Parser.c:	Parser.lalr
	lalr -c -d Parser.lalr;
```

```
transform.h transform.c:        Tree.TS trans.puma mytrans.h
        puma -cdipmk trans.puma

Tree.c Tree.h:  trans.cg mydtypes.h
        ast -cdim trans.cg

Tree.TS :       trans.cg Tree.h
        ast -c4 trans.cg

EnvAtt.h EnvAtt.c:      trans.cg trans.puma myscope.c mydtypes.h myscope.h
        echo EVAL EnvAtt SELECT AbstractSyntax EnvOutAtt \
        PROPERTY OUTPUT FOR EnvOutAtt | cat - trans.cg |ag -cDIW

PrepAtt.h PrepAtt.c:  trans.cg
        echo EVAL PrepAtt SELECT AbstractSyntax EnvOutAtt PrepOutAtt \
        PrepAtt \
        PROPERTY INPUT FOR EnvOutAtt \
        PROPERTY OUTPUT FOR PrepOutAtt | cat - trans.cg | ag -cDIW

TransAtt.h TransAtt.c:  trans.cg
        echo EVAL TransAtt SELECT AbstractSyntax TransOutAtt TransAtt \
        PROPERTY OUTPUT FOR TransOutAtt | cat - trans.cg |ag -cDIW

Transhelp1.h Transhelp1.c:      trans.cg
        echo EVAL Transhelp1 SELECT AbstractSyntax PrepOutAtt Transhelp1Int \
        TransOutAtt Transhelp1 | cat - trans.cg | ag -cDIW

Transhelp2.h Transhelp2.c:      trans.cg
        echo EVAL Transhelp2 SELECT AbstractSyntax PrepOutAtt TranshelpInt2 \
        TransOutAtt Transhelp2 | cat - trans.cg | ag -cDIW

Output.h Output.c:      trans.cg
        echo EVAL Output SELECT AbstractSyntax TransOutAtt OutputInt Output \
        PROPERTY INPUT FOR TransOutAtt | cat - trans.cg | ag -cDIW

Muster.h Muster.c:      trans.cg
        echo EVAL Muster SELECT AbstractSyntax Muster MusterInt \
        | cat - trans.cg | ag -cDIW

Tree.o: Tree.h Tree.c
        $(CC) -c -ansi -pedantic Tree.c -I $(REUSE)

Parser.o:       Parser.h Parser.c Tree.h
        $(CC) -c -ansi -pedantic Parser.c -I $(REUSE)

Scanner.o:      Scanner.h Scanner.c
```

```
        $(CC) -c -ansi -pedantic Scanner.c -I $(REUSE)

myprint.o:      myprint.h myprint.c
        $(CC) -c -ansi -pedantic myprint.c -I $(REUSE)

myscope.o:      myscope.h myscope.c mydtypes.h Tree.h
        $(CC) -c -ansi -pedantic myscope.c -I $(REUSE)

myset.o:        myset.h myset.c mydtypes.h global.h
        $(CC) -c -ansi -pedantic myset.c -I $(REUSE)

mytrans.o:      mytrans.h mytrans.c mydtypes.h Output.h TransAtt.h
        $(CC) -c -ansi -pedantic mytrans.c -I $(REUSE)

Output.o:       Output.h Output.c Tree.h TransAtt.h
        $(CC) -c -ansi -pedantic Output.c -I $(REUSE)

Transhelp1.o:   Transhelp1.h Transhelp1.c
        $(CC) -c -ansi -pedantic Transhelp1.c -I $(REUSE)

Transhelp2.o:   Transhelp2.h Transhelp2.c
        $(CC) -c -ansi -pedantic Transhelp2.c -I $(REUSE)

PrepAtt.o:     PrepAtt.h PrepAtt.c TransAtt.h
        $(CC) -c -ansi -pedantic PrepAtt.c -I $(REUSE)

TransAtt.o:     TransAtt.h TransAtt.c
        $(CC) -c -ansi -pedantic TransAtt.c -I $(REUSE)

EnvAtt.o:       EnvAtt.h EnvAtt.c
        $(CC) -c -ansi -pedantic EnvAtt.c -I $(REUSE)

transform.o:    transform.h transform.c mytrans.h Output.h TransAtt.h
        $(CC) -c -ansi -pedantic transform.c -I $(REUSE)

debugenv.o:     debugenv.h debugenv.c
        $(CC) -c -ansi -pedantic debugenv.c -I $(REUSE)

Muster.o:       Muster.h Muster.c
        $(CC) -c -ansi -pedantic Muster.c -I $(REUSE)

clean:
        rm -f Scanner.? Parser.? Tree.?
        rm -f Output.? PrepAtt.? TransAtt.? EnvAtt.? Muster.?
        rm -f Transhelp1.? Transhelp2.?
        rm -f transform.?
        rm -f trans.rex Parser.lalr Scanner.rpp
```

```
rm -f yy*.w *.o *.TS
rm -f $(NOWFILES)
```

C.3 Der Scanner und der Parser

Aan dieser Stelle werden beide Phasen beschrieben, da der verwendete Präprozessor `cg` (siehe Abschnitt 6.3) den größten Teil der `Scanner`-Spezifikation und die gesamte `Parser`-Spezifikation aus der Datei `trans.pars` (C.3.2) erzeugt. Der restliche Teil des `Scanner`s wird in der Datei `trans.scan` (C.3.1) spezifiziert.

In 6.1, 6.2, 6.3 und 7 befinden sich Erläuterungen zu den Dateien aus denen der Scanner und der Parser erzeugt werden. In [Gro92b] befindet sich auf Seite 2 ein Diagramm (Fig.1), das den Datenfluß während der Scanner- und der Parser-Generierung darstellt. Mit dem gerade zitierten Artikel und den Artikeln [Gro92d, GV92] erhält man einen vollständigen Einblick in die Grammatik der benötigten Dateien `trans.scan` und `trans.pars`.

C.3.1 Die Terminals, Kommentare und line-Direktiven

In der Datei `trans.scan` werden die Terminals (außer Schlüsselwörter), die Kommentare und die line-Direktiven von ProSet spezifiziert und Fehlermeldungen für den `Scanner` angegeben. Die Prozedur `auxCPPinfo` wurde der *Eli*-Datei `Scanner.c` entnommen und an *Cocktail* angepaßt. Sie werden für die line-Direktiven benötigt.

⟨*trans.scan*⟩≡
```
EXPORT
{
#include "Idents.h"
#include "Positions.h"
#include "global.h"

INSERT tScanAttribute
}

GLOBAL
{
#include <stdlib.h>
#include "Errors.h"
#include "StringMem.h"
extern char *strcat();
static char Word [256];

INSERT ErrorAttribute

void auxCPPinfo(char * start, /* start of characters recognized by reg expr */
           int length)   /* length of what was recognized in reg expr */
{
  register char c;
```

```
register char *p = start+1 ; /* first position after the @ */
register char *beginInt,len;
register char *Name;

if (strncmp (p, "line ", 5))
{
  Message("invalid macro processor information after @", 3,
          Attribute.Position);
  exit(-1);
}
p += 5;

while ((c = *p++) && isspace(c) && c!= '\n') ;

if (!isdigit(c))
{
  Message("invalid macro processor information after @", 3,
          Attribute.Position);
  exit(-1);
}

beginInt = p-1;
while ( (c = *p++) && isdigit(c) ) ;

/*  p points to position after string */
Attribute.Position.LineOffset = yyLineCount + 1 - atoi(beginInt);

/* process the file name string ( it should be simple, use auxCstring if )*/
/* not ok note, cpp seems to barf on (Haeckerchen) as a file name, so it is
   being ignored. */

while (isspace(c) && c != '\n')
  c = *p++;

if ( c == '\n' )
{
   yyLineCount++;
   yyLineStart = (unsigned char*)p-1;
   return;  /* we have a macro processor information without a file */
}

if ( c == '"' )
{
  len=0;
  Name = p; /* character AFTER (Haeckerchen) */
  while ( (c = *p++) && c != '"' )
  {
```

```
      len++;
      if ( c == '\n' )
      {
        Message("missing enclosing \" for macro processor file information", 3,
                Attribute.Position);
        exit(-1);
      }
      if (c == '\0')
      {
        p = TokenPtr + TokenLength;
        yyLineStart = (unsigned char*)p-1;
        if ( *p == '\0')
        {
          Message("file ends in string", 3, Attribute.Position);
          exit(-1);
        }
      }
    }
    /*  P is now one character after the (Haeckerchen) */
    /*  save the (Haeckerchen) and copy the string to SourceName */
    {
      int t=0;

      Attribute.Position.IncludeFile = MakeIdent(Name, len);
    }
    /* SourceName now has the name of the include file */
    /* ignore the rest of the line. */

    while( (c = *p++) && c != '\n' ) ;
 }
  else
  {
    Message("invalid macro processor information after @line <number>", 3,
            Attribute.Position);
    exit(-1);
  }
  yyLineStart = (unsigned char*)p-1;
/* we have a macro processor information */
}

}

LOCAL
{
char String[256], S[256];
}
```

```
DEFAULT
{
MessageI("illegal character", xxError, Attribute.Position, xxCharacter,
          TokenPtr);
}

EOF /* Fehlermeldung bei ueberraschendem Dateiende */
{
if(yyStartState == COMMENT) Message("unclosed comment",xxError,
                                     Attribute.Position);
else if(yyStartState == STRING) Message("unclosed string",xxError,
                                         Attribute.Position);
}

DEFINE
 digit = { 0-9 } .
 letter = { a-z A-Z _} .

START COMMENT STRING  /* moegliche Zustaende */
```

In den Klammerpaaren { } werden die Attribute für die einzelnen Terminals berechnet, Zustandsübergänge und die Rückgabewerte der erkannten Terminals angegeben. Die für ein Terminal definierten Attribute werden in der Datei trans.pars (C.3.2) angegeben.
⟨*trans.scan*⟩+≡

```
RULE

INSERT RULES #STD#

#STD#   digit + : {(void) GetWord (Word);
                   Attribute.intlit.Integer = atoi((char *)Word);
                   Attribute.intlit.Length = strlen((char *)Word);
                   return intlit;}

#STD#   (digit+ "." digit+)|(digit+ "." digit+ {e E}{+ \-}? digit+)
                : {(void) GetWord (Word);
                   Attribute.floatlit.Float = atof((char *)Word);
                   Attribute.floatlit.Length = strlen((char *)Word);
                   return floatlit;}

NOT #COMMENT#   "(*"
                :- {yyStart(COMMENT);}                  /* Modula-Kommentar */

#COMMENT# "*)"  :- {yyStart(STD);}

#COMMENT# (ANY | {\n})
                :- {}

        "--" ANY* >                                        /* ADA-Kommentar */
                :- {}

#STD# "         : {String[0] = '\0'; yyStart(STRING);}

#STRING# (-{"})* "
                :- {(void)GetWord(S); (void)strcat(String, "\"");
                    (void)strcat(String, S);
                    Attribute.str.Str = PutString(String, strlen(String));
                    Attribute.str.Length = strlen((char *)String);
                    yyStart (STD); return str;}

#STD#   letter ( letter | digit )*
                : { Attribute.id.Ident = MakeIdent(TokenPtr, TokenLength);
                    Attribute.id.Length = TokenLength;
                    if(Attribute.id.Length > MAXIDENTLENGTH)
                    {
                       ErrorMessageI(0, 2, NoPosition, 0,
                                     "maximal ident length is 1024");
```

```
                    }
                    return id;}

#STD#   "@" ANY* > :- {auxCPPinfo(TokenPtr, TokenLength);} /* line-Direktive */
```

C.3.2 Die konkrete Grammatik

In der Datei `trans.pars` wird zuerst die konkrete Grammatik spezifiziert und für die Terminals die Attribute angegeben und initialisiert. Dann folgt die Abbildung von der konkreten zur abstrakten Grammatik (C.3.3). Für weitere Informationen zum ersten Teil siehe [Gro88a] und für den zweiten Teil Kapitel 4:„Using the Generated Program Module" in [Gro92a]. Zur Herleitung der konkreten und abstrakten Grammatik siehe 7.

An dieser Stelle ist noch anzumerken, daß im folgenden meistens nur Teile der Grammatiken und Attributauswerter in den einzelnen Kapiteln angegeben werden, um ein besseres Lesen der Dokumentation zu ermöglichen. Die restlichen Teile befinden sich am Ende der Dokumentation.

Bei den Attributauswertern wird jedoch darauf geachtet, daß die für das Verständnis des Programms wichtigen Teile in den entsprechenden Kapiteln vorhanden sind.

⟨*trans.pars*⟩≡

```
PARSER

BEGIN
{
TreeRoot = NoTree;  /* setzen wir, um bei mehreren Transformationsdurchlaeufen
                       keine Reste vom alten AST in unseren Neuen
                       mitzuuebernehmen */
}

PROPERTY INPUT

RULE

xInitChain = xProgDefn .

xProgDefn = 'program' xId1:xId S1:';' xProgBody 'end' xId2:xId S2:';' .

xId = id .

xProgBody = xDecls xcBeginStmts xcPHMDefns .

xcPHMDefns = <
  new1xcPHMDefns = xcPHMDefns xPHMDefn .

  new2xcPHMDefns = .
>.

xPHMDefn = <
  new1xPHMDefn = 'procedure' xId1:xId xParamList S1:';' xProgBody 'end'
                 xId2:xId S2:';' .
```

```
  new2xPHMDefn = 'module' xId1:xId xModImport xModExport xProgBody 'end'
                 xId2:xId ';' .

  new3xPHMDefn = 'handler' xId1:xId xRdParamList xImplAsso S1:';' xProgBody
                 'end' xId2:xId S2:';' .
>.
```

⟨*transpars.fort*⟩

```
/* hierunter steht der Rest */

/******************************************************************************
        TERMINALS
******************************************************************************/

id : [Ident: tIdent] [Length] { Ident := NoIdent; Length := 0; } .

intlit : [Integer] [Length] { Integer := 0; Length := 0; } .

floatlit : [Float : float ] [Length] { Float := 0.0; Length := 0; } .

str : [Str : tStringRef] [Length] { Length := 0;
                         Str := {Word[0] = '\0'; Str = PutString(Word,0);};}.
```

Für *transpars.fort* siehe C.42.1 auf Seite 402

C.3.3 Die Abbildung der konkreten Grammatik auf die Abstrakte

Erläuterungen siehe [Gro92a], 6.4.1 und 7

⟨*trans.pars*⟩+≡

```
MODULE Tree

PARSER

GLOBAL
{
#include "Tree.h"
}

DECLARE  /*  hier sind alle Nonterminals (ausser xinitchain und xBinOp) aus
             der konkreten Grammatik aufgefuehrt  */
xProgDefn xId xProgBody xcPHMDefns xPHMDefn xParamList xcParams xParamMode
xModImport xModExport xIdList xRdParamList xImplAsso xDecls xDecl xcVars
xSingleVar xDeclKey xPersDecl xcBeginStmts xStmts xExplAsso xHandAsso xStmt
xStmtSignal xStmtNotify xStdIO xcPrimary xFrom xActuList xElIfStmt xElIfStmts
xElseStmts xCaseStmts xcCaseStmt xcCaseList xLabel xLoops xcLoopStmt xcForStmt
xcWhileStmt xcWhilefound xcUntilStmt xcTempList xTemplate xTempCond xcEFEmpty
xcEFList xExprFormal xInto xFormal xIterator xSimpleIts xSimpleIt xMapSel
xcLValList xLValue xcSimpleLV xcComps xcComp xSelector xcSetFormer xcTupFormer
xcTCList xcTupComp xFormer xExprList xInstantiate xInstExport xInstImport
xcPriSel xQualId xLambda xExpr xQuantifier xQualifier xElIfExprs xElIfExpr
xElseExpr xCaseExprs xcCaseExpr xOrOp xcOrTerm xAndOp xcAndTerm xcBoolOp
xcBoolTerm xcSetOp xcSetTerm xcAddOp xcAddTerm xcMulOp xcMulTerm xcPowOp
xcPowTerm xBinOp xUnOp xTypeList = [Tree : tTree] .

RULE

xInitChain       = { => { TreeRoot = mxInitChain(xProgDefn:Tree);};} .

xProgDefn        = { Tree := mxProgDefn('program':Position,
                            mxId(xId1:Tree), S1:Position, xProgBody:Tree,
                            'end':Position, mxId(xId2:Tree),S2:Position);} .

xId              = { Tree := mid(id:Position,id:Ident,id:Length);} .

xProgBody        = { Tree := mxProgBody(xDecls:Tree, xcBeginStmts:Tree,
                            xcPHMDefns:Tree);} .

new1xcPHMDefns   = { Tree := mnew4xPHMDefn(xcPHMDefns:Tree, xPHMDefn:Tree);}.

new2xcPHMDefns   = { Tree := mnew5xPHMDefn();} .

new1xPHMDefn     = { Tree := mnew1xPHMDefn('procedure':Position,
```

```
                                 mxId(xId1:Tree), xParamList:Tree, S1:Position,
                                 xProgBody:Tree, 'end':Position, mxId(xId2:Tree),
                                 S2:Position);} .

new2xPHMDefn      = { Tree := mnew2xPHMDefn('module':Position, mxId(xId1:Tree),
                                 xModImport:Tree, xModExport:Tree,xProgBody:Tree,
                                 'end':Position, mxId(xId2:Tree),';':Position);}.

new3xPHMDefn      = { Tree := mnew3xPHMDefn('handler':Position, mxId(xId1:Tree),
                                 xRdParamList:Tree, xImplAsso:Tree, S1:Position,
                                 xProgBody:Tree, 'end':Position, mxId(xId2:Tree),
                                 S2:Position);} .
```

⟨*transpars.abb*⟩

```
END Tree
```

Für *transpars.abb* siehe C.42.2 auf Seite 416.

C.4 Die Abstrakte Grammatik

Nachdem bereits die Abbildung von der konkreten zur abstrakten Grammatik angegeben wurde (C.3.3), folgt nun die abstrakte Grammatik. Sie ist in der Datei `trans.cg` spezifiziert und dient als Grundlage für die nachfolgenden Attributauswerter und für die Mustererkennung. Die Syntax für die Spezifikationssprache, in der die abstrakte Grammatik beschrieben ist, wird in Abschnitt 6.4 und in [Gro92a] erläutert.

⟨*trans.cg*⟩≡

```
MODULE AbstractSyntax

TREE

EXPORT
{
#define TREEFLAG == true
#include "Idents.h"
#include "Positions.h"
#if !defined(MYDTYPES)
#include "mydtypes.h"
#endif
}

IMPORT
{
#include "Errors.h"
}

PROPERTY INPUT

RULE

xInitChain = xProgDefn .

xProgDefn = [pos1:tPosition] xId1:xId [pos2:tPosition] xProgBody
            [pos3:tPosition] xId2:xId [pos4:tPosition] .
       /* = 'program' xId ';' xProgBody 'end' xId ';' */

xId = id .

xProgBody = xDecls xStmts xPHMDefn .

xPHMDefn = <
  new1xPHMDefn = [pos1:tPosition] xId1:xId xParamList [pos2:tPosition]
                 xProgBody [pos3:tPosition] xId2:xId [pos4:tPosition] .
            /* = 'procedure' xId xParamList ';' xProgBody 'end' xId ';' */
```

```
  new2xPHMDefn = [pos1:tPosition] xId1:xId xModImport xModExport xProgBody
                 [pos2:tPosition] xId2:xId [pos3:tPosition] .
            /* = 'module' xId xModImport xModExport xProgBody 'end' xId ';' */

  new3xPHMDefn = [pos1:tPosition] xId1:xId xRdParamList xImplAsso
                 [pos2:tPosition] xProgBody [pos3:tPosition] xId2:xId
                 [pos4:tPosition] .
            /* = 'handler' xId xRdParamList xImplAsso ';' xProgBody 'end'
                 xId ';' */

  new4xPHMDefn = No1:xPHMDefn No2:xPHMDefn .

  new5xPHMDefn = .
>.
```

⟨*transcg.abs*⟩

```
id = [Pos:tPosition] [Ident:tIdent] [Length] .

intlit = [Pos:tPosition] [Integer] [Length] .

floatlit = [Pos:tPosition] [Float:float] [Length] .

str = [Pos:tPosition] [Str:tStringRef] [Length] .

END AbstractSyntax
```

Für *transcg.abs* siehe C.42.3 auf Seite 436

C.5 Die Attributauswerter

Die Syntax der Spezifikationssprache für die Attributauswerter und deren Generierung ist in 6.4.2 und [Gro91a] beschrieben. Alle Attributauswerter benötigen auf jeden Fall die abstrakte Grammatik (C.4).

C.6 Der Attributauswerter `EnvAtt`

Dieser Attributauswerter setzt nur das Attribut `env` vom Typ `Scope`, damit die Funktion `mygettree` (C.7) die Prozedurnamentabelle erzeugen kann.
⟨*trans.cg*⟩+≡

```
MODULE EnvOutAtt
EVAL EnvAtt

DECLARE
xProgBody = [env : Scope] .

END EnvOutAtt
```

C.6.1 Die C-Struktur `Scope`

`Scope` ist in der Datei `mydtypes.h` als Zeiger auf `myscope` definiert. `myscope` spiegelt die Struktur der Prozedurnamentabelle wieder. `scopename` ist der Prozedur-, Handler- oder Modulname des dargestellten Scopes (für das Hauptprogramm wird scopename auf NoIdent gesetzt). `PhmList` ist eine verkettete Liste, in der die im Scope definierten Prozedur-, Handler- und Modulnamen (`phmname`) abgespeichert sind. `*outer` ist ein Zeiger auf den äußeren Scope (für das Hauptprogramm ist dieser Zeiger `NULL`).
⟨*mydtypesh*⟩≡

```
typedef struct phmlist { tIdent phmname;
                          struct phmlist *next;
                        } PhmList;

typedef struct scope { tIdent scopename;
                        PhmList *phmnames;
                        struct scope *outer;
                      } myscope ;

typedef myscope *Scope;
```

C.7 Die *puma*-Funktion `mygettree`

Die Funktion `mygettree` ist eine *puma*-Funktion (zu *puma* siehe 6.5 und [Gro91b]). `mygettree` bekommt den gesamten abstrakten Syntaxbaum (AST) als Eingabe übergeben. Nachdem das Pattern `xInitChain(...)` erkannt wurde, wird die Prozedurnamentabelle initialisiert (`XinitChain` ist die Wurzel des AST). Die Funktion `mygetscope` füllt nun die Prozedurnamentabelle, d.h. sie durchläuft den AST und trägt die vorkommenden Namen in die Prozedurnamentabelle ein.

Anmerkungen:

- Der Ausdruck Pattern wird benutzt, um eine deutliche Trennung der Begriffe Transformationsmuster (Muster) und Pattern (Musterïn *puma*-Funktionen) zu erhalten.
- ... wird als Abkürzung bei den Pattern verwendet, sofern die Eindeutigkeit des Pattern in dem Zusammenhang gewährleistet ist.

⟨*transpuma*⟩≡

```
PROCEDURE mygettree(tree:Tree)
xInitChain(
  xProgDefn:xProgDefn(_,_,_,
    xProgBody:xProgBody(_,_,
      new:xPHMDefn,
      env,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_,_,_),_,_)
? env = initscope(NoIdent,NULL);
  mygetscope(env, new); .
```

C.7.1 Die C-Funktion `initscope`

Die Funktion `initscope` ist in der Datei `myscope.c` definiert. Sie initialisiert für jeden Sichtbarkeitsbereich die Prozedurnamentabelle, indem sie einen Zeiger auf den über ihr liegenden Bereich setzt und den neuen Prozedur-, Handler- oder Modulnamen einträgt.

⟨*myscopec1*⟩≡

```
Scope initscope(tIdent scopename, Scope outer)
{
  Scope help;

  if((help = (Scope)malloc(sizeof(myscope))) == NULL)
  {
     Message("not enough memory", 2, NoPosition);
     exit(-1);
  }
  help->scopename = scopename;
  help->phmnames = NULL;
  help->outer = outer;
  return help;
}
```

⟨*myscopeh*⟩≡

```
extern Scope initscope(tIdent, Scope);
```

C.7.2 Die *puma*-Funktion `mygetscope`

`mygetscope` erhält als Parameter den aktuellen Scope und die Prozedur-, Handler- und Moduldeklarationen.

Zunächst wird der Name von `now` in die Prozedurnamentabelle eingetragen und für `now` ein neuer Bereich initialisiert. Hiernach wird `mygetscope` mit den benötigten Parametern aus `now` aufgerufen. `xPHMDefn` muß auf die angegebene Weise aufgeschlüsselt werden, da bei `new1xPHMDef`, `new2xPHMDef` und `new3xPHMDef` die benötigten Parameter an verschiedenen Positionen in der Grammatikregel stehen und `new4xPHMDef` und `new5xPHMDef` eine andere Darstellung haben und daher auch anders behandelt werden müssen.

⟨*transpuma*⟩+≡

```
PROCEDURE mygetscope(scope:Scope,now:xPHMDefn)
_,new1xPHMDefn(_,_,_,_,_,_,_,_,_,
    xId1:xId(id:id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_),_,_,
    xProgBody:xProgBody(_,
      newStmts:xStmts,
      newPHMDefn:xPHMDefn,
      env,_,_,_,_,_,_,_,_),_,_,_)
? scope->phmnames = idphmlist(scope->phmnames, Ident);
  env = initscope(Ident,scope);
  mygetscope(env,newPHMDefn); .

_,new2xPHMDefn(_,_,_,_,_,_,_,_,_,
    xId1:xId(id:id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_),_,_,
    xProgBody:xProgBody(_,
      xStmts:xStmts,
      xPHMDefn:xPHMDefn,
      env,_,_,_,_,_,_,_,_),_,_,_)
? scope->phmnames = idphmlist(scope->phmnames, Ident);
  env = initscope(Ident,scope);
  mygetscope(env,xPHMDefn); .

_,new3xPHMDefn(_,_,_,_,_,_,_,_,_,
    xId1:xId(id:id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_),_,_,_,
    xProgBody:xProgBody(_,
      xStmts:xStmts,
      xPHMDefn:xPHMDefn,
      env,_,_,_,_,_,_,_,_),_,_,_)
? scope->phmnames = idphmlist(scope->phmnames, Ident);
  env = initscope(Ident,scope);
  mygetscope(env,xPHMDefn); .

_,new4xPHMDefn(_,_,_,_,_,_,_,_,No1:xPHMDefn,No2:xPHMDefn)
?  mygetscope(scope,No1);
```

```
    mygetscope(scope,No2); .

  _,new5xPHMDefn ? .
```

C.7.3 Die C-Funktion idphmlist

Diese Funktion speichert Prozedur-, Handler und Modulnamen in der Prozedurnamentabelle.

⟨*myscopec2*⟩≡

```
PhmList *idphmlist(PhmList *list, tIdent phmname)
{
  PhmList *help;

  if((help = (PhmList *)malloc(sizeof(PhmList))) == NULL)
    {
      Message("not enough memory", 2, NoPosition);
      exit(-1);
    }
  help->phmname = phmname;
  help->next = list;
  list = help;
  return list;
}
```

⟨*myscopeh*⟩+≡

```
extern PhmList *idphmlist(PhmList *, tIdent);
```

C.8 Der Attributauswerter PrepAtt

C.8.1 Das Modul PrepOutAtt

Dem Nonterminal `xLoops` wird ein Attribut hinzugefügt, daß auf den zur Schleife gehörenden Sichtbarkeitsbereich zeigt. Das neue Attribut wird in nachfolgenden Attributauswertern benötigt.

⟨*trans.cg*⟩+≡

```

MODULE PrepOutAtt
EVAL PrepAtt

DECLARE
xLoops = [loopenv:Scope] .

END PrepOutAtt
```

C.8.2 Das Modul `PrepAtt`

`scope` wird bei `xProgBody` der gültige Sichtbarkeitsbereich zugewiesen.

Bei der Regel `xProgBody` können mit `printEnv` die dort sichtbaren Prozedur-, Handler- und Modulnamen ausgegeben werden (wurde nur zum Testen der Prozedurnamentabelle benötigt). `printEnv` ist in `debugenv.c` definiert. `env` wurde im Modul `EnvAtt` eingeführt. Bei `xLoops` wird das Attribut `loopenv` auf den aktuellen Prozedurnamenbereich gesetzt.

⟨*trans.cg*⟩+≡

```
MODULE PrepAtt
EVAL PrepAtt

IMPORT
{
#include "transform.h"
/* #include "debugenv.h" */
}

GLOBAL
{
   Scope scope;
}

RULE

xProgBody = { => { if ((scope = (Scope)malloc(sizeof(myscope))) == NULL)
                   {
                      Message("not enough memory", 2, NoPosition);
                   }
                   scope = env;
                   /* printEnv(env); */};}.

xLoops = { loopenv := NULL; }.

new6xLoops = { loopenv := NULL; }.

new7xLoops = { loopenv := { if ((loopenv = (Scope)malloc(sizeof(myscope)))
                                == NULL)
                            {
                               Message("not enough memory", 2, NoPosition);
                            }
                            loopenv = scope;
                         }; }.

new8xLoops = { loopenv := { if ((loopenv = (Scope)malloc(sizeof(myscope)))
                                == NULL)
                            {
```

```
                                Message("not enough memory", 2, NoPosition);
                             }
                             loopenv = scope;
                          }; }.

new9xLoops = { loopenv := { if ((loopenv = (Scope)malloc(sizeof(myscope)))
                                 == NULL)
                             {
                                Message("not enough memory", 2, NoPosition);
                             }
                             loopenv = scope;
                          }; }.

new10xLoops = { loopenv := { if ((loopenv = (Scope)malloc(sizeof(myscope)))
                                 == NULL)
                             {
                                Message("not enough memory", 2, NoPosition);
                             }
                             loopenv = scope;
                          }; }.
END PrepAtt
```

C.8.3 Die C-Funktion `printEnv`

`printEnv` gibt für `env` die sichtbaren Prozedur-, Handler- und Modulnamen aus.

⟨*debugenv.c*⟩≡

```
#include "debugenv.h"

void printEnv(Scope env)
{
   PhmList *helpphmlist;
   Scope helpscope;
   tIdent id;

   helpscope = env;
   fprintf(ofile, "actual Procedurename: ");
   WriteIdent(ofile, helpscope->scopename);
   fprintf(ofile, "\n");
   fprintf(ofile, "\n visible Procedure-, Module- or Handlername \n");
   while(helpscope != NULL)
   {
      helpphmlist = helpscope->phmnames;
      while(helpphmlist != NULL)
      {
         WriteIdent(ofile, helpphmlist->phmname);
         fprintf(ofile, "\n");
         helpphmlist = helpphmlist->next;
      }
      helpscope = helpscope->outer;
   }
   fprintf(ofile, "\n");
}
```

⟨*debugenv.h*⟩≡

```
#include "myscope.h"
#include "Idents.h"
#include "global.h"

extern void printEnv(Scope);
```

C.9 Der Attributauswerter `TransAtt`

C.9.1 Das Modul `TransOutAtt`

Im Modul `TransOutAtt` werden dem Nonterminal `xLoops` zwei Attribute zugefügt, an denen abgelesen werden kann, ob eine transformierbare Menge in der Schleife vorhanden ist (`transatt`) und welche freien und gebundenen Variablen in dieser Menge vorkommen (`variable`). Diese neuen Attribute werden auch noch in nachfolgenden Attributauswertern verwendet.
⟨*trans.cg*⟩+≡

```
MODULE TransOutAtt
EVAL TransAtt

DECLARE
xLoops = [ transatt:CODE ] [ variable:DSET ] .

END TransOutAtt
```

C.9.2 Das Modul `TransAtt`

`xLoops` erhält das ererbte Attribut `loop`. Es zeigt auf `xLoops`. Dieses Attribut wird benötigt, um für die Attributauswerter `Transhelp1` und `Transhelp2` den Knoten `xLoops` verfügbar zu haben, da in diesen Attributauswertern von `xLoops` die Attribute `variable` und `transatt` benötigt werden.

In diesem Modul wird nun ein Teil der Transformationen durchgeführt. Der Rest der Transformationen wird direkt in die Ausgabedatei geschrieben.

Bei den Knoten `new38xStmt`, `new39xStmt`, `new40xStmt`, `new41xStmt`, `new42xStmt` und `new43xStmt` wird `loop` gesetzt (es sind die einzigen Regeln, bei denen `xLoops` auf der rechten Seite steht).

Bei `new7xLoops`, `new8xLoops`, `new9xLoops` und `new10xLoops` werden nun Funktionen aufgerufen, die herausfinden, welche transformierbare Menge in der Schleife vorkommt (`Pattern` und `getcode`) und ob die Voraussetzungen zum Transformieren erfüllt sind (`CanIDelete`, `Transhelp1` und `Transhelp2`). Sind die Voraussetzungen erfüllt, wird diese Menge in der Schleife gelöscht (`Delete`). (`new6xLoops` wird nicht betrachtet, da dort mehrere Ausgänge aus der Schleife vorkommen können, und somit die Voraussetzungen für die endliche Differentiation nicht gegeben sind.).

⟨*trans.cg*⟩+≡

```
MODULE TransAtt
EVAL TransAtt

IMPORT
{
#include "transform.h"
/* #include "debugenv.h" */
}

GLOBAL
{
   Scope scope;
}

DECLARE
xLoops = [ loop:tTree INH] .

RULE

new38xStmt = { xLoops:loop := xLoops; } .

new39xStmt = { xLoops:loop := xLoops; } .

new40xStmt = { xLoops:loop := xLoops; } .
```

```
new41xStmt = { xLoops:loop := xLoops; } .

new42xStmt = { xLoops:loop := xLoops; } .

new43xStmt = { xLoops:loop := xLoops; } .

xLoops = { transatt := noset; variable := NULL; }.

new6xLoops = { variable := NULL; transatt := noset; }.

new7xLoops = { variable
               transatt := { Pattern(&xIterator, &variable, &transatt, code);
                             if(transatt == noset)
                                getcode(&xStmts, &variable, &transatt);
                             if(transatt != noset)
                                CanIDelete(xStmts, variable, &transatt);
                             if(transatt != noset) Transhelp1(loop);
                             if(transatt != noset) Transhelp2(loop);
                             if(transatt != noset)
                             {
                                Delete(&xIterator, variable, transatt);
                                Delete(&xStmts, variable, transatt);
                              }}; }.

new8xLoops = { variable
               transatt := { Pattern(&xExpr, &variable, &transatt, code);
                             if(transatt == noset)
                                getcode(&xStmts, &variable, &transatt);
                             if(transatt != noset)
                                CanIDelete(xStmts, variable, &transatt);
                             if(transatt != noset) Transhelp1(loop);
                             if(transatt != noset) Transhelp2(loop);
                             if(transatt != noset)
                             {
                                Delete(&xExpr, variable, transatt);
                                Delete(&xStmts, variable, transatt);
                              }}; }.

new9xLoops = { variable
               transatt := { Pattern(&xIterator, &variable, &transatt, code);
                             if(transatt == noset)
                                getcode(&xStmts, &variable, &transatt);
                             if(transatt != noset)
                                CanIDelete(xStmts, variable, &transatt);
                             if(transatt != noset) Transhelp1(loop);
                             if(transatt != noset) Transhelp2(loop);
                             if(transatt != noset)
```

```
                              {
                                 Delete(&xIterator, variable, transatt);
                                 Delete(&xStmts, variable, transatt);
                               }}; }.

new10xLoops = { variable
                transatt := { Pattern(&xExpr, &variable, &transatt, code);
                              if(transatt == noset)
                                 getcode(&xStmts, &variable, &transatt);
                              if(transatt != noset)
                                CanIDelete(xStmts, variable, &transatt);
                              if(transatt != noset) Transhelp1(loop);
                              if(transatt != noset) Transhelp2(loop);
                              if(transatt != noset)
                              {
                                 Delete(&xExpr, variable, transatt);
                                 Delete(&xStmts, variable, transatt);
                              }}; }.
END TransAtt
```

C.9.2.1 Der C-Aufzählungstyp CODE und die C-Struktur DSET

In `CODE` werden die Bezeichnungen angegeben, die im Anhang A den applikativen Ausdrücken zugwiesen wurden. An `CODE` kann also abgelesen werden, welche transformierbare Menge (mindestens) in einer Schleife vorkommt (Kennung). Innerhalb eines Transformationsdurchlaufes wird in jeder Schleife nur die erste vorkommende transformierbare Menge transformiert. `noset` wird gesetzt, wenn keine transformierbare Menge in der Schleife vorkommt.

In `DSET` wird für jede transformierbare Menge ein Typ definiert, in dem die freien und gebundenen Variablen abgespeichert werden können. Diese Typen können bei unterschiedlichen Mustern verschieden sein. Im Strukturnamen ist immer die Kennung eines dazugehörigen Musters enthalten. Kann eine Struktur für mehrere Muster verwendet werden, werden die übrigen Kennungen im Kommentar angefügt.

⟨*mydtypesh*⟩+≡

```
typedef enum { noset, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11,
               A12, A13, A14, A15, A16, A17, A18, A19, A20, A21, A22,
               A23, A24, A25, A26} CODE;

typedef struct { tIdent iterator;
                 tIdent r1sidename;
                 tIdent r2sidename;
               } tA1; /* A2 A9 */

typedef struct { tIdent iterator;
                 tIdent r1sidename;
                 tIdent r2sidename;
                 int integer;
               } tA3; /* A11 */

typedef struct { tIdent iterator;
                 tIdent r1sidename;
                 tIdent r2sidename;
                 tIdent rvariable;
               } tA4; /* A5 */

typedef struct { tIdent rsidename;
               } tA6;

typedef struct { tIdent iterator1;  /* j */
                 tIdent iterator2;  /* q */
                 tIdent rsidename;
               } tA8; /* A7 */

typedef struct { tIdent iterator;
                 tIdent rsidename;
```

```
               } tA10;

typedef struct { tIdent iterator1; /* j */
                 tIdent iterator2; /* k */
                 tIdent r1sidename;
                 tIdent r2sidename;
               } tA12;

typedef struct { tIdent iterator1; /* j */
                 tIdent iterator2; /* k */
                 tIdent iterator3; /* q */
                 tIdent r1sidename;
                 tIdent r2sidename;
               } tA13; /* A14 A15 A16  A17 A24 A25 */

typedef struct { tIdent iterator;
                 tIdent r1sidename;
                 tIdent r2sidename;
                 tIdent r3sidename;
               } tA18; /* A19 */

typedef struct { tIdent iterator1;
                 tIdent iterator2;
                 tIdent iterator3;
                 tIdent r1sidename;
                 tIdent r2sidename;
                 tIdent r3sidename;
               } tA20; /* A21 A22 A23 */

typedef struct { tIdent iterator;
                 tIdent rsidename;
                 unsigned short operator1;
                 unsigned short operator2;
                 int integer1;
                 int integer2;
               } tA26;

typedef struct { tIdent newlsidename;
                 int newlength;
                 union {
                         tA1 a1; /* A2 A9 */
                         tA3 a3; /* A11 */
                         tA4 a4; /* A5 */
                         tA6 a6;
                         tA8 a8; /* A7 */
                         tA10 a10;
                         tA12 a12;
```

```
                        tA13 a13; /* A14 A15 A16  A17 A24 A25 */
                        tA18 a18; /* A19 */
                        tA20 a20; /* A21 A22 A23 */
                        tA26 a26;
                 } A;
               } Dset;

typedef Dset * DSET;
```

C.9.3 Die *puma*-Funktion `getcode`

`getcode` liefert die Kennung der erkannten transformierbaren Menge. Die Kennung wird von der Funktion `Pattern` geliefert. Sobald `code != noset` ist, wird die Funktion beendet. Hierdurch wird sichergestellt, daß in jeder Schleife jeweils nur eine Menge transformiert wird. Kommt in `stmts` `new1xExplAsso` vor, kann bei der Anweisung `newstmt` eine Ausnahme ausgelöst werden. In diesem Fall kommt diese Anweisung nicht als Transformationsmöglichkeit in Frage.

In der Funktion `Pattern` sind die Muster spezifiziert. Damit nur in dieser Funktion die Pattern angegeben werden müssen, mußte die Funktion so weit gefaßt werden, daß nicht in allen Kontexten alle Parameter der Funktion benutzt werden. In diesem Kontext wird der Parameter 4 nicht benötigt. Der 5. Parameter (`code`) gibt an, daß die Funktion `Pattern` entweder von dem Attributauswerter `TransAtt` oder von der Funktion `getcode` aufgerufen wurde.

$\langle$*transpuma*$\rangle+\equiv$

```
PROCEDURE getcode(REF stmts:xStmts, REF variable:DSET, REF transatt:CODE)
new1xStmts(_,_,_,_,_,_,_,
  newstmts:xStmts,
  newstmt:xStmt,
  new2xExplAsso,_),_,_
? { getcode(&newstmts, &variable, &transatt);
    if(transatt == noset)
     Pattern(&newstmt, &variable, &transatt, code);
  }; .

new1xStmts(_,_,_,_,_,_,_,
  newstmts:xStmts,
  newstmt:xStmt,
  new1xExplAsso,_),_,_
? { getcode(&newstmts, &variable, &transatt);}; .
   /* Ausnahme kann ausgeloest werden */

new2xStmts(_,_,_,_,_,_,_,
  newstmt:xStmt,
  new2xExplAsso,_),_,_
? { Pattern(&newstmt, &variable, &transatt, code);}; .

new2xStmts(_,_,_,_,_,_,_,
  newstmt:xStmt,
  new1xExplAsso,_),_,_
? . /* Ausnahme kann ausgeloest werden */

new3xStmts(_,_,_,_,_,_,_,_,
  newstmts:xStmts),_,_
```

```
? { getcode(&newstmts, &variable, &transatt);}; .
```

C.10 Die *puma*-Funktion `Pattern`

In der Funktion `Pattern` werden alle Operationen durchgeführt, bei denen Transformationsmuster oder Modifikationsmuster (dx_i) eine Rolle spielen. Deshalb benötigt die Funktion mehr Parameter, als für die einzelnen Operationen nötig wären.

Ab Seite 181 werden die Pattern der differenzierbaren Ausdrücke aufgeführt. Auf Seite 231 beginnen dann die Modifikationsmuster und danach ab Seite 278 die Pattern, die auf enthaltene applikative Ausdrücke durchsucht werden (z.B. Zuweisungen und `put`-Anweisung).

$\langle$*transpuma1*$\rangle\equiv$

```
FUNCTION Pattern(REF pat:Tree, REF variable:DSET, REF transatt:CODE,
                 funcflag:FUNCT) int
LOCAL
{
   int result;
   tTree newvariable;
}
```

Hier wird der applikative Ausdruck $\boxed{\{j : j \textbf{ in } F \mid j \textbf{ in } G\}}$ spezifiziert.

$\langle$*transpuma1*$\rangle{+}\equiv$

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
        new11xBinOp,
        new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
```

im Falle `code` wird in `transatt` der Code des erkannten Musters (hier `A1`) eingefügt und die Funktion `getA1variable` speichert in `variable` die freien und gebundenen Variablen der Menge.

⟨*transpuma1*⟩+≡
```
         case code \:
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual1, qual4) == 1)
            {
               variable = getA1variable(qual1, qual3, qual5);
               transatt = A1;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
```

Dieses Muster darf nicht in der Schleife für eine freie Variable der zu transformierenden Menge auftreten. Dies wird im `case`-Zweig `helpstmt` sichergestellt.

⟨*transpuma1*⟩+≡
```
  case helpstmt \: /* wird in Transhelp2 benutzt */
      transatt = noset;
      result = 0;
      break;
```

Im Falle `delete` wird entschieden, ob die Anweisung, mit dem zu transformierenden Muster (`transatt`) und dessen Variablen (`variable`) übereinstimmt. In diesem Fall wird die Anweisung von der aufrufenden Funktion gelöscht.

⟨*transpuma1*⟩+≡
```
  case delete \: /* wird von Delete aufgerufen */
      if(transatt == A1)
      {
         result = checkA1variable(qual1, qual3, qual5, variable);
         if(result == 1)
         {
            expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                        NoPosition, variable->newlsidename,
                        variable->newlength))));
         }
      }
      else
      {
         result = 0;
      }
      break;
  default \: result = 0;
      }
    };.
```

Hier wird das Muster $\boxed{\{j : j \textbf{ in } F \mid j \textbf{ notin } G\}}$ spezifiziert. Die einzelnen Operationen sind die gleichen, wie bei dem vorherigen Muster.

⟨*transpuma1*⟩+≡

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
        new12xBinOp,
        new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \: /* fuegt in transatt die Bezeichnung ein und traegt
                       in variable die Variablen der Anweisung ein */
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual1, qual4) == 1)
            {
               variable = getA1variable(qual1, qual3, qual5);
               transatt = A2;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A2)
           {
              result = checkA1variable(qual1,qual3,qual5,variable);
              if(result == 1)
              {
                  expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                              NoPosition,
                              variable->newlsidename,
```

```
                              variable->newlength))));
          }
       }
       else
       {
          result = 0;
       }
       break;
    default \: result = 0;
 }
};.
```

Hier wird das Muster $\boxed{\{j : j \textbf{ in } F \mid G(j) = H\}}$ spezifiziert. H steht für einen konstanten Integerwert.

⟨*transpuma1*⟩+≡
```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
    new3xFormer(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
      new1xIterator(_,_,_,_,_,_,_,_,
        new2xSimpleIts(_,_,_,_,_,_,_,_,
          new1xSimpleIt(_,_,_,_,_,_,_,_,
            new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
            new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
        new50xExpr(_,_,_,_,_,_,_,_,
          new44xExpr(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
            new1xSelector(_,_,_,_,_,_,_,_,_,
              new2xExprList(_,_,_,_,_,_,_,_,
                new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)),_)),
          new5xBinOp,
          new13xExpr(_,_,_,_,_,_,_,_,intlit:intlit)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual1, qual5) == 1)
            {
               variable = getA3variable(qual1, qual3, qual4, intlit);
               transatt = A3;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A3)
           {
              result = checkA3variable(qual1,qual3,qual4,
                             intlit, variable);
              if(result == 1)
```

```
          {
             expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                         NoPosition,
                         variable->newlsidename,
                         variable->newlength))));
          }
        }
        else
        {
          result = 0;
        }
        break;
     default \: result = 0;
  }
};.
```

Hier wird das Muster $\boxed{\{[F(j), j] : j \textbf{ in } G\}}$ spezifiziert.

⟨*transpuma1*⟩+≡

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
    new3xFormer(_,_,_,_,_,_,_,_,
      new37xExpr(_,_,_,_,_,_,_,_,_,
        new4xFormer(_,_,_,_,_,_,_,_,
          new44xExpr(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),
            new1xSelector(_,_,_,_,_,_,_,_,_,
              new2xExprList(_,_,_,_,_,_,_,_,
                new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
          new2xExprList(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_),_,
      new2xIterator(_,_,_,_,_,_,_,_,
        new2xSimpleIts(_,_,_,_,_,_,_,_,
          new1xSimpleIt(_,_,_,_,_,_,_,_,
            new5xLValue(_,_,_,_,_,_,_,_,qual4:new2xQualId),_,
            new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId))))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \: /* fuegt in transatt die Bezeichnung ein und traegt
                        in variable die Variablen der Anweisung ein */
            if(EqualQualId(qual2, qual3) == 1
               && EqualQualId(qual2, qual4) == 1)
            {
               variable = getA4variable(qual2, qual1, qual5, qual5);
               transatt = A4;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A4)
           {
              result = checkA4variable(qual2,qual1,qual5,qual5,variable);
              if(result == 1)
              {
```

```
                    expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                                NoPosition,
                                variable->newlsidename,
                                variable->newlength))));
                 }
              }
              else
              {
                 result = 0;
              }
              break;
          default \: result = 0;
     }
   };.
```

Hier wird das Muster $\boxed{\{j : j \textbf{ in } G \mid F(j) = q\}}$ spezifiziert.

⟨*transpuma1*⟩+≡

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
          new50xExpr(_,_,_,_,_,_,_,_,
            new44xExpr(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
              new1xSelector(_,_,_,_,_,_,_,_,_,
                new2xExprList(_,_,_,_,_,_,_,_,
                  new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)),_)),
            new5xBinOp,
            new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \: /* fuegt in transatt die Bezeichnung ein und traegt
                       in variable die Variablen der Anweisung ein */
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual2, qual5) == 1)
            {
               variable = getA4variable(qual1, qual4, qual3, qual6);
               transatt = A5;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A5)
           {
              result = checkA4variable(qual1, qual4, qual3, qual6, variable);
              if(result == 1)
              {
```

```
               expr = mnew44xExpr(mnew38xExpr(mnew2xQualId(mxId(mid(
                           NoPosition,variable->newlsidename,
                           variable->newlength)))),
                      mnew2xSelector(NoPosition,mnew2xExprList(mnew38xExpr
                           (qual6)),NoPosition));
            }
         }
         else
         {
            result = 0;
         }
         break;
     default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{\sharp F}$.

$\langle$*transpuma1*$\rangle+\equiv$

```
expr:new52xExpr(_,_,_,_,_,_,_,_,new3xUnOp(_,_,_,_,_,_,_,_,pos),
   new38xExpr(_,_,_,_,_,_,_,_,qual:new2xQualId)),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            variable = getA6variable(qual);
            transatt = A6;
            result = 1;
            break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A6)
            {
                result = checkA6variable(qual, variable);
                if(result == 1)
                {
                   expr = mnew38xExpr(mnew2xQualId(mxId(
                              mid(NoPosition,variable->newlsidename,
                              variable->newlength))));
                }
            }
            else result = 0;
            break;
       default \: result = 0;
    }
  };.
```

Hier folgt das Muster $\boxed{\{[j, \sharp F\{j\}] : j \textbf{ in domain } F\}}$.

$\langle$*transpuma1*$\rangle$+≡

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new37xExpr(_,_,_,_,_,_,_,_,_,
      new4xFormer(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new52xExpr(_,_,_,_,_,_,_,_,
            new3xUnOp,
            new44xExpr(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual2:xQualId),
              new2xSelector(_,_,_,_,_,_,_,_,_,
                new2xExprList(_,_,_,_,_,_,_,_,
                  new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId)),_))))),_),_,
    new2xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual4:new2xQualId),_,
          new52xExpr(_,_,_,_,_,_,_,_,
            new9xUnOp,
            new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)))))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual1, qual4) == 1
               && EqualQualId(qual2, qual5) == 1)
            {
               variable = getA8variable(qual2, qual1, NoTree);
               transatt = A7;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A7)
```

```
          {
              result = checkA8variable(qual2, qual1, NoTree, variable);
              if(result == 1)
              {
                 expr = mnew38xExpr(mnew2xQualId(mxId(
                            mid(NoPosition,variable->newlsidename,
                            variable->newlength))));
              }
          }
          else result = 0;
          break;
     default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{\sharp F\{j\}}$.

⟨*transpuma1*⟩+≡

```
expr:new52xExpr(_,_,_,_,_,_,_,_,new3xUnOp(_,_,_,_,_,_,_,_,pos),
  new44xExpr(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),
    new2xSelector(_,_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_))),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            variable = getA8variable(qual1, NoTree, qual2);
            transatt = A8;
            result = 1;
            break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A8)
            {
                result = checkA8variable(qual1, NoTree, qual2, variable);
                if(result == 1)
                {
                   expr = mnew44xExpr(mnew38xExpr(mnew2xQualId(mxId(
                              mid(NoPosition,variable->newlsidename,
                              variable->newlength)))),
                              mnew1xSelector(NoPosition,
                              mnew2xExprList(mnew38xExpr(qual2)),
                              NoPosition));
                }
            }
            else result = 0;
            break;
       default \: result = 0;
    }
  };.
```

Hier folgt das Muster $\boxed{\{j : j \textbf{ in } F \mid G(j)\}}$.

$\langle \textit{transpuma1} \rangle{+}{\equiv}$

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
      new44xExpr(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
        new1xSelector(_,_,_,_,_,_,_,_,_,
          new2xExprList(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)),_)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual1, qual5) == 1)
            {
               variable = getA1variable(qual1, qual3, qual4);
               transatt = A9;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A9)
            {
                result = checkA1variable(qual1, qual3, qual4, variable);
                if(result == 1)
                {
                   expr = mnew38xExpr(mnew2xQualId(mxId(
                             mid(NoPosition,variable->newlsidename,
                             variable->newlength))));
                }
```

```
          }
          else result = 0;
          break;
      default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{\{j : j \textbf{ in } F\}}$.

⟨*transpuma1*⟩+≡
```
  expr:new36xExpr(_,_,_,_,_,_,_,_,_,
    new3xFormer(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
      new2xIterator(_,_,_,_,_,_,_,_,
        new2xSimpleIts(_,_,_,_,_,_,_,_,
          new1xSimpleIt(_,_,_,_,_,_,_,_,
            new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
            new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))))),_),_,_,_
  RETURN result;
  ? { switch(funcflag)
      {
         case code \:
              if(EqualQualId(qual1, qual2) == 1)
              {
                 variable = getA10variable(qual3, qual1);
                 transatt = A10;
                 result = 1;
              }
              else
              {
                 variable = NULL;
                 transatt = noset;
              }
              break;
         case helpstmt \:
              transatt = noset;
              result = 0;
              break;
         case delete \:
              if(transatt == A10)
              {
                  result = checkA10variable(qual3, qual1, variable);
                  if(result == 1)
                  {
                     expr = mnew38xExpr(mnew2xQualId(mxId(
                                 mid(NoPosition,variable->newlsidename,
                                 variable->newlength))));
                  }
              }
              else result = 0;
              break;
         default \: result = 0;
      }
    };.
```

Hier wird das Muster $\boxed{\{j : j \textbf{ in } F \mid G(j) \neq H\}}$ spezifiziert. `H` steht für einen konstanten Integerwert.

$\langle$*transpuma1*$\rangle+\equiv$

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
    new3xFormer(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
      new1xIterator(_,_,_,_,_,_,_,_,
        new2xSimpleIts(_,_,_,_,_,_,_,_,
          new1xSimpleIt(_,_,_,_,_,_,_,_,
            new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
            new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
        new50xExpr(_,_,_,_,_,_,_,_,
          new44xExpr(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
            new1xSelector(_,_,_,_,_,_,_,_,_,
              new2xExprList(_,_,_,_,_,_,_,_,
                new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)),_)),
          new6xBinOp,
          new13xExpr(_,_,_,_,_,_,_,_,intlit:intlit)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual1, qual5) == 1)
            {
               variable = getA3variable(qual1, qual3, qual4, intlit);
               transatt = A11;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A11)
           {
              result = checkA3variable(qual1,qual3,qual4,
                               intlit, variable);
              if(result == 1)
```

```
            {
               expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                           NoPosition,
                           variable->newlsidename,
                           variable->newlength))));
            }
         }
         else
         {
            result = 0;
         }
         break;
     default \: result = 0;
  }
};.
```

Hier wird das Muster $\boxed{\{[[F(j), k], j] : [k, j] \textbf{ in } G\}}$ spezifiziert.

⟨*transpuma1*⟩+≡

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new37xExpr(_,_,_,_,_,_,_,_,_,
      new4xFormer(_,_,_,_,_,_,_,_,
        new37xExpr(_,_,_,_,_,_,_,_,_,
          new4xFormer(_,_,_,_,_,_,_,_,
            new44xExpr(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),
              new1xSelector(_,_,_,_,_,_,_,_,_,
                new2xExprList(_,_,_,_,_,_,_,_,
                  new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
            new2xExprList(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_),_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId))),_),_,
    new2xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new2xLValue(_,_,_,_,_,_,_,_,_,
            new3xLValue(_,_,_,_,_,_,_,_,
              No1:new5xLValue(_,_,_,_,_,_,_,_,qual5:new2xQualId),_,
              No2:new5xLValue(_,_,_,_,_,_,_,_,qual6:new2xQualId)),_),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual7:new2xQualId))))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual2, qual4) == 1
               && EqualQualId(qual2, qual6) == 1
               && EqualQualId(qual3, qual5) == 1)
            {
               variable = getA12variable(qual2, qual3, qual1, qual7);
               transatt = A12;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
```

```
         break;
    case delete \: /* wird von Delete aufgerufen */
         if(transatt == A12)
         {
            result = checkA12variable(qual2, qual3, qual1, qual7, variable);
            if(result == 1)
            {
               expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                           NoPosition,
                           variable->newlsidename,
                           variable->newlength))));
            }
         }
         else
         {
            result = 0;
         }
         break;
     default \: result = 0;
  }
};.
```

Hier wird das Muster $\boxed{\{[j,k] : [j,k] \textbf{ in } F \mid k \textbf{ in } G\}}$ spezifiziert.

⟨*transpuma1*⟩+≡
```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new37xExpr(_,_,_,_,_,_,_,_,_,
      new4xFormer(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId))),_),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new2xLValue(_,_,_,_,_,_,_,_,_,
            new3xLValue(_,_,_,_,_,_,_,_,
              No1:new5xLValue(_,_,_,_,_,_,_,_,qual3:new2xQualId),_,
              No2:new5xLValue(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        Expr1:new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId),
        new11xBinOp,
        Expr2:new38xExpr(_,_,_,_,_,_,_,_,qual7:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual2, qual4) == 1
               && EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual6) == 1)
            {
               variable = getA13variable(qual1, qual2, NoTree, qual5, qual7);
               transatt = A13;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A13)
           {
```

```
            result = checkA13variable(qual1, qual2, NoTree, qual5, qual7,
                                      variable);
            if(result == 1)
            {
               expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                           NoPosition,
                           variable->newlsidename,
                           variable->newlength))));
            }
         }
         else
         {
            result = 0;
         }
         break;
      default \: result = 0;
  }
};.
```

Hier wird das Muster $\boxed{\{k : k \textbf{ in } F\{q\} \mid k \textbf{ in } G\}}$ spezifiziert.

⟨*transpuma1*⟩+≡
```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
          new44xExpr(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId),
            new2xSelector(_,_,_,_,_,_,_,_,_,
              new2xExprList(_,_,_,_,_,_,_,_,
                new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_)))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        Expr1:new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId),
        new11xBinOp,
        Expr2:new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual1, qual5) == 1)
            {
               variable = getA13variable(NoTree, qual1, qual4, qual3, qual6);
               transatt = A14;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A14)
           {
              result = checkA13variable(NoTree, qual1,qual4 , qual3, qual6,
                                        variable);
              if(result == 1)
              {
```

```
                expr = mnew44xExpr(mnew38xExpr(mnew2xQualId(mxId(mid(
                             NoPosition,variable->newlsidename,
                             variable->newlength)))),
                        mnew2xSelector(NoPosition,mnew2xExprList(mnew38xExpr
                             (qual4)),NoPosition));
              }
           }
           else
           {
              result = 0;
           }
           break;
       default \: result = 0;
  }
};.
```

Hier wird das Muster $\boxed{\{k : k \textbf{ in } G \mid k \textbf{ in } F\{q\}\}}$ spezifiziert.

⟨*transpuma1*⟩+≡
```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        Expr1:new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
        new11xBinOp,
        Expr2:new44xExpr(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId),
          new2xSelector(_,_,_,_,_,_,_,_,_,
            new2xExprList(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId)),_))))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual1, qual4) == 1)
            {
               variable = getA13variable(NoTree, qual1, qual6, qual5, qual3);
               transatt = A15;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A15)
           {
              result = checkA13variable(NoTree, qual1, qual6, qual5, qual3,
                                        variable);
              if(result == 1)
              {
```

```
              expr = mnew44xExpr(mnew38xExpr(mnew2xQualId(mxId(mid(
                           NoPosition,variable->newlsidename,
                           variable->newlength)))),
                      mnew2xSelector(NoPosition,mnew2xExprList(mnew38xExpr
                           (qual6)),NoPosition));
            }
         }
         else
         {
            result = 0;
         }
         break;
     default \: result = 0;
  }
};.
```

Hier wird das Muster $\boxed{\{[j,k] : [j,k] \textbf{ in } F \mid k \textbf{ notin } G\}}$ spezifiziert.

$\langle$*transpuma1*$\rangle+\equiv$

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new37xExpr(_,_,_,_,_,_,_,_,_,
      new4xFormer(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId))),_),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new2xLValue(_,_,_,_,_,_,_,_,_,
            new3xLValue(_,_,_,_,_,_,_,_,
              No1:new5xLValue(_,_,_,_,_,_,_,_,qual3:new2xQualId),_,
              No2:new5xLValue(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        Expr1:new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId),
        new12xBinOp,
        Expr2:new38xExpr(_,_,_,_,_,_,_,_,qual7:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual2, qual4) == 1
               && EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual6) == 1)
            {
               variable = getA13variable(qual1, qual2, NoTree, qual5, qual7);
               transatt = A16;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A16)
           {
```

```
            result = checkA13variable(qual1, qual2, NoTree, qual5, qual7,
                                      variable);
            if(result == 1)
            {
               expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                           NoPosition,
                           variable->newlsidename,
                           variable->newlength))));
            }
         }
         else
         {
            result = 0;
         }
         break;
      default \: result = 0;
   }
 };.
```

Hier wird das Muster $\boxed{\{k : k \textbf{ in } F\{q\} \mid k \textbf{ notin } G\}}$ spezifiziert.

⟨*transpuma1*⟩+≡
```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
          new44xExpr(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId),
            new2xSelector(_,_,_,_,_,_,_,_,_,
              new2xExprList(_,_,_,_,_,_,_,_,
                new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_)))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        Expr1:new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId),
        new12xBinOp,
        Expr2:new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual1, qual5) == 1)
            {
               variable = getA13variable(NoTree, qual1, qual4, qual3, qual6);
               transatt = A17;
               result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
       case helpstmt \: /* wird in transhelp2 benutzt */
            transatt = noset;
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
      case delete \: /* wird von Delete aufgerufen */
           if(transatt == A17)
           {
              result = checkA13variable(NoTree, qual1,qual4 , qual3, qual6,
                                        variable);
              if(result == 1)
              {
```

```
              expr = mnew44xExpr(mnew38xExpr(mnew2xQualId(mxId(mid(
                           NoPosition,variable->newlsidename,
                           variable->newlength)))),
                     mnew2xSelector(NoPosition,mnew2xExprList(mnew38xExpr
                           (qual4)),NoPosition));
            }
         }
         else
         {
            result = 0;
         }
         break;
     default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{V := \{j : j \textbf{ in } F \mid G(j) \textbf{ in } H\}}$.

$\langle$*transpuma1*$\rangle{+}{\equiv}$

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
   new3xFormer(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
      new1xIterator(_,_,_,_,_,_,_,_,
         new2xSimpleIts(_,_,_,_,_,_,_,_,
            new1xSimpleIt(_,_,_,_,_,_,_,_,
               new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
               new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
         new50xExpr(_,_,_,_,_,_,_,_,
            new44xExpr(_,_,_,_,_,_,_,_,
               new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
               new1xSelector(_,_,_,_,_,_,_,_,_,
                  new2xExprList(_,_,_,_,_,_,_,_,
                     new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)),_)),
            new11xBinOp,
            new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
          if(EqualQualId(qual1, qual2) == 1
             && EqualQualId(qual1, qual5) == 1)
          {
              variable = getA18variable(qual1, qual3, qual4, qual6);
              transatt = A18;
              result = 1;
          }
          else
          {
             variable = NULL;
             transatt = noset;
          }
          break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A18)
            {
                result = checkA18variable(qual1, qual3, qual4, qual6, variable);
                if(result == 1)
                {
                   expr = mnew38xExpr(mnew2xQualId(mxId(mid(
```

```
                    NoPosition, variable->newlsidename,
                    variable->newlength))));
               }
           }
           else result = 0;
           break;
      default \: result = 0;
   }
 };.
```

Hier folgt das Muster $\boxed{V := \{j : j \textbf{ in } F \mid G(j) \textbf{ notin } H\}}$.

$\langle$*transpuma1*$\rangle{+}\equiv$

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
   new3xFormer(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
      new1xIterator(_,_,_,_,_,_,_,_,
         new2xSimpleIts(_,_,_,_,_,_,_,_,
            new1xSimpleIt(_,_,_,_,_,_,_,_,
               new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
               new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
         new50xExpr(_,_,_,_,_,_,_,_,
            new44xExpr(_,_,_,_,_,_,_,_,
               new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
               new1xSelector(_,_,_,_,_,_,_,_,
                  new2xExprList(_,_,_,_,_,_,_,_,
                     new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId)),_)),
            new12xBinOp,
            new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
          if(EqualQualId(qual1, qual2) == 1
             && EqualQualId(qual1, qual5) == 1)
          {
              variable = getA18variable(qual1, qual3, qual4, qual6);
              transatt = A19;
              result = 1;
          }
          else
          {
             variable = NULL;
             transatt = noset;
          }
          break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A19)
            {
                result = checkA18variable(qual1, qual3, qual4, qual6, variable);
                if(result == 1)
                {
                   expr = mnew38xExpr(mnew2xQualId(mxId(mid(
```

```
                    NoPosition, variable->newlsidename,
                    variable->newlength))));
               }
           }
           else result = 0;
           break;
     default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{V := \{[j,k] : [j,k]\ \mathbf{in}\ F \mid G(j)\ \mathbf{in}\ H\}}$.

⟨*transpuma1*⟩+≡

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new37xExpr(_,_,_,_,_,_,_,_,_,
      new4xFormer(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId))),_),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new2xLValue(_,_,_,_,_,_,_,_,_,
            new3xLValue(_,_,_,_,_,_,_,_,
              No1:new5xLValue(_,_,_,_,_,_,_,_,qual3:new2xQualId),_,
              No2:new5xLValue(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        Expr1:new44xExpr(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId),
          new1xSelector(_,_,_,_,_,_,_,_,_,
            new2xExprList(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual7:new2xQualId)),_)),
        new11xBinOp,
        Expr2:new38xExpr(_,_,_,_,_,_,_,_,qual8:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
          if(EqualQualId(qual1, qual3) == 1
             && EqualQualId(qual2, qual4) == 1
             && EqualQualId(qual2, qual7) == 1)
          {
              variable = getA20variable(qual1, qual2, NoTree, qual5, qual6,
                                        qual8);
              transatt = A20;
              result = 1;
          }
          else
          {
             variable = NULL;
             transatt = noset;
          }
          break;
       case helpstmt \:
            transatt = noset;
```

```
          result = 0;
          break;
     case delete \:
          if(transatt == A20)
          {
              result = checkA20variable(qual1, qual2, NoTree, qual5, qual6,
                                         qual8, variable);
              if(result == 1)
              {
                 expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                   NoPosition, variable->newlsidename,
                   variable->newlength))));
              }
          }
          else result = 0;
          break;
     default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{V\{q\} := \{k : k \textbf{ in } F\{q\} \mid G(k) \textbf{ in } H\}}$.

⟨*transpuma1*⟩+≡

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
          new44xExpr(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId),
            new2xSelector(_,_,_,_,_,_,_,_,_,
              new2xExprList(_,_,_,_,_,_,_,_,
                new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_)))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        Expr1:new44xExpr(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId),
          new1xSelector(_,_,_,_,_,_,_,_,_,
            new2xExprList(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId)),_)),
        new11xBinOp,
        Expr2:new38xExpr(_,_,_,_,_,_,_,_,qual7:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
          if(EqualQualId(qual1, qual2) == 1
             && EqualQualId(qual1, qual6) == 1)
          {
              variable = getA20variable(NoTree, qual1, qual4, qual3, qual5,
                                        qual7);
              transatt = A21;
              result = 1;
          }
          else
          {
             variable = NULL;
             transatt = noset;
          }
          break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A21)
```

```
          {
              result = checkA20variable(NoTree, qual1, qual4, qual3, qual5,
                                         qual7, variable);
              if(result == 1)
              {
                 expr = mnew44xExpr(mnew38xExpr(mnew2xQualId(mxId(mid(
                          NoPosition,variable->newlsidename,
                          variable->newlength)))),
                       mnew2xSelector(NoPosition,mnew2xExprList(mnew38xExpr
                          (qual4)),NoPosition));
              }
          }
          else result = 0;
          break;
     default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{V := \{[j,k] : [j,k] \textbf{ in } F \mid G(j) \textbf{ notin } H\}}$.

⟨*transpuma1*⟩+≡

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new37xExpr(_,_,_,_,_,_,_,_,_,
      new4xFormer(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId))),_),_,
    new1xIterator(_,_,_,_,_,_,_,_,
      new2xSimpleIts(_,_,_,_,_,_,_,_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new2xLValue(_,_,_,_,_,_,_,_,_,
            new3xLValue(_,_,_,_,_,_,_,_,
              No1:new5xLValue(_,_,_,_,_,_,_,_,qual3:new2xQualId),_,
              No2:new5xLValue(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_),_,
          new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId))),_,
      new50xExpr(_,_,_,_,_,_,_,_,
        Expr1:new44xExpr(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId),
          new1xSelector(_,_,_,_,_,_,_,_,_,
            new2xExprList(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual7:new2xQualId)),_)),
        new12xBinOp,
        Expr2:new38xExpr(_,_,_,_,_,_,_,_,qual8:new2xQualId)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
          if(EqualQualId(qual1, qual3) == 1
             && EqualQualId(qual2, qual4) == 1
             && EqualQualId(qual2, qual7) == 1)
          {
              variable = getA20variable(qual1, qual2, NoTree, qual5, qual6,
                                        qual8);
              transatt = A22;
              result = 1;
          }
          else
          {
             variable = NULL;
             transatt = noset;
          }
          break;
       case helpstmt \:
            transatt = noset;
```

```
            result = 0;
            break;
        case delete \:
            if(transatt == A22)
            {
                result = checkA20variable(qual1, qual2, NoTree, qual5, qual6,
                                          qual8, variable);
                if(result == 1)
                {
                   expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                     NoPosition, variable->newlsidename,
                     variable->newlength))));
                }
            }
            else result = 0;
            break;
        default \: result = 0;
   }
 };.
```

Hier folgt das Muster $\boxed{V\{q\} := \{k : k \textbf{ in } F\{q\} \mid G(k) \textbf{ notin } H\}}$.

$\langle$*transpuma1*$\rangle$+≡

```
  expr:new36xExpr(_,_,_,_,_,_,_,_,_,
    new3xFormer(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
      new1xIterator(_,_,_,_,_,_,_,_,
        new2xSimpleIts(_,_,_,_,_,_,_,_,
          new1xSimpleIt(_,_,_,_,_,_,_,_,
            new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
            new44xExpr(_,_,_,_,_,_,_,_,
              new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId),
              new2xSelector(_,_,_,_,_,_,_,_,_,
                new2xExprList(_,_,_,_,_,_,_,_,
                  new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_)))),_,
        new50xExpr(_,_,_,_,_,_,_,_,
          Expr1:new44xExpr(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId),
            new1xSelector(_,_,_,_,_,_,_,_,_,
              new2xExprList(_,_,_,_,_,_,_,_,
                new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId)),_)),
          new12xBinOp,
          Expr2:new38xExpr(_,_,_,_,_,_,_,_,qual7:new2xQualId)))),_),_,_,_
  RETURN result;
  ? { switch(funcflag)
      {
         case code \:
            if(EqualQualId(qual1, qual2) == 1
               && EqualQualId(qual1, qual6) == 1)
            {
                variable = getA20variable(NoTree, qual1, qual4, qual3, qual5,
                                          qual7);
                transatt = A23;
                result = 1;
            }
            else
            {
               variable = NULL;
               transatt = noset;
            }
            break;
         case helpstmt \:
              transatt = noset;
              result = 0;
              break;
         case delete \:
              if(transatt == A23)
```

```
          {
              result = checkA20variable(NoTree, qual1, qual4, qual3, qual5,
                                         qual7, variable);
              if(result == 1)
              {
                 expr = mnew44xExpr(mnew38xExpr(mnew2xQualId(mxId(mid(
                          NoPosition,variable->newlsidename,
                          variable->newlength)))),
                       mnew2xSelector(NoPosition,mnew2xExprList(mnew38xExpr
                          (qual4)),NoPosition));
              }
          }
          else result = 0;
          break;
     default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{V := \{[j,k] : k \textbf{ in } F, j \textbf{ in } G\{k\}\}}$.

$\langle$*transpuma1*$\rangle{+}{\equiv}$

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new37xExpr(_,_,_,_,_,_,_,_,_,
      new4xFormer(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId))),_),_,
    new2xIterator(_,_,_,_,_,_,_,_,
      new1xSimpleIts(_,_,_,_,_,_,_,_,
        new2xSimpleIts(_,_,_,_,_,_,_,_,
          new1xSimpleIt(_,_,_,_,_,_,_,_,
            new5xLValue(_,_,_,_,_,_,_,_,qual3:new2xQualId),_,
            new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId))),_,
        new1xSimpleIt(_,_,_,_,_,_,_,_,
          new5xLValue(_,_,_,_,_,_,_,_,qual5:new2xQualId),_,
          new44xExpr(_,_,_,_,_,_,_,_,
            new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId),
            new2xSelector(_,_,_,_,_,_,_,_,_,
              new2xExprList(_,_,_,_,_,_,_,_,
                new38xExpr(_,_,_,_,_,_,_,_,qual7:new2xQualId)),_)))))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
          if(EqualQualId(qual1, qual5) == 1
             && EqualQualId(qual2, qual3) == 1
             && EqualQualId(qual2, qual7) == 1)
          {
              variable = getA13variable(qual1, qual2, NoTree, qual4, qual6);
              transatt = A24;
              result = 1;
          }
          else
          {
             variable = NULL;
             transatt = noset;
          }
          break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A24)
```

```
          {
              result = checkA13variable(qual1, qual2, NoTree, qual4, qual6,
                                         variable);
              if(result == 1)
              {
                 expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                   NoPosition, variable->newlsidename,
                   variable->newlength))));
              }
          }
          else result = 0;
          break;
     default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{V\{q\} := \{k : k \textbf{ in } F, q \textbf{ in } G\{k\}\}}$.

⟨*transpuma1*⟩+≡

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
      new2xIterator(_,_,_,_,_,_,_,_,
        new1xSimpleIts(_,_,_,_,_,_,_,_,
          new2xSimpleIts(_,_,_,_,_,_,_,_,
            new1xSimpleIt(_,_,_,_,_,_,_,_,
              new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
              new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
          new1xSimpleIt(_,_,_,_,_,_,_,_,
            new5xLValue(_,_,_,_,_,_,_,_,qual4:new2xQualId),_,
              new44xExpr(_,_,_,_,_,_,_,_,
                new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId),
                  new2xSelector(_,_,_,_,_,_,_,_,_,
                    new2xExprList(_,_,_,_,_,_,_,_,
                      new38xExpr(_,_,_,_,_,_,_,_,qual6:new2xQualId))
                    ,_)))))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
          if(EqualQualId(qual1, qual2) == 1
             && EqualQualId(qual1, qual6) == 1)
          {
              variable = getA13variable(NoTree, qual1, qual4, qual3, qual5);
              transatt = A25;
              result = 1;
          }
          else
          {
             variable = NULL;
             transatt = noset;
          }
          break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A24)
            {
                result = checkA13variable(NoTree, qual1, qual4, qual3, qual5,
                                          variable);
                if(result == 1)
```

```
              {
                 expr = mnew44xExpr(mnew38xExpr(mnew2xQualId(mxId(mid(
                          NoPosition,variable->newlsidename,
                          variable->newlength)))),
                     mnew2xSelector(NoPosition,mnew2xExprList(mnew38xExpr
                          (qual6)),NoPosition));
              }
          }
          else result = 0;
          break;
     default \: result = 0;
  }
};.
```

Hier folgt das Muster $\boxed{V := \{k : k \textbf{ in } F \mid k \textbf{ operator1 } int1 \textbf{ operator2 } int2\}}$.

$\langle$*transpuma1*$\rangle{+}\equiv$

```
expr:new36xExpr(_,_,_,_,_,_,_,_,_,
  new3xFormer(_,_,_,_,_,_,_,_,
    new38xExpr(_,_,_,_,_,_,_,_,qual1:new2xQualId),_,
      new1xIterator(_,_,_,_,_,_,_,_,
        new2xSimpleIts(_,_,_,_,_,_,_,_,
          new1xSimpleIt(_,_,_,_,_,_,_,_,
            new5xLValue(_,_,_,_,_,_,_,_,qual2:new2xQualId),_,
              new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId))),_,
          new50xExpr(_,_,_,_,_,_,_,_,
            Expr1:new50xExpr(_,_,_,_,_,_,_,_,
              Expr3:new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId),
              op1:xBinOp,
              Expr4:new13xExpr(_,_,_,_,_,_,_,_,int1:intlit)),
            op2:xBinOp,
            Expr2:new13xExpr(_,_,_,_,_,_,_,_,int2:intlit)))),_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
          if(EqualQualId(qual1, qual2) == 1
             && EqualQualId(qual1, qual4) == 1)
          {
              variable = getA26variable(qual1, qual3, op1, op2,
                                        (int1->intlit).Integer,
                                        (int2->intlit).Integer);
              transatt = A26;
              result = 1;
          }
          else
          {
             variable = NULL;
             transatt = noset;
          }
          break;
       case helpstmt \:
            transatt = noset;
            result = 0;
            break;
       case delete \:
            if(transatt == A26)
            {
                result = checkA26variable(qual1, qual3, op1, op2,
                                        (int1->intlit).Integer,
                                        (int2->intlit).Integer, variable);
```

```
            if(result == 1)
            {
               expr = mnew38xExpr(mnew2xQualId(mxId(mid(
                 NoPosition, variable->newlsidename,
                 variable->newlength))));
            }
        }
        else result = 0;
        break;
    default \: result = 0;
  }
};.
```

Nun beginnen die Modifikationsmuster (dx_i). Bevor Transformationscode in die Ausgabedatei geschrieben werden kann, müssen line-Direktiven ausgegeben werden.

Als Erstes wird das Transformationsmuster `with :=` spezifiziert. Im Falle `helpstmt` wird überprüft, ob diese Anweisung eine „Gültige“ ist und im Falle `preder` wird die entsprechende Ableitung in die Ausgabedatei geschrieben.

⟨*transpuma1*⟩+≡

```
new29xStmt(_,_,_,_,_,_,_,_,
  new5xLValue(_,_,_,_,_,_,_,_,
    qual1:new2xQualId(_,_,_,_,_,_,_,_,
      xId(id(_,Ident,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_))),
  new23xBinOp,_,
  new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_,_,_
RETURN result; /* with := */
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A3 && Ident != variable->A.a3.r1sidename)
            || (transatt == A4 && Ident != variable->A.a4.r2sidename)
            || (transatt == A5 && Ident != variable->A.a4.r2sidename)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9 && Ident != variable->A.a1.r1sidename)
            || (transatt == A11 && Ident != variable->A.a3.r1sidename)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18 && Ident != variable->A.a18.r1sidename)
            || (transatt == A19 && Ident != variable->A.a18.r1sidename)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24 && Ident != variable->A.a13.r1sidename)
            || (transatt == A25 && Ident != variable->A.a13.r1sidename)
            || (transatt == A26 && Ident != variable->A.a26.rsidename))
            {
               transatt = noset;
            }
            result = 0;
            break;
       case preder \:
            if(transatt == A1)
            {
```

```
  if(Ident == variable->A.a1.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA1r1with(qual2, Ident, variable);
     result = 1;
  }
  else if(Ident == variable->A.a1.r2sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA1r2with(qual2, Ident, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A2)
{
  if(Ident == variable->A.a1.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA2r1with(qual2, Ident, variable);
     result = 1;
  }
  else if(Ident == variable->A.a1.r2sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA2r2with(qual2, Ident, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A3)
{
  if(Ident == variable->A.a3.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA3with(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
```

```
  }
}
else if(transatt == A4)
{
  if(Ident == variable->A.a4.r2sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA4with(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A5)
{
  if(Ident == variable->A.a4.r2sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA4with(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A6)
{
  if(Ident == variable->A.a6.rsidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA6with(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A9)
{
  if(Ident == variable->A.a1.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA9with(qual2, variable);
```

```
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A10)
  {
    if(Ident == variable->A.a10.rsidename)
    {
      lineprint();fprintf(ofile,"\n");
      insertA10with(qual2, variable);
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A11)
  {
    if(Ident == variable->A.a3.r1sidename)
    {
      lineprint();fprintf(ofile,"\n");
      insertA11with(qual2, variable);
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A18)
  {
    if(Ident == variable->A.a18.r1sidename)
    {
      lineprint();fprintf(ofile,"\n");
      insertA18with(qual2, variable);
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A19)
```

```
{
  if(Ident == variable->A.a18.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA19with(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A24)
{
  if(Ident == variable->A.a13.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA24r1sidewith(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A25)
{
  if(Ident == variable->A.a13.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA24r1sidewith(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A26)
{
  if(Ident == variable->A.a26.rsidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA26rsidewith(qual2, variable);
     result = 1;
  }
  else
```

```
            {
               result = 0;
            }
          }
          else
          {
            result = 0;
          }
          break;
      default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationmuster $\boxed{\texttt{less :=}}$ spezifiziert. Der Rest ist wie gehabt.

⟨*transpuma1*⟩+≡

```
new29xStmt(_,_,_,_,_,_,_,_,
  new5xLValue(_,_,_,_,_,_,_,_,
    qual1:new2xQualId(_,_,_,_,_,_,_,_,
      xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
  new24xBinOp,_,
  new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_,_,_
RETURN result; /* less := */
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A3 && Ident != variable->A.a3.r1sidename)
            || (transatt == A4 && Ident != variable->A.a4.r2sidename)
            || (transatt == A5 && Ident != variable->A.a4.r2sidename)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9 && Ident != variable->A.a1.r1sidename)
            || (transatt == A11 && Ident != variable->A.a3.r1sidename)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18 && Ident != variable->A.a18.r1sidename)
            || (transatt == A19 && Ident != variable->A.a18.r1sidename)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24 && Ident != variable->A.a13.r1sidename)
            || (transatt == A25 && Ident != variable->A.a13.r1sidename)
            || (transatt == A26 && Ident != variable->A.a26.rsidename))
            {
               transatt = noset;
            }
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
       case preder \:
            if(transatt == A1)
            {
              if(Ident == variable->A.a1.r1sidename)
              {
                 lineprint();fprintf(ofile,"\n");
                 insertA1r1less(qual2, Ident, variable);
                 result = 1;
```

```
    }
    else if(Ident == variable->A.a1.r2sidename)
    {
       lineprint();fprintf(ofile,"\n");
       insertA1r2less(qual2, Ident, variable);
       result = 1;
    }
    else
    {
       result = 0;
    }
  }
  else if(transatt == A2)
  {
    if(Ident == variable->A.a1.r1sidename)
    {
       lineprint();fprintf(ofile,"\n");
       insertA2r1less(qual2, Ident, variable);
       result = 1;
    }
    else if(Ident == variable->A.a1.r2sidename)
    {
       lineprint();fprintf(ofile,"\n");
       insertA2r2less(qual2, Ident, variable);
       result = 1;
    }
    else
    {
       result = 0;
    }
  }
  else if(transatt == A3)
  {
    if(Ident == variable->A.a3.r1sidename)
    {
       lineprint();fprintf(ofile,"\n");
       insertA3less(qual2, variable);
       result = 1;
    }
    else
    {
       result = 0;
    }
  }
  else if(transatt == A4)
  {
    if(Ident == variable->A.a4.r2sidename)
```

```
  {
     lineprint();fprintf(ofile,"\n");
     insertA4less(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A5)
{
  if(Ident == variable->A.a4.r2sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA4less(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A6)
{
  if(Ident == variable->A.a6.rsidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA6less(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A9)
{
  if(Ident == variable->A.a1.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA9less(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
```

```
  }
}
else if(transatt == A10)
{
  if(Ident == variable->A.a10.rsidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA10less(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A11)
{
  if(Ident == variable->A.a3.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA11less(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A18)
{
  if(Ident == variable->A.a18.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA18less(qual2, variable);
     result = 1;
  }
  else
  {
     result = 0;
  }
}
else if(transatt == A19)
{
  if(Ident == variable->A.a18.r1sidename)
  {
     lineprint();fprintf(ofile,"\n");
     insertA19less(qual2, variable);
```

```
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A24)
  {
    if(Ident == variable->A.a13.r1sidename)
    {
      lineprint();fprintf(ofile,"\n");
      insertA24r1sideless(qual2, variable);
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A25)
  {
    if(Ident == variable->A.a13.r1sidename)
    {
      lineprint();fprintf(ofile,"\n");
      insertA24r1sideless(qual2, variable);
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A26)
  {
    if(Ident == variable->A.a26.rsidename)
    {
      lineprint();fprintf(ofile,"\n");
      insertA26rsideless(qual2, variable);
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else
```

```
          {
            result = 0;
          }
          break;
      default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `G(const1) := G(const1) + const2;` erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,
      qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new13xExpr(_,_,_,_,_,_,_,_,
          int1:intlit)),_)),_,
    new50xExpr(_,_,_,_,_,_,_,_,
      new44xExpr(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId),
        new1xSelector(_,_,_,_,_,_,_,_,_,
          new2xExprList(_,_,_,_,_,_,_,_,
            new13xExpr(_,_,_,_,_,_,_,_,
              int2:intlit)),_)),
        new31xBinOp,
        new13xExpr(_,_,_,_,_,_,_,_,int3:intlit))),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3 && Ident != variable->A.a3.r2sidename)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11 && Ident != variable->A.a3.r2sidename)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
```

```
          || (transatt == A24)
          || (transatt == A25)
          || (transatt == A26))
          {
             transatt = noset;
          }
          result = 0;  /* wird eigentlich nicht benoetigt */
          break;
     case preder \:
          if(transatt == A3)
          {
            if(EqualQualId(qual1, qual2) == 1
               && Equalintlit(int1, int2) == 1
               && Ident == variable->A.a3.r2sidename)
            {
               lineprint();fprintf(ofile,"\n");
               insertA3plus(int1, int2, variable);
               result = 1;
            }
            else
            {
               result = 0;
            }
          }
          else if(transatt == A11)
          {
            if(EqualQualId(qual1, qual2) == 1
               && Equalintlit(int1, int2) == 1
               && Ident == variable->A.a3.r2sidename)
            {
               lineprint();fprintf(ofile,"\n");
               insertA11plus(int1, int2, variable);
               result = 1;
            }
            else
            {
               result = 0;
            }
          }
          else
          {
            result = 0;
          }
          break;
      default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `G(const1) +:= const2;` erkannt.

⟨*transpuma1*⟩+≡

```
new29xStmt(_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,
      qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new13xExpr(_,_,_,_,_,_,_,_,int1:intlit)),_)),
  new31xBinOp,_,
  new13xExpr(_,_,_,_,_,_,_,_,int2:intlit)),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3 && Ident != variable->A.a3.r2sidename)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11 && Ident != variable->A.a3.r2sidename)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24)
            || (transatt == A25)
            || (transatt == A26))
            {
               transatt = noset;
            }
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
```

```
    case preder \:
         if(transatt == A3)
         {
           if(Ident == variable->A.a3.r2sidename)
           {
              lineprint();fprintf(ofile,"\n");
              insertA3plus(int1, int2, variable);
              result = 1;
           }
           else
           {
              result = 0;
           }
         }
         else if(transatt == A11)
         {
           if(Ident == variable->A.a3.r2sidename)
           {
              lineprint();fprintf(ofile,"\n");
              insertA11plus(int1, int2, variable);
              result = 1;
           }
           else
           {
              result = 0;
           }
         }
         else
         {
           result = 0;
         }
         break;
     default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster $\boxed{\texttt{G(x) := G(x) + const;}}$ erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,
      qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
  new50xExpr(_,_,_,_,_,_,_,_,
    new44xExpr(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId),
      new1xSelector(_,_,_,_,_,_,_,_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_)),
    new31xBinOp,
    new13xExpr(_,_,_,_,_,_,_,_,intlit(_,Integer,_,_,_,_,_,_,_,_,_))))
,_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3 && Ident != variable->A.a3.r2sidename)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11 && Ident != variable->A.a3.r2sidename)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24)
```

```
          || (transatt == A25)
          || (transatt == A26))
          {
             transatt = noset;
          }
          result = 0;  /* wird eigentlich nicht benoetigt */
          break;
     case preder \:
          if(transatt == A3)
          {
            if( EqualQualId(qual1, qual3) == 1
                && EqualQualId(qual2, qual4) == 1
                && Ident == variable->A.a3.r2sidename)
            {
               lineprint();fprintf(ofile,"\n");
               insertA3plus2(qual2, Integer, variable);
               result = 1;
            }
            else
            {
               result = 0;
            }
          }
          else if(transatt == A11)
          {
            if( EqualQualId(qual1, qual3) == 1
                && EqualQualId(qual2, qual4) == 1
                && Ident == variable->A.a3.r2sidename)
            {
               lineprint();fprintf(ofile,"\n");
               insertA11plus2(qual2, Integer, variable);
               result = 1;
            }
            else
            {
               result = 0;
            }
          }
          else
          {
            result = 0;
          }
          break;
      default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `G(const1) := G(const1) - const2;` erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,
      qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new13xExpr(_,_,_,_,_,_,_,_,
          int1:intlit)),_)),_,
  new50xExpr(_,_,_,_,_,_,_,_,
    new44xExpr(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId),
      new1xSelector(_,_,_,_,_,_,_,_,_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new13xExpr(_,_,_,_,_,_,_,_,
            int2:intlit)),_)),
    new32xBinOp,
    new13xExpr(_,_,_,_,_,_,_,_,int3:intlit))),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3 && Ident != variable->A.a3.r2sidename)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11 && Ident != variable->A.a3.r2sidename)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
```

```
        || (transatt == A23)
        || (transatt == A24)
        || (transatt == A25)
        || (transatt == A26))
        {
           transatt = noset;
        }
        result = 0;  /* wird eigentlich nicht benoetigt */
        break;
   case preder \:
        if(transatt == A3)
        {
          if( EqualQualId(qual1, qual2) == 1
              && Equalintlit(int1, int2) == 1
              && Ident == variable->A.a3.r2sidename)
          {
             lineprint();fprintf(ofile,"\n");
             insertA3minus(int1, int2, variable);
             result = 1;
          }
          else
          {
             result = 0;
          }
        }
        else if(transatt == A11)
        {
          if( EqualQualId(qual1, qual2) == 1
              && Equalintlit(int1, int2) == 1
              && Ident == variable->A.a3.r2sidename)
          {
             lineprint();fprintf(ofile,"\n");
             insertA11minus(int1, int2, variable);
             result = 1;
          }
          else
          {
             result = 0;
          }
        }
        else
        {
          result = 0;
        }
        break;
    default \: result = 0;
}
```

```
};.
```

Hier wird nun das Modifikationsmuster `G(x) -:= const;` erkannt.

⟨*transpuma1*⟩+≡

```
new29xStmt(_,_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,
      qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new13xExpr(_,_,_,_,_,_,_,_,
          int1:intlit)),_)),
  new32xBinOp,_,
  new13xExpr(_,_,_,_,_,_,_,_,int2:intlit)),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3 && Ident != variable->A.a3.r2sidename)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11 && Ident != variable->A.a3.r2sidename)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24)
            || (transatt == A25)
            || (transatt == A26))
            {
               transatt = noset;
            }
            result = 0;  /* wird eigentlich nicht benoetigt */
```

```
          break;
     case preder \:
          if(transatt == A3)
          {
            if(Ident == variable->A.a3.r2sidename)
            {
               lineprint();fprintf(ofile,"\n");
               insertA3minus(int1, int2, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
          else if(transatt == A11)
          {
            if(Ident == variable->A.a3.r2sidename)
            {
               lineprint();fprintf(ofile,"\n");
               insertA11minus(int1, int2, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
          else
          {
            result = 0;
          }
          break;
      default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `G(x) := G(x) - const;` erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,
      qual1:new2xQualId(_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
  new50xExpr(_,_,_,_,_,_,_,
    new44xExpr(_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId),
      new1xSelector(_,_,_,_,_,_,_,_,
        new2xExprList(_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_)),
    new32xBinOp,
    new13xExpr(_,_,_,_,_,_,_,_,intlit(_,Integer,_,_,_,_,_,_,_,_,_))))
,_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3 && Ident != variable->A.a3.r2sidename)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11 && Ident != variable->A.a3.r2sidename)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24)
```

```
          || (transatt == A25)
          || (transatt == A26))
          {
             transatt = noset;
          }
          result = 0;  /* wird eigentlich nicht benoetigt */
          break;
     case preder \:
          if(transatt == A3)
          {
            if( EqualQualId(qual1, qual3) == 1
                && EqualQualId(qual2, qual4) == 1
                && Ident == variable->A.a3.r2sidename)
            {
               lineprint();fprintf(ofile,"\n");
               insertA3minus2(qual2, Integer, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
          else if(transatt == A11)
          {
            if( EqualQualId(qual1, qual3) == 1
                && EqualQualId(qual2, qual4) == 1
                && Ident == variable->A.a3.r2sidename)
            {
               lineprint();fprintf(ofile,"\n");
               insertA11minus2(qual2, Integer, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
          else
          {
            result = 0;
          }
          break;
      default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `F(x) := om;` erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
  new17xExpr),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3)
            || (transatt == A4 && Ident != variable->A.a4.r1sidename)
            || (transatt == A5 && Ident != variable->A.a4.r1sidename)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18 && Ident != variable->A.a18.r2sidename)
            || (transatt == A19 && Ident != variable->A.a18.r2sidename)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24)
            || (transatt == A25)
            || (transatt == A26))
            {
               transatt = noset;
            }
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
       case preder \:
            if(transatt == A4 || transatt == A5)
```

```
        {
          lineprint();fprintf(ofile,"\n");
          insertA4ypredef(qual2, variable);
          result = 2;
        }
        else if(transatt == A18)
        {
          lineprint();fprintf(ofile,"\n");
          insertA18ypredef(qual2, variable);
          result = 2;
        }
        else if(transatt == A19)
        {
          lineprint();fprintf(ofile,"\n");
          insertA19ypredef(qual2, variable);
          result = 2;
        }
        else
        {
          result = 0;
        }
        break;
   case postder \:
        if(transatt == A4 || transatt == A5)
        {
          insertA4ypostdef(qual2, variable);
          result = 0; /* wird nicht benoetigt */
        }
        else if(transatt == A18)
        {
          insertA18ypostdef(qual2, variable);
          result = 0; /* wird nicht benoetigt */
        }
        else if(transatt == A19)
        {
          insertA19ypostdef(qual2, variable);
          result = 0; /* wird nicht benoetigt */
        }
    default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `F(x) := y;` erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
   new38xExpr(_,_,_,_,_,_,_,qual3:new2xQualId)),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3)
            || (transatt == A4 && Ident != variable->A.a4.r1sidename)
            || (transatt == A5 && Ident != variable->A.a4.r1sidename)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18 && Ident != variable->A.a18.r2sidename)
            || (transatt == A19 && Ident != variable->A.a18.r2sidename)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24)
            || (transatt == A25)
            || (transatt == A26))
            {
               transatt = noset;
            }
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
       case preder \:
            if(transatt == A4 || transatt == A5)
```

```
          {
            lineprint();fprintf(ofile,"\n");
            insertA4ypredef(qual2, variable);
            result = 2;
          }
          else if(transatt == A18)
          {
            lineprint();fprintf(ofile,"\n");
            insertA18ypredef(qual2, variable);
            result = 2;
          }
          else if(transatt == A19)
          {
            lineprint();fprintf(ofile,"\n");
            insertA19ypredef(qual2, variable);
            result = 2;
          }
          else
          {
            result = 0;
          }
          break;
     case postder \:
          if(transatt == A4 || transatt == A5)
          {
            insertA4ypostdef(qual2, variable);
            result = 0; /* wird nicht benoetigt */
          }
          else if(transatt == A18)
          {
            insertA18ypostdef(qual2, variable);
            result = 0; /* wird nicht benoetigt */
          }
          else if(transatt == A19)
          {
            insertA19ypostdef(qual2, variable);
            result = 0; /* wird nicht benoetigt */
          }
      default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster $\boxed{\texttt{F(x) := const;}}$ erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
   new13xExpr(_,_,_,_,_,_,_,_,intlit:intlit)),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3)
            || (transatt == A4 && Ident != variable->A.a4.r1sidename)
            || (transatt == A5 && Ident != variable->A.a4.r1sidename)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18 && Ident != variable->A.a18.r2sidename)
            || (transatt == A19 && Ident != variable->A.a18.r2sidename)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24)
            || (transatt == A25)
            || (transatt == A26))
            {
               transatt = noset;
            }
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
       case preder \:
            if(transatt == A4 || transatt == A5)
```

```
          {
            lineprint();fprintf(ofile,"\n");
            insertA4ypredef(qual2, variable);
            result = 2;
          }
          else if(transatt == A18)
          {
            lineprint();fprintf(ofile,"\n");
            insertA18ypredef(qual2, variable);
            result = 2;
          }
          else if(transatt == A19)
          {
            lineprint();fprintf(ofile,"\n");
            insertA19ypredef(qual2, variable);
            result = 2;
          }
          else
          {
            result = 0;
          }
          break;
     case postder \:
          if(transatt == A4 || transatt == A5)
          {
            insertA4ypostdef(qual2, variable);
            result = 0; /* wird nicht benoetigt */
          }
          else if(transatt == A18)
          {
            insertA18ypostdef(qual2, variable);
            result = 0; /* wird nicht benoetigt */
          }
          else if(transatt == A19)
          {
            insertA19ypostdef(qual2, variable);
            result = 0; /* wird nicht benoetigt */
          }
      default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `F{x} := F{x} with y;` erkannt.

⟨transpuma1⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new2xSelector(_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
  new50xExpr(_,_,_,_,_,_,_,_,
    new44xExpr(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId),
      new2xSelector(_,_,_,_,_,_,_,_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_)),
    new23xBinOp,
    new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId))),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11)
            || (transatt == A12 && Ident != variable->A.a12.r2sidename)
            || (transatt == A13 && Ident != variable->A.a13.r1sidename)
            || (transatt == A14 && Ident != variable->A.a13.r1sidename)
            || (transatt == A15 && Ident != variable->A.a13.r1sidename)
            || (transatt == A16 && Ident != variable->A.a13.r1sidename)
            || (transatt == A17 && Ident != variable->A.a13.r1sidename)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20 && Ident != variable->A.a20.r1sidename)
            || (transatt == A21 && Ident != variable->A.a20.r1sidename)
            || (transatt == A22 && Ident != variable->A.a20.r1sidename)
            || (transatt == A23 && Ident != variable->A.a20.r1sidename)
            || (transatt == A24 && Ident != variable->A.a13.r2sidename)
            || (transatt == A25 && Ident != variable->A.a13.r2sidename)
            || (transatt == A26))
            {
```

```
            transatt = noset;
          }
          result = 0;  /* wird eigentlich nicht benoetigt */
          break;
      case preder \:
          if(transatt == A7)
          {
            if(EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual4) == 1)
            {
               lineprint();fprintf(ofile,"\n");
               insertA8with(qual2, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
          else if(transatt == A8)
          {
            if(EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual4) == 1)
            {
               lineprint();fprintf(ofile,"\n");
               insertA8with(qual2, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
          else if(transatt == A12)
          {
            if(EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual4) == 1)
            {
               lineprint();fprintf(ofile,"\n");
               insertA12with(qual2, qual5, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
```

```
else if(transatt == A13)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA13with(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A14)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA13with(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A15)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA13with(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A16)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
```

```
      lineprint();fprintf(ofile,"\n");
      insertA16with(qual2, qual5, variable);
      result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A17)
{
  if(EqualQualId(qual1, qual3) == 1
      && EqualQualId(qual2, qual4) == 1)
  {
      lineprint();fprintf(ofile,"\n");
      insertA16with(qual2, qual5, variable);
      result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A20)
{
  if(EqualQualId(qual1, qual3) == 1
      && EqualQualId(qual2, qual4) == 1)
  {
      lineprint();fprintf(ofile,"\n");
      insertA20with(qual2, qual5, variable);
      result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A21)
{
  if(EqualQualId(qual1, qual3) == 1
      && EqualQualId(qual2, qual4) == 1)
  {
      lineprint();fprintf(ofile,"\n");
      insertA20with(qual2, qual5, variable);
      result = 1;
  }
  else
```

```
  {
    result = 0;
  }
}
else if(transatt == A22)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA22with(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A23)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA22with(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A24 && Ident == variable->A.a13.r2sidename)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA24r2sidewith(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A25 && Ident == variable->A.a13.r2sidename)
```

```
          {
            if(EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual4) == 1)
            {
               lineprint();fprintf(ofile,"\n");
               insertA24r2sidewith(qual2, qual5, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
          else
          {
            result = 0;
          }
          break;
     default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `F{x} := F{x} less y;` erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new2xSelector(_,_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
  new50xExpr(_,_,_,_,_,_,_,_,
    new44xExpr(_,_,_,_,_,_,_,_,
      new38xExpr(_,_,_,_,_,_,_,_,qual3:new2xQualId),
      new2xSelector(_,_,_,_,_,_,_,_,_,
        new2xExprList(_,_,_,_,_,_,_,_,
          new38xExpr(_,_,_,_,_,_,_,_,qual4:new2xQualId)),_)),
    new24xBinOp,
    new38xExpr(_,_,_,_,_,_,_,_,qual5:new2xQualId))),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A9)
            || (transatt == A10)
            || (transatt == A11)
            || (transatt == A12 && Ident != variable->A.a12.r2sidename)
            || (transatt == A13 && Ident != variable->A.a13.r1sidename)
            || (transatt == A14 && Ident != variable->A.a13.r1sidename)
            || (transatt == A15 && Ident != variable->A.a13.r1sidename)
            || (transatt == A16 && Ident != variable->A.a13.r1sidename)
            || (transatt == A17 && Ident != variable->A.a13.r1sidename)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20 && Ident != variable->A.a20.r1sidename)
            || (transatt == A21 && Ident != variable->A.a20.r1sidename)
            || (transatt == A22 && Ident != variable->A.a20.r1sidename)
            || (transatt == A23 && Ident != variable->A.a20.r1sidename)
            || (transatt == A24 && Ident != variable->A.a13.r2sidename)
            || (transatt == A25 && Ident != variable->A.a13.r2sidename)
            || (transatt == A26))
            {
```

```
            transatt = noset;
          }
          result = 0;  /* wird eigentlich nicht benoetigt */
          break;
     case preder \:
          if(transatt == A7)
          {
            if(EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual4) == 1)
            {
               lineprint();fprintf(ofile,"\n");
               insertA8less(qual2, variable);
               result = 1;
                    }
            else
            {
              result = 0;
            }
          }
          else if(transatt == A8)
          {
            if(EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual4) == 1)
            {
               lineprint();fprintf(ofile,"\n");
               insertA8less(qual2, variable);
               result = 1;
                    }
            else
            {
              result = 0;
            }
          }
          else if(transatt == A12)
          {
            if(EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual4) == 1)
            {
               lineprint();fprintf(ofile,"\n");
               insertA12less(qual2, qual5, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
```

```
else if(transatt == A13)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA13less(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A14)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA13less(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A15)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA13less(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A16)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
```

```
      lineprint();fprintf(ofile,"\n");
      insertA16less(qual2, qual5, variable);
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A17)
  {
    if(EqualQualId(qual1, qual3) == 1
       && EqualQualId(qual2, qual4) == 1)
    {
      lineprint();fprintf(ofile,"\n");
      insertA16less(qual2, qual5, variable);
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A20)
  {
    if(EqualQualId(qual1, qual3) == 1
       && EqualQualId(qual2, qual4) == 1)
    {
      lineprint();fprintf(ofile,"\n");
      insertA20less(qual2, qual5, variable);
      result = 1;
    }
    else
    {
      result = 0;
    }
  }
  else if(transatt == A21)
  {
    if(EqualQualId(qual1, qual3) == 1
       && EqualQualId(qual2, qual4) == 1)
    {
      lineprint();fprintf(ofile,"\n");
      insertA20less(qual2, qual5, variable);
      result = 1;
    }
    else
```

```
  {
    result = 0;
  }
}
else if(transatt == A22)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA22less(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A23)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA22less(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A24 && Ident == variable->A.a13.r2sidename)
{
  if(EqualQualId(qual1, qual3) == 1
     && EqualQualId(qual2, qual4) == 1)
  {
     lineprint();fprintf(ofile,"\n");
     insertA24r2sideless(qual2, qual5, variable);
     result = 1;
  }
  else
  {
    result = 0;
  }
}
else if(transatt == A25 && Ident == variable->A.a13.r2sidename)
```

```
          {
            if(EqualQualId(qual1, qual3) == 1
               && EqualQualId(qual2, qual4) == 1)
            {
               lineprint();fprintf(ofile,"\n");
               insertA24r2sideless(qual2, qual5, variable);
               result = 1;
            }
            else
            {
              result = 0;
            }
          }
          else
          {
            result = 0;
          }
          break;
     default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `G(x) := true;` erkannt. erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
  new15xExpr),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9 && Ident != variable->A.a1.r2sidename)
            || (transatt == A10)
            || (transatt == A11)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24)
            || (transatt == A25)
            || (transatt == A26))
            {
               transatt = noset;
            }
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
       case preder \:
            if(transatt == A9 && Ident == variable->A.a1.r2sidename)
```

```
          {
            lineprint();fprintf(ofile,"\n");
            insertA9true(qual2, variable);
            result = 1;
          }
          else
          {
            result = 0;
          }
          break;
     default \: result = 0;
  }
};.
```

Hier wird nun das Modifikationsmuster `G(x) := false;` erkannt. erkannt.

⟨*transpuma1*⟩+≡

```
new28xStmt(_,_,_,_,_,_,_,
  new6xLValue(_,_,_,_,_,_,_,_,
    new5xLValue(_,_,_,_,_,_,_,_,qual1:new2xQualId(_,_,_,_,_,_,_,_,
        xId(id(_,Ident,_,_,_,_,_,_,_,_,_),_,_,_,_,_,_,_,_))),
    new1xSelector(_,_,_,_,_,_,_,_,
      new2xExprList(_,_,_,_,_,_,_,
        new38xExpr(_,_,_,_,_,_,_,_,qual2:new2xQualId)),_)),_,
  new16xExpr),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case helpstmt \:
            if((transatt == A1)
            || (transatt == A2)
            || (transatt == A3)
            || (transatt == A4)
            || (transatt == A5)
            || (transatt == A6)
            || (transatt == A7)
            || (transatt == A8)
            || (transatt == A9 && Ident != variable->A.a1.r2sidename)
            || (transatt == A10)
            || (transatt == A11)
            || (transatt == A12)
            || (transatt == A13)
            || (transatt == A14)
            || (transatt == A15)
            || (transatt == A16)
            || (transatt == A17)
            || (transatt == A18)
            || (transatt == A19)
            || (transatt == A20)
            || (transatt == A21)
            || (transatt == A22)
            || (transatt == A23)
            || (transatt == A24)
            || (transatt == A25)
            || (transatt == A26))
            {
               transatt = noset;
            }
            result = 0;  /* wird eigentlich nicht benoetigt */
            break;
       case preder \:
            if(transatt == A9 && Ident == variable->A.a1.r2sidename)
```

```
              {
                lineprint();fprintf(ofile,"\n");
                insertA9false(qual2, variable);
                result = 1;
              }
              else
              {
                result = 0;
              }
              break;
         default \: result = 0;
      }
    };.
```

Hier folgen nun Grammatikregeln, die Muster enthalten können. Die Teilbäume, die mit einer Marke versehen sind, werden nach applikativen Ausdrücken durchsucht.

⟨*transpuma3*⟩≡
```
new13xStmt(_,_,_,_,_,_,_,_,xStdIO,marke:xActuList),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
           result = Pattern(&marke, &variable, &transatt, funcflag);
           break;
       case helpstmt \:
           transatt = noset;
           break;
       case delete \:
           result = Pattern(&marke, &variable, &transatt, funcflag);
           break;
       case preder \:
           break;
    }
    result = 0;
  };.
new28xStmt(_,_,_,_,_,_,_,_,xLValue,_,expr:xExpr),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
           result = Pattern(&expr, &variable, &transatt, funcflag);
           break;
       case helpstmt \:
           transatt = noset;
           break;
       case delete \:
           result = Pattern(&expr, &variable, &transatt, funcflag);
           break;
       case preder \:
           break;
    }
    result = 0;
  };.
new29xStmt(_,_,_,_,_,_,_,_,xLValue,xBinOp,_,expr:xExpr),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
           result = Pattern(&expr, &variable, &transatt, funcflag);
           break;
       case helpstmt \:
           transatt = noset;
           break;
```

```
       case delete \:
           result = Pattern(&expr, &variable, &transatt, funcflag);
           break;
       case preder \:
           break;
    }
    result = 0;
  };.
new32xStmt(_,_,_,_,_,_,_,_,No1,xFrom,No2),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
           result = Pattern(&No1, &variable, &transatt, funcflag);
           if(result == 0)
           {
              result = Pattern(&No2, &variable, &transatt, funcflag);
           }
           break;
       case helpstmt \:
           transatt = noset;
           break;
       case delete \:
           result = Pattern(&No1, &variable, &transatt, funcflag);
           if(result == 0)
           {
              result = Pattern(&No2, &variable, &transatt, funcflag);
           }
           break;
       case preder \:
           break;
    }
    result = 0;
  };.
new36xExpr(_,_,_,_,_,_,_,_,_,marke:xFormer,_),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
           result = Pattern(&marke, &variable, &transatt, funcflag);
           break;
       case helpstmt \:
           transatt = noset;
           break;
       case delete \:
           result = Pattern(&marke, &variable, &transatt, funcflag);
           break;
```

```
      case preder \:
          break;
    }
    result = 0;
  };.
new50xExpr(_,_,_,_,_,_,_,_,Expr1,xBinOp,Expr2),_,_,_
RETURN result;
? { switch(funcflag)
    {
      case code \:
          result = Pattern(&Expr1, &variable, &transatt, funcflag);
          if(result == 0)
          {
             result = Pattern(&Expr2, &variable, &transatt, funcflag);
          }
          break;
      case helpstmt \:
          transatt = noset;
          break;
      case delete \:
          result = Pattern(&Expr1, &variable, &transatt, funcflag);
          if(result == 0)
          {
             result = Pattern(&Expr2, &variable, &transatt, funcflag);
          }
          break;
      case preder \:
          break;
    }
    result = 0;
  };.
new52xExpr(_,_,_,_,_,_,_,_,xUnOp,expr:xExpr),_,_,_
RETURN result;
? { switch(funcflag)
    {
      case code \:
          result = Pattern(&expr, &variable, &transatt, funcflag);
          break;
      case helpstmt \:
          transatt = noset;
          break;
      case delete \:
          result = Pattern(&expr, &variable, &transatt, funcflag);
          break;
      case preder \:
          break;
    }
```

```
    result = 0;
  };.
new1xExprList(_,_,_,_,_,_,_,_,marke:xExprList,_,expr:xExpr),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
           result = Pattern(&marke, &variable, &transatt, funcflag);
           if(result == 0)
           {
              result = Pattern(&expr, &variable, &transatt, funcflag);
           }
           break;
       case helpstmt \:
           transatt = noset;
           break;
       case delete \:
           result = Pattern(&marke, &variable, &transatt, funcflag);
           if(result == 0)
           {
              result = Pattern(&expr, &variable, &transatt, funcflag);
           }
           break;
       case preder \:
           break;
    }
    result = 0;
  };.
new2xExprList(_,_,_,_,_,_,_,_,expr:xExpr),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
           result = Pattern(&expr, &variable, &transatt, funcflag);
           break;
       case helpstmt \:
           transatt = noset;
           break;
       case delete \:
           result = Pattern(&expr, &variable, &transatt, funcflag);
           break;
       case preder \:
           break;
    }
    result = 0;
  };.
new3xExprList(_,_,_,_,_,_,_,_,_,marke:xExprList,_),_,_,_
```

```
RETURN result;
? { switch(funcflag)
    {
       case code \:
           result = Pattern(&marke, &variable, &transatt, funcflag);
           break;
       case helpstmt \:
           transatt = noset;
           break;
       case delete \:
           result = Pattern(&marke, &variable, &transatt, funcflag);
           break;
       case preder \:
           break;
    }
    result = 0;
  };.
new3xFormer(_,_,_,_,_,_,_,_,expr:xExpr,_,marke:xIterator),_,_,_
RETURN result;
? { switch(funcflag)
    {
       case code \:
           result = Pattern(&expr, &variable, &transatt, funcflag);
           if(result == 0)
           {
              result = Pattern(&marke, &variable, &transatt, funcflag);
           }
           break;
       case helpstmt \:
           transatt = noset;
           break;
       case delete \:
           result = Pattern(&expr, &variable, &transatt, funcflag);
           if(result == 0)
           {
              result = Pattern(&marke, &variable, &transatt, funcflag);
           }
           break;
       case preder \:
           break;
    }
    result = 0;
  };.
new1xIterator(_,_,_,_,_,_,_,_,marke:xSimpleIts,_,expr:xExpr),_,_,_
RETURN result;
? { switch(funcflag)
    {
```

```
        case code \:
            result = Pattern(&marke, &variable, &transatt, funcflag);
            if(result == 0)
            {
               result = Pattern(&expr, &variable, &transatt, funcflag);
            }
            break;
        case helpstmt \:
            transatt = noset;
            break;
        case delete \:
            result = Pattern(&marke, &variable, &transatt, funcflag);
            if(result == 0)
            {
               result = Pattern(&expr, &variable, &transatt, funcflag);
            }
            break;
        case preder \:
            break;
     }
     result = 0;
   };.
new2xSimpleIts(_,_,_,_,_,_,_,_,marke:xSimpleIt),_,_,_
RETURN result;
? { switch(funcflag)
     {
        case code \:
            result = Pattern(&marke, &variable, &transatt, funcflag);
            break;
        case helpstmt \:
            transatt = noset;
            break;
        case delete \:
            result = Pattern(&marke, &variable, &transatt, funcflag);
            break;
        case preder \:
            break;
     }
     result = 0;
   };.
new1xSimpleIt(_,_,_,_,_,_,_,_,xLValue,_,expr:xExpr),_,_,_
RETURN result;
? { switch(funcflag)
     {
        case code \:
            result = Pattern(&expr, &variable, &transatt, funcflag);
            break;
```

```
      case helpstmt \:
          transatt = noset;
          break;
      case delete \:
          result = Pattern(&expr, &variable, &transatt, funcflag);
          break;
      case preder \:
          break;
   }
   result = 0;
  };.
new1xActuList(_,_,_,_,_,_,_,_,_,marke:xExprList,_),_,_,_
RETURN result;
? { switch(funcflag)
   {
      case code \:
          result = Pattern(&marke, &variable, &transatt, funcflag);
          break;
      case helpstmt \:
          transatt = noset;
          break;
      case delete \:
          result = Pattern(&marke, &variable, &transatt, funcflag);
          break;
      case preder \:
          break;
   }
   result = 0;
  };.
```

Dieser letzte Fall wird benutzt, wenn keins der angegebenen Pattern zum Zuge gekommen ist. Dieses Muster muß immer zu allerletzt kommen.

⟨*transpuma2*⟩≡

```
_,_,_,_
RETURN result;/* Dieses Muster muss zu allerletzt kommen */
? { switch(funcflag)
    {
       case code \:
           variable = NULL;
           transatt = noset;
           break;
       case helpstmt \:
           transatt = noset;
           break;
       case delete \:
           break;
       case preder \:
           break;
    }
    result = 0;
  };.
```

C.10.1 Die C-Funktion `psttvariable`

Die Funktion `psttvariable` erzeugt die eindeutigen virtuellen Variablen `psttint`, wobei `int` eine ganze Zahl ist. Der Name der neuen Variablen wird in `newvar` gespeichert und dann mit allen Bezeichnern in der Bezeichnertabelle verglichen, die durch die Funktion `GetString` in der Variablen `identtable` gespeichert wurden. In `max` ist der größte Index der Bezeichnertabelle gespeichert.

⟨*mysetc*⟩≡
```
tTree psttvariable()
{
   static int numbervar = 0;
   float help;
   int help1 = 1, i;
   char *newvar;
   char *newnum;
   char *identtable;
   tIdent newvariable = 0;
   tTree newqualid;
   tIdent max;

   max = MaxIdent();
   newvar = (char *) malloc (1);
   do
   {
      free(newvar);
      numbervar++;
      identtable = (char *) malloc (MAXIDENTLENGTH);
      help = (float) numbervar;
      help = help / 10.0;
      help1 = 1;
      while(help>=1)
      {
         help1++;
         help = help / 10.0;
      }
      newnum = (char *) malloc (help1);
      sprintf(newnum, "%d", numbervar);
      newvar = (char *) malloc (help1 + 5);
      sprintf(newvar, "pstt");
      strcat(newvar,newnum);
      free(newnum);
      i = 1;
      while(i <= max)
      {
         GetString((tIdent) i, identtable);
         if(strcmp(identtable, newvar) == 0)
```

```
          {
             break;
          }
          i++;
       }
       if(i == max+1)
       {
          newvariable = MakeIdent(newvar,help1 + 4);
       }
    }
    while(newvariable == 0);
    free(newvar);
    free(identtable);
    newqualid = mnew2xQualId(mxId(mid(NoPosition, newvariable,help1 + 4)));
    return newqualid;
 }
```

⟨*myseth*⟩≡

```
 extern tTree psttvariable();
```

⟨*global.h*⟩≡

```
 #define MAXIDENTLENGTH     1024
```

C.10.2 Die C-Funktion `lineprint`

Die Funktion `lineprint` schreibt eine line-Direktive in die Ausgabedatei, die angben soll, daß hier neuer Code eingefügt wurde. Die Funktion gibt natürlich nur etwas aus, sofern die option `-NL` nicht angegeben wurde.

⟨*myprintc*⟩≡

```
 void lineprint()
 {
    if(!lineprint)
    {
       fprintf(ofile,"\n@line 1 \"PSTT\"");
    }
 }
```

⟨*myprinth*⟩≡

```
 extern void lineprint();
```

C.11 Die Ableitungsfunktionen

Die folgenden Funktionen schreiben die Ableitungen in die Ausgabedatei.

C.11.1 Die C-Funktion `insertA1r1with`

Schreibt für die Menge $\{j : j \text{ in } F \mid j \text{ in } G\}$; die Ableitung für „$F$ with := x;“ in die Ausgabedatei.

⟨*mytransc*⟩≡
```
void insertA1r1with(tTree qual, tIdent Ident, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ( ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a1.r2sidename);
   fprintf(ofile, " ) then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\n  end if ; \n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩≡
```
extern void insertA1r1with(tTree, tIdent, DSET);
```

C.11.2 Die C-Funktion `insertA1r2with`

Schreibt für die Menge $\{j : j \text{ in } F \mid j \text{ in } G\}$; die Ableitung für „$G$ with := x;“ in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA1r2with(tTree qual, tIdent Ident, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a1.r1sidename);
   fprintf(ofile, " ) then\n      ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\n  end if ; \n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA1r2with(tTree, tIdent, DSET);
```

C.11.3 Die C-Funktion `insertA1r1less`

Schreibt für die Menge $\{j : j \text{ in } F \mid j \text{ in } G\}$; die Ableitung für „$F$ less := x;“ in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA1r1less(tTree qual, tIdent Ident, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, " if ( ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a1.r2sidename);
   fprintf(ofile, " ) then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\n  end if ; \n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA1r1less(tTree, tIdent, DSET);
```

C.11.4 Die C-Funktion `insertA1r2less`

Schreibt für die Menge $\{j : j \text{ in } F \mid j \text{ in } G\}$; die Ableitung für „$G$ less := x;“ in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA1r2less(tTree qual, tIdent Ident, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a1.r1sidename);
   fprintf(ofile, " ) then\n      ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\n  end if ; \n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA1r2less(tTree, tIdent, DSET);
```

C.11.5 Die C-Funktion `insertA2r1with`

Schreibt für die Menge $\{j : j \text{ in } F \mid j \text{ notin } G\}$; die Ableitung für „$F$ with := x;“ in die Ausgabedatei.

$\langle$*mytransc*$\rangle$+≡

```
void insertA2r1with(tTree qual, tIdent Ident, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " notin ");
   WriteIdent(ofile, variable->A.a1.r2sidename);
   fprintf(ofile, " ) then\n      ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\n  end if ; \n");
   linedirflag = 1;
}
```

$\langle$*mytransh*$\rangle$+≡

```
extern void insertA2r1with(tTree, tIdent, DSET);
```

C.11.6 Die C-Funktion `insertA2r2with`

Schreibt für die Menge $\{j : j \text{ in } F \mid j \text{ notin } G\}$; die Ableitung für „$G$ with := x;“in die Ausgabedatei.

$\langle$*mytransc*$\rangle$+≡

```
void insertA2r2with(tTree qual, tIdent Ident, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a1.r1sidename);
   fprintf(ofile, " ) then\n      ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\n  end if ; \n");
   linedirflag = 1;
}
```

$\langle$*mytransh*$\rangle$+≡

```
extern void insertA2r2with(tTree, tIdent, DSET);
```

C.11.7 Die C-Funktion `insertA2r1less`

Schreibt für die Menge $\{j : j \text{ in } F \mid j \text{ notin } G\}$; die Ableitung für „$F$ less := x;“ in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA2r1less(tTree qual, tIdent Ident, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " notin ");
   WriteIdent(ofile, variable->A.a1.r2sidename);
   fprintf(ofile, " ) then\n      ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\n  end if ; \n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA2r1less(tTree, tIdent, DSET);
```

C.11.8 Die C-Funktion `insertA2r2less`

Schreibt für die Menge $\{j : j \text{ in } F \mid j \text{ notin } G\}$; die Ableitung für „$G$ less := x;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA2r2less(tTree qual, tIdent Ident, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a1.r1sidename);
   fprintf(ofile, " ) then\n      ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\n  end if ; \n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA2r2less(tTree, tIdent, DSET);
```

C.11.9 Die C-Funktion `insertA3with`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) = H\}$; die Ableitung für „$F$ with := x;“ in die Ausgabedatei.

$\langle mytransc \rangle + \equiv$
```
void insertA3with(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = ");
   fprintf(ofile, "%d ) then\n     ", variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\nend if ; \n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle + \equiv$
```
extern void insertA3with(tTree, DSET);
```

C.11.10 Die C-Funktion `insertA3less`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) = H\}$; die Ableitung für „$F$ less := x“; in die Ausgabedatei.

$\langle mytransc \rangle + \equiv$
```
void insertA3less(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = ");
   fprintf(ofile, "%d ) then\n     ", variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\nend if ; \n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}{\equiv}$

```
extern void insertA3less(tTree, DSET);
```

C.11.11 Die C-Funktion `insertA3plus`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) = H\}$; die Ableitung für die Definition „$G(x) := G(x) + const$;“ in die Ausgabedatei.

$\langle mytransc \rangle{+}{\equiv}$

```
void insertA3plus(tTree constante1, tTree constante2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (%d /= 0) then\n   if ( %d in ",
           (constante2->intlit).Integer, (constante1->intlit).Integer);
   WriteIdent(ofile, variable->A.a3.r1sidename);
   fprintf(ofile, " ) then\n   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(%d) = %d ) then\n     ", (constante1->intlit).Integer,
           variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := %d;\n elseif ( ", (constante1->intlit).Integer);
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(%d) = %d - %d) then\n    ", (constante1->intlit).Integer,
           variable->A.a3.integer, (constante2->intlit).Integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := %d ;\nend if; \nend if;\nend if;",
           (constante1->intlit).Integer);
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}{\equiv}$

```
extern void insertA3plus(tTree, tTree, DSET);
```

C.11.12 Die C-Funktion `insertA3minus`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) = H\}$; die Ableitung für die Definition „$G(x)$:= $G(x)$ - *const*;“ in die Ausgabedatei.

$\langle mytransc \rangle {+}{\equiv}$

```
void insertA3minus(tTree constante1, tTree constante2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (%d /= 0) then\n   if ( %d in ",
           (constante2->intlit).Integer, (constante1->intlit).Integer);
   WriteIdent(ofile, variable->A.a3.r1sidename);
   fprintf(ofile, " ) then\n   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(%d) = %d ) then\n     ", (constante1->intlit).Integer,
           variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := %d;\n elseif ( ", (constante1->intlit).Integer);
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(%d) = %d + %d) then\n     ", (constante1->intlit).Integer,
           variable->A.a3.integer, (constante2->intlit).Integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := %d ;\nend if; \nend if;\nend if;",
           (constante1->intlit).Integer);
   linedirflag = 1;
}
```

$\langle mytransh \rangle {+}{\equiv}$

```
extern void insertA3minus(tTree, tTree, DSET);
```

C.11.13 Die C-Funktion `insertA3plus2`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) = H\}$; die Ableitung für die Definition „$G(int) := G(int) + const$;“ in die Ausgabedatei. H ist dabei eine ganze Zahl.

$\langle mytransc\rangle{+}{\equiv}$
```
void insertA3plus2(tTree qual, int constante, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (%d /= 0) then\n   if ( ",constante);
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a3.r1sidename);
   fprintf(ofile, " ) then\n   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = ");
   fprintf(ofile, "%d ) then\n     ", variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n elseif ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = ");
   fprintf(ofile, "%d - %d) then\n     ", variable->A.a3.integer, constante);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\nend if; \nend if; \nend if;");
   linedirflag = 1;
}
```

$\langle mytransh\rangle{+}{\equiv}$
```
extern void insertA3plus2(tTree, int, DSET);
```

C.11.14 Die C-Funktion `insertA3minus2`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) = H\}$; die Ableitung für die Definition „$G(int)$:= $G(int)$ - $const$;“ in die Ausgabedatei. H ist dabei eine ganze Zahl.

$\langle mytransc \rangle{+}{\equiv}$

```
void insertA3minus2(tTree qual, int constante, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (%d /= 0) then\n   if ( ",constante);
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a3.r1sidename);
   fprintf(ofile, " ) then\n   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = ");
   fprintf(ofile, "%d ) then\n     ", variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n elseif ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = ");
   fprintf(ofile, "%d + %d) then\n     ", variable->A.a3.integer, constante);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\nend if; \nend if; \nend if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}{\equiv}$

```
extern void insertA3minus2(tTree,int, DSET);
```

C.11.15 Die C-Funktion `insertA4with`

Schreibt für die Menge $\{[F(j), j] : j \text{ in } G\}$; die Ableitung für „$G$ with := x;“ in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA4with(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := [");
   WriteIdent(ofile, variable->A.a4.r1sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, "),");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ];\n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA4with(tTree, DSET);
```

C.11.16 Die C-Funktion `insertA4less`

Schreibt für die Menge $\{[F(j), j] : j \text{ in } G\}$; die Ableitung für „$G$ less := x;“ in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA4less(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := [");
   WriteIdent(ofile, variable->A.a4.r1sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, "),");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ];\n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA4less(tTree, DSET);
```

C.11.17 Die C-Funktion `insertA4ypredef`

Schreibt für die Menge $\{[F(j), j] : j \text{ in } G\}$; die Vorableitungen für die Definitionen „$F(x)$:= *om*;“ und „$F(x) := y$;“ in die Ausgabedatei.

$\langle mytransc \rangle + \equiv$

```
void insertA4ypredef(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a4.r2sidename);
   fprintf(ofile, " ) then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := [");
   WriteIdent(ofile, variable->A.a4.r1sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, "),");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ]");
   fprintf(ofile, " ;\nend if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle + \equiv$

```
extern void insertA4ypredef(tTree, DSET);
```

C.11.18 Die C-Funktion `insertA4ypostdef`

Schreibt für die Menge $\{[F(j), j] : j \text{ in } G\};$ die Nachableitung für „$F(x) := y;$“in die Ausgabedatei.

$\langle mytransc \rangle +\equiv$
```
void insertA4ypostdef(tTree qual, DSET variable)
{
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a4.r2sidename);
   fprintf(ofile, " ) then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := [");
   WriteIdent(ofile, variable->A.a4.r1sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, "),");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ]");
   fprintf(ofile, " ;\nend if;\n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle +\equiv$
```
extern void insertA4ypostdef(tTree, DSET);
```

C.11.19 Die C-Funktion `insertA6with`

Schreibt für die Menge $\sharp F$ die Ableitung für die Definition „F with $:= x;$“in die Ausgabedatei.

$\langle mytransc \rangle +\equiv$
```
void insertA6with(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " notin ");
   WriteIdent(ofile, variable->A.a6.rsidename);
   fprintf(ofile, " ) then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " +:= 1;\nend if;\n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA6with(tTree, DSET);
```

C.11.20 Die C-Funktion `insertA6less`

Schreibt für die Menge $\sharp F$ die Ableitung für die Definition „F less := x;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA6less(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a6.rsidename);
   fprintf(ofile, " ) then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, "-:= 1;\nend if;\n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA6less(tTree, DSET);
```

C.11.21 Die C-Funktion `insertA8with`

Schreibt für die Menge $\{[j, \#F\{j\}] : j \text{ in domain } F\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ with y;“in die Ausgabedatei.

$\langle mytransc \rangle{+}{\equiv}$

```
void insertA8with(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ( ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " notin ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " ) then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") := 1;\n else ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") := ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") + 1;\n");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}{\equiv}$

```
extern void insertA8with(tTree, DSET);
```

C.11.22 Die C-Funktion `insertA8less`

Schreibt für die Menge $\{[j, \#F\{j\}] : j \text{ in domain } F\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ less y;“in die Ausgabedatei.

$\langle mytransc\rangle{+}{\equiv}$

```
void insertA8less(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = 1 then\n    ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") := om;\n else ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") - 1;\n");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh\rangle{+}{\equiv}$

```
extern void insertA8less(tTree, DSET);
```

C.11.23 Die C-Funktion `insertA9with`

Schreibt für die Menge $\{j : j \text{ in } F|G(j)\}$; die Ableitung für die Definition „F with := x;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA9with(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteIdent(ofile, variable->A.a1.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n end if;");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA9with(tTree, DSET);
```

C.11.24 Die C-Funktion `insertA9less`

Schreibt für die Menge $\{j : j \text{ in } F|G(j)\}$; die Ableitung für die Definition „F less := x;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA9less(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteIdent(ofile, variable->A.a1.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n end if;");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA9less(tTree, DSET);
```

C.11.25 Die C-Funktion `insertA9true`

Schreibt für die Menge $\{j : j \text{ in } F|G(j)\}$; die Ableitung für die Definition „$G(x) :=$ true;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA9true(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if not ");
   WriteIdent(ofile, variable->A.a1.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") then\n   ");
   fprintf(ofile, "if ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a1.r1sidename);
   fprintf(ofile, " then\n      ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n    end if;\n    end if;");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA9true(tTree, DSET);
```

C.11.26 Die C-Funktion `insertA9false`

Schreibt für die Menge $\{j : j \text{ in } F|G(j)\}$; die Ableitung für die Definition „$G(x) :=$ false;“in die Ausgabedatei.

$\langle mytransc \rangle{+}\equiv$
```
void insertA9false(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteIdent(ofile, variable->A.a1.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") then\n   ");
   fprintf(ofile, "if ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a1.r1sidename);
   fprintf(ofile, " then\n      ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n    end if;\n    end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}\equiv$
```
extern void insertA9false(tTree, DSET);
```

C.11.27 Die C-Funktion `insertA10with`

Schreibt für die Menge $\{j : j \text{ in } F\}$; die Ableitung für die Definition „F with $:= x$;“in die Ausgabedatei.

$\langle mytransc \rangle{+}\equiv$
```
void insertA10with(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}\equiv$
```
extern void insertA10with(tTree, DSET);
```

C.11.28 Die C-Funktion `insertA10less`

Schreibt für die Menge $\{j : j \text{ in } F\}$; die Ableitung für die Definition „F less := x;“in die Ausgabedatei.

$\langle mytransc \rangle{+}{\equiv}$
```
void insertA10less(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}{\equiv}$
```
extern void insertA10less(tTree, DSET);
```

C.11.29 Die C-Funktion `insertA11with`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j)/=H\}$; die Ableitung für „$F$ with := x;“ in die Ausgabedatei.

$\langle mytransc \rangle{+}{\equiv}$
```
void insertA11with(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") /= ");
   fprintf(ofile, "%d ) then\n     ", variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\nend if ; \n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}{\equiv}$
```
extern void insertA11with(tTree, DSET);
```

C.11.30 Die C-Funktion `insertA11less`

Schreibt für die Menge $\{j : j \text{ in } F \mid \ G(j)/=H\}$; die Ableitung für „$F$ less := x“; in die Ausgabedatei.

⟨*mytransc*⟩+≡

```
void insertA11less(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") /= ");
   fprintf(ofile, "%d ) then\n     ", variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\nend if ; \n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡

```
extern void insertA11less(tTree, DSET);
```

C.11.31 Die C-Funktion `insertA11plus`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j)/ = H\}$; die Ableitung für die Definition „$G(x)$:= $G(x)$ + *const*;“ in die Ausgabedatei.

$\langle mytransc \rangle + \equiv$

```
void insertA11plus(tTree constante1, tTree constante2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (%d /= 0) then\n   if ( %d in ",
           (constante2->intlit).Integer, (constante1->intlit).Integer);
   WriteIdent(ofile, variable->A.a3.r1sidename);
   fprintf(ofile, " ) then\n   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(%d) = %d ) then\n     ", (constante1->intlit).Integer,
           variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := %d;\n elseif ( ", (constante1->intlit).Integer);
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(%d) = %d - %d) then\n    ", (constante1->intlit).Integer,
           variable->A.a3.integer, (constante2->intlit).Integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := %d ;\nend if; \nend if;\nend if;",
           (constante1->intlit).Integer);
   linedirflag = 1;
}
```

$\langle mytransh \rangle + \equiv$

```
extern void insertA11plus(tTree, tTree, DSET);
```

C.11.32 Die C-Funktion `insertA11minus`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) /= H\}$; die Ableitung für die Definition „$G(x)$:= $G(x)$ - *const*;“ in die Ausgabedatei.

$\langle mytransc \rangle {+}{\equiv}$
```
void insertA11minus(tTree constante1, tTree constante2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (%d /= 0) then\n   if ( %d in ",
           (constante2->intlit).Integer, (constante1->intlit).Integer);
   WriteIdent(ofile, variable->A.a3.r1sidename);
   fprintf(ofile, " ) then\n   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(%d) = %d ) then\n     ", (constante1->intlit).Integer,
           variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := %d;\n elseif ( ", (constante1->intlit).Integer);
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(%d) = %d + %d) then\n     ", (constante1->intlit).Integer,
           variable->A.a3.integer, (constante2->intlit).Integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := %d ;\nend if; \nend if;\nend if;",
           (constante1->intlit).Integer);
   linedirflag = 1;
}
```

$\langle mytransh \rangle {+}{\equiv}$
```
extern void insertA11minus(tTree, tTree, DSET);
```

C.11.33 Die C-Funktion `insertA11plus2`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j)/=H\}$; die Ableitung für die Definition „$G(int) := G(int) + const;$“ in die Ausgabedatei. H ist dabei eine ganze Zahl.

$\langle mytransc \rangle {+}{\equiv}$

```
void insertA11plus2(tTree qual, int constante, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (%d /= 0) then\n   if ( ",constante);
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a3.r1sidename);
   fprintf(ofile, " ) then\n   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = %d ) then\n     ", variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n elseif ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile,") = %d - %d) then\n     ",variable->A.a3.integer,constante);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\nend if; \nend if; \nend if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle {+}{\equiv}$

```
extern void insertA11plus2(tTree, int, DSET);
```

C.11.34 Die C-Funktion `insertA11minus2`

Schreibt für die Menge $\{j : j \text{ in } F \mid \; G(j)/ = H\}$; die Ableitung für die Definition „$G(int) := G(int)$ - $const$;" in die Ausgabedatei. H ist dabei eine ganze Zahl.

$\langle mytransc \rangle + \equiv$

```
void insertA11minus2(tTree qual, int constante, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (%d /= 0) then\n   if ( ",constante);
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a3.r1sidename);
   fprintf(ofile, " ) then\n   if ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = ");
   fprintf(ofile, "%d ) then\n     ", variable->A.a3.integer);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n elseif ( ");
   WriteIdent(ofile, variable->A.a3.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") = ");
   fprintf(ofile, "%d + %d) then\n     ", variable->A.a3.integer, constante);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ;\nend if; \nend if; \nend if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle + \equiv$

```
extern void insertA11minus2(tTree,int, DSET);
```

C.11.35 Die C-Funktion `insertA12with`

Schreibt für die Menge $\{[[F(j),k],j]:[k,j] \text{ in } G\}$; die Ableitung für die Definition „$G\{x\}$:= $G\{x\}$ with y;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA12with(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := [[");
   WriteIdent(ofile, variable->A.a12.r1sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "), ");
   WriteQualId(ofile, qual1);
   fprintf(ofile, "], ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA12with(tTree, tTree, DSET);
```

C.11.36 Die C-Funktion `insertA12less`

Schreibt für die Menge $\{[[F(j),k],j]:[k,j] \text{ in } G\}$; die Ableitung für die Definition „$G\{x\}$:= $G\{x\}$ less y;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA12less(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := [[");
   WriteIdent(ofile, variable->A.a12.r1sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "), ");
   WriteQualId(ofile, qual1);
   fprintf(ofile, "], ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle +\equiv$
```
extern void insertA12less(tTree, tTree, DSET);
```

C.11.37 Die C-Funktion `insertA13with`

Schreibt für die Menge $\{[j,k] : [j,k]$ in $F \mid k$ in $G\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ with y;“in die Ausgabedatei.

$\langle mytransc \rangle +\equiv$
```
void insertA13with(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a13.r2sidename);
   fprintf(ofile, " then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := [");
   WriteQualId(ofile, qual1);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n");
   fprintf(ofile, "end if;\n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle +\equiv$
```
extern void insertA13with(tTree, tTree, DSET);
```

C.11.38 Die C-Funktion `insertA13less`

Schreibt für die Menge $\{[j, k] : [j, k] \text{ in } F \mid k \text{ in } G\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ less y;“in die Ausgabedatei.

$\langle mytransc \rangle + \equiv$

```
void insertA13less(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a13.r2sidename);
   fprintf(ofile, " then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := [");
   WriteQualId(ofile, qual1);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n");
   fprintf(ofile, "end if;\n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle + \equiv$

```
extern void insertA13less(tTree, tTree, DSET);
```

C.11.39 Die C-Funktion `insertA16with`

Schreibt für die Menge $\{[j,k] : [j,k] \text{ in } F \mid k \text{ notin } G\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ with y;“in die Ausgabedatei.

$\langle$*mytransc*$\rangle$+≡

```
void insertA16with(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, " notin ");
   WriteIdent(ofile, variable->A.a13.r2sidename);
   fprintf(ofile, " then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := [");
   WriteQualId(ofile, qual1);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n");
   fprintf(ofile, "end if;\n");
   linedirflag = 1;
}
```

$\langle$*mytransh*$\rangle$+≡

```
extern void insertA16with(tTree, tTree, DSET);
```

C.11.40 Die C-Funktion `insertA16less`

Schreibt für die Menge $\{[j,k] : [j,k] \text{ in } F \mid k \text{ notin } G\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ less y;"in die Ausgabedatei.

$\langle mytransc \rangle + \equiv$
```
void insertA16less(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, " notin ");
   WriteIdent(ofile, variable->A.a13.r2sidename);
   fprintf(ofile, " then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := [");
   WriteQualId(ofile, qual1);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n");
   fprintf(ofile, "end if;\n");
   linedirflag = 1;
}
```

$\langle mytransh \rangle + \equiv$
```
extern void insertA16less(tTree, tTree, DSET);
```

C.11.41 Die C-Funktion `insertA18with`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) \text{ in } H\}$; die Ableitung für die Definition „F with:= x;“in die Ausgabedatei.

$\langle mytransc \rangle{+}{\equiv}$

```
void insertA18with(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "    if ");
   WriteIdent(ofile, variable->A.a18.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") in ");
   WriteIdent(ofile, variable->A.a18.r3sidename);
   fprintf(ofile, " then\n       ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n   ");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}{\equiv}$

```
extern void insertA18with(tTree, DSET);
```

C.11.42 Die C-Funktion `insertA18less`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) \text{ in } H\}$; die Ableitung für die Definition „F less:= x;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA18less(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "    if ");
   WriteIdent(ofile, variable->A.a18.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") in ");
   WriteIdent(ofile, variable->A.a18.r3sidename);
   fprintf(ofile, " then\n       ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n   ");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA18less(tTree, DSET);
```

C.11.43 Die C-Funktion `insertA18ypredef`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) \text{ in } H\}$; die Ableitung für die Definition „$F\{x\}$:= om;“und die Vorableitung für die Definition „$F\{x\}$:= y;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA18ypredef(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a18.r1sidename);
   fprintf(ofile, ") then\n");
   insertA18less(qual, variable);
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}\equiv$
```
extern void insertA18ypredef(tTree, DSET);
```

C.11.44 Die C-Funktion `insertA18ypostdef`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) \text{ in } H\}$; die Nachableitung für die Definition „$F\{x\} := y$;“in die Ausgabedatei.

$\langle mytransc \rangle{+}\equiv$
```
void insertA18ypostdef(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a18.r1sidename);
   fprintf(ofile, ") then\n");
   insertA18with(qual, variable);
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}\equiv$
```
extern void insertA18ypostdef(tTree, DSET);
```

C.11.45 Die C-Funktion `insertA19with`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) \text{ notin } H\}$; die Ableitung für die Definition „F with:= x;“in die Ausgabedatei.

$\langle mytransc \rangle{+}\equiv$

```
void insertA19with(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "    if ");
   WriteIdent(ofile, variable->A.a18.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") notin ");
   WriteIdent(ofile, variable->A.a18.r3sidename);
   fprintf(ofile, " then\n       ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n   ");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}\equiv$

```
extern void insertA19with(tTree, DSET);
```

C.11.46 Die C-Funktion `insertA19less`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) \text{ notin } H\}$; die Ableitung für die Definition „F less:= x;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA19less(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "    if ");
   WriteIdent(ofile, variable->A.a18.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual);
   fprintf(ofile, ") notin ");
   WriteIdent(ofile, variable->A.a18.r3sidename);
   fprintf(ofile, " then\n       ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n   ");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA19less(tTree, DSET);
```

C.11.47 Die C-Funktion `insertA19ypredef`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) \text{ notin } H\}$; die Ableitung für die Definition „$F\{x\} := om$;“und die Vorableitung für die Definition „$F\{x\} := y$;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA19ypredef(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a18.r1sidename);
   fprintf(ofile, ") then\n");
   insertA19less(qual, variable);
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle + \equiv$
```
extern void insertA19ypredef(tTree, DSET);
```

C.11.48 Die C-Funktion `insertA19ypostdef`

Schreibt für die Menge $\{j : j \text{ in } F \mid G(j) \text{ notin } H\}$; die Nachableitung für die Definition „$F\{x\} := y;$“in die Ausgabedatei.

$\langle mytransc \rangle + \equiv$
```
void insertA19ypostdef(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a18.r1sidename);
   fprintf(ofile, ") then\n");
   insertA19with(qual, variable);
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle + \equiv$
```
extern void insertA19ypostdef(tTree, DSET);
```

C.11.49 Die C-Funktion `insertA20with`

Schreibt für die Menge $\{[j,k] : [j,k] \text{ in } F \mid G(j) \text{ in } H\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ with y;“in die Ausgabedatei.

$\langle mytransc \rangle{+}\!\equiv$

```
void insertA20with(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "    if ");
   WriteIdent(ofile, variable->A.a20.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual2);
   fprintf(ofile, ") in ");
   WriteIdent(ofile, variable->A.a20.r3sidename);
   fprintf(ofile, " then\n       ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := [");
   WriteQualId(ofile, qual1);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n   ");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}\!\equiv$

```
extern void insertA20with(tTree, tTree, DSET);
```

C.11.50 Die C-Funktion `insertA20less`

Schreibt für die Menge $\{[j,k] : [j,k] \text{ in } F \mid G(j) \text{ in } H\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ less y;“in die Ausgabedatei.

$\langle mytransc \rangle{+}{\equiv}$
```
void insertA20less(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "    if ");
   WriteIdent(ofile, variable->A.a20.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual2);
   fprintf(ofile, ") in ");
   WriteIdent(ofile, variable->A.a20.r3sidename);
   fprintf(ofile, " then\n       ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := [");
   WriteQualId(ofile, qual1);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n   ");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}{\equiv}$
```
extern void insertA20less(tTree, tTree, DSET);
```

C.11.51 Die C-Funktion `insertA22with`

Schreibt für die Menge $\{[j,k] : [j,k] \text{ in } F \mid G(j) \text{ notin } H\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ with y;“in die Ausgabedatei.

$\langle mytransc \rangle +\equiv$

```
void insertA22with(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "    if ");
   WriteIdent(ofile, variable->A.a20.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual2);
   fprintf(ofile, ") notin ");
   WriteIdent(ofile, variable->A.a20.r3sidename);
   fprintf(ofile, " then\n       ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := [");
   WriteQualId(ofile, qual1);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n   ");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle +\equiv$

```
extern void insertA22with(tTree, tTree, DSET);
```

C.11.52 Die C-Funktion `insertA22less`

Schreibt für die Menge $\{[j,k] : [j,k] \text{ in } F \mid G(j) \text{ notin } H\}$; die Ableitung für die Definition „$F\{x\} := F\{x\}$ less y;“in die Ausgabedatei.

$\langle mytransc \rangle{+}\equiv$
```
void insertA22less(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "    if ");
   WriteIdent(ofile, variable->A.a20.r2sidename);
   fprintf(ofile, "(");
   WriteQualId(ofile, qual2);
   fprintf(ofile, ") notin ");
   WriteIdent(ofile, variable->A.a20.r3sidename);
   fprintf(ofile, " then\n       ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := [");
   WriteQualId(ofile, qual1);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual2);
   fprintf(ofile, "];\n   ");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}\equiv$
```
extern void insertA22less(tTree, tTree, DSET);
```

C.11.53 Die C-Funktion `insertA24r1sidewith`

Schreibt für die Menge $\{[j,k] : k \text{ in } F, j \text{ in } G\{j\}\ \}$; die Ableitung für die Definition „F with := x;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA24r1sidewith(tTree qual, DSET variable)
{
   tTree newvar;

   newvar = psttvariable();
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "for ");
   WriteQualId(ofile, newvar);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a13.r2sidename);
   fprintf(ofile, "{");
   WriteQualId(ofile, qual);
   fprintf(ofile, "} do\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := [");
   WriteQualId(ofile, newvar);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual);
   fprintf(ofile, "];\n");
   fprintf(ofile, "end for;");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA24r1sidewith(tTree, DSET);
```

C.11.54 Die C-Funktion `insertA24r1sideless`

Schreibt für die Menge $\{[j,k] : k \text{ in } F, j \text{ in } G\{j\}\ \}$; die Ableitung für die Definition „F less := x;“ in die Ausgabedatei.

$\langle mytransc \rangle{+}\!\equiv$

```
void insertA24r1sideless(tTree qual, DSET variable)
{
   tTree newvar;

   newvar = psttvariable();
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "for ");
   WriteQualId(ofile, newvar);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a13.r2sidename);
   fprintf(ofile, "{");
   WriteQualId(ofile, qual);
   fprintf(ofile, "} do\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := [");
   WriteQualId(ofile, newvar);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual);
   fprintf(ofile, "];\n");
   fprintf(ofile, "end for;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}\!\equiv$

```
extern void insertA24r1sideless(tTree, DSET);
```

C.11.55 Die C-Funktion `insertA24r2sidewith`

Schreibt für die Menge $\{[j, k] : k \text{ in } F, j \text{ in } G\{k\}\ \}$; die Ableitung für die Definition „G{x} := G{x} with y;“in die Ausgabedatei.

⟨*mytransc*⟩+≡
```
void insertA24r2sidewith(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteQualId(ofile, qual1);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a13.r1sidename);
   fprintf(ofile, " then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := [");
   WriteQualId(ofile, qual2);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual1);
   fprintf(ofile, "];\n");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA24r2sidewith(tTree, tTree, DSET);
```

C.11.56 Die C-Funktion `insertA24r2sideless`

Schreibt für die Menge $\{[j,k] : k \text{ in } F, j \text{ in } G\{k\}\ \}$; die Ableitung für die Definition „G{x} := G{x} less y;“in die Ausgabedatei.

$\langle mytransc \rangle{+}\equiv$

```
void insertA24r2sideless(tTree qual1, tTree qual2, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if ");
   WriteQualId(ofile, qual1);
   fprintf(ofile, " in ");
   WriteIdent(ofile, variable->A.a13.r1sidename);
   fprintf(ofile, " then\n   ");
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := [");
   WriteQualId(ofile, qual2);
   fprintf(ofile, ", ");
   WriteQualId(ofile, qual1);
   fprintf(ofile, "];\n");
   fprintf(ofile, "end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle{+}\equiv$

```
extern void insertA24r2sideless(tTree, tTree, DSET);
```

C.11.57 Die C-Funktion `insertA26rsidewith`

Schreibt für die Menge $\{k : k \text{ in } F \mid k \text{ op1 } int1 \text{ op2 } int2\}$; die Ableitung für die Definition „F with := x;“in die Ausgabedatei.

$\langle mytransc \rangle {+}{\equiv}$

```
void insertA26rsidewith(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ");
   WriteBinOp((variable->A.a26).operator1);
   fprintf(ofile, " %d ", variable->A.a26.integer1);
   WriteBinOp((variable->A.a26).operator2);
   fprintf(ofile, " %d) then\n   ", variable->A.a26.integer2);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " with := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n   end if;");
   linedirflag = 1;
}
```

$\langle mytransh \rangle {+}{\equiv}$

```
extern void insertA26rsidewith(tTree, DSET);
```

C.11.58 Die C-Funktion `insertA26rsideless`

Schreibt für die Menge $\{k : k \text{ in } F \mid k \text{ op1 } int1 \text{ op2 } int2\}$; die Ableitung für die Definition „F less := x;“in die Ausgabedatei.

$\langle mytransc \rangle {+}{\equiv}$

```
void insertA26rsideless(tTree qual, DSET variable)
{
   fprintf(ofile, "\n-- insert transformation code\n");
   fprintf(ofile, "if (");
   WriteQualId(ofile, qual);
   fprintf(ofile, " ");
   WriteBinOp(variable->A.a26.operator1);
   fprintf(ofile, " %d ", variable->A.a26.integer1);
   WriteBinOp(variable->A.a26.operator2);
   fprintf(ofile, " %d) then\n   ", variable->A.a26.integer2);
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " less := ");
   WriteQualId(ofile, qual);
   fprintf(ofile, ";\n   end if;");
   linedirflag = 1;
}
```

⟨*mytransh*⟩+≡
```
extern void insertA26rsideless(tTree, DSET);
```

C.12 Hilfsfunktionen und Typen

Die nächsten drei Funktionen werden benötigt, da es bei *puma* Probleme ga, wenn auf ein gleichnamiges Attribut zweier Nachfolger des gleichen Typs einer Grammatikregel zugegriffen wurde.

C.12.1 Die C-Funktion `EqualQualId`

Die Funktion `EqualQualId` vergleicht zwei Knotentypen vom Typ `new2xQualId`.

⟨*mysetc*⟩+≡
```
int EqualQualId(tTree qual1, tTree qual2)
{
   if(((((qual1->new2xQualId).xId)->xId).id)->id).Ident
      == (((((qual2->new2xQualId).xId)->xId).id)->id).Ident)
   {
      return 1;
   }
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int EqualQualId (tTree , tTree);
```

C.12.2 Die C-Funktion `Equalintlit`

Die Funktion `Equalintlit` vergleicht zwei Knotentypen vom Typ `intlit`.

⟨*mysetc*⟩+≡
```
int Equalintlit(tTree int1, tTree int2)
{
   if((int1->intlit).Integer == (int2->intlit).Integer)
   {
      return 1;
   }
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int Equalintlit (tTree , tTree);
```

C.12.3 Die C-Funktion `WriteQualId`

`WriteQualId` ruft die Funktion `WriteIdent` mit `Ident` von `qualid` auf. Diese Funktion wird benötigt, da in einer *puma*-Funktion, der Zugriff auf Attribute nicht immer möglich ist.

⟨*mysetc*⟩+≡

```
void WriteQualId (FILE *file, tTree qualid)
{
   WriteIdent(file, (((((qualid->new2xQualId).xId)->xId).id)->id).Ident);
}
```

⟨*myseth*⟩+≡

```
extern void WriteQualId (FILE *, tTree);
```

C.12.4 Die C-Aufzählungstypen `FUNCT` und `ISET`

`FUNCT` wird für die Funktion `Pattern` verwendet. Dieser Aufzählungstyp gibt an, in welchem Kontext die Funktion aufgerufen wurde.

`code`	wird von `TransAtt` oder `getcode` aufgerufen
`delete`	wird von `Delete` aufgerufen
`helpstmt`	wird von `Transhelp2` aufgerufen
`preder`	wird von `Statement` verwendet
`postder`	wird von `Output` verwendet.

`ISET` wird im Attributauswerter `Transhelp2` in den Funktionen `Pattern` und `testident` verwendet. `testident` liefert als Rückgabewert, ob ein in der Schleife vorkommender Bezeichner auf der der rechten Seite (`rside`) oder gar nicht (`noside`) in der zu transformierenden Menge vorkommt.

⟨*mydtypesh*⟩+≡

```
typedef enum { code, delete, helpstmt, preder, postder } FUNCT;
typedef enum { noside, rside } ISET;
```

C.13 Das Einlesen der Variablen

C.13.1 Die C-Funktion `getA1variable`

Die Funktion `getA1variable` speichert die Bezeichner aus einem Muster in der Union `variable->A.a1`.

⟨*mytransc*⟩+≡
```
DSET getA1variable(tTree iterator, tTree r1sidename, tTree r2sidename)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator != NoTree)
   {
      variable->A.a1.iterator =
      (((((iterator->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a1.r1sidename =
      (((((r1sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a1.r2sidename =
      (((((r2sidename->new2xQualId).xId)->xId).id)->id).Ident;
   return variable;
}
```

⟨*mytransh*⟩+≡
```
extern DSET getA1variable(tTree, tTree, tTree);
```

C.13.2 Die C-Funktion `getA3variable`

Die Funktion `getA3variable` speichert die Bezeichner aus einem Muster in der Union `variable->A.a3`.

⟨*mytransc*⟩+≡

```
DSET getA3variable(tTree iterator, tTree r1sidename, tTree r2sidename, tTree integer)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator != NoTree)
   {
      variable->A.a3.iterator =
      (((((iterator->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a3.r1sidename =
      (((((r1sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a3.r2sidename =
      (((((r2sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a3.integer = (integer->intlit).Integer;
   return variable;
}
```

⟨*mytransh*⟩+≡

```
extern DSET getA3variable(tTree, tTree, tTree, tTree);
```

C.13.3 Die C-Funktion `getA4variable`

Die Funktion **getA4variable** speichert die Bezeichner aus einem Muster in der Union `variable->A.a4`.

⟨*mytransc*⟩+≡
```
DSET getA4variable(tTree iterator, tTree r1sidename, tTree r2sidename,
                   tTree rvariable)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator != NoTree)
   {
      variable->A.a4.iterator =
      (((((iterator->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a4.r1sidename =
      (((((r1sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a4.r2sidename =
      (((((r2sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a4.rvariable =
      (((((rvariable->new2xQualId).xId)->xId).id)->id).Ident;
   return variable;
}
```

⟨*mytransh*⟩+≡
```
extern DSET getA4variable(tTree, tTree, tTree, tTree);
```

C.13.4 Die C-Funktion `getA6variable`

Die Funktion **getA6variable** speichert die Bezeichner aus einem Muster in der Union `variable->A.a6`.

⟨*mytransc*⟩+≡
```
DSET getA6variable(tTree rsidename)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   variable->A.a6.rsidename =
      (((((rsidename->new2xQualId).xId)->xId).id)->id).Ident;
   return variable;
}
```

⟨*mytransh*⟩+≡
```
extern DSET getA6variable(tTree);
```

C.13.5 Die C-Funktion `getA8variable`

Die Funktion **getA8variable** speichert die Bezeichner aus einem Muster in der Union `variable->A.a8`.

⟨*mytransc*⟩+≡
```
DSET getA8variable(tTree rsidename, tTree iterator1, tTree iterator2)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator1 != NoTree)
   {
      variable->A.a8.iterator1 =
         (((((iterator1->new2xQualId).xId)->xId).id)->id).Ident;
   }
   if(iterator2 != NoTree)
   {
      variable->A.a8.iterator2 =
         (((((iterator2->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a8.rsidename =
      (((((rsidename->new2xQualId).xId)->xId).id)->id).Ident;
   return variable;
}
```

⟨*mytransh*⟩+≡
```
extern DSET getA8variable(tTree, tTree, tTree);
```

C.13.6 Die C-Funktion `getA10variable`

Die Funktion `getA10variable` speichert die Bezeichner aus einem Muster in der Union `variable->A.a10`.

⟨*mytransc*⟩+≡
```
DSET getA10variable(tTree rsidename, tTree iterator)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator != NoTree)
   {
      variable->A.a10.iterator =
         (((((iterator->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a10.rsidename =
      (((((rsidename->new2xQualId).xId)->xId).id)->id).Ident;
   return variable;
}
```

⟨*mytransh*⟩+≡
```
extern DSET getA10variable(tTree, tTree);
```

C.13.7 Die C-Funktion `getA12variable`

Die Funktion `getA12variable` speichert die Bezeichner aus einem Muster in der Union `variable->A.a12`.

⟨*mytransc*⟩+≡
```
DSET getA12variable(tTree iterator1, tTree iterator2, tTree r1sidename,
                    tTree r2sidename)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator1 != NoTree)
   {
      variable->A.a12.iterator1 =
      (((((iterator1->new2xQualId).xId)->xId).id)->id).Ident;
   }
   if(iterator2 != NoTree)
   {
      variable->A.a12.iterator2 =
      (((((iterator2->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a12.r1sidename =
      (((((r1sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a12.r2sidename =
      (((((r2sidename->new2xQualId).xId)->xId).id)->id).Ident;
   return variable;
}
```

⟨*mytransh*⟩+≡
```
extern DSET getA12variable(tTree, tTree, tTree, tTree);
```

C.13.8 Die C-Funktion `getA13variable`

Die Funktion **`getA13variable`** speichert die Bezeichner aus einem Muster in der Union **`variable->`**`A.a13`.

⟨*mytransc*⟩+≡
```
DSET getA13variable(tTree iterator1, tTree iterator2, tTree iterator3,
                    tTree r1sidename, tTree r2sidename)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator1 != NoTree)
   {
      variable->A.a13.iterator1 =
      (((((iterator1->new2xQualId).xId)->xId).id)->id).Ident;
   }
   if(iterator2 != NoTree)
   {
      variable->A.a13.iterator2 =
      (((((iterator2->new2xQualId).xId)->xId).id)->id).Ident;
   }
   if(iterator3 != NoTree)
   {
      variable->A.a13.iterator3 =
      (((((iterator3->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a13.r1sidename =
      (((((r1sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a13.r2sidename =
      (((((r2sidename->new2xQualId).xId)->xId).id)->id).Ident;
   return variable;
}
```

⟨*mytransh*⟩+≡
```
extern DSET getA13variable(tTree, tTree, tTree, tTree, tTree);
```

C.13.9 Die C-Funktion `getA18variable`

Die Funktion **`getA18variable`** speichert die Bezeichner aus einem Muster in der Union `variable->A.a18`.

⟨*mytransc*⟩+≡
```
DSET getA18variable(tTree iterator, tTree r1sidename, tTree r2sidename,
                    tTree r3sidename)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator != NoTree)
   {
      variable->A.a18.iterator =
      (((((iterator->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a18.r1sidename =
      (((((r1sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a18.r2sidename =
      (((((r2sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a18.r3sidename =
      (((((r3sidename->new2xQualId).xId)->xId).id)->id).Ident;
   return variable;
}
```

⟨*mytransh*⟩+≡
```
extern DSET getA18variable(tTree, tTree, tTree, tTree);
```

C.13.10 Die C-Funktion `getA20variable`

Die Funktion **`getA20variable`** speichert die Bezeichner aus einem Muster in der Union **`variable->A.a20`**.

⟨*mytransc*⟩+≡

```
DSET getA20variable(tTree iterator1, tTree iterator2, tTree iterator3,
                    tTree r1sidename, tTree r2sidename, tTree r3sidename)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator1 != NoTree)
   {
      variable->A.a20.iterator1 =
      (((((iterator1->new2xQualId).xId)->xId).id)->id).Ident;
   }
   if(iterator2 != NoTree)
   {
      variable->A.a20.iterator2 =
      (((((iterator2->new2xQualId).xId)->xId).id)->id).Ident;
   }
   if(iterator3 != NoTree)
   {
      variable->A.a20.iterator3 =
      (((((iterator3->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a20.r1sidename =
      (((((r1sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a20.r2sidename =
      (((((r2sidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a20.r3sidename =
      (((((r3sidename->new2xQualId).xId)->xId).id)->id).Ident;
   return variable;
}
```

⟨*mytransh*⟩+≡

```
extern DSET getA20variable(tTree, tTree, tTree, tTree, tTree, tTree);
```

C.13.11 Die C-Funktion `getA26variable`

Die Funktion `getA26variable` speichert die Bezeichner aus einem Muster in der Union `variable->A.a26`.

⟨*mytransc*⟩+≡
```
DSET getA26variable(tTree iterator, tTree rsidename, tTree operator1,
                    tTree operator2, int konst1, int konst2)
{
   DSET variable;
   tTree newqual;

   variable = getmemory();
   newqual = psttvariable(NoPosition);
   variable->newlsidename =
      (((((newqual->new2xQualId).xId)->xId).id)->id).Ident;
   variable->newlength = (((((newqual->new2xQualId).xId)->xId).id)->id).Length;
   if(iterator != NoTree)
   {
      variable->A.a26.iterator =
      (((((iterator->new2xQualId).xId)->xId).id)->id).Ident;
   }
   variable->A.a26.rsidename =
      (((((rsidename->new2xQualId).xId)->xId).id)->id).Ident;
   variable->A.a26.operator1 = ((operator1->xBinOp).yyHead).yyKind;
   variable->A.a26.operator2 = ((operator2->xBinOp).yyHead).yyKind;
   variable->A.a26.integer1 = konst1;
   variable->A.a26.integer2 = konst2;
   return variable;
}
```

⟨*mytransh*⟩+≡
```
extern DSET getA26variable(tTree, tTree, tTree, tTree, int, int);
```

C.13.12 Die C-Funktion `getmemory`

Diese Funktion allokiert Speicherplatz für die Struktur `DSET`.

⟨*mytransc*⟩+≡

```
DSET getmemory()
{
   DSET variable;

   if((variable = (DSET)malloc(sizeof(Dset))) == NULL)
   {
      Message("not enough memory", 2, NoPosition);
      exit(-1);
   }
   return variable;
}
```

⟨*mytransh*⟩+≡

```
extern DSET getmemory();
```

C.14 Der Vergleich von Variablen

Die folgenden Funktionen werden von der Funktion `Pattern` aufgerufen, um zu vergleichen, ob die angegebenen Bezeichner mit denen in der Variable `variable` übereinstimmen.

C.14.1 Die C-Funktion `checkA1variable`

`checkA1variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen. Bei dieser Funktion ist darauf zu achten, daß die einzelnen Komponenten mit den richtigen Bezeichnern überprüft werden. Bei den Bezeichnern muß der Typ `tTree` angegeben werden, jedoch ist durch den Aufruf sichergestellt, daß sie vom Typ `new2xQualId` sind.

⟨*mysetc*⟩+≡

```
int checkA1variable(tTree qual1, tTree qual2, tTree qual3, DSET variable)
{
   if((((((qual1->new2xQualId).xId)->xId).id)->id).Ident ==
                              variable->A.a1.iterator
   && (((((qual2->new2xQualId).xId)->xId).id)->id).Ident ==
      variable->A.a1.r1sidename
   && (((((qual3->new2xQualId).xId)->xId).id)->id).Ident ==
      variable->A.a1.r2sidename)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡

```
extern int checkA1variable(tTree, tTree, tTree, DSET);
```

C.14.2 Die C-Funktion `checkA3variable`

`checkA3variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen.

⟨*mysetc*⟩+≡

```
int checkA3variable(tTree qual1, tTree qual2, tTree qual3, tTree integer, DSET variable)
{
   if(((((qual1->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a3.iterator
      && (((((qual2->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a3.r1sidename
      && (((((qual3->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a3.r2sidename
      && (integer->intlit).Integer == variable->A.a3.integer)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡

```
extern int checkA3variable(tTree, tTree, tTree, tTree, DSET);
```

C.14.3 Die C-Funktion `checkA4variable`

`checkA4variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen. Bei dieser Funktion ist darauf zu achten, daß die einzelnen Komponenten mit den richtigen Bezeichnern überprüft werden. Bei den Bezeichnern muß der Typ `tTree` angegeben werden, jedoch ist durch den Aufruf sichergestellt, daß sie vom Typ `new2xQualId` sind.

⟨*mysetc*⟩+≡

```
int checkA4variable(tTree qual1, tTree qual2, tTree qual3, tTree qual4,
                    DSET variable)
{
   if((((((qual1->new2xQualId).xId)->xId).id)->id).Ident ==
      variable->A.a4.iterator
   && (((((qual2->new2xQualId).xId)->xId).id)->id).Ident ==
      variable->A.a4.r1sidename
   && (((((qual3->new2xQualId).xId)->xId).id)->id).Ident ==
      variable->A.a4.r2sidename
   && (((((qual4->new2xQualId).xId)->xId).id)->id).Ident ==
      variable->A.a4.rvariable)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡

```
extern int checkA4variable(tTree, tTree, tTree, tTree, DSET);
```

C.14.4 Die C-Funktion `checkA6variable`

`checkA6variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen. Bei dieser Funktion ist darauf zu achten, daß die einzelnen Komponenten mit den richtigen Bezeichnern überprüft werden. Bei den Bezeichnern muß der Typ `tTree` angegeben werden, jedoch ist durch den Aufruf sichergestellt, daß sie vom Typ `new2xQualId` sind.

⟨*mysetc*⟩+≡

```
int checkA6variable(tTree qual, DSET variable)
{
   if((((((qual->new2xQualId).xId)->xId).id)->id).Ident ==
      variable->A.a6.rsidename)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int checkA6variable(tTree, DSET);
```

C.14.5 Die C-Funktion checkA8variable

`checkA8variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen.

⟨*mysetc*⟩+≡
```
int checkA8variable(tTree qual1, tTree qual2, tTree qual3, DSET variable)
{
   if((((((qual1->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a8.rsidename
      && (qual2 == NoTree || (((((qual2->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a8.iterator1)
      && (qual3 == NoTree || (((((qual3->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a8.iterator2))
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int checkA8variable(tTree, tTree, tTree, DSET);
```

C.14.6 Die C-Funktion checkA10variable

`checkA10variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen.

⟨*mysetc*⟩+≡
```
int checkA10variable(tTree qual1, tTree qual2, DSET variable)
{
   if((((((qual1->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a10.rsidename
      && (((((qual2->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a10.iterator)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int checkA10variable(tTree, tTree, DSET);
```

C.14.7 Die C-Funktion `checkA12variable`

`checkA12variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen.

⟨*mysetc*⟩+≡
```
int checkA12variable(tTree qual1, tTree qual2, tTree qual3, tTree qual4,
                     DSET variable)
{
   if((qual1 == NoTree || (((((qual1->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a12.iterator1)
      && (qual2 == NoTree || (((((qual2->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a12.iterator2)
      && (((((qual3->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a12.r1sidename
      && (((((qual4->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a12.r2sidename)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int checkA12variable(tTree, tTree, tTree, tTree, DSET);
```

C.14.8 Die C-Funktion `checkA13variable`

`checkA13variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen.

⟨*mysetc*⟩+≡
```
int checkA13variable(tTree qual1, tTree qual2, tTree qual3, tTree qual4,
                     tTree qual5, DSET variable)
{
   if((qual1 == NoTree || (((((qual1->new2xQualId).xId)->xId).id)->id).Ident
                           == variable->A.a13.iterator1)
      && (qual2 == NoTree || (((((qual2->new2xQualId).xId)->xId).id)->id).Ident
                              == variable->A.a13.iterator2)
      && (qual3 == NoTree || (((((qual3->new2xQualId).xId)->xId).id)->id).Ident
                              == variable->A.a13.iterator3)
      && (((((qual4->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a13.r1sidename
      && (((((qual5->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a13.r2sidename)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int checkA13variable(tTree, tTree, tTree, tTree, tTree, DSET);
```

C.14.9 Die C-Funktion `checkA18variable`

`checkA18variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen.

⟨*mysetc*⟩+≡
```
int checkA18variable(tTree qual1, tTree qual2, tTree qual3, tTree qual4,
                     DSET variable)
{
   if((((((qual1->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a18.iterator
      && (((((qual2->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a18.r1sidename
      && (((((qual3->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a18.r2sidename
      && (((((qual4->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a18.r3sidename)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int checkA18variable(tTree, tTree, tTree, tTree, DSET);
```

C.14.10 Die C-Funktion `checkA20variable`

`checkA20variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen.

⟨*mysetc*⟩+≡
```
int checkA20variable(tTree qual1, tTree qual2, tTree qual3, tTree qual4,
                     tTree qual5, tTree qual6, DSET variable)
{
   if((qual1 == NoTree || (((((qual1->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a20.iterator1)
      && (qual2 == NoTree || (((((qual2->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a20.iterator2)
      && (qual3 == NoTree || (((((qual3->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a20.iterator3)
      && (((((qual4->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a20.r1sidename
      && (((((qual5->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a20.r2sidename
      && (((((qual6->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a20.r3sidename)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int checkA20variable(tTree, tTree, tTree, tTree, tTree, tTree, DSET);
```

C.14.11 Die C-Funktion checkA26variable

`checkA26variable` überprüft, ob die angegebenen Bezeichner mit denen in `variable` übereinstimmen.

⟨*mysetc*⟩+≡
```
int checkA26variable(tTree iterator, tTree rsidename, tTree operator1,
                     tTree operator2, int konst1, int konst2, DSET variable)
{
   if((((((iterator->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a26.iterator
      && (((((rsidename->new2xQualId).xId)->xId).id)->id).Ident
      == variable->A.a26.rsidename
      && ((operator1->xBinOp).yyHead).yyKind
      == variable->A.a26.operator1
      && ((operator2->xBinOp).yyHead).yyKind
      == variable->A.a26.operator2
      && konst1 == variable->A.a26.integer1
      && konst2 == variable->A.a26.integer2)
      return 1;
   else return 0;
}
```

⟨*myseth*⟩+≡
```
extern int checkA26variable(tTree, tTree, tTree, tTree, int, int, DSET);
```

C.15 Die *puma*-Funktion CanIDelete

Diese Funktion, die Funktion `CanIDelete1` und die Attributauswerter `Transhelp1` und `Transhelp2` durchlaufen die Schleife, um festzustellen, ob die Voraussetzungen zum Transformieren gegeben sind. Sollte eine Voraussetzung nicht erfüllt sein, wird `transatt` auf `noset` gesetzt. Welche Voraussetzungen überprüft werden, wird bei den jeweiligen Funktionen erläutert.

⟨*transpuma*⟩+≡
```
PROCEDURE CanIDelete(stmts:xStmts, variable:DSET, REF transatt:CODE)
new1xStmts(_,_,_,_,_,_,_,_,newstmts:xStmts,newstmt:xStmt,_,_),_,_
? CanIDelete(newstmts, variable, transatt);
  CanIDelete1(newstmt, variable, transatt); .

new2xStmts(_,_,_,_,_,_,_,_,newstmt:xStmt,_,_),_,_
? CanIDelete1(newstmt, variable, transatt); .

new3xStmts(_,_,_,_,_,_,_,_,_,newstmts:xStmts),_,_
? CanIDelete(newstmts, variable, transatt); .
```

C.15.1 Die *puma*-Funktion `CanIDelete1`

`CanIDelete1` überprüft, ob in der Schleife Anweisungen vorkommen, bei denen nicht transformiert werden darf, da Seiteneffekte auf die Variablen in der zu transformierenden Menge nicht ausgeschlossen werden können. Dazu zählen unter anderem `stop`- und `return`-Anweisungen, Tupelraumoperationen oder `lambda`-Funktionen.

Dasgleiche gilt momentan auch für `get`, `fget`, `getf` und `fgetf`, da für diese Funktionen die Grammatik noch überarbeitet wird.

⟨*transpuma*⟩+≡
```
PROCEDURE CanIDelete1(stmt:xStmt, variable:DSET, REF transatt:CODE)
new4xStmt,_,_ ? transatt = noset;. /* new4xStmt = 'return' xExpr . */

new5xStmt,_,_ ? transatt = noset;. /* new5xStmt = 'return' . */

new13xStmt(_,_,_,_,_,_,_,_,new7xStdIO,_),_,_ ? transatt = noset;.
                         /* new13xStmt = 'get' xActuList . */

new54xStmt,_,_ ? transatt = noset;. /* getf */

new55xStmt,_,_ ? transatt = noset;. /* fget */

new56xStmt,_,_ ? transatt = noset;. /* fgetf */

new34xStmt,_,_ ? transatt = noset; . /* new34xStmt = xLambda xActuList . */

new35xStmt,_,_ ? transatt = noset; .  /* new35xStmt = 'self' xActuList . */

new44xStmt,_,_ ? transatt = noset;.  /* new44xStmt = 'quit' . */

new45xStmt,_,_ ? transatt = noset;. /* new45xStmt = 'quit' xId . */

new46xStmt,_,_ ? transatt = noset;.  /* new46xStmt = 'continue' . */

new47xStmt,_,_ ? transatt = noset;. /* new47xStmt = 'continue' xId . */

new48xStmt,_,_ ? transatt = noset; .    /* new48xStmt = '||' xExpr . */

new49xStmt,_,_ ? transatt = noset; .
                         /* new49xStmt = dep1:'deposit' Expr1:xExpr 'at'
                            Expr2:xExpr 'end' dep2:'deposit' . */

new50xStmt,_,_ ? transatt = noset; .
                          /* new50xStmt = dep1:'deposit' Expr1:xExpr 'at'
                             Expr2:xExpr 'blockiffull' 'end' dep2:'deposit' . */

new51xStmt,_,_ ? transatt = noset; .
```

```
                          /* new51xStmt = fetch1:'fetch' xcTempList 'at' xExpr
                             xElseStmts 'end' fetch2:'fetch' . */

new52xStmt,_,_ ? transatt = noset; .
                          /* new52xStmt = meet1:'meet' xcTempList 'at' xExpr
                             xElseStmts 'end' meet2:'meet' . */
```

C.16 Der Attributauswerter `Transhelp1`

`Transhelp1` untersucht, ob in der Schleife eine Prozedur aufgerufen wird, da in diesem Fall Seiteneffekte auf die Transformationsmöglichkeiten nicht ausgeschlossen werden können.

Für diesen Attributauswerter wird ein `THREAD`-Attribut benötigt, da es auf die Reihenfolge der Anweisungen ankommt.
⟨*trans.cg*⟩+≡

```
MODULE Transhelp1Int

EVAL Transhelp1

DECLARE
xProgDefn xProgBody xPHMDefn xParamList xParamMode xModImport xModExport
xIdList xRdParamList xImplAsso xDecls xDecl xSingleVar xDeclKey xPersDecl
xStmts xExplAsso xHandAsso xStmt xStmtSignal xStmtNotify xId xUnOp xStdIO
xExpr xFrom xActuList xElIfStmt xElIfStmts xElseStmts xCaseStmts xExprList
xLabel xLoops xTemplate xTempCond xExprFormal xInto xFormal xIterator
xSimpleIts xSimpleIt xMapSel xLValue xSelector xFormer xInstantiate
xInstExport xInstImport xQualId xLambda xQuantifier xQualifier xElIfExprs
xElIfExpr xElseExpr xCaseExprs xBinOp id intlit floatlit str
xTypeList = [ help THREAD ] .

END Transhelp1Int



MODULE Transhelp1

EVAL Transhelp1

IMPORT
{
#include "transform.h"
}

GLOBAL
{
CODE tflag;  /* tflag = transatt */
int tflagflag = 0;  /* tflagflag == 1 => aeusserste Schleife des
                   uebergebenden Baumes  */
Scope scope;  /* zeigt auf den gueltigen Sichtbarkeitsbereich */
}

RULE
```

Mit der folgenden Regel wird der Attributauswerter gestartet.

⟨*trans.cg*⟩+≡

```
xInitChain = { xProgDefn:helpIn := 1;}.
```

Bei den folgenden Regeln werden nur die Variablen `scope` und `tflag` gesetzt, damit r im weiteren die Werte von `loopenv` und `transatt` verfügbar sind.

⟨*trans.cg*⟩+≡

```
xLoops = { helpOut := { helpOut = helpIn;
                        if(tflagflag == 0)
                        {
                           scope = loopenv;
                           tflag = transatt;
                        }
                        tflagflag++;};}.

new6xLoops = { xStmts:helpIn :- helpIn;
               helpOut :- xStmts:helpOut;}.

new7xLoops = { xIterator:helpIn := { xIterator:helpIn = helpIn;
                                     if(tflagflag == 0)
                                     {
                                        scope = loopenv;
                                        tflag = transatt;
                                     }
                                     tflagflag++;};
               xStmts:helpIn :- xIterator:helpOut;
               helpOut := { helpOut = xStmts:helpOut;
                            if(tflagflag == 1)
                            {
                               transatt = tflag;
                            }
                            tflagflag--;};}.

new8xLoops = { xExpr:helpIn := { xExpr:helpIn = helpIn;
                                 if(tflagflag == 0)
                                 {
                                    scope = loopenv;
                                    tflag = transatt;
                                 }
                                 tflagflag++;};
               xStmts:helpIn :- xExpr:helpOut;
               helpOut := { helpOut = xStmts:helpOut;
                            if(tflagflag == 1)
                            {
                               transatt = tflag;
                            }
                            tflagflag--;};}.

new9xLoops = { xIterator:helpIn := { xIterator:helpIn = helpIn;
                                     if(tflagflag == 0)
                                     {
```

```
                                    scope = loopenv;
                                    tflag = transatt;
                                 }
                                 tflagflag++;};
                 xStmts:helpIn :- xIterator:helpOut;
                 helpOut := { helpOut = xStmts:helpOut;
                              if(tflagflag == 1)
                              {
                                 transatt = tflag;
                              }
                              tflagflag--;};}.

  new10xLoops = { xStmts:helpIn := { xStmts:helpIn = helpIn;
                                     if(tflagflag == 0)
                                     {
                                        scope = loopenv;
                                        tflag = transatt;
                                     }
                                     tflagflag++;};
                  xExpr:helpIn :- xStmts:helpOut;
                  helpOut := { helpOut = xStmts:helpOut;
                               if(tflagflag == 1)
                               {
                                  transatt = tflag;
                               }
                               tflagflag--;};}.
```

Für jeden Bezeichner in der Schleife muß in der Prozedurnamentabelle nachgesehen werden, ob es sich bei diesem Bezeichner um einen Prozedurnamen handelt.

⟨*trans.cg*⟩+≡

```
  id = { helpOut := { helpOut = helpIn;
                      if (idenv(Ident, scope))
                      {
                         tflag = noset;
                      }};
     }.

  ⟨transhelp1⟩

  END Transhelp1
```

Für *transhelp1* siehe Abschnitt C.42.4.

C.16.1 Die C-Funktion `idenv`

`idenv` durchsucht den aktuellen Prozedurnamenbereich nach dem Bezeichner `ident`.

⟨*myscopec3*⟩≡
```
int idenv(tIdent ident, Scope scope)
{
   PhmList *helpphmlist;
   Scope helpscope;
   tIdent id;

   helpscope = scope;
   if (ident == helpscope->scopename)
   {
      return 1;
   }
   while(helpscope != NULL)
   {
      helpphmlist = helpscope->phmnames;
      while(helpphmlist != NULL)
      {
         if (ident == helpphmlist->phmname)
         {
            return 1;
         }
         helpphmlist = helpphmlist->next;
      }
      helpscope = helpscope->outer;
   }
   return 0;
}
```

⟨*myscopeh*⟩+≡
```
extern int idenv(tIdent, Scope);
```

C.17 Der Attributauswerter `Transhelp2`

`Transhelp2` untersucht, ob die freien Variablen aus der zu transformierenden Menge nur in „erlaubten“ Anweisungen als l-Values auftreten. Diese Untersuchung ist notwendig, um zum einen sicherzustellen, daß auf die `rsidename`'s nur die in Anhang A für dieses Muster angegebenen Operationen durchgeführt werden. Bei anderen Zuweisungen kann keine Aussage über die Veränderung zwischen dem alten und dem neuen Wert gemacht werden bzw. die Transformationen wären nicht effizient. Für diesen Attributauswerter wird wiederum ein `THREAD`-Attribut benötigt.

⟨*trans.cg*⟩+≡

```
MODULE TranshelpInt2

EVAL Transhelp2

DECLARE
xProgDefn xProgBody xPHMDefn xParamList xParamMode = [ help2 THREAD ] .
xModImport xModExport xIdList xRdParamList xImplAsso = [ help2 THREAD ] .
xDecls xDecl xSingleVar xDeclKey xPersDecl xStmts = [ help2 THREAD ] .
xExplAsso xHandAsso xStmt xStmtSignal xStmtNotify xId = [ help2 THREAD ] .
xUnOp xStdIO xExpr xFrom xActuList xElIfStmt xElIfStmts = [ help2 THREAD ] .
xElseStmts xCaseStmts xExprList xLabel xLoops xTemplate = [ help2 THREAD ] .
xTempCond xExprFormal xInto xFormal xIterator xSimpleIts = [ help2 THREAD ] .
xSimpleIt xMapSel xLValue xSelector xFormer xInstantiate = [ help2 THREAD ] .
xInstExport xInstImport xQualId xLambda xQuantifier = [ help2 THREAD ] .
xQualifier xElIfExprs xElIfExpr xElseExpr xCaseExprs = [ help2 THREAD ] .
xBinOp id intlit floatlit str xTypeList = [ help2 THREAD ] .

END TranshelpInt2



MODULE Transhelp2

EVAL Transhelp2

IMPORT
{
#include "transform.h"
}

GLOBAL
{
   ISET rflag2 = noside; /* Returnwert bei id ;
                            == rside : Ident == r*sidename;
                            == noside : sonst; */
```

```
   int lflag2 = 0; /* wird zu Beginn von new5xLValue auf 1 gesetzt und
                      am Ende wieder auf 0 */
   int simpleflag2 = 0; /* wird bei new1xSimpleIt auf 1 gesetzt, da dieser
                           l-Value an die Menge gebunden ist. */
   CODE tflag2;  /* tflag2 = transatt */
   int tflag2flag = 0;  /* tflag2flag == 1 => aeusserste Schleife des
                           uebergebenden Baumes  */
   DSET vflag2;  /* vflag2 = variable */
}

RULE
```

Mit der folgenden Regel wird wieder der Attributauswerter gestartet.

⟨*trans.cg*⟩+≡

```
xInitChain = { xProgDefn:help2In := 1;}.
```

Bei den folgenden Regeln werden auch wieder die Variablen `vflag2` und `tflag2` gesetzt, damit wieder die Werte von `variable` und `transatt` verfügbar sind.

⟨*trans.cg*⟩+≡

```
xLoops = { help2Out := { help2Out = help2In;
                         if(tflag2flag == 0)
                         {
                            vflag2 = variable;
                            tflag2 = transatt;
                         }
                         tflag2flag++;};}.

new6xLoops = { xStmts:help2In :- help2In;
               help2Out :- xStmts:help2Out;}.

new7xLoops = { xIterator:help2In := { xIterator:help2In = help2In;
                                      if(tflag2flag == 0)
                                      {
                                         vflag2 = variable;
                                         tflag2 = transatt;
                                      }
                                      tflag2flag++;};
               xStmts:help2In :- xIterator:help2Out;
               help2Out := { help2Out = xStmts:help2Out;
                             if(tflag2flag == 1)
                             {
                                transatt = tflag2;
                             }
                             tflag2flag--;};}.

new8xLoops = { xExpr:help2In := { xExpr:help2In = help2In;
                                  if(tflag2flag == 0)
                                  {
                                     vflag2 = variable;
                                     tflag2 = transatt;
                                  }
                                  tflag2flag++;};
               xStmts:help2In :- xExpr:help2Out;
               help2Out := { help2Out = xStmts:help2Out;
                             if(tflag2flag == 1)
                             {
                                transatt = tflag2;
                             }
                             tflag2flag--;};}.

new9xLoops = { xIterator:help2In := { xIterator:help2In = help2In;
                                      if(tflag2flag == 0)
                                      {
```

```
                                    vflag2 = variable;
                                    tflag2 = transatt;
                                 }
                                 tflag2flag++;};
               xStmts:help2In :- xIterator:help2Out;
               help2Out := { help2Out = xStmts:help2Out;
                             if(tflag2flag == 1)
                             {
                                transatt = tflag2;
                             }
                             tflag2flag--;};}.

new10xLoops = { xStmts:help2In := { xStmts:help2In = help2In;
                                    if(tflag2flag == 0)
                                    {
                                       vflag2 = variable;
                                       tflag2 = transatt;
                                    }
                                    tflag2flag++;};
                xExpr:help2In :- xStmts:help2Out;
                help2Out := { help2Out = xStmts:help2Out;
                              if(tflag2flag == 1)
                              {
                                 transatt = tflag2;
                              }
                              tflag2flag--;};}.
```

⟨*trans.cg*⟩+≡

```
new1xSimpleIt = { xLValue:help2In := { simpleflag2 = 1;
                                       xLValue:help2In = help2In;};
                  xExpr:help2In :- xLValue:help2Out;
                  help2Out :- xExpr:help2Out;}.
```

Hier wird vor dem Eintritt in das Nonterminal `xQualId` die Variable `lflag1` auf `1` gesetzt, und beim Austritt aus `new5xLValue` wieder auf `0`. Damit kann in `id` entschieden werden, ob es sich bei `Ident` um einen l-Value handelt oder nicht.

⟨*trans.cg*⟩+≡
```
new5xLValue = { xQualId:help2In :=
                    { xQualId:help2In = help2In;
                      if(simpleflag2 == 0)
                      {
                         lflag2 = 1;
                      }
                    };
                 help2Out :=
                    { help2Out = xQualId:help2Out;
                      lflag2 = 0;
                    };}.
```

Die Funktion `testident` übergibt an `rflag2`, ob `Ident` in der zu transformierenden Menge vorkommt.

⟨*trans.cg*⟩+≡
```
id = { help2Out := { help2Out = help2In;
                      if(lflag2 == 1)
                      {
                         rflag2 = testident(tflag2, vflag2, Ident);
                      }
                    }; }.
```

Kommt `Ident` in der zu transformierenden Menge vor, wird nun mit der Funktion `Pattern` überprüft, ob in `xStmt` eine erlaubte Operation verwendet wurde. Im Anschluß daran, wird `rflag2` wieder auf `noside` gesetzt. In `Pattern` wird nötigenfalls `tflag2` modifiziert.

⟨*trans.cg*⟩+≡
```
new1xStmts  = { xStmts:help2In :- help2In;
                xStmt:help2In :- xStmts:help2Out;
                xExplAsso:help2In := { xExplAsso:help2In = xStmt:help2Out;
                                       if(rflag2 != noside)
                                       { Pattern(&xStmt,&vflag2,&tflag2,
                                                 helpstmt);
                                         rflag2 = noside;
                                       }
                                     };
                help2Out :- xExplAsso:help2Out;}.

new2xStmts  = { xStmt:help2In :- help2In;
                xExplAsso:help2In := { xExplAsso:help2In = xStmt:help2Out;
                                       if(rflag2 != noside)
                                       {
                                         Pattern(&xStmt,&vflag2,&tflag2,
                                                 helpstmt);
                                         rflag2 = noside;
                                       }
                                     };
                help2Out :- xExplAsso:help2Out;}.

⟨transhelp2⟩

END Transhelp2
```

Für *transhelp2* siehe Abschnitt C.42.5

C.17.1 Die C-Funktion `testident`

`testident` testet, ob `Ident` als freie Variable (`vflag`) im Muster `tflag` vorkommt.

⟨*mysetc*⟩+≡

```
ISET testident(CODE tflag, DSET vflag, tIdent Ident)
{
   switch(tflag)
   {
      case A1: case A2: case A9:
               if(vflag->A.a1.r1sidename == Ident
                       || vflag->A.a1.r2sidename == Ident)
               {
                  return rside;
               }
               else
               {
                  return noside;
               }
               break;
     case A3: case A11:
               if(vflag->A.a3.r1sidename == Ident
                       || vflag->A.a3.r2sidename == Ident)
               {
                  return rside;
               }
               else
               {
                  return noside;
               }
               break;
      case A4: case A5:
               if(vflag->A.a4.r1sidename == Ident
                       || vflag->A.a4.r2sidename == Ident)
               {
                  return rside;
               }
               else
               {
                  return noside;
               }
               break;
     case A6: if(vflag->A.a6.rsidename == Ident)
               {
                  return rside;
               }
               else
               {
```

```
            return noside;
         }
         break;
case A7: if(vflag->A.a8.rsidename == Ident)
         {
            return rside;
         }
         else
         {
            return noside;
         }
         break;
case A8: if(vflag->A.a8.rsidename == Ident)
         {
            return rside;
         }
         else
         {
            return noside;
         }
         break;
case A10: if(vflag->A.a10.rsidename == Ident)
         {
            return rside;
         }
         else
         {
            return noside;
         }
         break;
case A12: if(vflag->A.a12.r1sidename == Ident
            || vflag->A.a12.r2sidename == Ident)
         {
            return rside;
         }
         else
         {
            return noside;
         }
         break;
case A13: case A14: case A15: case A16: case A17: case A24: case A25:
         if(vflag->A.a13.r1sidename == Ident
            || vflag->A.a13.r2sidename == Ident)
         {
            return rside;
         }
         else
```

```
                {
                   return noside;
                }
                break;
       case A18: case A19:
                if(vflag->A.a18.r1sidename == Ident
                   || vflag->A.a18.r2sidename == Ident
                   || vflag->A.a18.r3sidename == Ident)
                {
                   return rside;
                }
                else
                {
                   return noside;
                }
                break;
       case A20: case A21: case A22: case A23:
                if(vflag->A.a20.r1sidename == Ident
                   || vflag->A.a20.r2sidename == Ident
                   || vflag->A.a20.r3sidename == Ident)
                {
                   return rside;
                }
                else
                {
                   return noside;
                }
                break;
       case A26:
                if(vflag->A.a26.rsidename == Ident)
                {
                   return rside;
                }
                else
                {
                   return noside;
                }
                break;
     }
  }
```

$\langle myseth \rangle{+}{\equiv}$

```
  extern ISET testident(CODE, DSET, tIdent);
```

C.18 Die *puma*-Funktion `Delete`

Nachdem nun endgültig entschieden ist, daß transformiert werden darf, löscht die Funktion `Pattern` den applikativen Ausdruck mit der Kennung `transatt` und den Variablen aus `variable`.

$\langle transpuma \rangle {+}{\equiv}$

```
PROCEDURE Delete (REF del:Tree, variable:DSET, transatt:CODE)
new1xStmts(_,_,_,_,_,_,_,_,newstmts:xStmts,stmt:xStmt,_,_),_,_
? Delete(newstmts, variable, transatt);
  { Pattern(&stmt,&variable,&transatt,delete);}; .

new2xStmts(_,_,_,_,_,_,_,_,stmt:xStmt,_,_),_,_
? { Pattern(&stmt,&variable,&transatt,delete);}; .


_,_,_
? { Pattern(&del,&variable,&transatt,delete);}; .
```

C.19 Der Attributauswerter `Output`

Der Attributauswerter `Output` schreibt den transformierten AST in die Ausgabedatei, wobei an einigen Stellen noch Transformationen „auswertet“ und die Position der Schlüsselwörter und Terminals in der Eingabedatei berücksichtigt werden.

Auch für diesen Attributauswerter benötigen wir ein `THREAD`-Attribut.

Für jedes Nonterminal werden die Schlüsselwörter und die Terminals ausgegeben, wobei durch das `THREAD`-Attribut die Reihenfolge so festgelegt ist, daß ein gültiges ProSet-Programm ausgegeben wird.

⟨*trans.cg*⟩+≡

```
MODULE OutputInt

EVAL Output

IMPORT
{
#include "myprint.h"
#include "Idents.h"
#include "transform.h"
#include "TransAtt.h"
}

DECLARE
xProgDefn xProgBody xPHMDefn xParamList xParamMode = [ write THREAD ] .
xModImport xModExport xIdList xRdParamList xImplAsso = [ write THREAD ] .
xDecls xDecl xSingleVar xDeclKey xPersDecl xStmts = [ write THREAD ] .
xExplAsso xHandAsso xStmt xStmtSignal xStmtNotify xId = [ write THREAD ] .
xUnOp xStdIO xExpr xFrom xActuList xElIfStmt xElIfStmts = [ write THREAD ] .
xElseStmts xCaseStmts xExprList xLabel xLoops xTemplate = [ write THREAD ] .
xTempCond xExprFormal xInto xFormal xIterator xSimpleIts = [ write THREAD ] .
xSimpleIt xMapSel xLValue xSelector xFormer xInstantiate = [ write THREAD ] .
xInstExport xInstImport xQualId xLambda xQuantifier = [ write THREAD ] .
xQualifier xElIfExprs xElIfExpr xElseExpr xCaseExprs = [ write THREAD ] .
xBinOp id intlit floatlit str xTypeList = [ write THREAD ] .

END OutputInt



MODULE Output

EVAL Output
```

`tflag` und `vflag` benötigen wir für die Funktion `Statement`.

⟨*trans.cg*⟩+≡

```
GLOBAL
{
CODE tflag;       /* tflag = transatt */
DSET vflag;       /* vflag = variable */
int loopflag = 0; /* loopflag = 1 : bin in einer Schleife */
}

RULE
```

Bei `new1xStmts` und `new2xStmts` wird entsprechend der vorgefundenen Anweisung durch `Statement` differenzierter Code eingefügt, sofern `new1xStmts` bzw. `new2xStmts` in einer Schleife stehen.

Die Funktion `putmykey` schreibt das angegebene Schlüsselwort in die Ausgabedatei. Die Funktion `lineprint` gibt line-Direktiven aus.

⟨*trans.cg*⟩+≡
```
new1xStmts  = { xStmts:writeIn :- writeIn;
                xStmt:writeIn := { if(loopflag == 1)
                                   {
                                      xStmt:writeIn = Statement(xExplAsso,
                                                          tflag, &xStmt,vflag,
                                                          xStmts:writeOut);
                                   }
                                   else
                                   {
                                      xStmt:writeIn = xStmts:writeOut;
                                   }
                                 };
                xExplAsso:writeIn :- xStmt:writeOut;
                writeOut := { writeOut = putmykey(";", pos,
                                             xExplAsso:writeOut);
                              if(loopflag == 1 && xStmt:writeIn == 1)
                              {
                                 lineprint();fprintf(ofile,"\n");
                                 fprintf(ofile,
                                     "\n-- end insert transformation code\n");
                                 linedirflag = 1;
                              }
                              else if(loopflag == 1 && xStmt:writeIn == 2)
                              {
                                 lineprint();fprintf(ofile,"\n");
                                 Pattern(&xStmt, &vflag, &tflag, postder);
                                 fprintf(ofile,
                                      "\n-- end insert transformation code\n");
                                 linedirflag = 1;
                              }
                           };}.

new2xStmts  = { xStmt:writeIn := { if(loopflag == 1)
                                   {
                                      xStmt:writeIn = Statement(xExplAsso,
                                                          tflag,&xStmt, vflag,
                                                          writeIn);
                                   }
                                   else
                                   {
```

```
                        xStmt:writeIn = writeIn;
                     }
                   };
xExplAsso:writeIn :- xStmt:writeOut;
writeOut := { writeOut = putmykey(";", pos,
                                    xExplAsso:writeOut);
              if(loopflag == 1 && xStmt:writeIn == 1)
              {
                 lineprint();fprintf(ofile,"\n");
                 fprintf(ofile,
                      "\n-- end insert transformation code\n");
                 linedirflag = 1;
              }
              else if(loopflag == 1 && xStmt:writeIn == 2)
              {
                 lineprint();fprintf(ofile,"\n");
                 Pattern(&xStmt, &vflag, &tflag, postder);
                 fprintf(ofile,
                      "\n-- end insert transformation code\n");
                 linedirflag = 1;
              }
           };}.
```

Hier werden `vflag`, `tflag` und `loopflag` für die Funktion `Statement` gesetzt und, falls `transatt != noset`, der Initialisierungscode vor der Schleife eingefügt (`inittrans`).

⟨*trans.cg*⟩+≡

```
xLoops = { writeOut := { writeOut = writeIn;
                         vflag = variable;
                         tflag = transatt;};}.

new6xLoops = { xStmts:writeIn := putmykey("loop", pos, writeIn);
               writeOut :- xStmts:writeOut;}.

new7xLoops = { xIterator:writeIn := { xIterator:writeIn = writeIn;
                                      loopflag = 1;
                                      vflag = variable;
                                      tflag = transatt;
                                      if(transatt != noset)
                                      {
                                        inittrans(transatt, variable);
                                      }
                                      putmykey("for", pos1, writeIn);};
               xStmts:writeIn := putmykey("do", pos2, xIterator:writeOut);
               writeOut := { writeOut = xStmts:writeOut;
                             loopflag = 0;};}.

new8xLoops = { xExpr:writeIn := { xExpr:writeIn = writeIn;
                                  loopflag = 1;
                                  vflag = variable;
                                  tflag = transatt;
                                  if(transatt != noset)
                                  {
                                     inittrans(transatt, variable);
                                  }
                                  putmykey("while", pos1, 1);};
               xStmts:writeIn := putmykey("do", pos2, xExpr:writeOut);
               writeOut := { writeOut = xStmts:writeOut;
                             loopflag = 0;};}.

new9xLoops = { xIterator:writeIn := { xIterator:writeIn = writeIn;
                                      loopflag = 1;
                                      vflag = variable;
                                      tflag = transatt;
                                      if(transatt != noset)
                                      {
                                        inittrans(transatt, variable);
                                      }
                                      putmykey("whilefound", pos1, writeIn);};
               xStmts:writeIn := putmykey("do", pos2, xIterator:writeOut);
                writeOut := { writeOut = xStmts:writeOut;
```

```
                              loopflag = 0;};}.

new10xLoops = { xStmts:writeIn := { xStmts:writeIn = writeIn;
                                    loopflag = 1;
                                    vflag = variable;
                                    tflag = transatt;
                                    if(transatt != noset)
                                    {
                                       inittrans(transatt, variable);
                                    }
                                    putmykey("repeat", pos1, writeIn);};
                xExpr:writeIn := putmykey("until", pos2, xStmts:writeOut);
                writeOut := { writeOut = xExpr:writeOut;
                              loopflag = 0;};}.
```

Die folgende Regel startet den Attributauswerter und fügt, nachdem alle Nonterminals durchlaufen wurden, ein letztes „`\n`"in die Ausgabedatei (`putmyend`).

⟨*trans.cg*⟩+≡
```
xInitChain = { xProgDefn:writeIn := 1;
               => putmyend(xProgDefn:writeOut);}.
```

`putmyid`, `putmyint`, `putmyfloat` und `putmystr` fügen Bezeichner, ganze Zahlen und reelle Zahlen sowie Zeichenketten in die Ausgabedatei ein.

⟨*trans.cg*⟩+≡
```
id = { writeOut := putmyid( Ident, Pos, Length, writeIn);}.
intlit = { writeOut := putmyint( Integer, Pos, Length, writeIn);}.
floatlit = { writeOut := putmyfloat( Float, Pos, Length, writeIn);}.
str = { writeOut := putmystr( Str, Pos, Length, writeIn);}.

⟨output⟩

END Output
```

Für *output* siehe Abschnitt C.42.6

C.19.1 Die *puma*-Funktion `Statement`

Wurde bei `stmt` keine Ausnahme angegeben (`new2xExplAsso`), wird die Funktion `Pattern` aufgerufen. Der Rückgabewert der Funktion `Pattern` wird durchgereicht. An dem Rückgabewert erkennt der Attributauswerter, wie er weiter zu verfahren hat.

Rückgabewert	Arbeitsweise des Attributauswerters `Output`
0	es wurde keine Transformation durchgeführt; der Attributauswerter arbeitet normal weiter;
1	Output schreibt den Kommentar „`-- end insert;`“ in die Ausgabedatei;
2	Output ruft die Funktion `Pattern` für eine Nachableitung auf; Output schreibt den Kommentar „`-- end insert;`“ in die Ausgabedatei;

⟨*transpuma*⟩+≡

```
FUNCTION Statement(expl:xExplAsso,tflag:CODE,REF stmt:xStmt,vflag:DSET,
                   att:int) int
LOCAL
{
   int Return;
}
new2xExplAsso,_,_,_,_ RETURN Return;
? { Return = Pattern(&stmt,&vflag,&tflag,preder);}; .

_,_,_,_,_ RETURN 0; ? .
```

C.20 Die Initialisierung der virtuellen Variablen: Die C-Funktion `inittrans`

`inittrans` fügt für die einzelnen Muster die Initialisierung vor die Schleife in der Ausgabedatei ein und setzt `TRANSFLAG` auf `1`, um anzuzeigen, daß eine Transformation in diesem Transformationsdurchlauf stattgefunden hat. `linedirflag = 1` zeigt an, daß eine line-Direktive ausgegeben werden muß.

⟨*mytransc*⟩+≡

```
void inittrans(CODE transatt, DSET variable)
{
   if(transatt != noset)
   {
      TRANSFLAG = 1;
      lineprint();
      fprintf(ofile, "\n-- begin insert initialisation code\n");
      switch(transatt)
      {
       case A1: initA1(variable); break;
       case A2: initA2(variable); break;
       case A3: initA3(variable); break;
       case A4: initA4(variable); break;
       case A5: initA4(variable); break;
       case A6: initA6(variable); break;
       case A7: initA8(variable); break;
       case A8: initA8(variable); break;
       case A9: initA9(variable); break;
       case A10: initA10(variable); break;
       case A11: initA11(variable); break;
       case A12: initA12(variable); break;
       case A13: initA13(variable); break;
       case A14: initA13(variable); break;
       case A15: initA13(variable); break;
       case A16: initA16(variable); break;
       case A17: initA16(variable); break;
       case A18: initA18(variable); break;
       case A19: initA19(variable); break;
       case A20: initA20(variable); break;
       case A21: initA20(variable); break;
       case A22: initA22(variable); break;
       case A23: initA22(variable); break;
       case A24: initA24(variable); break;
       case A25: initA24(variable); break;
       case A26: initA26(variable); break;
      }
      fprintf(ofile, "\n-- end insert initialisation code\n");
      linedirflag = 1;
```

```
    }
}
```

⟨*mytransh*⟩+≡

```
extern void inittrans(CODE, DSET);
```

C.21 Die C-Funktion `initA1`

`initA1` initialisiert die virtuelle Variable für das Muster `A1`.

⟨*mytransc*⟩+≡

```
void initA1(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := { ");
   WriteIdent(ofile, (variable->A.a1).iterator);
   fprintf(ofile, " : ");
   WriteIdent(ofile, (variable->A.a1).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a1).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a1).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a1).r2sidename);
   fprintf(ofile, " };");
}
```

⟨*mytransh*⟩+≡

```
extern void initA1(DSET);
```

C.22 Die C-Funktion `initA2`

`initA2` initialisiert die virtuelle Variable für das Muster `A2`.

⟨*mytransc*⟩+≡

```
void initA2(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := { ");
   WriteIdent(ofile, (variable->A.a1).iterator);
   fprintf(ofile, " : ");
   WriteIdent(ofile, (variable->A.a1).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a1).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a1).iterator);
   fprintf(ofile, " notin ");
   WriteIdent(ofile, (variable->A.a1).r2sidename);
   fprintf(ofile, " };");
}
```

⟨*mytransh*⟩+≡

```
extern void initA2(DSET);
```

C.23 Die C-Funktion `initA3`

`initA3` initialisiert die virtuelle Variable für das Muster `A3`.

⟨*mytransc*⟩+≡

```
void initA3(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := { ");
   WriteIdent(ofile, (variable->A.a3).iterator);
   fprintf(ofile, " : ");
   WriteIdent(ofile, (variable->A.a3).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a3).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a3).r2sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.a3).iterator);
   fprintf(ofile, ") = %d};", (variable->A.a3).integer);
}
```

⟨*mytransh*⟩+≡

```
extern void initA3(DSET);
```

C.24 Die C-Funktion `initA4`

`initA4` initialisiert die virtuelle Variable für die Muster `A4` und `A5`.

⟨*mytransc*⟩+≡

```
void initA4(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := { [");
   WriteIdent(ofile, (variable->A.a4).r1sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.a4).iterator);
   fprintf(ofile, "),");
   WriteIdent(ofile, (variable->A.a4).iterator);
   fprintf(ofile, "] : ");
   WriteIdent(ofile, (variable->A.a4).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a4).r2sidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡

```
extern void initA4(DSET);
```

C.25 Die C-Funktion `initA6`

`initA6` initialisiert die virtuelle Variable für das Muster `A6`.

⟨*mytransc*⟩+≡

```
void initA6(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := #");
   WriteIdent(ofile, (variable->A.a6).rsidename);
   fprintf(ofile, ";");
}
```

⟨*mytransh*⟩+≡

```
extern void initA6(DSET);
```

C.26 Die C-Funktion `initA8`

`initA8` initialisiert die virtuelle Variable für das Muster `A8`.

⟨*mytransc*⟩+≡

```
void initA8(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := { [");
   WriteIdent(ofile, (variable->A.a8).iterator1);
   fprintf(ofile, ",#");
   WriteIdent(ofile, (variable->A.a8).rsidename);
   fprintf(ofile, "{");
   WriteIdent(ofile, (variable->A.a8).iterator1);
   fprintf(ofile, "}] : ");
   WriteIdent(ofile, (variable->A.a8).iterator1);
   fprintf(ofile, " in domain ");
   WriteIdent(ofile, (variable->A.a8).rsidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡

```
extern void initA8(DSET);
```

C.27 Die C-Funktion `initA9`

`initA9` initialisiert die virtuelle Variable für das Muster `A9`.

⟨*mytransc*⟩+≡

```
void initA9(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := { ");
   WriteIdent(ofile, (variable->A.a1).iterator);
   fprintf(ofile, ":");
   WriteIdent(ofile, (variable->A.a1).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a1).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a1).r2sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.a1).iterator);
   fprintf(ofile, ")");
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡
```
extern void initA9(DSET);
```

C.28 Die C-Funktion `initA10`

`initA10` initialisiert die virtuelle Variable für das Muster `A10`.

⟨*mytransc*⟩+≡
```
void initA10(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := { ");
   WriteIdent(ofile, (variable->A.a10).iterator);
   fprintf(ofile, ":");
   WriteIdent(ofile, (variable->A.a10).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a10).rsidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡
```
extern void initA10(DSET);
```

C.29 Die C-Funktion `initA11`

`initA11` initialisiert die virtuelle Variable für das Muster `A11`.

⟨*mytransc*⟩+≡
```
void initA11(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := { ");
   WriteIdent(ofile, (variable->A.a3).iterator);
   fprintf(ofile, " : ");
   WriteIdent(ofile, (variable->A.a3).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a3).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a3).r2sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.a3).iterator);
   fprintf(ofile, ") /= %d};", (variable->A.a3).integer);
}
```

⟨*mytransh*⟩+≡
```
extern void initA11(DSET);
```

C.30 Die C-Funktion `initA12`

`initA12` initialisiert die virtuelle Variable für das Muster `A12`.

⟨*mytransc*⟩+≡
```
void initA12(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := {[[ ");
   WriteIdent(ofile, (variable->A.a12).r1sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.a12).iterator1);
   fprintf(ofile, "), ");
   WriteIdent(ofile, (variable->A.a12).iterator2);
   fprintf(ofile, "], ");
   WriteIdent(ofile, (variable->A.a12).iterator1);
   fprintf(ofile, "] : [ ");
   WriteIdent(ofile, (variable->A.a12).iterator2);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a12).iterator1);
   fprintf(ofile, "] in  ");
   WriteIdent(ofile, (variable->A.a12).r2sidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡
```
extern void initA12(DSET);
```

C.31 Die C-Funktion `initA13`

`initA13` initialisiert die virtuelle Variable für das Muster `A13`.

⟨*mytransc*⟩+≡
```
void initA13(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := {[ ");
   WriteIdent(ofile, (variable->A.a13).iterator1);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a13).iterator2);
   fprintf(ofile, "] : [ ");
   WriteIdent(ofile, (variable->A.a13).iterator1);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a13).iterator2);
   fprintf(ofile, "] in ");
   WriteIdent(ofile, (variable->A.a13).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a13).iterator2);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a13).r2sidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡
```
extern void initA13(DSET);
```

C.32 Die C-Funktion initA16

initA16] initialisiert die virtuelle Variable für das Muster [[A16.

⟨*mytransc*⟩+≡

```
void initA16(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := {[ ");
   WriteIdent(ofile, (variable->A.a13).iterator1);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a13).iterator2);
   fprintf(ofile, "] : [ ");
   WriteIdent(ofile, (variable->A.a13).iterator1);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a13).iterator2);
   fprintf(ofile, "] in ");
   WriteIdent(ofile, (variable->A.a13).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a13).iterator2);
   fprintf(ofile, " notin ");
   WriteIdent(ofile, (variable->A.a13).r2sidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡

```
extern void initA16(DSET);
```

C.33 Die C-Funktion `initA18`

`initA18` initialisiert die virtuelle Variable für das Muster `A18`.

⟨*mytransc*⟩+≡
```
void initA18(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := {");
   WriteIdent(ofile, (variable->A.a18).iterator);
   fprintf(ofile, " : ");
   WriteIdent(ofile, (variable->A.a18).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a18).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a18).r2sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.a18).iterator);
   fprintf(ofile, ")");
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a18).r3sidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡
```
extern void initA18(DSET);
```

C.34 Die C-Funktion `initA19`

`initA19` initialisiert die virtuelle Variable für das Muster `A19`.

⟨*mytransc*⟩+≡
```
void initA19(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := {");
   WriteIdent(ofile, (variable->A.a18).iterator);
   fprintf(ofile, " : ");
   WriteIdent(ofile, (variable->A.a18).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a18).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a18).r2sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.a18).iterator);
   fprintf(ofile, ")");
   fprintf(ofile, " notin ");
   WriteIdent(ofile, (variable->A.a18).r3sidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡
```
extern void initA19(DSET);
```

C.35 Die C-Funktion `initA20`

`initA20` initialisiert die virtuelle Variable für das Muster `A20`.

⟨*mytransc*⟩+≡

```
void initA20(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := {[");
   WriteIdent(ofile, (variable->A.a20).iterator1);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a20).iterator2);
   fprintf(ofile, "] : [");
   WriteIdent(ofile, (variable->A.a20).iterator1);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a20).iterator2);
   fprintf(ofile, "] in ");
   WriteIdent(ofile, (variable->A.a20).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a20).r2sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.a20).iterator2);
   fprintf(ofile, ")");
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a20).r3sidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡

```
extern void initA20(DSET);
```

C.36 Die C-Funktion `initA22`

`initA22` initialisiert die virtuelle Variable für das Muster `A22`.

⟨*mytransc*⟩+≡
```
void initA22(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := {[");
   WriteIdent(ofile, (variable->A.a20).iterator1);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a20).iterator2);
   fprintf(ofile, "] : [");
   WriteIdent(ofile, (variable->A.a20).iterator1);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a20).iterator2);
   fprintf(ofile, "] in ");
   WriteIdent(ofile, (variable->A.a20).r1sidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a20).r2sidename);
   fprintf(ofile, "(");
   WriteIdent(ofile, (variable->A.a20).iterator2);
   fprintf(ofile, ")");
   fprintf(ofile, " notin ");
   WriteIdent(ofile, (variable->A.a20).r3sidename);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡
```
extern void initA22(DSET);
```

C.37 Die C-Funktion `initA24`

`initA24` initialisiert die virtuelle Variable für das Muster `A24`.

⟨*mytransc*⟩+≡

```
void initA24(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := {[");
   WriteIdent(ofile, (variable->A.a13).iterator1);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a13).iterator2);
   fprintf(ofile, "] : ");
   WriteIdent(ofile, (variable->A.a13).iterator2);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a13).r1sidename);
   fprintf(ofile, ", ");
   WriteIdent(ofile, (variable->A.a13).iterator1);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a13).r2sidename);
   fprintf(ofile, "{");
   WriteIdent(ofile, (variable->A.a13).iterator2);
   fprintf(ofile, "}};");
}
```

⟨*mytransh*⟩+≡

```
extern void initA24(DSET);
```

C.38 Die C-Funktion `initA26`

`initA26` initialisiert die virtuelle Variable für das Muster `A26`.

⟨*mytransc*⟩+≡
```
void initA26(DSET variable)
{
   WriteIdent(ofile, variable->newlsidename);
   fprintf(ofile, " := {");
   WriteIdent(ofile, (variable->A.a26).iterator);
   fprintf(ofile, " : ");
   WriteIdent(ofile, (variable->A.a26).iterator);
   fprintf(ofile, " in ");
   WriteIdent(ofile, (variable->A.a26).rsidename);
   fprintf(ofile, " | ");
   WriteIdent(ofile, (variable->A.a26).iterator);
   WriteBinOp((variable->A.a26).operator1);
   fprintf(ofile, " %d ", variable->A.a26.integer1);
   WriteBinOp((variable->A.a26).operator2);
   fprintf(ofile, " %d ", variable->A.a26.integer2);
   fprintf(ofile, "};");
}
```

⟨*mytransh*⟩+≡
```
extern void initA26(DSET);
```

C.39 Die C-Funktion `WriteBinOp`

Die Funktion `WriteBinOp` schreibt den Operator in die Ausgabedatei.

⟨*mytransc*⟩+≡

```
void WriteBinOp(unsigned short operator)
{
   switch(operator)
   {
      case knew1xBinOp : fprintf(ofile, " or "); break;
      case knew2xBinOp : fprintf(ofile, " or % "); break;
      case knew3xBinOp : fprintf(ofile, " and "); break;
      case knew4xBinOp : fprintf(ofile, " and % "); break;
      case knew5xBinOp : fprintf(ofile, " = "); break;
      case knew6xBinOp : fprintf(ofile, " /= "); break;
      case knew7xBinOp : fprintf(ofile, " < "); break;
      case knew8xBinOp : fprintf(ofile, " <= "); break;
      case knew9xBinOp : fprintf(ofile, " > "); break;
      case knew10xBinOp : fprintf(ofile, " >= "); break;
      case knew11xBinOp : fprintf(ofile, " in "); break;
      case knew12xBinOp : fprintf(ofile, " notin "); break;
      case knew13xBinOp : fprintf(ofile, " subset "); break;
      case knew14xBinOp : fprintf(ofile, " = % "); break;
      case knew15xBinOp : fprintf(ofile, " /= % "); break;
      case knew16xBinOp : fprintf(ofile, " < % "); break;
      case knew17xBinOp : fprintf(ofile, " <= % "); break;
      case knew18xBinOp : fprintf(ofile, " > % "); break;
      case knew19xBinOp : fprintf(ofile, " >= % "); break;
      case knew20xBinOp : fprintf(ofile, " in % "); break;
      case knew21xBinOp : fprintf(ofile, " notin % "); break;
      case knew22xBinOp : fprintf(ofile, " subset % "); break;
      case knew23xBinOp : fprintf(ofile, " with "); break;
      case knew24xBinOp : fprintf(ofile, " less "); break;
      case knew25xBinOp : fprintf(ofile, " lessf "); break;
      case knew27xBinOp : fprintf(ofile, " with % "); break;
      case knew28xBinOp : fprintf(ofile, " less % "); break;
      case knew29xBinOp : fprintf(ofile, " lessf % "); break;
      case knew31xBinOp : fprintf(ofile, " + "); break;
      case knew32xBinOp : fprintf(ofile, " - "); break;
      case knew33xBinOp : fprintf(ofile, " max "); break;
      case knew34xBinOp : fprintf(ofile, " min "); break;
      case knew35xBinOp : fprintf(ofile, " + % "); break;
      case knew36xBinOp : fprintf(ofile, " - % "); break;
      case knew37xBinOp : fprintf(ofile, " max % "); break;
      case knew38xBinOp : fprintf(ofile, " min % "); break;
      case knew39xBinOp : fprintf(ofile, " * "); break;
      case knew40xBinOp : fprintf(ofile, " / "); break;
```

```
      case knew41xBinOp : fprintf(ofile, " mod "); break;
      case knew42xBinOp : fprintf(ofile, " * % "); break;
      case knew43xBinOp : fprintf(ofile, " / % "); break;
      case knew44xBinOp : fprintf(ofile, " mod % "); break;
      case knew45xBinOp : fprintf(ofile, " ** "); break;
      case knew46xBinOp : fprintf(ofile, " ** % "); break;
      case knew47xBinOp : fprintf(ofile, " npow "); break;
   }
}
```

$\langle mytransh \rangle{+}{\equiv}$

```
extern void WriteBinOp(unsigned short operator);
```

C.40 Die Ausgabe der Terminals

Für die Ausgabe der Terminals wurden die folgenden Funktionen verwendet:

C.40.1 Die C-Funktion `putmybetween`

Schreibt line-Direktiven und `Length` viele Leerzeichen in die Ausgabedatei `ofile`.

⟨*myprintc*⟩+≡

```
void putmybetween(tPosition pos, int Length)
{
   static int oldline = 1;
   static int oldcolumn = 1;
   static int oldlength = 0;
   static int oldlineoffset = 0;
   static int oldincludefile = 0;
   int line, col, i;

   line = pos.Line - oldline;
   if(line != 0)
   {
      fprintf(ofile, "\n");
      if(!linedir &&
         ((oldlineoffset != pos.LineOffset||oldincludefile != pos.IncludeFile)
          ||linedirflag ==1))
      {
         switch(pos.IncludeFile)
         {
          case 0:
            fprintf(ofile, "\n@line %d \"%s\" \n", pos.Line, ifile);
            break;
          default:
            fprintf(ofile, "\n@line %d \"", pos.Line - pos.LineOffset);
            WriteIdent(ofile, pos.IncludeFile);
            fprintf(ofile, "\"\n");
         }
      linedirflag = 0;
      }
      col = pos.Column - 1;
      for(i = 0 ; i < col ; i++)
      {
         fprintf(ofile, "%s", " ");
      }
   }
   else
   {
```

```
      if(!linedir &&
         ((oldlineoffset != pos.LineOffset||oldincludefile != pos.IncludeFile)
          ||linedirflag ==1))
      {
         switch(pos.IncludeFile)
         {
          case 0:
            fprintf(ofile, "\n@line %d \"%s\" \n", pos.Line, ifile);
            break;
          default:
            fprintf(ofile, "\n@line %d \"", pos.Line - pos.LineOffset);
            WriteIdent(ofile, pos.IncludeFile);
            fprintf(ofile, "\"\n");
         }
      linedirflag = 0;
      }
      col = pos.Column - (oldcolumn + oldlength);
      for(i = 0 ; i < col ; i++)
      {
         fprintf(ofile, "%s", " ");
      }
   }
   oldline = pos.Line;
   oldcolumn = pos.Column;
   oldlineoffset = pos.LineOffset;
   oldincludefile = pos.IncludeFile;
   oldlength = Length;
}
```

C.40.2 Die C-Funktion `putmykey`

Schreibt das Schlüsselwort „`key`“ in die Ausgabedatei.

⟨*myprintc*⟩+≡

```
int putmykey(const char *key, tPosition pos, int att)
{
   int Length;

   Length = strlen(key);
   putmybetween(pos, Length);
   fprintf(ofile, "%s", key);
   return 1;
}
```

⟨*myprinth*⟩+≡

```
extern int putmykey(const char *, tPosition, int);
```

C.40.3 Die C-Funktion `putmyid`

Schreibt den Bezeichner „`id`“ in die Ausgabedatei.

$\langle myprintc \rangle$+≡

```
int putmyid(tIdent id, tPosition pos, int Length, int att)
{
   putmybetween(pos, Length);
   WriteIdent(ofile, id);
   return 1;
}
```

$\langle myprinth \rangle$+≡

```
extern int putmyid(tIdent, tPosition, int, int);
```

C.40.4 Die C-Funktion `putmyint`

Schreibt den Integer „`integer`“ in die Ausgabedatei.

$\langle myprintc \rangle$+≡

```
int putmyint(const int integer, tPosition pos, int Length, int att)
{
   putmybetween(pos, Length);
   fprintf(ofile, "%d", integer);
   return 1;
}
```

$\langle myprinth \rangle$+≡

```
extern int putmyint(const int, tPosition, int, int);
```

C.40.5 Die C-Funktion `putmyfloat`

Schreibt die reelle Zahl „`real`“ in die Ausgabedatei.

$\langle myprintc \rangle$+≡

```
int putmyfloat(const double real, tPosition pos, int Length, int att)
{
   putmybetween(pos, Length);
   fprintf(ofile, "%f", real);
   return 1;
}
```

$\langle myprinth \rangle$+≡

```
extern int putmyfloat(const double, tPosition, int, int);
```

C.40.6 Die C-Funktion `putmystr`

Schreibt die Zeichenkette „`str`“ in die Ausgabedatei.

$\langle myprintc \rangle + \equiv$

```
int putmystr(tStringRef str, tPosition pos, int Length, int att)
{
   putmybetween(pos, Length);
   WriteString(ofile, str);
   return 1;
}
```

$\langle myprinth \rangle + \equiv$

```
extern int putmystr(tStringRef, tPosition, int, int);
```

C.40.7 Die C-Funktion `putmyend`

Schreibt ein letztes „`\n`“ in die Ausgabedatei.

$\langle myprintc \rangle + \equiv$

```
void putmyend(int att)
{
   fprintf(ofile, "\n");
}
```

$\langle myprinth \rangle + \equiv$

```
extern void putmyend(int);
```

C.41 Der Attributauswerter `Muster`

Dieser Attributauswerter gibt die Terminals und Nonterminals des AST in Preorder aus. `Muster` wird nur benutzt, um für neue Muster die Grammatikstruktur zu erhalten.

⟨*trans.cg*⟩+≡

```
MODULE MusterInt

EVAL Muster

DECLARE
xProgDefn xProgBody xPHMDefn xParamList xParamMode xModImport xModExport
xIdList xRdParamList xImplAsso xDecls xDecl xSingleVar xDeclKey xPersDecl
xStmts xExplAsso xHandAsso xStmt xStmtSignal xStmtNotify xId xUnOp xStdIO
xExpr xFrom xActuList xElIfStmt xElIfStmts xElseStmts xCaseStmts xExprList
xLabel xLoops xTemplate xTempCond xExprFormal xInto xFormal xIterator
xSimpleIts xSimpleIt xMapSel xLValue str xSelector xFormer xInstantiate
xInstExport xInstImport xQualId xLambda xQuantifier xQualifier xElIfExprs id
xElIfExpr xElseExpr xCaseExprs xBinOp intlit floatlit
xTypeList xInitChain = [ muster THREAD ] .

END MusterInt

MODULE Muster

EVAL Muster

IMPORT
{
#include "global.h"
}

RULE
xInitChain = { xProgDefn:musterIn := fprintf(ofile, "xInitChain(");
               musterOut := fprintf(ofile, ")\n", xProgDefn:musterOut); }.

xProgDefn = { xId1:musterIn := fprintf(ofile, "xProgDefn(", musterIn);
              xProgBody:musterIn := fprintf(ofile, ",", xId1:musterOut);
              xId2:musterIn := fprintf(ofile, ",", xProgBody:musterOut);
              musterOut := fprintf(ofile, ")\n", xId2:musterOut); }.

xId = { id:musterIn := fprintf(ofile, "xId(", musterIn);
        musterOut := fprintf(ofile, ")\n", id:musterOut); }.

xProgBody = { xDecls:musterIn := fprintf(ofile, "xProgBody(", musterIn);
              xStmts:musterIn := fprintf(ofile, ",", xDecls:musterOut);
              xPHMDefn:musterIn := fprintf(ofile, ",", xStmts:musterOut);
```

```
              musterOut := fprintf(ofile, ")\n", xPHMDefn:musterOut); }.

xPHMDefn = { musterOut :- musterIn; }.

new1xPHMDefn = { xId1:musterIn := fprintf(ofile, "new1xPHMDefn(", musterIn);
                 xParamList:musterIn := fprintf(ofile, ",", xId1:musterOut);
                 xProgBody:musterIn := fprintf(ofile, ",", xParamList:musterOut);
                 xId2:musterIn := fprintf(ofile, ",", xProgBody:musterOut);
                 musterOut := fprintf(ofile, ")\n", xId2:musterOut); }.

new2xPHMDefn = { xId1:musterIn := fprintf(ofile, "new2xPHMDefn \n", musterIn);
                 xModImport:musterIn := fprintf(ofile, ",", xId1:musterOut);
                 xModExport:musterIn := fprintf(ofile, ",", xModImport:musterOut) ;
                 xProgBody:musterIn := fprintf(ofile, ",", xModExport:musterOut);
                 xId2:musterIn := fprintf(ofile, ",", xProgBody:musterOut);
                 musterOut := fprintf(ofile, ")\n", xId2:musterOut);}.

new3xPHMDefn = { xId1:musterIn := fprintf(ofile, "new3xPHMDefn \n", musterIn);
                 xRdParamList:musterIn := fprintf(ofile, ",", xId1:musterOut);
                 xImplAsso:musterIn := fprintf(ofile, ",", xRdParamList:musterOut);
                 xProgBody:musterIn := fprintf(ofile, ",", xImplAsso:musterOut);
                 xId2:musterIn := fprintf(ofile, ",", xProgBody:musterOut);
                 musterOut := fprintf(ofile, ")\n", xId2:musterOut);}.

new4xPHMDefn = { No1:musterIn := fprintf(ofile, "new4xPHMDefn \n", musterIn);
                 No2:musterIn := fprintf(ofile, ",", No1:musterOut);
                 musterOut := fprintf(ofile, ")\n", No2:musterOut);} .

new5xPHMDefn = { musterOut := fprintf(ofile, "new5xPHMDefn \n", musterIn); } .

id = { musterOut := { fprintf(ofile, "id(", musterIn);
                      WriteIdent(ofile, Ident);
                      musterOut = fprintf(ofile, ")\n");};} .
```

⟨*muster*⟩

```
END Muster
```

Für *muster* siehe C.42.7 auf Seite 533.

C.42 Fortsetzungen von Attributauswertern und Dateien

C.42.1 Die Fortsetzung der konkreten Grammatik

⟨*transpars.fort*⟩≡

```
/* Parameter list: */
xParamList = <
  new1xParamList = '(' ')' .
  new2xParamList = '(' xcParams ')' .
>.
xcParams = <
  new1xcParams = xcParams ',' xParamMode xId .
  new2xcParams = xParamMode xId .
>.
/* Parameter mode: */
xParamMode = <
  new1xParamMode = .
  new2xParamMode = 'rd' .
  new3xParamMode = 'rw' .
  new4xParamMode = 'wr' .
>.

xModImport = <
  new1xModImport = 'import' xIdList ';' .
  new2xModImport = .
>.
xModExport = <
  new1xModExport = 'export' xIdList ';' .
  new2xModExport = .
>.
xIdList = <
  new1xIdList = xIdList ',' xId .
  new2xIdList = xId .
>.

/* Rd-Parameter list: */
xRdParamList = <
  new1xRdParamList = '(' ')' .
  new2xRdParamList = '(' xIdList ')' .
>.

/* Implicit handler association: */
xImplAsso = <
  new1xImplAsso = 'for' xIdList .
  new2xImplAsso = 'for' 'others' .
  new3xImplAsso = .
>.
```

```
/*****************************************************************************
        Declarations:
*****************************************************************************/
xDecls = <
  new1xDecls = xDecls xDecl .
  new2xDecls = .
>.
xDecl = <
  new1xDecl = xDeclKey xcVars xExplAsso ';' .
  new2xDecl = xPersDecl xIdList ':' xExpr xExplAsso ';' .
>.
xcVars = <
  new1xcVars = xcVars ',' xSingleVar .
  new2xcVars = xSingleVar .
>.
xSingleVar = <
  new1xSingleVar = xId ':=' xExpr .
  new2xSingleVar = xId .
>.
xDeclKey = <
  new1xDeclKey = 'visible' .
  new2xDeclKey = 'hidden' .
  new3xDeclKey = 'visible' 'constant' .
  new4xDeclKey = 'hidden' 'constant' .
  new5xDeclKey = 'constant' .
>.

xPersDecl = <
  new1xPersDecl = 'visible' 'persistent'.
  new2xPersDecl = 'hidden' 'persistent' .
  new3xPersDecl = 'persistent' .
  new4xPersDecl = 'visible' 'persistent' 'constant'.
  new5xPersDecl = 'hidden' 'persistent' 'constant'.
  new6xPersDecl = 'persistent' 'constant' .
>.
/*****************************************************************************
        Statements:
*****************************************************************************/
xcBeginStmts = 'begin' xStmts .
xStmts = <
  new1xStmts = xStmts xStmt xExplAsso ';' .
  new2xStmts = xStmt xExplAsso ';' .
>.

xExplAsso = <
  new1xExplAsso = xExplAsso 'when' xHandAsso .
  new2xExplAsso = .
```

```
>.

xHandAsso = <
  new1xHandAsso = xIdList 'use' xId .
  new2xHandAsso = 'others' 'use' xId .
>.

xStmt = <
  new1xStmt = 'pass' .    /* Simple Statements: */
  new2xStmt = 'stop' .
  new3xStmt = 'stop' xExpr .
  new4xStmt = 'return' xExpr .
  new5xStmt = 'return' .
  new6xStmt = 'return' 'commit' xExpr .
  new7xStmt = 'return' 'commit' .
  new8xStmt = 'resume' xExpr .
  new9xStmt = 'resume' .
  new10xStmt = xStmtSignal .
  new11xStmt = xStmtNotify .
  new12xStmt = 'escape' xId xActuList .
  new13xStmt = xStdIO xActuList .
  new14xStmt = 'any' '(' xcSimpleLV ',' xExpr ')' .
  new15xStmt = 'rany' '(' xcSimpleLV ',' xExpr ')' .
  new16xStmt = 'break' '(' xcSimpleLV ',' xExpr ')' .
  new17xStmt = 'rbreak' '(' xcSimpleLV ',' xExpr ')' .
  new18xStmt = 'len' '(' xcSimpleLV ',' xExpr ')' .
  new19xStmt = 'rlen' '(' xcSimpleLV ',' xExpr ')' .
  new20xStmt = 'lpad' '(' xcSimpleLV ',' xExpr ')' .
  new21xStmt = 'rpad' '(' xcSimpleLV ',' xExpr ')' .
  new22xStmt = 'match' '(' xcSimpleLV ',' xExpr ')' .
  new23xStmt = 'rmatch' '(' xcSimpleLV ',' xExpr ')' .
  new24xStmt = 'notany' '(' xcSimpleLV ',' xExpr ')' .
  new25xStmt = 'rnotany' '(' xcSimpleLV ',' xExpr ')' .
  new26xStmt = 'span' '(' xcSimpleLV ',' xExpr ')' .
  new27xStmt = 'rspan' '(' xcSimpleLV ',' xExpr ')' .
  new28xStmt = xLValue ':=' xExpr .        /* Assignments: */
  new29xStmt = xLValue xBinOp ':=' xExpr .
  new30xStmt = xLValue xAndOp ':=' xExpr .
  new31xStmt = xLValue xOrOp ':=' xExpr .
  new32xStmt = xLValue xFrom xcSimpleLV .
  new33xStmt = xcSimpleLV xActuList .   /* Function calls: */
  new34xStmt = xLambda xActuList .   /* Direct lambda calls: */
  new35xStmt = 'self' xActuList .   /* Recursive lambda calls: */
  new36xStmt = if1:'if' xExpr 'then' xStmts xElIfStmts xElseStmts 'end'
               if2:'if' .  /* Conditional statements: */
  new37xStmt = case1:'case' xExpr xCaseStmts  xElseStmts 'end' case2:'case' .
  new38xStmt = xcLoopStmt 'end' 'loop' .    /* Loop statements: */
```

```
  new39xStmt = xcForStmt 'end' 'for' .
  new40xStmt = xcWhileStmt 'end' 'while' .
  new41xStmt = xcWhilefound 'end' 'whilefound' .
  new42xStmt = xcUntilStmt 'end' 'repeat' .
  new43xStmt = xLabel xLoops 'end' xId .
  new44xStmt = 'quit' .
  new45xStmt = 'quit' xId .
  new46xStmt = 'continue' .
  new47xStmt = 'continue' xId .
  new48xStmt = '||' xExpr .   /* Process spawing statement: */
  new49xStmt = dep1:'deposit' Expr1:xExpr 'at' Expr2:xExpr 'end'
               dep2:'deposit' .   /* Tuple-Space Operations: */
  new50xStmt = dep1:'deposit' Expr1:xExpr 'at' Expr2:xExpr 'blockiffull' 'end'
               dep2:'deposit' .
  new51xStmt = fetch1:'fetch' xcTempList 'at' xExpr xElseStmts 'end'
               fetch2:'fetch' .
  new52xStmt = meet1:'meet' xcTempList 'at' xExpr xElseStmts 'end'
               meet2:'meet' .
  new53xStmt = 'c_fct_call' xId '(' xTypeList ')' .
  new54xStmt = 'getf' '(' xExpr ',' xExprList ')' .
  new55xStmt = 'fget' '(' xExpr ',' xExprList ')' .
  new56xStmt = 'fgetf' '(' xExpr1:xExpr ',' xExpr2:xExpr ',' xExprList ')' .
>.

xStmtSignal = <
  new1xStmtSignal = 'signal' xLValue ':=' xId xActuList .
  new2xStmtSignal = 'signal' xcSimpleLV xActuList .
>.

xStmtNotify = <
  new1xStmtNotify = 'notify' xLValue ':=' xId xActuList .
  new2xStmtNotify = 'notify' xcSimpleLV xActuList .
>.

xStdIO = <
  new1xStdIO = 'put' .
  new2xStdIO = 'eput' .
  new3xStdIO = 'fput' .
  new4xStdIO = 'putf' .
  new5xStdIO = 'eputf' .
  new6xStdIO = 'fputf' .
  new7xStdIO = 'get' .
/*  new8xStdIO = 'fget' . */
/*  new9xStdIO = 'getf' . */
/*  new10xStdIO = 'fgetf' . */
  new11xStdIO = 'fopen' .
  new12xStdIO = 'fclose' .
```

```
>.

xcPrimary = <
  new1xcPrimary = 'any' '(' xcSimpleLV ',' xExpr ')' .
  new2xcPrimary = 'rany' '(' xcSimpleLV ',' xExpr ')' .
  new3xcPrimary = 'break' '(' xcSimpleLV ',' xExpr ')' .
  new4xcPrimary = 'rbreak' '(' xcSimpleLV ',' xExpr ')' .
  new5xcPrimary = 'len' '(' xcSimpleLV ',' xExpr ')' .
  new6xcPrimary = 'rlen' '(' xcSimpleLV ',' xExpr ')' .
  new7xcPrimary = 'match' '(' xcSimpleLV ',' xExpr ')' .
  new8xcPrimary = 'rmatch' '(' xcSimpleLV ',' xExpr ')' .
  new9xcPrimary = 'notany' '(' xcSimpleLV ',' xExpr ')' .
  new10xcPrimary = 'rnotany' '(' xcSimpleLV ',' xExpr ')' .
  new11xcPrimary = 'span' '(' xcSimpleLV ',' xExpr ')' .
  new12xcPrimary = 'rspan' '(' xcSimpleLV ',' xExpr ')' .
  new13xcPrimary = intlit .  /*  Primary Expressions without possible
                                 selections:  */
  new14xcPrimary = floatlit .
  new15xcPrimary = 'true' .
  new16xcPrimary = 'false' .
  new17xcPrimary = 'om' .
  new18xcPrimary = 'atom' .
  new19xcPrimary = 'boolean' .
  new20xcPrimary = 'integer' .
  new21xcPrimary = 'real' .
  new22xcPrimary = 'string' .
  new23xcPrimary = 'tuple' .
  new24xcPrimary = 'set' .
  new25xcPrimary = 'function' .
  new26xcPrimary = 'modtype' .
  new27xcPrimary = 'instance' .
  new28xcPrimary = '{' '}' .
  new29xcPrimary = '[' ']' .
  new30xcPrimary = 'newat' '(' ')' .  /* Newat: */
  new31xcPrimary = xInstantiate . /* module instantiations: */
  new32xcPrimary = xcPriSel . /*Primary Expressions with possible selections:*/
>.

/* Assignments: */
xFrom = <
  new1xFrom = 'from' .
  new2xFrom = 'frome' .
  new3xFrom = 'fromb' .
>.

/* Function calls: */
xActuList = <
```

```
  new1xActuList = '(' xExprList ')' .
  new2xActuList = '(' ')' .
>.

/* Conditional statements: */
xElIfStmt = 'elseif' xExpr 'then' xStmts .
xElIfStmts = <
  new1xElIfStmts = xElIfStmts xElIfStmt .
  new2xElIfStmts = .
>.

xElseStmts = <
  new1xElseStmts = 'else' xStmts .
  new2xElseStmts = .
>.

/* Case statements: */
xCaseStmts = <
  new1xCaseStmts = xCaseStmts xcCaseStmt .
  new2xCaseStmts = xcCaseStmt .
>.
xcCaseStmt = xcCaseList xStmts .
xcCaseList = 'when' xExprList '=>' .

/* Loop statements: */
xLabel = xId ':' .
xLoops = <
  new1xLoops = xcLoopStmt .
  new2xLoops = xcForStmt .
  new3xLoops = xcWhileStmt .
  new4xLoops = xcWhilefound .
  new5xLoops = xcUntilStmt .
>.
xcLoopStmt = 'loop' xStmts .
xcForStmt = 'for' xIterator 'do' xStmts .
xcWhileStmt = 'while' xExpr 'do' xStmts .
xcWhilefound = 'whilefound' xIterator 'do' xStmts .
xcUntilStmt = 'repeat' xStmts 'until' xExpr .

/* Tuple-Space Operations: */
xcTempList = <
  new1xcTempList = xcTempList 'xor' xTemplate .
  new2xcTempList = xTemplate .
>.
xTemplate = <
  new1xTemplate = '(' xcEFEmpty xTempCond ')' '=>' xStmts .
  new2xTemplate = '(' xcEFEmpty xTempCond ')' .
```

```
>.
xTempCond = <
  new1xTempCond = '|' xExpr .
  new2xTempCond = .
>.
xcEFEmpty = <
  new1xcEFEmpty = .
  new2xcEFEmpty = xcEFList .
>.
xcEFList = <
  new1xcEFList = xcEFList ',' xExprFormal .
  new2xcEFList = xExprFormal .
>.
xExprFormal = <
  new1xExprFormal = xExpr .
  new2xExprFormal = '?' xFormal xInto .
>.
xInto = <
  new1xInto = 'into' xExpr .
  new2xInto = .
>.
xFormal = <
  new1xFormal = xcSimpleLV .
  new2xFormal = .
>.

/*******************************************************************************
        Iterators:
*******************************************************************************/
xIterator = <
  new1xIterator = xSimpleIts '|' xExpr .
  new2xIterator = xSimpleIts .
>.
xSimpleIts = <
  new1xSimpleIts = xSimpleIts ',' xSimpleIt .
  new2xSimpleIts = xSimpleIt .
>.
xSimpleIt = <
  new1xSimpleIt = xLValue 'in' xExpr .
  new2xSimpleIt = xLValue '=' xId xMapSel .
>.
/*******************************************************************************
        Map Selectors for simple iterators:
*******************************************************************************/
xMapSel = <
  new1xMapSel = '(' xcLValList ')' .
  new2xMapSel = '{' xcLValList '}' .
```

```
>.
xcLValList = <
  new1xcLValList = xcLValList ',' xLValue .
  new2xcLValList = xLValue .
>.
/******************************************************************************
        Left hand side values:
******************************************************************************/
xLValue = <
  new1xLValue = xcSimpleLV .
  new2xLValue = '[' xcComps ']' .
>.
xcSimpleLV = <
  new1xcSimpleLV = xQualId .
  new2xcSimpleLV = xcSimpleLV xSelector .
>.
xcComps = <
  new1xcComps = xcComps ',' xcComp .
  new2xcComps = xcComp .
>.
xcComp = <
  new1xcComp = xLValue .
  new2xcComp = '-' .
>.
/******************************************************************************
        Selectors:
******************************************************************************/
xSelector = <
  new1xSelector = '(' xExprList ')' .
  new2xSelector = '{' xExprList '}' .
  new3xSelector = '(' xExpr '..' ')' .
  new4xSelector = '(' Expr1:xExpr '..' Expr2:xExpr ')' .
>.
/******************************************************************************
        Former:
******************************************************************************/
/* sets: */
xcSetFormer = <
  new1xcSetFormer = xFormer .
  new2xcSetFormer = xExpr ',' xExprList .
  new3xcSetFormer = Expr1:xExpr ',' Expr2:xExpr '..' Expr3:xExpr .
>.
/* tuples: */
xcTupFormer = <
  new1xcTupFormer = xFormer .
  new2xcTupFormer = xcTupComp ',' Expr1:xExpr '..' Expr2:xExpr .
  new3xcTupFormer = xcTupComp ',' xcTCList .
```

```
>.
xcTCList = <
  new1xcTCList = xcTCList ',' xcTupComp .
  new2xcTCList = xcTupComp .
>.
xcTupComp = <
  new1xcTupComp = xExpr .
  new2xcTupComp = '-' .
>.
/* general: */
xFormer = <
  new1xFormer = xExpr .
  new2xFormer = Expr1:xExpr '..' Expr2:xExpr .
  new3xFormer = xExpr ':' xIterator .
>.
/*******************************************************************************
        Expressions:
*******************************************************************************/
xExprList = <
  new1xExprList = xExprList ',' xExpr .
  new2xExprList = xExpr .
>.
/*******************************************************************************
        Primary Expressions without possible selections:
*******************************************************************************/

/* module instantiations: */
xInstantiate = ins1:'instantiate' xExpr xInstExport xInstImport 'end'
               ins2:'instantiate' .
xInstExport = <
  new1xInstExport = 'export' xExprList ';' .
  new2xInstExport = .
>.
xInstImport = <
  new1xInstImport = 'import' xIdList ';' .
  new2xInstImport = .
>.
/*******************************************************************************
        Primary Expressions with possible selections:
*******************************************************************************/
xcPriSel = <
  new1xcPriSel = str .
  new2xcPriSel = '$' .
  new3xcPriSel = 'argv' .
  new4xcPriSel = '{' xcSetFormer '}' .
  new5xcPriSel = '[' xcTupFormer ']' .
  new6xcPriSel = xQualId .  /* Identifier: */
```

```
  new7xcPriSel = xLambda xActuList.  /* Lambda Expressions: */
  new8xcPriSel = 'self' xActuList .  /* Recursive lambda calls: */
  new9xcPriSel = 'closure' 'self' .  /* Closures: */
  new10xcPriSel = 'closure' xLambda .
  new11xcPriSel = 'closure' xQualId .
  new12xcPriSel = xcPriSel xSelector .
  new13xcPriSel = xcPriSel '(' ')' .
  new14xcPriSel = xcPriSel '[' xHandAsso ']' .  /* Explicit handler
                                                   associations: */
  new15xcPriSel = if1:'if' Expr1:xExpr 'then' Expr2:xExpr xElIfExprs xElseExpr
                  'end' if2:'if' . /* Conditional Expressions: */
  new16xcPriSel = case1:'case' xExpr xCaseExprs xElseExpr 'end' case2:'case' .
  new17xcPriSel = '(' xExpr ')' .
  new18xcPriSel = 'c_fct_call' Id1:xId '(' xTypeList ')' Id2:xId .
>.
/* Identifier: */
xQualId = <
  new1xQualId = Id1:xId '.' Id2:xId .
  new2xQualId = xId .
>.

/* Lambda Expressions: */
xLambda = lam1:'lambda' xParamList ':' xProgBody 'end' lam2:'lambda'.

xExpr = <
  new1xExpr = xExpr xOrOp xcOrTerm .
  new2xExpr = xcOrTerm .
  new3xExpr = xQuantifier .  /* Quantifiers: */
>.

xQuantifier = xQualifier xSimpleIts '|' xcPowTerm .
xQualifier = <
  new1xQualifier = 'exists' .
  new2xQualifier = 'forall' .
>.

/* Conditional Expressions: */
xElIfExprs = <
  new1xElIfExprs = xElIfExprs xElIfExpr .
  new2xElIfExprs = .
>.
xElIfExpr = 'elseif' Expr1:xExpr 'then' Expr2:xExpr .

xElseExpr = <
  new1xElseExpr = 'else' xExpr .
  new2xElseExpr = .
>.
```

```
/* Case Expressions: */
xCaseExprs = <
  new1xCaseExprs = xCaseExprs xcCaseExpr .
  new2xCaseExprs = xcCaseExpr .
>.
xcCaseExpr = xcCaseList xExpr .

/******************************************************************************
        Binary operations:
******************************************************************************/
xOrOp = <
  new1xOrOp = 'or' .
  new2xOrOp = 'or' '%' .
>.
xcOrTerm = <
  new1xcOrTerm = xcOrTerm xAndOp xcAndTerm .
  new2xcOrTerm = xcAndTerm .
>.
xAndOp = <
  new1xAndOp = 'and' .
  new2xAndOp = 'and' '%' .
>.
xcAndTerm = <
  new1xcAndTerm = xcAndTerm xcBoolOp xcBoolTerm .
  new2xcAndTerm = xcBoolTerm .
>.
xcBoolOp = <
  new1xcBoolOp = '=' .
  new2xcBoolOp = '/=' .
  new3xcBoolOp = '<' .
  new4xcBoolOp = '<=' .
  new5xcBoolOp = '>' .
  new6xcBoolOp = '>=' .
  new7xcBoolOp = 'in' .
  new8xcBoolOp = 'notin' .
  new9xcBoolOp = 'subset' .
  new10xcBoolOp = '=' '%' .
  new11xcBoolOp = '/=' '%' .
  new12xcBoolOp = '<' '%' .
  new13xcBoolOp = '<=' '%' .
  new14xcBoolOp = '>' '%' .
  new15xcBoolOp = '>=' '%' .
  new16xcBoolOp = 'in' '%' .
  new17xcBoolOp = 'notin' '%' .
  new18xcBoolOp = 'subset' '%' .
>.
```

```
xcBoolTerm = <
  new1xcBoolTerm = xcBoolTerm xcSetOp xcSetTerm .
  new2xcBoolTerm = xcSetTerm .
>.
xcSetOp = <
  new1xcSetOp = 'with' .
  new2xcSetOp = 'less' .
  new3xcSetOp = 'lessf' .
  new4xcSetOp = '!' xQualId .
  new5xcSetOp = 'with' '%' .
  new6xcSetOp = 'less' '%' .
  new7xcSetOp = 'lessf' '%' .
  new8xcSetOp = '!' xQualId '%' .
  new9xcSetOp = 'npow' .
>.
xcSetTerm = <
  new1xcSetTerm = xcSetTerm xcAddOp xcAddTerm .
  new2xcSetTerm = xcAddTerm .
>.
xcAddOp = <
  new1xcAddOp = '+' .
  new2xcAddOp = '-' .
  new3xcAddOp = 'max' .
  new4xcAddOp = 'min' .
  new5xcAddOp = '+' '%' .
  new6xcAddOp = '-' '%' .
  new7xcAddOp = 'max' '%' .
  new8xcAddOp = 'min' '%' .
>.
xcAddTerm = <
  new1xcAddTerm = xcAddTerm xcMulOp xcMulTerm .
  new2xcAddTerm = xcMulTerm .
>.
xcMulOp = <
  new1xcMulOp = '*' .
  new2xcMulOp = '/' .
  new3xcMulOp = 'mod' .
  new4xcMulOp = '*' '%' .
  new5xcMulOp = '/' '%' .
  new6xcMulOp = 'mod' '%' .
>.
xcMulTerm = <
  new1xcMulTerm = xcMulTerm xcPowOp xcPowTerm .
  new2xcMulTerm = xcPowTerm .
>.
xcPowOp = <
  new1xcPowOp = '**' .
```

```
  new2xcPowOp = '**' '%' .
>.
xcPowTerm = <
  new1xcPowTerm = xUnOp xcPowTerm .
  new2xcPowTerm = xcPrimary .
>.

xBinOp = <
  new1xBinOp = xcBoolOp .
  new2xBinOp = xcSetOp .
  new3xBinOp = xcAddOp .
  new4xBinOp = xcMulOp .
  new5xBinOp = xcPowOp .
>.
/********************************************************************************
        Unary Operators:
********************************************************************************/
xUnOp = <
  new1xUnOp = '+' .
  new2xUnOp = '-' .
  new3xUnOp = '#' .
  new4xUnOp = '||' .
  new5xUnOp = 'not' .
  new6xUnOp = 'pow' .
  new7xUnOp = 'arb' .
  new8xUnOp = 'random' .
  new9xUnOp = 'domain' .
  new10xUnOp = 'range' .
  new11xUnOp = 'type' .
  new12xUnOp = 'is_map' .
  new13xUnOp = 'is_smap' .
  new14xUnOp = '!' xQualId .
  new15xUnOp = 'or' '%' .
  new16xUnOp = 'and' '%' .
  new17xUnOp = '=' '%' .
  new18xUnOp = '/=' '%' .
  new19xUnOp = '<' '%' .
  new20xUnOp = '<=' '%' .
  new21xUnOp = '>' '%' .
  new22xUnOp = '>=' '%' .
  new23xUnOp = 'in' '%' .
  new24xUnOp = 'notin' '%' .
  new25xUnOp = 'subset' '%' .
  new26xUnOp = 'with' '%' .
  new27xUnOp = 'less' '%' .
  new28xUnOp = 'lessf' '%' .
  new29xUnOp = '!' xQualId '%' .
```

```
  new30xUnOp = '+' '%' .
  new31xUnOp = '-' '%' .
  new32xUnOp = 'max' '%' .
  new33xUnOp = 'min' '%' .
  new34xUnOp = '*' '%' .
  new35xUnOp = '/' '%' .
  new36xUnOp = 'mod' '%' .
  new37xUnOp = '**' '%' .
>.

xTypeList = <
  new1xTypeList = .
  new2xTypeList = Id1:xId ':' Id2:xId .
  new3xTypeList = xTypeList ',' Id1:xId ':' Id2:xId .
>.
```

C.42.2 Die Fortsetzung der Abbildung zwischen der konkreten und abstrakten Grammatik

⟨*transpars.abb*⟩≡

```
new1xParamList   = { Tree := mnew1xParamList('(':Position, ')':Position);} .

new2xParamList   = { Tree := mnew2xParamList('(':Position, xcParams:Tree,
                                ')':Position);} .

new1xcParams     = { Tree := mnew3xParamList(xcParams:Tree, ',':Position,
                                xParamMode:Tree, mxId(xId:Tree));} .

new2xcParams     = { Tree := mnew4xParamList(xParamMode:Tree,mxId(xId:Tree));}.

new1xParamMode   = { Tree := mnew1xParamMode();} .

new2xParamMode   = { Tree := mnew2xParamMode('rd':Position);} .

new3xParamMode   = { Tree := mnew3xParamMode('rw':Position);} .

new4xParamMode   = { Tree := mnew4xParamMode('wr':Position);} .

new1xModImport   = { Tree := mnew1xModImport('import':Position, xIdList:Tree,
                                ';':Position);} .

new2xModImport   = { Tree := mnew2xModImport();} .

new1xModExport   = { Tree := mnew1xModExport('export':Position, xIdList:Tree,
                                ';':Position);} .

new2xModExport   = { Tree := mnew2xModExport();} .

new1xIdList      = { Tree := mnew1xIdList(xIdList:Tree, ',':Position,
                                mxId(xId:Tree));} .

new2xIdList      = { Tree := mnew2xIdList(mxId(xId:Tree));} .

new1xRdParamList = { Tree := mnew1xRdParamList('(':Position, ')':Position);} .

new2xRdParamList = { Tree := mnew2xRdParamList('(':Position, xIdList:Tree,
                                ')':Position);} .

new1xImplAsso    = { Tree := mnew1xImplAsso('for':Position, xIdList:Tree);} .

new2xImplAsso    = { Tree := mnew2xImplAsso('for':Position,
                                'others':Position);} .
```

```
new3xImplAsso    = { Tree := mnew3xImplAsso();} .

new1xDecls       = { Tree := mnew1xDecls(xDecls:Tree, xDecl:Tree);} .

new2xDecls       = { Tree := mnew2xDecls();} .

new1xDecl        = { Tree := mnew1xDecl(xDeclKey:Tree, xcVars:Tree,
                       xExplAsso:Tree, ';':Position);} .

new2xDecl        = { Tree := mnew2xDecl(xPersDecl:Tree, xIdList:Tree,
                       ':':Position, xExpr:Tree, xExplAsso:Tree,
                       ';':Position);}.

new1xcVars       = { Tree := mnew3xSingleVar(xcVars:Tree, ',':Position,
                       xSingleVar:Tree);} .

new2xcVars       = { Tree :- xSingleVar:Tree; } .

new1xSingleVar   = { Tree := mnew1xSingleVar(mxId(xId:Tree), ':=':Position,
                       xExpr:Tree);} .

new2xSingleVar   = { Tree := mnew2xSingleVar(mxId(xId:Tree));} .

new1xDeclKey     = { Tree := mnew1xDeclKey('visible':Position);} .

new2xDeclKey     = { Tree := mnew2xDeclKey('hidden':Position);} .

new3xDeclKey     = { Tree := mnew3xDeclKey('visible':Position,
                       'constant':Position);} .

new4xDeclKey     = { Tree := mnew4xDeclKey('hidden':Position,
                       'constant':Position);} .

new5xDeclKey     = { Tree := mnew5xDeclKey('constant':Position);} .

new1xPersDecl    = { Tree := mnew1xPersDecl('visible':Position,
                       'persistent':Position);} .

new2xPersDecl    = { Tree := mnew2xPersDecl('hidden':Position,
                       'persistent':Position);} .

new3xPersDecl    = { Tree := mnew3xPersDecl('persistent':Position);} .

new4xPersDecl    = { Tree := mnew4xPersDecl('visible':Position,
                       'persistent':Position, 'constant':Position);} .

new5xPersDecl    = { Tree := mnew5xPersDecl('hidden':Position,
```

```
                               'persistent':Position, 'constant':Position);} .

new6xPersDecl    = { Tree := mnew6xPersDecl('persistent':Position,
                               'constant':Position);} .

xcBeginStmts     = { Tree := mnew3xStmts('begin':Position, xStmts:Tree);} .

new1xStmts       = { Tree := mnew1xStmts(xStmts:Tree, xStmt:Tree,
                               xExplAsso:Tree, ';':Position);} .

new2xStmts       = { Tree := mnew2xStmts(xStmt:Tree, xExplAsso:Tree,
                               ';':Position);} .

new1xExplAsso    = { Tree := mnew1xExplAsso(xExplAsso:Tree, 'when':Position,
                               xHandAsso:Tree);} .

new2xExplAsso    = { Tree := mnew2xExplAsso();} .

new1xHandAsso    = { Tree := mxHandAsso(xIdList:Tree, 'use':Position,
                               mxId(xId:Tree));} .

new2xHandAsso    = { Tree := mxHandAsso('others':Position, 'use':Position,
                               mxId(xId:Tree));} .

new1xStmt        = { Tree := mnew1xStmt('pass':Position);} .

new2xStmt        = { Tree := mnew2xStmt('stop':Position);} .

new3xStmt        = { Tree := mnew3xStmt('stop':Position, xExpr:Tree);} .

new4xStmt        = { Tree := mnew4xStmt('return':Position, xExpr:Tree);} .

new5xStmt        = { Tree := mnew5xStmt('return':Position);} .

new6xStmt        = { Tree := mnew6xStmt('return':Position, 'commit':Position,
                               xExpr:Tree);} .

new7xStmt        = { Tree := mnew7xStmt('return':Position,
                               'commit':Position);} .

new8xStmt        = { Tree := mnew8xStmt('resume':Position, xExpr:Tree);} .

new9xStmt        = { Tree := mnew9xStmt('resume':Position);} .

new10xStmt       = { Tree := mnew10xStmt(xStmtSignal:Tree);} .

new11xStmt       = { Tree := mnew11xStmt(xStmtNotify:Tree);} .
```

```
new12xStmt       = { Tree := mnew12xStmt('escape':Position, mxId(xId:Tree),
                            xActuList:Tree);} .

new13xStmt       = { Tree := mnew13xStmt(xStdIO:Tree, xActuList:Tree);} .

new14xStmt       = { Tree := mnew14xStmt('any':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .

new15xStmt       = { Tree := mnew15xStmt('rany':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .

new16xStmt       = { Tree := mnew16xStmt('break':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .

new17xStmt       = { Tree := mnew17xStmt('rbreak':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .

new18xStmt       = { Tree := mnew18xStmt('len':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .

new19xStmt       = { Tree := mnew19xStmt('rlen':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .

new20xStmt       = { Tree := mnew20xStmt('lpad':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .

new21xStmt       = { Tree := mnew21xStmt('rpad':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .
new22xStmt       = { Tree := mnew22xStmt('match':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .

new23xStmt       = { Tree := mnew23xStmt('rmatch':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                            ')':Position);} .

new24xStmt       = { Tree := mnew24xStmt('notany':Position, '(':Position,
                            xcSimpleLV:Tree, ',':Position, xExpr:Tree,
```

```
                           ')':Position);} .

new25xStmt       = { Tree := mnew25xStmt('rnotany':Position, '(':Position,
                           xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                           ')':Position);} .

new26xStmt       = { Tree := mnew26xStmt('span':Position, '(':Position,
                           xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                           ')':Position);} .

new27xStmt       = { Tree := mnew27xStmt('rspan':Position, '(':Position,
                           xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                           ')':Position);} .

new28xStmt       = { Tree := mnew28xStmt(xLValue:Tree, ':=':Position,
                           xExpr:Tree);} .

new29xStmt       = { Tree := mnew29xStmt(xLValue:Tree, xBinOp:Tree,
                           ':=':Position, xExpr:Tree);} .

new30xStmt       = { Tree := mnew29xStmt(xLValue:Tree, xAndOp:Tree,
                           ':=':Position, xExpr:Tree);} .

new31xStmt       = { Tree := mnew29xStmt(xLValue:Tree, xOrOp:Tree,
                           ':=':Position, xExpr:Tree);} .

new32xStmt       = { Tree := mnew32xStmt(xLValue:Tree, xFrom:Tree,
                           xcSimpleLV:Tree);} .

new33xStmt       = { Tree := mnew33xStmt(xcSimpleLV:Tree, xActuList:Tree);} .

new34xStmt       = { Tree := mnew34xStmt(xLambda:Tree, xActuList:Tree);} .

new35xStmt       = { Tree := mnew35xStmt('self':Position, xActuList:Tree);} .

new36xStmt       = { Tree := mnew36xStmt(if1:Position, xExpr:Tree,
                           'then':Position, xStmts:Tree, xElIfStmts:Tree,
                           xElseStmts:Tree,'end':Position,if2:Position);}.

new37xStmt       = { Tree := mnew37xStmt(case1:Position, xExpr:Tree,
                           xCaseStmts:Tree, xElseStmts:Tree,'end':Position,
                           case2:Position);} .

new38xStmt       = { Tree := mnew38xStmt(xcLoopStmt:Tree, 'end':Position,
                           'loop':Position);} .

new39xStmt       = { Tree := mnew39xStmt(xcForStmt:Tree, 'end':Position,
```

```
                                 'for':Position);} .

new40xStmt       = { Tree := mnew40xStmt(xcWhileStmt:Tree, 'end':Position,
                                 'while':Position);} .

new41xStmt       = { Tree := mnew41xStmt(xcWhilefound:Tree, 'end':Position,
                                 'whilefound':Position);} .

new42xStmt       = { Tree := mnew42xStmt(xcUntilStmt:Tree, 'end':Position,
                                 'repeat':Position);} .

new43xStmt       = { Tree := mnew43xStmt(xLabel:Tree, xLoops:Tree,
                                 'end':Position, mxId(xId:Tree));} .

new44xStmt       = { Tree := mnew44xStmt('quit':Position);} .

new45xStmt       = { Tree := mnew45xStmt('quit':Position, mxId(xId:Tree));} .

new46xStmt       = { Tree := mnew46xStmt('continue':Position);} .

new47xStmt       = { Tree := mnew47xStmt('continue':Position,
                                 mxId(xId:Tree));} .

new48xStmt       = { Tree := mnew48xStmt('||':Position, xExpr:Tree);} .

new49xStmt       = { Tree := mnew49xStmt(dep1:Position, Expr1:Tree,
                                 'at':Position, Expr2:Tree, 'end':Position,
                                 dep2:Position);} .

new50xStmt       = { Tree := mnew50xStmt(dep1:Position, Expr1:Tree,
                                 'at':Position,Expr2:Tree,'blockiffull':Position,
                                 'end':Position, dep2:Position);} .

new51xStmt       = { Tree := mnew51xStmt(fetch1:Position, xcTempList:Tree,
                                 'at':Position, xExpr:Tree, xElseStmts:Tree,
                                 'end':Position, fetch2:Position);} .

new52xStmt       = { Tree := mnew52xStmt(meet1:Position, xcTempList:Tree,
                                 'at':Position, xExpr:Tree, xElseStmts:Tree,
                                 'end':Position, meet2:Position);} .

new53xStmt       = { Tree := mnew53xStmt('c_fct_call':Position, mxId(xId:Tree),
                                 '(':Position, xTypeList:Tree, ')':Position);} .

new54xStmt       = { Tree := mnew54xStmt('getf':Position, '(':Position,
                                 xExpr:Tree, ',':Position, xExprList:Tree,
                                 ')':Position);} .
```

```
new55xStmt       = { Tree := mnew55xStmt('fget':Position, '(':Position,
                        xExpr:Tree, ',':Position, xExprList:Tree,
                        ')':Position);} .

new56xStmt       = { Tree := mnew56xStmt('fgetf':Position, '(':Position,
                        xExpr1:Tree, ',':Position, xExpr2:Tree,
                        ',':Position, xExprList:Tree, ')':Position);} .

new1xStmtSignal  = { Tree := mnew1xStmtSignal('signal':Position, xLValue:Tree,
                       ':=':Position, mxId(xId:Tree),xActuList:Tree);}.

new2xStmtSignal  = { Tree := mnew2xStmtSignal('signal':Position,
                       xcSimpleLV:Tree, xActuList:Tree);} .

new1xStmtNotify  = { Tree := mnew1xStmtNotify('notify':Position, xLValue:Tree,
                       ':=':Position, mxId(xId:Tree),xActuList:Tree);}.

new2xStmtNotify  = { Tree := mnew2xStmtNotify('notify':Position,
                       xcSimpleLV:Tree, xActuList:Tree);} .

new1xStdIO       = { Tree := mnew1xStdIO('put':Position);} .

new2xStdIO       = { Tree := mnew2xStdIO('eput':Position);} .

new3xStdIO       = { Tree := mnew3xStdIO('fput':Position);} .

new4xStdIO       = { Tree := mnew4xStdIO('putf':Position);} .

new5xStdIO       = { Tree := mnew5xStdIO('eputf':Position);} .

new6xStdIO       = { Tree := mnew6xStdIO('fputf':Position);} .

new7xStdIO       = { Tree := mnew7xStdIO('get':Position);} .

/* new8xStdIO       = { Tree := mnew8xStdIO('fget':Position);} . */
/* new9xStdIO       = { Tree := mnew9xStdIO('getf':Position);} . */
/* new10xStdIO      = { Tree := mnew10xStdIO('fgetf':Position);} . */

new11xStdIO      = { Tree := mnew11xStdIO('fopen':Position);} .

new12xStdIO      = { Tree := mnew12xStdIO('fclose':Position);} .

new1xcPrimary    = { Tree := mnew1xExpr('any':Position, '(':Position,
                       xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                       ')':Position);} .
```

```
new2xcPrimary    = { Tree := mnew2xExpr('rany':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new3xcPrimary    = { Tree := mnew3xExpr('break':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new4xcPrimary    = { Tree := mnew4xExpr('rbreak':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new5xcPrimary    = { Tree := mnew5xExpr('len':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new6xcPrimary    = { Tree := mnew6xExpr('rlen':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new7xcPrimary    = { Tree := mnew7xExpr('match':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new8xcPrimary    = { Tree := mnew8xExpr('rmatch':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new9xcPrimary    = { Tree := mnew9xExpr('notany':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new10xcPrimary   = { Tree := mnew10xExpr('rnotany':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new11xcPrimary   = { Tree := mnew11xExpr('span':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new12xcPrimary   = { Tree := mnew12xExpr('rspan':Position, '(':Position,
                          xcSimpleLV:Tree, ',':Position, xExpr:Tree,
                          ')':Position);} .

new13xcPrimary   = { Tree := mnew13xExpr(mintlit(intlit:Position,
                          intlit:Integer,intlit:Length));} .
```

```
new14xcPrimary   = { Tree := mnew14xExpr(mfloatlit(floatlit:Position,
                               floatlit:Float,floatlit:Length));} .

new15xcPrimary   = { Tree := mnew15xExpr('true':Position);} .

new16xcPrimary   = { Tree := mnew16xExpr('false':Position);} .

new17xcPrimary   = { Tree := mnew17xExpr('om':Position);} .

new18xcPrimary   = { Tree := mnew18xExpr('atom':Position);} .

new19xcPrimary   = { Tree := mnew19xExpr('boolean':Position);} .

new20xcPrimary   = { Tree := mnew20xExpr('integer':Position);} .

new21xcPrimary   = { Tree := mnew21xExpr('real':Position);} .

new22xcPrimary   = { Tree := mnew22xExpr('string':Position);} .

new23xcPrimary   = { Tree := mnew23xExpr('tuple':Position);} .

new24xcPrimary   = { Tree := mnew24xExpr('set':Position);} .

new25xcPrimary   = { Tree := mnew25xExpr('function':Position);} .

new26xcPrimary   = { Tree := mnew26xExpr('modtype':Position);} .

new27xcPrimary   = { Tree := mnew27xExpr('instance':Position);} .

new28xcPrimary   = { Tree := mnew28xExpr('{':Position, '}':Position);} .

new29xcPrimary   = { Tree := mnew29xExpr('[':Position, ']':Position);} .

new30xcPrimary   = { Tree := mnew30xExpr('newat':Position, '(':Position,
                               ')':Position);} .

new31xcPrimary   = { Tree := mnew31xExpr(xInstantiate:Tree);} .

new32xcPrimary   = { Tree :- xcPriSel:Tree; } .

new1xFrom        = { Tree := mnew1xFrom('from':Position);} .

new2xFrom        = { Tree := mnew2xFrom('frome':Position);} .

new3xFrom        = { Tree := mnew3xFrom('fromb':Position);} .

new1xActuList    = { Tree := mnew1xActuList('(':Position, xExprList:Tree,
```

```
                                 ')':Position);} .

new2xActuList    = { Tree := mnew2xActuList('(':Position, ')':Position);} .

xElIfStmt        = { Tree := mxElIfStmt('elseif':Position, xExpr:Tree,
                                 'then':Position, xStmts:Tree);} .

new1xElIfStmts   = { Tree := mnew1xElIfStmts(xElIfStmts:Tree,
                                 xElIfStmt:Tree);} .

new2xElIfStmts   = { Tree := mnew2xElIfStmts();} .

new1xElseStmts   = { Tree := mnew1xElseStmts('else':Position, xStmts:Tree);} .

new2xElseStmts   = { Tree := mnew2xElseStmts();} .

new1xCaseStmts   = { Tree := mnew1xCaseStmts(xCaseStmts:Tree,
                                 xcCaseStmt:Tree);} .

new2xCaseStmts   = { Tree := mnew2xCaseStmts(xcCaseStmt:Tree);} .

xcCaseStmt       = { Tree := mnew3xCaseStmts(xcCaseList:Tree, xStmts:Tree);} .

xcCaseList       = { Tree := mnew3xExprList('when':Position, xExprList:Tree,
                                 '=>':Position);} .

xLabel           = { Tree := mxLabel(mxId(xId:Tree), ':':Position);} .

xcLoopStmt       = { Tree := mnew6xLoops('loop':Position, xStmts:Tree);} .

xcForStmt        = { Tree := mnew7xLoops('for':Position, xIterator:Tree,
                                 'do':Position, xStmts:Tree);} .

xcWhileStmt      = { Tree := mnew8xLoops('while':Position, xExpr:Tree,
                                 'do':Position, xStmts:Tree);} .

xcWhilefound     = { Tree := mnew9xLoops('whilefound':Position,
                                 xIterator:Tree, 'do':Position, xStmts:Tree);} .

xcUntilStmt      = { Tree := mnew10xLoops('repeat':Position, xStmts:Tree,
                                 'until':Position, xExpr:Tree);} .

new1xcTempList   = { Tree := mnew3xTemplate(xcTempList:Tree, 'xor':Position,
                                 xTemplate:Tree);} .

new2xcTempList   = { Tree :- xTemplate:Tree; } .
```

```
new1xTemplate    = { Tree := mnew1xTemplate('(':Position, xcEFEmpty:Tree,
                              xTempCond:Tree, ')':Position, '=>':Position,
                              xStmts:Tree);} .

new2xTemplate    = { Tree := mnew2xTemplate('(':Position, xcEFEmpty:Tree,
                              xTempCond:Tree, ')':Position);} .

new1xTempCond    = { Tree := mnew1xTempCond('|':Position, xExpr:Tree);} .

new2xTempCond    = { Tree := mnew2xTempCond();} .

new1xcEFEmpty    = { Tree := mnew3xExprFormal();} .

new2xcEFEmpty    = { Tree :- xcEFList:Tree; } .

new1xcEFList     = { Tree := mnew4xExprFormal(xcEFList:Tree, ',':Position,
                              xExprFormal:Tree);} .

new2xcEFList     = { Tree :- xExprFormal:Tree; } .

new1xExprFormal  = { Tree := mnew1xExprFormal(xExpr:Tree);} .

new2xExprFormal  = { Tree := mnew2xExprFormal('?':Position, xFormal:Tree,
                              xInto:Tree);} .

new1xInto        = { Tree := mnew1xInto('into':Position, xExpr:Tree);} .

new2xInto        = { Tree := mnew2xInto();} .

new1xFormal      = { Tree := mnew1xFormal(xcSimpleLV:Tree);} .

new2xFormal      = { Tree := mnew2xFormal();} .

new1xIterator    = { Tree := mnew1xIterator(xSimpleIts:Tree, '|':Position,
                              xExpr:Tree);} .

new2xIterator    = { Tree := mnew2xIterator(xSimpleIts:Tree);} .

new1xSimpleIts   = { Tree := mnew1xSimpleIts(xSimpleIts:Tree, ',':Position,
                              xSimpleIt:Tree);} .

new2xSimpleIts   = { Tree := mnew2xSimpleIts(xSimpleIt:Tree);} .

new1xSimpleIt    = { Tree := mnew1xSimpleIt(xLValue:Tree, 'in':Position,
                              xExpr:Tree);} .

new2xSimpleIt    = { Tree := mnew2xSimpleIt(xLValue:Tree, '=':Position,
```

```
                              mxId(xId:Tree), xMapSel:Tree);} .

new1xMapSel      = { Tree := mnew1xMapSel('(':Position, xcLValList:Tree,
                              ')':Position);} .

new2xMapSel      = { Tree := mnew2xMapSel('{':Position, xcLValList:Tree,
                              '}':Position);} .

new1xcLValList   = { Tree := mnew3xLValue(xcLValList:Tree, ',':Position,
                              xLValue:Tree);} .

new2xcLValList   = { Tree :- xLValue:Tree; } .

new1xLValue      = { Tree :- xcSimpleLV:Tree; } .

new2xLValue      = { Tree := mnew2xLValue('[':Position, xcComps:Tree,
                              ']':Position);} .

new1xcSimpleLV   = { Tree := mnew5xLValue(xQualId:Tree);} .

new2xcSimpleLV   = { Tree := mnew6xLValue(xcSimpleLV:Tree,
                              xSelector:Tree);} .

new1xcComps      = { Tree := mnew3xLValue(xcComps:Tree, ',':Position,
                              xcComp:Tree);} .

new2xcComps      = { Tree :- xcComp:Tree; } .

new1xcComp       = { Tree :- xLValue:Tree; } .

new2xcComp       = { Tree := mnew4xLValue('-':Position);} .

new1xSelector    = { Tree := mnew1xSelector('(':Position, xExprList:Tree,
                              ')':Position);} .

new2xSelector    = { Tree := mnew2xSelector('{':Position, xExprList:Tree,
                              '}':Position);} .

new3xSelector    = { Tree := mnew3xSelector('(':Position, xExpr:Tree,
                              '..':Position, ')':Position);} .

new4xSelector    = { Tree := mnew4xSelector('(':Position, Expr1:Tree,
                              '..':Position, Expr2:Tree, ')':Position);} .

new1xcSetFormer  = { Tree :- xFormer:Tree; } .

new2xcSetFormer  = { Tree := mnew4xFormer(xExpr:Tree, ',':Position,
```

```
                              xExprList:Tree);} .

new3xcSetFormer  = { Tree := mnew5xFormer(Expr1:Tree, ',':Position,
                              Expr2:Tree, '..':Position, Expr3:Tree);} .

new1xcTupFormer  = { Tree :- xFormer:Tree; } .

new2xcTupFormer  = { Tree := mnew5xFormer(xcTupComp:Tree, ',':Position,
                              Expr1:Tree, '..':Position, Expr2:Tree);} .

new3xcTupFormer  = { Tree := mnew4xFormer(xcTupComp:Tree, ',':Position,
                              xcTCList:Tree);} .

new1xcTCList     = { Tree := mnew1xExprList(xcTCList:Tree, ',':Position,
                              xcTupComp:Tree);} .

new2xcTCList     = { Tree := mnew2xExprList(xcTupComp:Tree);} .

new1xcTupComp    = { Tree :- xExpr:Tree; } .

new2xcTupComp    = { Tree := mnew32xExpr('-':Position);} .

new1xFormer      = { Tree := mnew1xFormer(xExpr:Tree);} .

new2xFormer      = { Tree := mnew2xFormer(Expr1:Tree, '..':Position,
                              Expr2:Tree);} .

new3xFormer      = { Tree := mnew3xFormer(xExpr:Tree, ':':Position,
                              xIterator:Tree);} .

new1xExprList    = { Tree := mnew1xExprList(xExprList:Tree, ',':Position,
                              xExpr:Tree);} .
new2xExprList    = { Tree := mnew2xExprList(xExpr:Tree);} .

xInstantiate     = { Tree := mxInstantiate(ins1:Position, xExpr:Tree,
                              xInstExport:Tree, xInstImport:Tree,
                              'end':Position, ins2:Position);} .

new1xInstExport  = { Tree := mnew1xInstExport('export':Position,
                              xExprList:Tree, ';':Position);} .

new2xInstExport  = { Tree := mnew2xInstExport();} .

new1xInstImport  = { Tree := mnew1xInstImport('import':Position, xIdList:Tree,
                              ';':Position);} .

new2xInstImport  = { Tree := mnew2xInstImport();} .
```

```
new1xcPriSel     = { Tree := mnew33xExpr(mstr(str:Position,str:Str,
                                      str:Length));} .

new2xcPriSel     = { Tree := mnew34xExpr('$':Position);} .

new3xcPriSel     = { Tree := mnew35xExpr('argv':Position);} .

new4xcPriSel     = { Tree := mnew36xExpr('{':Position, xcSetFormer:Tree,
                      '}':Position);} .

new5xcPriSel     = { Tree := mnew37xExpr('[':Position, xcTupFormer:Tree,
                      ']':Position);} .

new6xcPriSel     = { Tree := mnew38xExpr(xQualId:Tree);} .

new7xcPriSel     = { Tree := mnew39xExpr(xLambda:Tree, xActuList:Tree);} .

new8xcPriSel     = { Tree := mnew40xExpr('self':Position, xActuList:Tree);} .

new9xcPriSel     = { Tree := mnew41xExpr('closure':Position,
                      'self':Position);} .

new10xcPriSel    = { Tree := mnew42xExpr('closure':Position, xLambda:Tree);} .

new11xcPriSel    = { Tree := mnew43xExpr('closure':Position, xQualId:Tree);} .

new12xcPriSel    = { Tree := mnew44xExpr(xcPriSel:Tree, xSelector:Tree);} .

new13xcPriSel    = { Tree := mnew45xExpr(xcPriSel:Tree, '(':Position,
                      ')':Position);} .

new14xcPriSel   = { Tree := mnew46xExpr(xcPriSel:Tree, '[':Position,
                     xHandAsso:Tree, ']':Position);} .

new15xcPriSel   = { Tree := mnew47xExpr(if1:Position, Expr1:Tree,
                     'then':Position, Expr2:Tree, xElIfExprs:Tree,
                      xElseExpr:Tree, 'end':Position, if2:Position);}.

new16xcPriSel   = { Tree := mnew48xExpr(case1:Position, xExpr:Tree,
                     xCaseExprs:Tree, xElseExpr:Tree, 'end':Position,
                     case2:Position);} .

new17xcPriSel   = { Tree := mnew49xExpr('(':Position, xExpr:Tree,
                     ')':Position);} .

new18xcPriSel   = { Tree := mnew53xExpr('c_fct_call':Position, mxId(Id1:Tree),
```

```
                                   '(':Position, xTypeList:Tree, ')':Position,
                                   mxId(Id2:Tree));} .

new1xQualId     = { Tree := mnew1xQualId(mxId(Id1:Tree), '.':Position,
                                   mxId(Id2:Tree));} .

new2xQualId     = { Tree := mnew2xQualId(mxId(xId:Tree));} .

xLambda         = { Tree := mxLambda(lam1:Position, xParamList:Tree,
                                   ':':Position, xProgBody:Tree, 'end':Position,
                                   lam2:Position);} .

new1xExpr       = { Tree := mnew50xExpr(xExpr:Tree, xOrOp:Tree,
                                   xcOrTerm:Tree);} .

new2xExpr       = { Tree :- xcOrTerm:Tree; } .

new3xExpr       = { Tree := mnew51xExpr(xQuantifier:Tree);} .

xQuantifier     = { Tree := mxQuantifier(xQualifier:Tree, xSimpleIts:Tree,
                                   '|':Position, xcPowTerm:Tree);} .

new1xQualifier  = { Tree := mnew1xQualifier('exists':Position);} .

new2xQualifier  = { Tree := mnew2xQualifier('forall':Position);} .

new1xElIfExprs  = { Tree := mnew1xElIfExprs(xElIfExprs:Tree,
                                   xElIfExpr:Tree);} .

new2xElIfExprs  = { Tree := mnew2xElIfExprs();} .

xElIfExpr       = { Tree := mxElIfExpr('elseif':Position, Expr1:Tree,
                                   'then':Position, Expr2:Tree);} .

new1xElseExpr   = { Tree := mnew1xElseExpr('else':Position, xExpr:Tree);} .

new2xElseExpr   = { Tree := mnew2xElseExpr();} .

new1xCaseExprs  = { Tree := mnew1xCaseExprs(xCaseExprs:Tree,
                                   xcCaseExpr:Tree);} .

new2xCaseExprs  = { Tree := mnew2xCaseExprs(xcCaseExpr:Tree);} .

xcCaseExpr      = { Tree := mnew3xCaseExprs(xcCaseList:Tree, xExpr:Tree);} .

new1xOrOp       = { Tree := mnew1xBinOp('or':Position);} .
```

```
new2xOrOp       = { Tree := mnew2xBinOp('or':Position, '%':Position);} .

new1xcOrTerm    = { Tree := mnew50xExpr(xcOrTerm:Tree, xAndOp:Tree,
                                xcAndTerm:Tree);} .

new2xcOrTerm    = { Tree :- xcAndTerm:Tree; } .

new1xAndOp      = { Tree := mnew3xBinOp('and':Position);} .

new2xAndOp      = { Tree := mnew4xBinOp('and':Position, '%':Position);} .

new1xcAndTerm   = { Tree := mnew50xExpr(xcAndTerm:Tree, xcBoolOp:Tree,
                                xcBoolTerm:Tree);} .

new2xcAndTerm   = { Tree :- xcBoolTerm:Tree; } .

new1xcBoolOp    = { Tree := mnew5xBinOp('=':Position);} .

new2xcBoolOp    = { Tree := mnew6xBinOp('/=':Position);} .

new3xcBoolOp    = { Tree := mnew7xBinOp('<':Position);} .

new4xcBoolOp    = { Tree := mnew8xBinOp('<=':Position);} .

new5xcBoolOp    = { Tree := mnew9xBinOp('>':Position);} .

new6xcBoolOp    = { Tree := mnew10xBinOp('>=':Position);} .

new7xcBoolOp    = { Tree := mnew11xBinOp('in':Position);} .

new8xcBoolOp    = { Tree := mnew12xBinOp('notin':Position);} .

new9xcBoolOp    = { Tree := mnew13xBinOp('subset':Position);} .

new10xcBoolOp   = { Tree := mnew14xBinOp('=':Position, '%':Position);} .

new11xcBoolOp   = { Tree := mnew15xBinOp('/=':Position, '%':Position);} .

new12xcBoolOp   = { Tree := mnew16xBinOp('<':Position, '%':Position);} .

new13xcBoolOp   = { Tree := mnew17xBinOp('<=':Position, '%':Position);} .

new14xcBoolOp   = { Tree := mnew18xBinOp('>':Position, '%':Position);} .

new15xcBoolOp   = { Tree := mnew19xBinOp('>=':Position, '%':Position);} .

new16xcBoolOp   = { Tree := mnew20xBinOp('in':Position, '%':Position);} .
```

```
new17xcBoolOp   = { Tree := mnew21xBinOp('notin':Position, '%':Position);} .

new18xcBoolOp   = { Tree := mnew22xBinOp('subset':Position, '%':Position);} .

new1xcBoolTerm  = { Tree := mnew50xExpr(xcBoolTerm:Tree, xcSetOp:Tree,
                              xcSetTerm:Tree);} .

new2xcBoolTerm  = { Tree :- xcSetTerm:Tree; } .

new1xcSetOp     = { Tree := mnew23xBinOp('with':Position);} .

new2xcSetOp     = { Tree := mnew24xBinOp('less':Position);} .

new3xcSetOp     = { Tree := mnew25xBinOp('lessf':Position);} .

new4xcSetOp     = { Tree := mnew26xBinOp('!':Position, xQualId:Tree);} .

new5xcSetOp     = { Tree := mnew27xBinOp('with':Position, '%':Position);} .

new6xcSetOp     = { Tree := mnew28xBinOp('less':Position, '%':Position);} .

new7xcSetOp     = { Tree := mnew29xBinOp('lessf':Position, '%':Position);} .

new8xcSetOp     = { Tree := mnew30xBinOp('!':Position, xQualId:Tree,
                              '%':Position);} .

new9xcSetOp     = { Tree := mnew47xBinOp('npow':Position);} .

new1xcSetTerm   = { Tree := mnew50xExpr(xcSetTerm:Tree, xcAddOp:Tree,
                              xcAddTerm:Tree);} .

new2xcSetTerm   = { Tree :- xcAddTerm:Tree; } .

new1xcAddOp     = { Tree := mnew31xBinOp('+':Position);} .

new2xcAddOp     = { Tree := mnew32xBinOp('-':Position);} .

new3xcAddOp     = { Tree := mnew33xBinOp('max':Position);} .

new4xcAddOp     = { Tree := mnew34xBinOp('min':Position);} .

new5xcAddOp     = { Tree := mnew35xBinOp('+':Position, '%':Position);} .

new6xcAddOp     = { Tree := mnew36xBinOp('-':Position, '%':Position);} .

new7xcAddOp     = { Tree := mnew37xBinOp('max':Position, '%':Position);} .
```

```
new8xcAddOp     = { Tree := mnew38xBinOp('min':Position, '%':Position);} .

new1xcAddTerm   = { Tree := mnew50xExpr(xcAddTerm:Tree, xcMulOp:Tree,
                             xcMulTerm:Tree);} .

new2xcAddTerm   = { Tree :- xcMulTerm:Tree; } .

new1xcMulOp     = { Tree := mnew39xBinOp('*':Position);} .

new2xcMulOp     = { Tree := mnew40xBinOp('/':Position);} .

new3xcMulOp     = { Tree := mnew41xBinOp('mod':Position);} .

new4xcMulOp     = { Tree := mnew42xBinOp('*':Position, '%':Position);} .

new5xcMulOp     = { Tree := mnew43xBinOp('/':Position, '%':Position);} .

new6xcMulOp     = { Tree := mnew44xBinOp('mod':Position, '%':Position);} .

new1xcMulTerm   = { Tree := mnew50xExpr(xcMulTerm:Tree, xcPowOp:Tree,
                             xcPowTerm:Tree);} .

new2xcMulTerm   = { Tree :- xcPowTerm:Tree; } .

new1xcPowOp     = { Tree := mnew45xBinOp('**':Position);} .

new2xcPowOp     = { Tree := mnew46xBinOp('**':Position, '%':Position);} .

new1xcPowTerm   = { Tree := mnew52xExpr(xUnOp:Tree, xcPowTerm:Tree);} .

new2xcPowTerm   = { Tree :- xcPrimary:Tree; } .

new1xUnOp       = { Tree := mnew1xUnOp('+':Position);} .

new2xUnOp       = { Tree := mnew2xUnOp('-':Position);} .

new3xUnOp       = { Tree := mnew3xUnOp('#':Position);} .

new4xUnOp       = { Tree := mnew4xUnOp('||':Position);} .

new5xUnOp       = { Tree := mnew5xUnOp('not':Position);} .

new6xUnOp       = { Tree := mnew6xUnOp('pow':Position);} .

new7xUnOp       = { Tree := mnew7xUnOp('arb':Position);} .
```

```
new8xUnOp      = { Tree := mnew8xUnOp('random':Position);} .

new9xUnOp      = { Tree := mnew9xUnOp('domain':Position);} .

new10xUnOp     = { Tree := mnew10xUnOp('range':Position);} .

new11xUnOp     = { Tree := mnew11xUnOp('type':Position);} .

new12xUnOp     = { Tree := mnew12xUnOp('is_map':Position);} .

new13xUnOp     = { Tree := mnew13xUnOp('is_smap':Position);} .

new14xUnOp     = { Tree := mnew14xUnOp('!':Position, xQualId:Tree);} .

new15xUnOp     = { Tree := mnew15xUnOp('or':Position, '%':Position);} .

new16xUnOp     = { Tree := mnew16xUnOp('and':Position, '%':Position);} .

new17xUnOp     = { Tree := mnew17xUnOp('=':Position, '%':Position);} .

new18xUnOp     = { Tree := mnew18xUnOp('/=':Position, '%':Position);} .

new19xUnOp     = { Tree := mnew19xUnOp('<':Position, '%':Position);} .

new20xUnOp     = { Tree := mnew20xUnOp('<=':Position, '%':Position);} .

new21xUnOp     = { Tree := mnew21xUnOp('>':Position, '%':Position);} .

new22xUnOp     = { Tree := mnew22xUnOp('>=':Position, '%':Position);} .

new23xUnOp     = { Tree := mnew23xUnOp('in':Position, '%':Position);} .

new24xUnOp     = { Tree := mnew24xUnOp('notin':Position, '%':Position);} .

new25xUnOp     = { Tree := mnew25xUnOp('subset':Position, '%':Position);} .

new26xUnOp     = { Tree := mnew26xUnOp('with':Position, '%':Position);} .

new27xUnOp     = { Tree := mnew27xUnOp('less':Position, '%':Position);} .

new28xUnOp     = { Tree := mnew28xUnOp('lessf':Position, '%':Position);} .

new29xUnOp     = { Tree := mnew29xUnOp('!':Position, xQualId:Tree,
                               '%':Position);} .

new30xUnOp     = { Tree := mnew30xUnOp('+':Position, '%':Position);} .
```

```
new31xUnOp      = { Tree := mnew31xUnOp('-':Position, '%':Position);} .

new32xUnOp      = { Tree := mnew32xUnOp('max':Position, '%':Position);} .

new33xUnOp      = { Tree := mnew33xUnOp('min':Position, '%':Position);} .

new34xUnOp      = { Tree := mnew34xUnOp('*':Position, '%':Position);} .

new35xUnOp      = { Tree := mnew35xUnOp('/':Position, '%':Position);} .

new36xUnOp      = { Tree := mnew36xUnOp('mod':Position, '%':Position);} .

new37xUnOp      = { Tree := mnew37xUnOp('**':Position, '%':Position);} .

new1xTypeList = { Tree := mnew1xTypeList();} .

new2xTypeList = { Tree := mnew2xTypeList(mxId(Id1:Tree), ':':Position,
                            mxId(Id2:Tree));} .

new3xTypeList = { Tree := mnew3xTypeList(xTypeList:Tree, ',':Position,
                            mxId(Id1:Tree), ':':Position, mxId(Id2:Tree));} .
```

C.42.3 Die Fortsetzung der abstrakte Grammatik

⟨*transcg.abs*⟩≡

```
xParamList = <
  new1xParamList = [pos1:tPosition] [pos2:tPosition] .
            /* = '(' ')' */

  new2xParamList = [pos1:tPosition] xParamList [pos2:tPosition] .
            /* = (' xParamList ')' */

  new3xParamList = xParamList [pos:tPosition] xParamMode xId .
          /* = xParamList ',' xParamMode xId */

  new4xParamList = xParamMode xId .
>.

xParamMode = <
  new1xParamMode = .

  new2xParamMode = [pos:tPosition] .
            /* = 'rd' */

  new3xParamMode = [pos:tPosition] .
            /* = 'rw' */

  new4xParamMode = [pos:tPosition] .
            /* = 'wr' */
>.

xModImport = <
  new1xModImport = [pos1:tPosition] xIdList [pos2:tPosition] .
            /* = 'import' xIdList ';' */

  new2xModImport = .
>.

xModExport = <
  new1xModExport = [pos1:tPosition] xIdList [pos2:tPosition] .
            /* = 'export' xIdList ';' */

  new2xModExport = .
>.

xIdList = <
  new1xIdList = xIdList [pos:tPosition] xId .
         /* = xIdList ',' xId */
  new2xIdList = xId .
```

```
>.

xRdParamList = <
  new1xRdParamList = [pos1:tPosition] [pos2:tPosition] .
                /* = '(' ')' */

  new2xRdParamList = [pos1:tPosition] xIdList [pos2:tPosition] .
                /* = '(' xIdList ')' */
>.

xImplAsso = <
  new1xImplAsso = [pos:tPosition] xIdList .
             /* = 'for' xIdList */

  new2xImplAsso = [pos1:tPosition] [pos2:tPosition] .
             /* = 'for' 'others' */

  new3xImplAsso = .
>.

xDecls = <
  new1xDecls = xDecls xDecl .

  new2xDecls = .
>.

xDecl = <
  new1xDecl = xDeclKey xSingleVar xExplAsso [pos:tPosition] .
         /* = xDeclKey xSingleVar xExplAsso ';' */

  new2xDecl = xPersDecl xIdList [pos1:tPosition] xExpr xExplAsso
              [pos2:tPosition] .
         /* = xPersDecl xIdList ':' xExpr xExplAsso ';' */
>.


xSingleVar = <
  new1xSingleVar = xId [pos:tPosition] xExpr .
              /* = xId ':=' xExpr */

  new2xSingleVar = xId .

  new3xSingleVar = No1:xSingleVar [pos:tPosition] No2:xSingleVar .
          /* = xSingleVar ',' xSingleVar */
>.

xDeclKey = <
```

```
  new1xDeclKey = [pos:tPosition] .
           /* = 'visible' */

  new2xDeclKey = [pos:tPosition] .
           /* = 'hidden' */

  new3xDeclKey = [pos1:tPosition] [pos2:tPosition] .
           /* = 'visible' 'constant' */

  new4xDeclKey = [pos1:tPosition] [pos2:tPosition] .
           /* = 'hidden' 'constant' */

  new5xDeclKey = [pos:tPosition] .
           /* = 'constant' */
>.

xPersDecl = <
  new1xPersDecl = [pos1:tPosition] [pos2:tPosition] .
            /* = 'visible' 'persistent' */

  new2xPersDecl = [pos1:tPosition] [pos2:tPosition] .
            /* = 'hidden' 'persistent' */

  new3xPersDecl = [pos:tPosition] .
            /* = 'persistent' */

  new4xPersDecl = [pos1:tPosition] [pos2:tPosition] [pos3:tPosition] .
            /* = 'visible' 'persistent' 'constant' */

  new5xPersDecl = [pos1:tPosition] [pos2:tPosition] [pos3:tPosition] .
            /* = 'hidden' 'persistent' 'constant' */

  new6xPersDecl = [pos1:tPosition] [pos2:tPosition] .
            /* = 'persistent' 'constant' */
>.

xStmts = <
  new1xStmts  = xStmts xStmt xExplAsso [pos:tPosition] .
          /* = xStmts xStmt xExplAsso ';' */

  new2xStmts = xStmt xExplAsso [pos:tPosition] .
         /* = xStmt xExplAsso ';' */

  new3xStmts = [pos:tPosition] xStmts .
         /* = 'begin' xStmts */

>.
```

```
xExplAsso = <
  new1xExplAsso = xExplAsso [pos:tPosition] xHandAsso .
            /* = xExplAsso 'when' xHandAsso */

  new2xExplAsso = .
>.

xHandAsso = <
  new1xHandAsso = xIdList [pos:tPosition] xId .
            /* = xIdList 'use' xId */

  new2xHandAsso = [pos1:tPosition] [pos2:tPosition] xId .
            /* = 'others' 'use' xId */
>.

xStmt = <
  new1xStmt = [pos:tPosition] .
        /* = 'pass' */

  new2xStmt = [pos:tPosition] .
        /* = 'stop' */

  new3xStmt = [pos:tPosition] xExpr .
        /* = 'stop' xExpr */

  new4xStmt = [pos:tPosition] xExpr .
        /* = 'return' xExpr */

  new5xStmt = [pos:tPosition] .
        /* = 'return' */

  new6xStmt = [pos1:tPosition] [pos2:tPosition] xExpr .
        /* = 'return' 'commit' xExpr */

  new7xStmt = [pos1:tPosition] [pos2:tPosition] .
        /* = 'return' 'commit' */

  new8xStmt = [pos:tPosition] xExpr .
        /* = 'resume' xExpr */

  new9xStmt = [pos:tPosition] .
        /* = 'resume' */

  new10xStmt = xStmtSignal .

  new11xStmt = xStmtNotify .
```

```
new12xStmt = [pos:tPosition] xId xActuList .
        /* = 'escape' xId xActuList */

new13xStmt = xStdIO xActuList .

new14xStmt = [pos1:tPosition] [pos2:tPosition] xLValue
             [pos3:tPosition] xExpr [pos4:tPosition] .
        /* = 'any' '(' xLValue ',' xExpr ')' */

new15xStmt = [pos1:tPosition] [pos2:tPosition] xLValue
             [pos3:tPosition] xExpr [pos4:tPosition] .
        /* = 'rany' '(' xLValue ',' xExpr ')' */

new16xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
             xExpr [pos4:tPosition] .
        /* = 'break' '(' xLValue ',' xExpr ')' */

new17xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
             xExpr [pos4:tPosition] .
        /* = 'rbreak' '(' xLValue ',' xExpr ')' */

new18xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
             xExpr [pos4:tPosition] .
        /* = 'len' '(' xLValue ',' xExpr ')' */

new19xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
             xExpr [pos4:tPosition] .
        /* = 'rlen' '(' xLValue ',' xExpr ')' */

new20xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
             xExpr [pos4:tPosition] .
        /* = 'lpad' '(' xLValue ',' xExpr ')' */

new21xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
             xExpr [pos4:tPosition] .
        /* = 'rpad' '(' xLValue ',' xExpr ')' */

new22xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
             xExpr [pos4:tPosition] .
        /* = 'match' '(' xLValue ',' xExpr ')' */

new23xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
             xExpr [pos4:tPosition] .
        /* = 'rmatch' '(' xLValue ',' xExpr ')' */

new24xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
```

```
                 xExpr [pos4:tPosition] .
         /* = 'notany' '(' xLValue ',' xExpr ')' */

  new25xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
                 xExpr [pos4:tPosition] .
         /* = 'rnotany' '(' xLValue ',' xExpr ')' */

  new26xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
                 xExpr [pos4:tPosition] .
         /* = 'span' '(' xLValue ',' xExpr ')' */

  new27xStmt = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
                 xExpr [pos4:tPosition] .
         /* = 'rspan' '(' xLValue ',' xExpr ')' */

  new28xStmt = xLValue [pos:tPosition] xExpr .
         /* = xLValue ':=' xExpr */

  new29xStmt = xLValue xBinOp [pos:tPosition] xExpr .
         /* = xLValue xBinOp ':=' xExpr */

/*  new30xStmt und new31xStmt sind weggefallen  */

  new32xStmt = No1:xLValue xFrom No2:xLValue .

  new33xStmt = xLValue xActuList .

  new34xStmt = xLambda xActuList .

  new35xStmt = [pos:tPosition] xActuList .
         /* = 'self' xActuList */

  new36xStmt = [pos1:tPosition] xExpr [pos2:tPosition] xStmts xElIfStmts
                 xElseStmts [pos3:tPosition] [pos4:tPosition] .
         /* = 'if' xExpr 'then' xStmts xElIfStmts xElseStmts 'end' 'if' */

  new37xStmt = [pos1:tPosition] xExpr xCaseStmts  xElseStmts [pos2:tPosition]
                 [pos3:tPosition] .
         /* = 'case' xExpr xCaseStmts  xElseStmts 'end' 'case' */

  new38xStmt = xLoops [pos1:tPosition] [pos2:tPosition] .
         /* = xLoops 'end' 'loop' */

  new39xStmt = xLoops [pos1:tPosition] [pos2:tPosition] .
         /* = xLoops 'end' 'for' */

  new40xStmt = xLoops [pos1:tPosition] [pos2:tPosition] .
```

```
        /* = xLoops 'end' 'while' */

new41xStmt = xLoops [pos1:tPosition] [pos2:tPosition] .
        /* = xLoops 'end' 'whilefound' */

new42xStmt = xLoops [pos1:tPosition] [pos2:tPosition] .
        /* = xLoops 'end' 'repeat' */

new43xStmt = xLabel xLoops [pos:tPosition] xId .
        /* = xLabel xLoops 'end' xId */

new44xStmt = [pos:tPosition] .
        /* = 'quit' */

new45xStmt = [pos:tPosition] xId .
        /* = 'quit' xId */

new46xStmt = [pos:tPosition] .
        /* = 'continue' */

new47xStmt = [pos:tPosition] xId .
        /* = 'continue' xId */

new48xStmt = [pos:tPosition] xExpr .
        /* = '||' xExpr */

new49xStmt = [pos1:tPosition] Expr1:xExpr [pos2:tPosition] Expr2:xExpr
             [pos3:tPosition] [pos4:tPosition] .
        /* = 'deposit' xExpr 'at' xExpr 'end' 'deposit' */

new50xStmt = [pos1:tPosition] Expr1:xExpr [pos2:tPosition] Expr2:xExpr
             [pos3:tPosition] [pos4:tPosition] [pos5:tPosition] .
        /* = 'deposit' xExpr 'at' xExpr 'blockiffull' 'end' 'deposit' */

new51xStmt = [pos1:tPosition] xTemplate [pos2:tPosition] xExpr xElseStmts
             [pos3:tPosition] [pos4:tPosition] .
        /* = 'fetch' xTemplate 'at' xExpr xElseStmts 'end' 'fetch' */

new52xStmt = [pos1:tPosition] xTemplate [pos2:tPosition] xExpr xElseStmts
             [pos3:tPosition] [pos4:tPosition] .
        /* = 'meet' xTemplate 'at' xExpr xElseStmts 'end' 'meet' */

new53xStmt = [pos1:tPosition] xId [pos2:tPosition] xTypeList
             [pos3:tPosition] .
        /* = 'c_fct_call' xId '(' xTypeList ')' */

new54xStmt = [pos1:tPosition] [pos2:tPosition] xExpr [pos3:tPosition]
```

```
                xExprList [pos4:tPosition] .
           /* = 'getf' '(' xExpr ',' xExprList ')' */

  new55xStmt = [pos1:tPosition] [pos2:tPosition] xExpr [pos3:tPosition]
                xExprList [pos4:tPosition] .
           /* = 'fget' '(' xExpr ',' xExprList ')' */

  new56xStmt = [pos1:tPosition] [pos2:tPosition] xExpr1:xExpr [pos3:tPosition]
                 xExpr2:xExpr [pos4:tPosition] xExprList [pos5:tPosition] .
           /* = 'fgetf' '(' xExpr ',' xExpr ',' xExprList ')' */
>.

xStmtSignal = <
  new1xStmtSignal = [pos1:tPosition] xLValue [pos2:tPosition] xId xActuList .
                /* = 'signal' xLValue ':=' xId xActuList */

  new2xStmtSignal = [pos:tPosition] xLValue xActuList .
                /* = 'signal' xLValue xActuList */
>.

xStmtNotify = <
  new1xStmtNotify = [pos1:tPosition] xLValue [pos2:tPosition] xId xActuList .
                /* = 'notify' xLValue ':=' xId xActuList */

  new2xStmtNotify = [pos:tPosition] xLValue xActuList .
                /* = 'notify' xLValue xActuList */
>.

xStdIO = <
  new1xStdIO = [pos:tPosition] .
           /* = 'put' */

  new2xStdIO = [pos:tPosition] .
           /* = 'eput' */

  new3xStdIO = [pos:tPosition] .
           /* = 'fput' */

  new4xStdIO = [pos:tPosition] .
           /* = 'putf' */

  new5xStdIO = [pos:tPosition] .
           /* = 'eputf' */

  new6xStdIO = [pos:tPosition] .
           /* = 'fputf' */
```

```
  new7xStdIO = [pos:tPosition] .
          /* = 'get' */

/*   new8xStdIO = [pos:tPosition] . */
          /* = 'fget' */

/*   new9xStdIO = [pos:tPosition] . */
          /* = 'getf' */

/*   new10xStdIO = [pos:tPosition] . */
           /* = 'fgetf' */

  new11xStdIO = [pos:tPosition] .
           /* = 'fopen' */

  new12xStdIO = [pos:tPosition] .
           /* = 'fclose' */
>.

xExpr = <
  new1xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
              xExpr [pos4:tPosition] .
         /* = 'any' '(' xLValue ',' xExpr ')' */

  new2xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
              xExpr [pos4:tPosition] .
         /* = 'rany' '(' xLValue ',' xExpr ')' */

  new3xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
              xExpr [pos4:tPosition] .
         /* = 'break' '(' xLValue ',' xExpr ')' */

  new4xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
              xExpr [pos4:tPosition] .
         /* = 'rbreak' '(' xLValue ',' xExpr ')' */

  new5xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
              xExpr[pos4:tPosition]  .
         /* = 'len' '(' xLValue ',' xExpr ')' */

  new6xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
              xExpr [pos4:tPosition] .
         /* = 'rlen' '(' xLValue ',' xExpr ')' */

  new7xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
              xExpr [pos4:tPosition] .
         /* = 'match' '(' xLValue ',' xExpr ')' */
```

```
new8xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
            xExpr [pos4:tPosition] .
       /* = 'rmatch' '(' xLValue ',' xExpr ')' */

new9xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
            xExpr [pos4:tPosition] .
       /* = 'notany' '(' xLValue ',' xExpr ')' */

new10xExpr =  [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
              xExpr [pos4:tPosition] .
        /* = 'rnotany' '(' xLValue ',' xExpr ')' */

new11xExpr =  [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
              xExpr [pos4:tPosition] .
        /* = 'span' '(' xLValue ',' xExpr ')' */

new12xExpr = [pos1:tPosition] [pos2:tPosition] xLValue [pos3:tPosition]
             xExpr [pos4:tPosition] .
        /* = 'rspan' '(' xLValue ',' xExpr ')' */

new13xExpr = intlit .

new14xExpr = floatlit .

new15xExpr = [pos:tPosition] .
        /* = 'true' */

new16xExpr = [pos:tPosition] .
        /* = 'false' */

new17xExpr = [pos:tPosition] .
        /* = 'om' */

new18xExpr = [pos:tPosition] .
        /* = 'atom' */

new19xExpr = [pos:tPosition] .
        /* = 'boolean' */

new20xExpr = [pos:tPosition] .
        /* = 'integer' */

new21xExpr = [pos:tPosition] .
        /* = 'real' */

new22xExpr = [pos:tPosition] .
```

```
        /* = 'string' */

new23xExpr = [pos:tPosition] .
        /* = 'tuple' */

new24xExpr = [pos:tPosition] .
        /* = 'set' */

new25xExpr = [pos:tPosition] .
        /* = 'function' */

new26xExpr = [pos:tPosition] .
        /* = 'modtype' */

new27xExpr = [pos:tPosition] .
        /* = 'instance' */

new28xExpr = [pos1:tPosition] [pos2:tPosition] .
        /* = '{' '}' */

new29xExpr = [pos1:tPosition] [pos2:tPosition] .
        /* = '[' ']' */

new30xExpr = [pos1:tPosition] [pos2:tPosition] [pos3:tPosition] .
        /* = 'newat' '(' ')' */

new31xExpr = xInstantiate .

new32xExpr = [pos:tPosition] .
        /* = '-' */

new33xExpr = str .

new34xExpr = [pos:tPosition] .
        /* = '$' */

new35xExpr = [pos:tPosition] .
        /* = 'argv' */

new36xExpr = [pos1:tPosition] xFormer [pos2:tPosition] .
        /* = '{' xFormer '}' */

new37xExpr = [pos1:tPosition] xFormer [pos2:tPosition] .
        /* = '[' xFormer ']' */

new38xExpr = xQualId .
```

```
  new39xExpr = xLambda xActuList.

  new40xExpr = [pos:tPosition] xActuList .
          /* = 'self' xActuList */

  new41xExpr = [pos1:tPosition] [pos2:tPosition] .
          /* = 'closure' 'self' */

  new42xExpr = [pos:tPosition] xLambda .
          /* = 'closure' xLambda */

  new43xExpr = [pos:tPosition] xQualId .
          /* = 'closure' xQualId */

  new44xExpr = xExpr xSelector .

  new45xExpr = xExpr [pos1:tPosition] [pos2:tPosition] .
          /* = xExpr '(' ')' */

  new46xExpr = xExpr [pos1:tPosition] xHandAsso [pos2:tPosition] .
          /* = xExpr '[' xHandAsso ']' */

  new47xExpr = [pos1:tPosition] Expr1:xExpr [pos2:tPosition] Expr2:xExpr
               xElIfExprs xElseExpr [pos3:tPosition] [pos4:tPosition] .
          /* = 'if' xExpr 'then' xExpr xElIfExprs xElseExpr 'end' 'if' */

  new48xExpr = [pos1:tPosition] xExpr xCaseExprs xElseExpr [pos2:tPosition]
               [pos3:tPosition] .
          /* = 'case' xExpr xCaseExprs xElseExpr 'end' 'case' */

  new49xExpr = [pos1:tPosition] xExpr [pos2:tPosition] .
          /* = '(' xExpr ')' */

  new50xExpr =  Expr1:xExpr xBinOp Expr2:xExpr .

  new51xExpr = xQuantifier .

  new52xExpr = xUnOp xExpr .

  new53xExpr = [pos1:tPosition] Id1:xId [pos2:tPosition] xTypeList
               [pos3:tPosition] Id2:xId .
          /* = 'c_fct_call' Id1:xId '(' xTypeList ')' Id2:xId */
>.

xFrom = <
  new1xFrom = [pos:tPosition] .
         /* = 'from' */
```

```
  new2xFrom = [pos:tPosition] .
         /* = 'frome' */

  new3xFrom = [pos:tPosition] .
         /* = 'fromb' */
>.

xActuList = <
  new1xActuList = [pos1:tPosition] xExprList [pos2:tPosition] .
             /* = '(' xExprList ')' */

  new2xActuList = [pos1:tPosition] [pos2:tPosition] .
             /* = '(' ')' */
>.

xElIfStmt = [pos1:tPosition] xExpr [pos2:tPosition] xStmts .
       /* = 'elseif' xExpr 'then' xStmts */

xElIfStmts = <
  new1xElIfStmts = xElIfStmts xElIfStmt .

  new2xElIfStmts = .
>.

xElseStmts = <
  new1xElseStmts = [pos:tPosition] xStmts .
              /* = 'else' xStmts */

  new2xElseStmts = .
>.

xCaseStmts = <
  new1xCaseStmts = No1:xCaseStmts No2:xCaseStmts .

  new2xCaseStmts = xCaseStmts .

  new3xCaseStmts = xExprList xStmts .
>.

xLabel = xId [pos:tPosition] .
    /* = xId ':' */

xLoops = <
  new6xLoops  = [pos:tPosition] xStmts .
           /* = 'loop' xStmts */
```

```
  new7xLoops  = [pos1:tPosition] xIterator [pos2:tPosition] xStmts .
          /* = 'for' xIterator 'do' xStmts */

  new8xLoops  = [pos1:tPosition] xExpr [pos2:tPosition] xStmts .
          /* = 'while' xExpr 'do' xStmts */

  new9xLoops  = [pos1:tPosition] xIterator [pos2:tPosition] xStmts .
          /* = 'whilefound' xIterator 'do' xStmts */

  new10xLoops = [pos1:tPosition] xStmts [pos2:tPosition] xExpr .
          /* = 'repeat' xStmts 'until' xExpr */
>.

xTemplate = <
  new1xTemplate = [pos1:tPosition] xExprFormal xTempCond [pos2:tPosition]
                  [pos3:tPosition] xStmts .
            /* = '(' xExprFormal xTempCond ')' '=>' xStmts */

  new2xTemplate = [pos1:tPosition] xExprFormal xTempCond [pos2:tPosition] .
            /* = '(' xExprFormal xTempCond ')' */

  new3xTemplate = No1:xTemplate [pos:tPosition] No2:xTemplate .
             /* = xTemplate 'xor' xTemplate */
>.

xTempCond = <
  new1xTempCond = [pos:tPosition] xExpr .
            /* = '|' xExpr */

  new2xTempCond = .
>.

xExprFormal = <
  new1xExprFormal = xExpr .

  new2xExprFormal = [pos:tPosition] xFormal xInto .
              /* = '?' xFormal xInto */

  new3xExprFormal = .

  new4xExprFormal = No1:xExprFormal [pos:tPosition] No2:xExprFormal .
              /* = xExprFormal ',' xExprFormal */
>.

xInto = <
  new1xInto = [pos:tPosition] xExpr .
         /* = 'into' xExpr */
```

```
  new2xInto = .
>.

xFormal = <
  new1xFormal = xLValue .

  new2xFormal = .
>.

xIterator = <
  new1xIterator = xSimpleIts [pos:tPosition] xExpr .
            /* = xSimpleIts '|' xExpr */

  new2xIterator = xSimpleIts .
>.

xSimpleIts = <
  new1xSimpleIts = xSimpleIts [pos:tPosition] xSimpleIt .
             /* = xSimpleIts ',' xSimpleIt */

  new2xSimpleIts = xSimpleIt .
>.

xSimpleIt = <
  new1xSimpleIt = xLValue [pos:tPosition] xExpr .
            /* = xLValue 'in' xExpr */

  new2xSimpleIt = xLValue [pos:tPosition] xId xMapSel .
            /* = xLValue '=' xId xMapSel */
>.

xMapSel = <
  new1xMapSel = [pos1:tPosition] xLValue  [pos2:tPosition].
          /* = '(' xLValue ')' */

  new2xMapSel = [pos1:tPosition] xLValue [pos2:tPosition] .
          /* '{' xLValue '}' */
>.

xLValue = <
  new2xLValue = [pos1:tPosition] xLValue [pos2:tPosition] .
          /* = '[' xLValue ']' */

  new3xLValue = No1:xLValue [pos:tPosition] No2:xLValue .
          /* = xLValue ',' xLValue */
```

```
  new4xLValue = [pos:tPosition] .
           /* = '-' */

  new5xLValue = xQualId .

  new6xLValue = xLValue xSelector .
>.

xSelector = <
  new1xSelector = [pos1:tPosition] xExprList [pos2:tPosition] .
             /* = '(' xExprList ')' */

  new2xSelector = [pos1:tPosition] xExprList [pos2:tPosition] .
             /* = '{' xExprList '}' */

  new3xSelector = [pos1:tPosition] xExpr [pos2:tPosition] [pos3:tPosition] .
             /* = '(' xExpr '..' ')' */

  new4xSelector = [pos1:tPosition] Expr1:xExpr [pos2:tPosition] Expr2:xExpr
                  [pos3:tPosition] .
             /* = '(' xExpr '..' xExpr ')' */
>.

xFormer = <
  new1xFormer = xExpr .

  new2xFormer = Expr1:xExpr [pos:tPosition] Expr2:xExpr .
           /* = xExpr '..' xExpr */

  new3xFormer = xExpr [pos:tPosition] xIterator .
           /* = xExpr ':' xIterator */

  new4xFormer = xExpr [pos:tPosition] xExprList .
           /* = xExpr ',' xExprList */

  new5xFormer = Expr1:xExpr [pos1:tPosition] Expr2:xExpr [pos2:tPosition]
                Expr3:xExpr .
           /* = xExpr ',' xExpr '..' xExpr */
>.

xExprList = <
  new1xExprList = xExprList [pos:tPosition] xExpr .
             /* = xExprList ',' xExpr */

  new2xExprList = xExpr .

  new3xExprList = [pos1:tPosition] xExprList [pos2:tPosition] .
```

```
            /* = 'when' xExprList '=>' */

>.

xInstantiate = [pos1:tPosition] xExpr xInstExport xInstImport [pos2:tPosition]
               [pos3:tPosition] .
          /* = 'instantiate' xExpr xInstExport xInstImport 'end'
               'instantiate' */

xInstExport = <
  new1xInstExport = [pos1:tPosition] xExprList [pos2:tPosition] .
             /* = 'export' xExprList ';' */

  new2xInstExport = .
>.

xInstImport = <
  new1xInstImport = [pos1:tPosition] xIdList [pos2:tPosition] .
             /* = 'import' xIdList ';' */

  new2xInstImport = .
>.

xQualId = <
  new1xQualId = Id1:xId [pos:tPosition] Id2:xId .
          /* = xId '.' xId */

  new2xQualId = xId .
>.

xLambda = [pos1:tPosition] xParamList [pos2:tPosition] xProgBody
          [pos3:tPosition] [pos4:tPosition] .
     /* = 'lambda' xParamList ':' xProgBody 'end' 'lambda' */

xQuantifier = xQualifier xSimpleIts [pos:tPosition] xExpr .
         /* = xQualifier xSimpleIts '|' xExpr */

xQualifier = <
  new1xQualifier = [pos:tPosition] .
            /* = 'exists' */

  new2xQualifier = [pos:tPosition] .
            /* = 'forall' */
>.

xElIfExprs = <
  new1xElIfExprs = xElIfExprs xElIfExpr .
```

```
  new2xElIfExprs = .
>.

xElIfExpr = [pos1:tPosition] Expr1:xExpr [pos2:tPosition] Expr2:xExpr .
       /* = 'elseif' xExpr 'then' xExpr */

xElseExpr = <
  new1xElseExpr = [pos:tPosition] xExpr .
             /* ='else' xExpr */

  new2xElseExpr = .
>.

xCaseExprs = <
  new1xCaseExprs = No1:xCaseExprs No2:xCaseExprs .

  new2xCaseExprs = xCaseExprs .

  new3xCaseExprs = xExprList xExpr .
>.

xBinOp = <
  new1xBinOp = [pos:tPosition] .
          /* = 'or' */

  new2xBinOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'or' '%' */

  new3xBinOp = [pos:tPosition] .
          /* = 'and' */

  new4xBinOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'and' '%' */

  new5xBinOp = [pos:tPosition] .
          /* = '=' */

  new6xBinOp = [pos:tPosition] .
          /* = '/=' */

  new7xBinOp = [pos:tPosition] .
          /* = '<' */

  new8xBinOp = [pos:tPosition] .
          /* = '<=' */
```

```
new9xBinOp = [pos:tPosition] .
        /* = '>' */

new10xBinOp = [pos:tPosition] .
         /* = '>=' */

new11xBinOp = [pos:tPosition] .
         /* = 'in' */

new12xBinOp = [pos:tPosition] .
         /* = 'notin' */

new13xBinOp = [pos:tPosition] .
         /* = 'subset' */

new14xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = '=' '%' */

new15xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = '/=' '%' */

new16xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = '<' '%' */

new17xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = '<=' '%' */

new18xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = '>' '%' */

new19xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = '>=' '%' */

new20xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = 'in' '%' */

new21xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = 'notin' '%' */

new22xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = 'subset' '%' */

new23xBinOp = [pos:tPosition] .
         /* = 'with' */

new24xBinOp = [pos:tPosition] .
         /* = 'less' */
```

```
new25xBinOp = [pos:tPosition] .
         /* = 'lessf' */

new26xBinOp = [pos:tPosition] xQualId .
         /* = '!' xQualId */

new27xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = 'with' '%' */

new28xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = 'less' '%' */

new29xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = 'lessf' '%' */

new30xBinOp = [pos1:tPosition] xQualId [pos2:tPosition] .
         /* = '!' xQualId '%' */

new31xBinOp = [pos:tPosition] .
         /* = '+' */

new32xBinOp = [pos:tPosition] .
         /* = '-' */

new33xBinOp = [pos:tPosition] .
         /* = 'max' */

new34xBinOp = [pos:tPosition] .
         /* = 'min' */

new35xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = '+' '%' */

new36xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = '-' '%' */

new37xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = 'max' '%' */

new38xBinOp = [pos1:tPosition] [pos2:tPosition] .
         /* = 'min' '%' */

new39xBinOp = [pos:tPosition] .
         /* = '*' */

new40xBinOp = [pos:tPosition] .
```

```
          /* = '/' */

  new41xBinOp = [pos:tPosition] .
          /* = 'mod' */

  new42xBinOp = [pos1:tPosition] [pos2:tPosition] .
          /* = '*' '%' */

  new43xBinOp = [pos1:tPosition] [pos2:tPosition] .
          /* = '/' '%' */

  new44xBinOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'mod' '%' */

  new45xBinOp = [pos:tPosition] .
          /* = '**' */

  new46xBinOp = [pos1:tPosition] [pos2:tPosition] .
          /* = '**' '%' */

  new47xBinOp = [pos:tPosition] .
          /* = 'npow' */

>.

xUnOp = <
  new1xUnOp = [pos:tPosition] .
        /* = '+' */

  new2xUnOp = [pos:tPosition] .
        /* = '-' */

  new3xUnOp = [pos:tPosition] .
        /* = '#' */

  new4xUnOp = [pos:tPosition] .
        /* = '||' */

  new5xUnOp = [pos:tPosition] .
        /* = 'not' */

  new6xUnOp = [pos:tPosition] .
        /* = 'pow' */

  new7xUnOp = [pos:tPosition] .
        /* = 'arb' */
```

```
new8xUnOp = [pos:tPosition] .
       /* = 'random' */

new9xUnOp = [pos:tPosition] .
       /* = 'domain' */

new10xUnOp = [pos:tPosition] .
        /* = 'range' */

new11xUnOp = [pos:tPosition] .
        /* = 'type' */

new12xUnOp = [pos:tPosition] .
        /* = 'is_map' */

new13xUnOp = [pos:tPosition] .
        /* = 'is_smap' */

new14xUnOp = [pos:tPosition] xQualId .
        /* = '!' xQualId */

new15xUnOp = [pos1:tPosition] [pos2:tPosition] .
        /* = 'or' '%' */

new16xUnOp = [pos1:tPosition] [pos2:tPosition] .
        /* = 'and' '%' */

new17xUnOp = [pos1:tPosition] [pos2:tPosition] .
        /* = '=' '%' */

new18xUnOp = [pos1:tPosition] [pos2:tPosition] .
        /* = '/=' '%' */

new19xUnOp = [pos1:tPosition] [pos2:tPosition] .
        /* = '<' '%' */

new20xUnOp = [pos1:tPosition] [pos2:tPosition] .
        /* = '<=' '%' */

new21xUnOp = [pos1:tPosition] [pos2:tPosition] .
        /* = '>' '%' */

new22xUnOp = [pos1:tPosition] [pos2:tPosition] .
        /* = '>=' '%' */

new23xUnOp = [pos1:tPosition] [pos2:tPosition] .
        /* = 'in' '%' */
```

```
  new24xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'notin' '%' */

  new25xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'subset' '%' */

  new26xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'with' '%' */

  new27xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'less' '%' */

  new28xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'lessf' '%' */

  new29xUnOp = [pos1:tPosition] xQualId [pos2:tPosition] .
          /* = '!' xQualId '%' */

  new30xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = '+' '%' */

  new31xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = '-' '%' */

  new32xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'max' '%' */

  new33xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'min' '%' */

  new34xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = '*' '%' */

  new35xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = '/' '%' */

  new36xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = 'mod' '%' */

  new37xUnOp = [pos1:tPosition] [pos2:tPosition] .
          /* = '**' '%' */
>.

xTypeList = <
  new1xTypeList = .
```

```
  new2xTypeList = Id1:xId [pos:tPosition]  Id2:xId .
            /* = Id1:xId ':' Id2:xId */

  new3xTypeList = xTypeList [pos1:tPosition] Id1:xId [pos2:tPosition] Id2:xId .
            /* = xTypeList ',' Id1:xId ':' Id2:xId */
>.
```

C.42.4 Die Fortsetzung des Attributauswerters Transhelp1

⟨*transhelp1*⟩≡
```
xLValue     = { helpOut :- helpIn;}.

new2xLValue = { xLValue:helpIn :- helpIn;
                helpOut :- xLValue:helpOut;}.

new3xLValue = { No1:helpIn :- helpIn;
                No2:helpIn :- No1:helpOut;
                helpOut :- No2:helpOut;}.

new4xLValue = { helpOut :- helpIn;}.

new5xLValue = { xQualId:helpIn :- helpIn;
                helpOut :- xQualId:helpOut;}.

new6xLValue = { xLValue:helpIn :- helpIn;
                xSelector:helpIn :- xLValue:helpOut;
                helpOut :- xSelector:helpOut;}.

xProgDefn = { xId1:helpIn :- helpIn;
              xProgBody:helpIn :- xId1:helpOut;
              xId2:helpIn :- xProgBody:helpOut;
              helpOut :- xId2:helpOut;}.

xProgBody = { xDecls:helpIn :- helpIn;
              xStmts:helpIn :- xDecls:helpOut;
              xPHMDefn:helpIn :- xStmts:helpOut;
              helpOut :- xPHMDefn:helpOut;}.

xId = { id:helpIn :- helpIn;
        helpOut :- id:helpOut;}.

xPHMDefn     = { helpOut :- helpIn;}.

new1xPHMDefn = { xId1:helpIn :- helpIn;
                 xParamList:helpIn :- xId1:helpOut;
                 xProgBody:helpIn :- xParamList:helpOut;
                 xId2:helpIn :- xProgBody:helpOut;
                 helpOut :- xId2:helpOut;}.

new2xPHMDefn = { xId1:helpIn :- helpIn;
                 xModImport:helpIn :- xId1:helpOut;
                 xModExport:helpIn :- xModImport:helpOut ;
                 xProgBody:helpIn :- xModExport:helpOut;
                 xId2:helpIn :- xProgBody:helpOut;
```

```
                 helpOut :- xId2:helpOut;}.

new3xPHMDefn = { xId1:helpIn :- helpIn;
                 xRdParamList:helpIn :- xId1:helpOut;
                 xImplAsso:helpIn :- xRdParamList:helpOut;
                 xProgBody:helpIn :- xImplAsso:helpOut;
                 xId2:helpIn :- xProgBody:helpOut;
                 helpOut :- xId2:helpOut;}.

new4xPHMDefn = { No1:helpIn :- helpIn;
                 No2:helpIn :- No1:helpOut;
                 helpOut :- No2:helpOut;}.

new5xPHMDefn = { helpOut :- helpIn;}.

xParamList     = { helpOut :- helpIn;}.

new1xParamList = { helpOut :- helpIn;}.

new2xParamList = { xParamList:helpIn :- helpIn;
                   helpOut :- xParamList:helpOut;}.

new3xParamList = { xParamList:helpIn :- helpIn;
                   xParamMode:helpIn :- xParamList:helpOut;
                   xId:helpIn :- xParamMode:helpOut;
                   helpOut :- xId:helpOut;}.

new4xParamList = { xParamMode:helpIn :- helpIn;
                   xId:helpIn :- xParamMode:helpOut;
                   helpOut :- xId:helpOut;}.

xParamMode     = { helpOut :- helpIn;}.

new1xParamMode = { helpOut :- helpIn;}.

new2xParamMode = { helpOut :- helpIn;}.

new3xParamMode = { helpOut :- helpIn;}.

new4xParamMode = { helpOut :- helpIn;}.

xModImport     = { helpOut :- helpIn;}.

new1xModImport = { xIdList:helpIn :- helpIn;
                   helpOut :- xIdList:helpOut;}.

new2xModImport = { helpOut :- helpIn;}.
```

```
xModExport     = { helpOut :- helpIn;}.

new1xModExport = { xIdList:helpIn :- helpIn;
                   helpOut :- xIdList:helpOut;}.

new2xModExport = { helpOut :- helpIn;}.

xIdList     = { helpOut :- helpIn;}.

new1xIdList = { xIdList:helpIn :- helpIn;
                xId:helpIn :- xIdList:helpOut;
                helpOut :- xId:helpOut;}.

new2xIdList = { xId:helpIn :- helpIn;
                helpOut :- xId:helpOut;}.

xRdParamList     = { helpOut :- helpIn;}.

new1xRdParamList = { helpOut :- helpIn;}.

new2xRdParamList = { xIdList:helpIn :- helpIn;
                     helpOut :- xIdList:helpOut;}.

xImplAsso     = { helpOut :- helpIn;}.

new1xImplAsso = { xIdList:helpIn :- helpIn;
                  helpOut :- xIdList:helpOut;}.

new2xImplAsso = { helpOut :- helpIn;}.

new3xImplAsso = { helpOut :- helpIn;}.

xDecls     = { helpOut :- helpIn;}.

new1xDecls = { xDecls:helpIn :- helpIn;
               xDecl:helpIn :- xDecls:helpOut;
               helpOut :- xDecl:helpOut;}.

new2xDecls = { helpOut :- helpIn;}.

xDecl     = { helpOut :- helpIn;}.

new1xDecl = { xDeclKey:helpIn :- helpIn;
              xSingleVar:helpIn :- xDeclKey:helpOut;
              xExplAsso:helpIn :- xSingleVar:helpOut;
              helpOut :- xExplAsso:helpOut;}.
```

```
new2xDecl = { xPersDecl:helpIn :- helpIn;
              xIdList:helpIn :- xPersDecl:helpOut;
              xExpr:helpIn :- xIdList:helpOut;
              xExplAsso:helpIn :- xExpr:helpOut;
              helpOut :- xExplAsso:helpOut;}.

xSingleVar     = { helpOut :- helpIn;}.

new1xSingleVar = { xId:helpIn :- helpIn;
                   xExpr:helpIn :- xId:helpOut;
                   helpOut :- xExpr:helpOut; } .

new2xSingleVar = { xId:helpIn :- helpIn;
                   helpOut :- xId:helpOut;}.

new3xSingleVar = { No1:helpIn :- helpIn;
                   No2:helpIn :- No1:helpOut;
                   helpOut :- No2:helpOut;}.

xDeclKey     = { helpOut :- helpIn;}.

new1xDeclKey = { helpOut :- helpIn;}.

new2xDeclKey = { helpOut :- helpIn;}.

new3xDeclKey = { helpOut :- helpIn;}.

new4xDeclKey = { helpOut :- helpIn;}.

new5xDeclKey = { helpOut :- helpIn;}.

xPersDecl     = { helpOut :- helpIn;}.

new1xPersDecl = { helpOut :- helpIn;}.

new2xPersDecl = { helpOut :- helpIn;}.

new3xPersDecl = { helpOut :- helpIn;}.

new4xPersDecl = { helpOut :- helpIn;}.

new5xPersDecl = { helpOut :- helpIn;}.

new6xPersDecl = { helpOut :- helpIn;}.

xStmts      = { helpOut :- helpIn;}.
```

```
new1xStmts  = { xStmts:helpIn :- helpIn;
                xStmt:helpIn :- xStmts:helpOut;
                xExplAsso:helpIn :- xStmt:helpOut;
                helpOut :- xExplAsso:helpOut;}.

new2xStmts  = { xStmt:helpIn :- helpIn;
                xExplAsso:helpIn :- xStmt:helpOut;
                helpOut :- xExplAsso:helpOut;}.

new3xStmts  = { xStmts:helpIn :- helpIn;
                helpOut :- xStmts:helpIn;}.

xExplAsso = { helpOut :- helpIn;}.

new1xExplAsso = { xExplAsso:helpIn :- helpIn;
                  xHandAsso:helpIn :- xExplAsso:helpOut;
                  helpOut :- xHandAsso:helpOut;}.

new2xExplAsso = { helpOut :- helpIn;}.

xHandAsso = { helpOut :- helpIn;}.

new1xHandAsso = { xIdList:helpIn :- helpIn;
                  xId:helpIn :- xIdList:helpOut;
                  helpOut :- xId:helpOut;}.

new2xHandAsso = { xId:helpIn :- helpIn;
                  helpOut :- xId:helpOut;}.

xStmt      = { helpOut :- helpIn;}.

new1xStmt  = { helpOut :- helpIn;}.

new2xStmt  = { helpOut :- helpIn;}.

new3xStmt  = { xExpr:helpIn :- helpIn;
               helpOut :- xExpr:helpOut;}.

new4xStmt  = { xExpr:helpIn :- helpIn;
               helpOut :- xExpr:helpOut;}.

new5xStmt  = { helpOut :- helpIn;}.

new6xStmt  = { xExpr:helpIn :- helpIn;
               helpOut :- xExpr:helpOut;}.
```

```
new7xStmt  = { helpOut :- helpIn;}.

new8xStmt  = { xExpr:helpIn :- helpIn;
               helpOut :- xExpr:helpOut;}.

new9xStmt  = { helpOut :- helpIn;}.

new10xStmt = { xStmtSignal:helpIn :- helpIn;
               helpOut :- xStmtSignal:helpOut;}.

new11xStmt = { xStmtNotify:helpIn :- helpIn;
               helpOut :- xStmtNotify:helpOut;}.

new12xStmt = { xId:helpIn :- helpIn;
               xActuList:helpIn :- xId:helpOut;
               helpOut :- xActuList:helpOut;}.

new13xStmt = { xStdIO:helpIn :- helpIn;
               xActuList:helpIn :- xStdIO:helpOut;
               helpOut :- xActuList:helpOut;}.

new14xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new15xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new16xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new17xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new18xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new19xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new20xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
```

```
               helpOut :- xExpr:helpOut;}.

new21xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new22xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new23xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new24xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new25xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new26xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new27xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new28xStmt = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new29xStmt = { xLValue:helpIn :- helpIn;
               xBinOp:helpIn :- xLValue:helpOut;
               xExpr:helpIn :- xBinOp:helpOut;
               helpOut :- xExpr:helpOut;}.

new32xStmt = { No1:helpIn :- helpIn;
               xFrom:helpIn :- No1:helpOut;
               No2:helpIn :- xFrom:helpOut;
               helpOut :- No2:helpOut;}.

new33xStmt = { xLValue:helpIn :- helpIn;
               xActuList:helpIn :- xLValue:helpOut;
               helpOut :- xActuList:helpOut;}.
```

```
new34xStmt = { xLambda:helpIn :- helpIn;
               xActuList:helpIn :- xLambda:helpOut;
               helpOut :- xActuList:helpOut;}.

new35xStmt = { xActuList:helpIn :- helpIn;
               helpOut :- xActuList:helpOut;}.

new36xStmt = { xExpr:helpIn :- helpIn;
               xStmts:helpIn :- xExpr:helpOut;
               xElIfStmts:helpIn :- xStmts:helpOut;
               xElseStmts:helpIn :- xElIfStmts:helpOut;
               helpOut :- xElseStmts:helpOut;}.

new37xStmt = { xExpr:helpIn :- helpIn;
               xCaseStmts:helpIn :- xExpr:helpOut;
               xElseStmts:helpIn :- xCaseStmts:helpOut;
               helpOut :- xElseStmts:helpOut;}.

new38xStmt = { xLoops:helpIn :- helpIn;
               helpOut :- xLoops:helpOut;}.

new39xStmt = { xLoops:helpIn :- helpIn;
               helpOut :- xLoops:helpOut;}.

new40xStmt = { xLoops:helpIn :- helpIn;
               helpOut :- xLoops:helpOut;}.

new41xStmt = { xLoops:helpIn :- helpIn;
               helpOut :- xLoops:helpOut;}.

new42xStmt = { xLoops:helpIn :- helpIn;
               helpOut :- xLoops:helpOut;}.

new43xStmt = { xLabel:helpIn :- helpIn;
               xLoops:helpIn :- xLabel:helpOut;
               xId:helpIn :- xLoops:helpOut;
               helpOut :- xId:helpOut;}.

new44xStmt = { helpOut :- helpIn;}.

new45xStmt = { xId:helpIn :- helpIn;
               helpOut :- xId:helpOut;}.

new46xStmt = { helpOut :- helpIn;}.

new47xStmt = { xId:helpIn :- helpIn;
```

```
              helpOut :- xId:helpOut;}.

new48xStmt = { xExpr:helpIn :- helpIn;
               helpOut :- xExpr:helpOut;}.

new49xStmt = { Expr1:helpIn :- helpIn;
               Expr2:helpIn :- Expr1:helpOut;
               helpOut :- Expr2:helpOut;}.

new50xStmt = { Expr1:helpIn :- helpIn;
               Expr2:helpIn :- Expr1:helpOut;
               helpOut :- Expr2:helpOut;}.

new51xStmt = { xTemplate:helpIn :- helpIn;
               xExpr:helpIn :- xTemplate:helpOut;
               xElseStmts:helpIn :- xExpr:helpOut;
               helpOut :- xElseStmts:helpOut;}.

new52xStmt = { xTemplate:helpIn :- helpIn;
               xExpr:helpIn :- xTemplate:helpOut;
               xElseStmts:helpIn :- xExpr:helpOut;
               helpOut :- xElseStmts:helpOut;}.

new53xStmt = { xId:helpIn :- helpIn;
               xTypeList:helpIn :- xId:helpOut;
               helpOut :- xTypeList:helpOut;}.

new54xStmt = { xExpr:helpIn :- helpIn;
               xExprList:helpIn :- xExpr:helpOut;
               helpOut :- xExprList:helpOut;}.

new55xStmt = { xExpr:helpIn :- helpIn;
               xExprList:helpIn :- xExpr:helpOut;
               helpOut :- xExprList:helpOut;}.

new56xStmt = { xExpr1:helpIn :- helpIn;
               xExpr2:helpIn :- xExpr1:helpOut;
               xExprList:helpIn :- xExpr2:helpOut;
               helpOut :- xExprList:helpOut;}.

xStmtSignal     = { helpOut :- helpIn;}.

new1xStmtSignal = { xLValue:helpIn :- helpIn;
                    xId:helpIn :- xLValue:helpOut;
                    xActuList:helpIn :- xId:helpOut;
                    helpOut :- xActuList:helpOut;}.
```

```
new2xStmtSignal = { xLValue:helpIn :- helpIn;
                    xActuList:helpIn :- xLValue:helpOut;
                    helpOut :- xActuList:helpOut;}.

xStmtNotify     = { helpOut :- helpIn;}.

new1xStmtNotify = { xLValue:helpIn :- helpIn;
                    xId:helpIn :- xLValue:helpOut;
                    xActuList:helpIn :- xId:helpOut;
                    helpOut :- xActuList:helpOut;}.

new2xStmtNotify = { xLValue:helpIn :- helpIn;
                    xActuList:helpIn :- xLValue:helpOut;
                    helpOut :- xActuList:helpOut;}.

xStdIO      = { helpOut :- helpIn;}.

new1xStdIO  = { helpOut :- helpIn;}.

new2xStdIO  = { helpOut :- helpIn;}.

new3xStdIO  = { helpOut :- helpIn;}.

new4xStdIO  = { helpOut :- helpIn;}.

new5xStdIO  = { helpOut :- helpIn;}.

new6xStdIO  = { helpOut :- helpIn;}.

new7xStdIO  = { helpOut :- helpIn;}.

/* new8xStdIO  = { helpOut :- helpIn;}. */
/* new9xStdIO  = { helpOut :- helpIn;}. */
/* new10xStdIO = { helpOut :- helpIn;}. */

new11xStdIO = { helpOut :- helpIn;}.

new12xStdIO = { helpOut :- helpIn;}.

xExpr      = { helpOut :- helpIn;}.

new1xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new2xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
```

```
               helpOut :- xExpr:helpOut;}.

new3xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new4xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new5xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new6xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new7xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new8xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new9xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new10xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new11xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new12xExpr  = { xLValue:helpIn :- helpIn;
               xExpr:helpIn :- xLValue:helpOut;
               helpOut :- xExpr:helpOut;}.

new13xExpr = { intlit:helpIn :- helpIn;
               helpOut :- intlit:helpOut;}.

new14xExpr = { floatlit:helpIn :- helpIn;
               helpOut :- floatlit:helpOut;}.
```

```
new15xExpr = { helpOut :- helpIn;}.

new16xExpr = { helpOut :- helpIn;}.

new17xExpr = { helpOut :- helpIn;}.

new18xExpr = { helpOut :- helpIn;}.

new19xExpr = { helpOut :- helpIn;}.

new20xExpr = { helpOut :- helpIn;}.

new21xExpr = { helpOut :- helpIn;}.

new22xExpr = { helpOut :- helpIn;}.

new23xExpr = { helpOut :- helpIn;}.

new24xExpr = { helpOut :- helpIn;}.

new25xExpr = { helpOut :- helpIn;}.

new26xExpr = { helpOut :- helpIn;}.

new27xExpr = { helpOut :- helpIn;}.

new28xExpr = { helpOut :- helpIn;}.

new29xExpr = { helpOut :- helpIn;}.

new30xExpr = { helpOut :- helpIn;}.

new31xExpr = { xInstantiate:helpIn :- helpIn;
               helpOut :- xInstantiate:helpOut;}.

new32xExpr = { helpOut :- helpIn;}.

new33xExpr = { str:helpIn :- helpIn;
               helpOut :- str:helpOut;}.

new34xExpr = { helpOut :- helpIn;}.

new35xExpr = { helpOut :- helpIn;}.

new36xExpr = { xFormer:helpIn :- helpIn;
               helpOut :- xFormer:helpOut;}.
```

```
new37xExpr = { xFormer:helpIn :- helpIn;
               helpOut :- xFormer:helpOut;}.

new38xExpr = { xQualId:helpIn :- helpIn;
               helpOut :- xQualId:helpOut;}.

new39xExpr = { xLambda:helpIn :- helpIn;
              xActuList:helpIn :- xLambda:helpOut;
              helpOut :- xActuList:helpOut;}.

new40xExpr = { xActuList:helpIn :- helpIn;
               helpOut :- xActuList:helpOut;}.

new41xExpr = { helpOut :- helpIn;}.

new42xExpr = { xLambda:helpIn :- helpIn;
               helpOut :- xLambda:helpOut;}.

new43xExpr = { xQualId:helpIn :- helpIn;
               helpOut :- xQualId:helpOut;}.

new44xExpr = { xExpr:helpIn :- helpIn;
               xSelector:helpIn :- xExpr:helpOut;}.

new45xExpr = { xExpr:helpIn :- helpIn;
               helpOut :- xExpr:helpOut;}.

new46xExpr = { xExpr:helpIn :- helpIn;
               xHandAsso:helpIn :- xExpr:helpOut;
               helpOut :- xHandAsso:helpOut;}.

new47xExpr = { Expr1:helpIn :- helpIn;
               Expr2:helpIn :- Expr1:helpOut;
               xElIfExprs:helpIn :- Expr2:helpOut;
               xElseExpr:helpIn :- xElIfExprs:helpOut;
               helpOut :- xElseExpr:helpOut;}.

new48xExpr = { xExpr:helpIn :- helpIn;
               xCaseExprs:helpIn :- xExpr:helpOut;
               xElseExpr:helpIn :- xCaseExprs:helpOut;
               helpOut :- xElseExpr:helpOut;}.

new49xExpr = { xExpr:helpIn :- helpIn;
               helpOut :- xExpr:helpOut;}.

new50xExpr = { Expr1:helpIn :- helpIn;
```

```
               xBinOp:helpIn :- Expr1:helpOut;
               Expr2:helpIn :- xBinOp:helpOut;
               helpOut :- Expr2:helpOut;}.

new51xExpr = { xQuantifier:helpIn :- helpIn;
               helpOut :- xQuantifier:helpOut;}.

new52xExpr = { xUnOp:helpIn :- helpIn;
               xExpr:helpIn :- xUnOp:helpOut;
               helpOut :- xExpr:helpOut;}.

new53xExpr = { Id1:helpIn :- helpIn;
               xTypeList:helpIn :- Id1:helpOut;
               Id2:helpIn :- xTypeList:helpOut;
               helpOut :- Id2:helpOut;}.

xFrom     = { helpOut :- helpIn;}.

new1xFrom = { helpOut :- helpIn;}.

new2xFrom = { helpOut :- helpIn;}.

new3xFrom = { helpOut :- helpIn;}.

xActuList     = { helpOut :- helpIn;}.

new1xActuList = { xExprList:helpIn :- helpIn;
                  helpOut :- xExprList:helpOut;}.

new2xActuList = { helpOut :- helpIn;}.

xElIfStmt = { xExpr:helpIn :- helpIn;
              xStmts:helpIn :- xExpr:helpOut;
              helpOut :- xStmts:helpOut;}.

xElIfStmts     = { helpOut :- helpIn;}.

new1xElIfStmts = { xElIfStmts:helpIn :- helpIn;
                   xElIfStmt:helpIn :- xElIfStmts:helpOut;
                   helpOut :- xElIfStmt:helpOut;}.

new2xElIfStmts = { helpOut :- helpIn;}.

xElseStmts     = { helpOut :- helpIn;}.

new1xElseStmts = { xStmts:helpIn :- helpIn;
                   helpOut :- xStmts:helpOut;}.
```

```
new2xElseStmts = { helpOut :- helpIn;}.

xCaseStmts     = { helpOut :- helpIn;}.

new1xCaseStmts = { No1:helpIn :- helpIn;
                   No2:helpIn :- No1:helpOut;
                   helpOut :- No2:helpOut;}.

new2xCaseStmts = { xCaseStmts:helpIn :- helpIn;
                   helpOut :- xCaseStmts:helpOut;}.

new3xCaseStmts = { xExprList:helpIn :- helpIn;
                   xStmts:helpIn :- xExprList:helpOut;
                   helpOut :- xStmts:helpOut;}.

xLabel = { xId:helpIn :- helpIn;
           helpOut :- xId:helpOut;}.

xTemplate     = { helpOut :- helpIn;}.

new1xTemplate = { xExprFormal:helpIn :- helpIn;
                  xTempCond:helpIn :- xExprFormal:helpOut;
                  xStmts:helpIn :- xTempCond:helpOut;
                  helpOut :- xStmts:helpOut;}.

new2xTemplate = { xExprFormal:helpIn :- helpIn;
                  xTempCond:helpIn :- xExprFormal:helpOut;
                  helpOut :- xTempCond:helpOut;}.

new3xTemplate = { No1:helpIn :- helpIn;
                  No2:helpIn :- No1:helpOut;
                  helpOut :- No2:helpOut;}.

xTempCond     = { helpOut :- helpIn;}.

new1xTempCond = { xExpr:helpIn :- helpIn;
                  helpOut :- xExpr:helpOut;}.

new2xTempCond = { helpOut :- helpIn;}.

xExprFormal     = { helpOut :- helpIn;}.

new1xExprFormal = { xExpr:helpIn :- helpIn;
                    helpOut :- xExpr:helpOut;}.

new2xExprFormal = { xFormal:helpIn :- helpIn;
```

```
                    xInto:helpIn :- xFormal:helpOut;
                    helpOut :- xInto:helpOut;}.

new3xExprFormal = { helpOut :- helpIn;}.

new4xExprFormal = { No1:helpIn :- helpIn;
                    No2:helpIn :- No1:helpOut;
                    helpOut :- No2:helpOut;}.

xInto     = { helpOut :- helpIn;}.

new1xInto = { xExpr:helpIn :- helpIn;
              helpOut :- xExpr:helpOut;}.

new2xInto = { helpOut :- helpIn;}.

xFormal     = { helpOut :- helpIn;}.

new1xFormal = { xLValue:helpIn :- helpIn;
                helpOut :- xLValue:helpOut;}.

new2xFormal = { helpOut :- helpIn;}.

xIterator     = { helpOut :- helpIn;}.

new1xIterator = { xSimpleIts:helpIn :- helpIn;
                  xExpr:helpIn :- xSimpleIts:helpOut;
                  helpOut :- xExpr:helpOut;}.

new2xIterator = { xSimpleIts:helpIn :- helpIn;
                  helpOut :- xSimpleIts:helpOut;}.

xSimpleIts     = { helpOut :- helpIn;}.

new1xSimpleIts = { xSimpleIts:helpIn :- helpIn;
                   xSimpleIt:helpIn :- xSimpleIts:helpOut;
                   helpOut :- xSimpleIt:helpOut;}.

new2xSimpleIts = { xSimpleIt:helpIn :- helpIn;
                   helpOut :- xSimpleIt:helpOut;}.

xSimpleIt = { helpOut :- helpIn;}.

new1xSimpleIt = { xLValue:helpIn :- helpIn;
                  xExpr:helpIn :- xLValue:helpOut;
                  helpOut :- xExpr:helpOut;}.
```

```
new2xSimpleIt = { xLValue:helpIn :- helpIn;
                  xId:helpIn :- xLValue:helpOut;
                  xMapSel:helpIn :- xId:helpOut;
                  helpOut :- xMapSel:helpOut;}.

xMapSel     = { helpOut :- helpIn;}.

new1xMapSel = { xLValue:helpIn :- helpIn;
                helpOut :- xLValue:helpOut;}.

new2xMapSel = { xLValue:helpIn :- helpIn;
                helpOut :- xLValue:helpOut;}.

xSelector     = { helpOut :- helpIn;}.

new1xSelector = { xExprList:helpIn :- helpIn;
                  helpOut :- xExprList:helpOut;}.

new2xSelector = { xExprList:helpIn :- helpIn;
                  helpOut :- xExprList:helpOut;}.

new3xSelector = { xExpr:helpIn :- helpIn;
                  helpOut :-  xExpr:helpOut;}.

new4xSelector = { Expr1:helpIn :- helpIn;
                  Expr2:helpIn :- Expr1:helpOut;
                  helpOut :- Expr2:helpOut;}.

xFormer     = { helpOut :- helpIn;}.

new1xFormer = { xExpr:helpIn :- helpIn;
                helpOut :- xExpr:helpOut;}.

new2xFormer = { Expr1:helpIn :- helpIn;
                Expr2:helpIn :- Expr1:helpOut;
                helpOut :- Expr2:helpOut;}.

new3xFormer = { xExpr:helpIn :- helpIn;
                xIterator:helpIn :- xExpr:helpOut;
                helpOut :- xIterator:helpOut;}.

new4xFormer = { xExpr:helpIn :- helpIn;
                xExprList:helpIn :- xExpr:helpOut;
                helpOut :- xExprList:helpOut;}.

new5xFormer = { Expr1:helpIn :- helpIn;
                Expr2:helpIn :- Expr1:helpOut;
```

```
                Expr3:helpIn :- Expr2:helpOut;
                helpOut :- Expr3:helpOut;}.

xExprList     = { helpOut :- helpIn;}.

new1xExprList = { xExprList:helpIn :- helpIn;
                  xExpr:helpIn :- xExprList:helpOut;
                  helpOut :- xExpr:helpOut;}.

new2xExprList = { xExpr:helpIn :- helpIn;
                  helpOut :- xExpr:helpOut;}.

new3xExprList = { xExprList:helpIn :- helpIn;
                  helpOut :- xExprList:helpOut;}.

xInstantiate = { xExpr:helpIn :- helpIn;
                 xInstExport:helpIn :- xExpr:helpOut;
                 xInstImport:helpIn :- xInstExport:helpOut;
                 helpOut :- xInstImport:helpOut;}.

xInstExport     = { helpOut :- helpIn;}.

new1xInstExport = { xExprList:helpIn :- helpIn;
                    helpOut :- xExprList:helpOut;}.

new2xInstExport = { helpOut :- helpIn;}.

xInstImport     = { helpOut :- helpIn;}.

new1xInstImport = { xIdList:helpIn :- helpIn;
                    helpOut :- xIdList:helpOut;}.

new2xInstImport = { helpOut :- helpIn;}.

xQualId     = { helpOut :- helpIn;}.

new1xQualId = { Id1:helpIn :- helpIn;
                Id2:helpIn :- Id1:helpOut;
                helpOut :- Id2:helpOut;}.

new2xQualId = { xId:helpIn :- helpIn;
                helpOut :- xId:helpOut;}.

xLambda = { xParamList:helpIn :- helpIn;
            xProgBody:helpIn :- xParamList:helpOut;
            helpOut :- xProgBody:helpOut;}.
```

```
xQuantifier = { xQualifier:helpIn :- helpIn;
                xSimpleIts:helpIn :- xQualifier:helpOut;
                xExpr:helpIn :- xSimpleIts:helpOut;
                helpOut :- xExpr:helpOut;}.

xQualifier     = { helpOut :- helpIn;}.

new1xQualifier = { helpOut :- helpIn;}.

new2xQualifier = { helpOut :- helpIn;}.

xElIfExprs     = { helpOut :- helpIn;}.

new1xElIfExprs = { xElIfExprs:helpIn :- helpIn;
                   xElIfExpr:helpIn :- xElIfExprs:helpOut;
                   helpOut :- xElIfExpr:helpOut;}.

new2xElIfExprs = { helpOut :- helpIn;}.

xElIfExpr = { Expr1:helpIn :- helpIn;
              Expr2:helpIn :- Expr1:helpOut;
              helpOut :- Expr2:helpOut;}.

xElseExpr     = { helpOut :- helpIn;}.

new1xElseExpr = { xExpr:helpIn :- helpIn;
                  helpOut :- xExpr:helpOut;}.

new2xElseExpr = { helpOut :- helpIn;}.

xCaseExprs     = { helpOut :- helpIn;}.

new1xCaseExprs = { No1:helpIn :- helpIn;
                   No2:helpIn :- No1:helpOut;
                   helpOut :- No2:helpOut;}.

new2xCaseExprs = { xCaseExprs:helpIn :- helpIn;
                   helpOut :- xCaseExprs:helpOut;}.

new3xCaseExprs = { xExprList:helpIn :- helpIn;
                   xExpr:helpIn :- xExprList:helpOut;
                   helpOut :- xExpr:helpOut;}.

xBinOp      = { helpOut :- helpIn;}.

new1xBinOp  = { helpOut :- helpIn;}.
```

```
new2xBinOp  = { helpOut :- helpIn;}.

new3xBinOp  = { helpOut :- helpIn;}.

new4xBinOp  = { helpOut :- helpIn;}.

new5xBinOp  = { helpOut :- helpIn;}.

new6xBinOp  = { helpOut :- helpIn;}.

new7xBinOp  = { helpOut :- helpIn;}.

new8xBinOp  = { helpOut :- helpIn;}.

new9xBinOp  = { helpOut :- helpIn;}.

new10xBinOp = { helpOut :- helpIn;}.

new11xBinOp = { helpOut :- helpIn;}.

new12xBinOp = { helpOut :- helpIn;}.

new13xBinOp = { helpOut :- helpIn;}.

new14xBinOp = { helpOut :- helpIn;}.

new15xBinOp = { helpOut :- helpIn;}.

new16xBinOp = { helpOut :- helpIn;}.

new17xBinOp = { helpOut :- helpIn;}.

new18xBinOp = { helpOut :- helpIn;}.

new19xBinOp = { helpOut :- helpIn;}.

new20xBinOp = { helpOut :- helpIn;}.

new21xBinOp = { helpOut :- helpIn;}.

new22xBinOp = { helpOut :- helpIn;}.

new23xBinOp = { helpOut :- helpIn;}.

new24xBinOp = { helpOut :- helpIn;}.

new25xBinOp = { helpOut :- helpIn;}.
```

```
new26xBinOp = { xQualId:helpIn :- helpIn;
                helpOut :- xQualId:helpOut;}.

new27xBinOp = { helpOut :- helpIn;}.

new28xBinOp = { helpOut :- helpIn;}.

new29xBinOp = { helpOut :- helpIn;}.

new30xBinOp = { xQualId:helpIn :- helpIn;
                helpOut :- xQualId:helpOut;}.

new31xBinOp = { helpOut :- helpIn;}.

new32xBinOp = { helpOut :- helpIn;}.

new33xBinOp = { helpOut :- helpIn;}.

new34xBinOp = { helpOut :- helpIn;}.

new35xBinOp = { helpOut :- helpIn;}.

new36xBinOp = { helpOut :- helpIn;}.

new37xBinOp = { helpOut :- helpIn;}.

new38xBinOp = { helpOut :- helpIn;}.

new39xBinOp = { helpOut :- helpIn;}.

new40xBinOp = { helpOut :- helpIn;}.

new41xBinOp = { helpOut :- helpIn;}.

new42xBinOp = { helpOut :- helpIn;}.

new43xBinOp = { helpOut :- helpIn;}.

new44xBinOp = { helpOut :- helpIn;}.

new45xBinOp = { helpOut :- helpIn;}.

new46xBinOp = { helpOut :- helpIn;}.

new47xBinOp = { helpOut :- helpIn;}.
```

```
xUnOp      = { helpOut :- helpIn;}.

new1xUnOp  = { helpOut :- helpIn;}.

new2xUnOp  = { helpOut :- helpIn;}.

new3xUnOp  = { helpOut :- helpIn;}.

new4xUnOp  = { helpOut :- helpIn;}.

new5xUnOp  = { helpOut :- helpIn;}.

new6xUnOp  = { helpOut :- helpIn;}.

new7xUnOp  = { helpOut :- helpIn;}.

new8xUnOp  = { helpOut :- helpIn;}.

new9xUnOp  = { helpOut :- helpIn;}.

new10xUnOp = { helpOut :- helpIn;}.

new11xUnOp = { helpOut :- helpIn;}.

new12xUnOp = { helpOut :- helpIn;}.

new13xUnOp = { helpOut :- helpIn;}.

new14xUnOp = { xQualId:helpIn :- helpIn;
               helpOut :- xQualId:helpOut;}.

new15xUnOp = { helpOut :- helpIn;}.

new16xUnOp = { helpOut :- helpIn;}.

new17xUnOp = { helpOut :- helpIn;}.

new18xUnOp = { helpOut :- helpIn;}.

new19xUnOp = { helpOut :- helpIn;}.

new20xUnOp = { helpOut :- helpIn;}.

new21xUnOp = { helpOut :- helpIn;}.

new22xUnOp = { helpOut :- helpIn;}.
```

```
new23xUnOp = { helpOut :- helpIn;}.

new24xUnOp = { helpOut :- helpIn;}.

new25xUnOp = { helpOut :- helpIn;}.

new26xUnOp = { helpOut :- helpIn;}.

new27xUnOp = { helpOut :- helpIn;}.

new28xUnOp = { helpOut :- helpIn;}.

new29xUnOp = { xQualId:helpIn :- helpIn;
               helpOut :- xQualId:helpOut;}.

new30xUnOp = { helpOut :- helpIn;}.

new31xUnOp = { helpOut :- helpIn;}.

new32xUnOp = { helpOut :- helpIn;}.

new33xUnOp = { helpOut :- helpIn;}.

new34xUnOp = { helpOut :- helpIn;}.

new35xUnOp = { helpOut :- helpIn;}.

new36xUnOp = { helpOut :- helpIn;}.

new37xUnOp = { helpOut :- helpIn;}.

xTypeList = { helpOut :- helpIn;}.

new1xTypeList = { helpOut :- helpIn;}.

new2xTypeList = { Id1:helpIn :- helpIn;
                  Id2:helpIn :- Id1:helpOut;
                  helpOut :- Id2:helpOut;}.

new3xTypeList = { xTypeList:helpIn :- helpIn;
                  Id1:helpIn :- xTypeList:helpOut;
                  Id2:helpIn :- Id1:helpOut;
                  helpOut :- Id2:helpOut;}.

intlit = { helpOut :- helpIn;}.

floatlit = { helpOut :- helpIn;}.
```

```
str = { helpOut :- helpIn;}.
```

C.42.5 Die Fortsetzung des Attributauswerters Transhelp2

⟨*transhelp2*⟩≡
```
xStmts      = { help2Out :- help2In;}.

new3xStmts  = { xStmts:help2In :- help2In;
                help2Out :- xStmts:help2In;}.

xLValue     = { help2Out :- help2In;}.

new2xLValue = { xLValue:help2In :- help2In;
                help2Out :- xLValue:help2Out;}.

new3xLValue = { No1:help2In :- help2In;
                No2:help2In :- No1:help2Out;
                help2Out :- No2:help2Out;}.

new4xLValue = { help2Out :- help2In;}.

new6xLValue = { xLValue:help2In :- help2In;
                xSelector:help2In :- xLValue:help2Out;
                help2Out :- xSelector:help2Out;}.

xProgDefn = { xId1:help2In :- help2In;
              xProgBody:help2In :- xId1:help2Out;
              xId2:help2In :- xProgBody:help2Out;
              help2Out :- xId2:help2Out;}.

xId = { id:help2In :- help2In;
        help2Out :- id:help2Out;}.

xProgBody = { xDecls:help2In :- help2In;
              xStmts:help2In :- xDecls:help2Out;
              xPHMDefn:help2In :- xStmts:help2Out;
              help2Out :- xPHMDefn:help2Out;}.

xPHMDefn     = { help2Out :- help2In;}.

new1xPHMDefn = { xId1:help2In :- help2In;
                 xParamList:help2In :- xId1:help2Out;
                 xProgBody:help2In :- xParamList:help2Out;
                 xId2:help2In :- xProgBody:help2Out;
                 help2Out :- xId2:help2Out;}.

new2xPHMDefn = { xId1:help2In :- help2In;
                 xModImport:help2In :- xId1:help2Out;
                 xModExport:help2In :- xModImport:help2Out ;
```

```
                 xProgBody:help2In :- xModExport:help2Out;
                 xId2:help2In :- xProgBody:help2Out;
                 help2Out :- xId2:help2Out;}.

new3xPHMDefn = { xId1:help2In :- help2In;
                 xRdParamList:help2In :- xId1:help2Out;
                 xImplAsso:help2In :- xRdParamList:help2Out;
                 xProgBody:help2In :- xImplAsso:help2Out;
                 xId2:help2In :- xProgBody:help2Out;
                 help2Out :- xId2:help2Out;}.

new4xPHMDefn = { No1:help2In :- help2In;
                 No2:help2In :- No1:help2Out;
                 help2Out :- No2:help2Out;}.
new5xPHMDefn = { help2Out :- help2In;}.

xParamList     = { help2Out :- help2In;}.

new1xParamList = { help2Out :- help2In;}.

new2xParamList = { xParamList:help2In :- help2In;
                   help2Out :- xParamList:help2Out;}.

new3xParamList = { xParamList:help2In :- help2In;
                   xParamMode:help2In :- xParamList:help2Out;
                   xId:help2In :- xParamMode:help2Out;
                   help2Out :- xId:help2Out;}.

new4xParamList = { xParamMode:help2In :- help2In;
                   xId:help2In :- xParamMode:help2Out;
                   help2Out :- xId:help2Out;}.

xParamMode     = { help2Out :- help2In;}.

new1xParamMode = { help2Out :- help2In;}.

new2xParamMode = { help2Out :- help2In;}.

new3xParamMode = { help2Out :- help2In;}.

new4xParamMode = { help2Out :- help2In;}.

xModImport     = { help2Out :- help2In;}.

new1xModImport = { xIdList:help2In :- help2In;
                   help2Out :- xIdList:help2Out;}.
```

```
new2xModImport = { help2Out :- help2In;}.

xModExport     = { help2Out :- help2In;}.

new1xModExport = { xIdList:help2In :- help2In;
                   help2Out :- xIdList:help2Out;}.

new2xModExport = { help2Out :- help2In;}.

xIdList     = { help2Out :- help2In;}.

new1xIdList = { xIdList:help2In :- help2In;
                xId:help2In :- xIdList:help2Out;
                help2Out :- xId:help2Out;}.

new2xIdList = { xId:help2In :- help2In;
                help2Out :- xId:help2Out;}.

xRdParamList     = { help2Out :- help2In;}.

new1xRdParamList = { help2Out :- help2In;}.

new2xRdParamList = { xIdList:help2In :- help2In;
                     help2Out :- xIdList:help2Out;}.

xImplAsso     = { help2Out :- help2In;}.

new1xImplAsso = { xIdList:help2In :- help2In;
                  help2Out :- xIdList:help2Out;}.

new2xImplAsso = { help2Out :- help2In;}.

new3xImplAsso = { help2Out :- help2In;}.

xDecls     = { help2Out :- help2In;}.

new1xDecls = { xDecls:help2In :- help2In;
               xDecl:help2In :- xDecls:help2Out;
               help2Out :- xDecl:help2Out;}.

new2xDecls = { help2Out :- help2In;}.

xDecl     = { help2Out :- help2In;}.

new1xDecl = { xDeclKey:help2In :- help2In;
              xSingleVar:help2In :- xDeclKey:help2Out;
              xExplAsso:help2In :- xSingleVar:help2Out;
```

```
              help2Out :- xExplAsso:help2Out;}.

new2xDecl = { xPersDecl:help2In :- help2In;
              xIdList:help2In :- xPersDecl:help2Out;
              xExpr:help2In :- xIdList:help2Out;
              xExplAsso:help2In :- xExpr:help2Out;
              help2Out :- xExplAsso:help2Out;}.

xSingleVar     = { help2Out :- help2In;}.

new1xSingleVar = { xId:help2In :- help2In;
                   xExpr:help2In :- xId:help2Out;
                   help2Out :- xExpr:help2Out; } .

new2xSingleVar = { xId:help2In :- help2In;
                   help2Out :- xId:help2Out;}.

new3xSingleVar = { No1:help2In :- help2In;
                   No2:help2In :- No1:help2Out;
                   help2Out :- No2:help2Out;}.

xDeclKey     = { help2Out :- help2In;}.

new1xDeclKey = { help2Out :- help2In;}.

new2xDeclKey = { help2Out :- help2In;}.

new3xDeclKey = { help2Out :- help2In;}.

new4xDeclKey = { help2Out :- help2In;}.

new5xDeclKey = { help2Out :- help2In;}.

xPersDecl     = { help2Out :- help2In;}.

new1xPersDecl = { help2Out :- help2In;}.

new2xPersDecl = { help2Out :- help2In;}.

new3xPersDecl = { help2Out :- help2In;}.

new4xPersDecl = { help2Out :- help2In;}.

new5xPersDecl = { help2Out :- help2In;}.

new6xPersDecl = { help2Out :- help2In;}.
```

```
xExplAsso = { help2Out :- help2In;}.

new1xExplAsso = { xExplAsso:help2In :- help2In;
                  xHandAsso:help2In :- xExplAsso:help2Out;
                  help2Out :- xHandAsso:help2Out;}.

new2xExplAsso = { help2Out :- help2In;}.

xHandAsso = { help2Out :- help2In;}.

new1xHandAsso = { xIdList:help2In :- help2In;
                  xId:help2In :- xIdList:help2Out;
                  help2Out :- xId:help2Out;}.

new2xHandAsso = { xId:help2In :- help2In;
                  help2Out :- xId:help2Out;}.

xStmt      = { help2Out :- help2In;}.

new1xStmt  = { help2Out :- help2In;}.

new2xStmt  = { help2Out :- help2In;}.

new3xStmt  = { xExpr:help2In :- help2In;
               help2Out :- xExpr:help2Out;}.

new4xStmt  = { xExpr:help2In :- help2In;
               help2Out :- xExpr:help2Out;}.

new5xStmt  = { help2Out :- help2In;}.

new6xStmt  = { xExpr:help2In :- help2In;
               help2Out :- xExpr:help2Out;}.

new7xStmt  = { help2Out :- help2In;}.

new8xStmt  = { xExpr:help2In :- help2In;
               help2Out :- xExpr:help2Out;}.

new9xStmt  = { help2Out :- help2In;}.

new10xStmt = { xStmtSignal:help2In :- help2In;
               help2Out :- xStmtSignal:help2Out;}.

new11xStmt = { xStmtNotify:help2In :- help2In;
               help2Out :- xStmtNotify:help2Out;}.
```

```
new12xStmt = { xId:help2In :- help2In;
               xActuList:help2In :- xId:help2Out;
               help2Out :- xActuList:help2Out;}.

new13xStmt = { xStdIO:help2In :- help2In;
               xActuList:help2In :- xStdIO:help2Out;
               help2Out :- xActuList:help2Out;}.

new14xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new15xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new16xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new17xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new18xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new19xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new20xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new21xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new22xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new23xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.
```

```
new24xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new25xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new26xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new27xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new28xStmt = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new29xStmt = { xLValue:help2In :- help2In;
               xBinOp:help2In :- xLValue:help2Out;
               xExpr:help2In :- xBinOp:help2Out;
               help2Out :- xExpr:help2Out;}.

new32xStmt = { No1:help2In :- help2In;
               xFrom:help2In :- No1:help2Out;
               No2:help2In :- xFrom:help2Out;
               help2Out :- No2:help2Out;}.

new33xStmt = { xLValue:help2In :- help2In;
               xActuList:help2In :- xLValue:help2Out;
               help2Out :- xActuList:help2Out;}.

new34xStmt = { xLambda:help2In :- help2In;
               xActuList:help2In :- xLambda:help2Out;
               help2Out :- xActuList:help2Out;}.

new35xStmt = { xActuList:help2In :- help2In;
               help2Out :- xActuList:help2Out;}.

new36xStmt = { xExpr:help2In :- help2In;
               xStmts:help2In :- xExpr:help2Out;
               xElIfStmts:help2In :- xStmts:help2Out;
               xElseStmts:help2In :- xElIfStmts:help2Out;
               help2Out :- xElseStmts:help2Out;}.
```

```
new37xStmt = { xExpr:help2In :- help2In;
               xCaseStmts:help2In :- xExpr:help2Out;
               xElseStmts:help2In :- xCaseStmts:help2Out;
               help2Out :- xElseStmts:help2Out;}.

new38xStmt = { xLoops:help2In :- help2In;
               help2Out :- xLoops:help2Out;}.

new39xStmt = { xLoops:help2In :- help2In;
               help2Out :- xLoops:help2Out;}.

new40xStmt = { xLoops:help2In :- help2In;
               help2Out :- xLoops:help2Out;}.

new41xStmt = { xLoops:help2In :- help2In;
               help2Out :- xLoops:help2Out;}.

new42xStmt = { xLoops:help2In :- help2In;
               help2Out :- xLoops:help2Out;}.

new43xStmt = { xLabel:help2In :- help2In;
               xLoops:help2In :- xLabel:help2Out;
               xId:help2In :- xLoops:help2Out;
               help2Out :- xId:help2Out;}.

new44xStmt = { help2Out :- help2In;}.

new45xStmt = { xId:help2In :- help2In;
               help2Out :- xId:help2Out;}.

new46xStmt = { help2Out :- help2In;}.

new47xStmt = { xId:help2In :- help2In;
               help2Out :- xId:help2Out;}.

new48xStmt = { xExpr:help2In :- help2In;
               help2Out :- xExpr:help2Out;}.

new49xStmt = { Expr1:help2In :- help2In;
               Expr2:help2In :- Expr1:help2Out;
               help2Out :- Expr2:help2Out;}.

new50xStmt = { Expr1:help2In :- help2In;
               Expr2:help2In :- Expr1:help2Out;
               help2Out :- Expr2:help2Out;}.
```

```
new51xStmt = { xTemplate:help2In :- help2In;
               xExpr:help2In :- xTemplate:help2Out;
               xElseStmts:help2In :- xExpr:help2Out;
               help2Out :- xElseStmts:help2Out;}.

new52xStmt = { xTemplate:help2In :- help2In;
               xExpr:help2In :- xTemplate:help2Out;
               xElseStmts:help2In :- xExpr:help2Out;
               help2Out :- xElseStmts:help2Out;}.

new53xStmt = { xId:help2In :- help2In;
               xTypeList:help2In :- xId:help2Out;
               help2Out :- xTypeList:help2Out;}.

new54xStmt = { xExpr:help2In :- help2In;
               xExprList:help2In :- xExpr:help2Out;
               help2Out :- xExprList:help2Out;}.

new55xStmt = { xExpr:help2In :- help2In;
               xExprList:help2In :- xExpr:help2Out;
               help2Out :- xExprList:help2Out;}.

new56xStmt = { xExpr1:help2In :- help2In;
               xExpr2:help2In :- xExpr1:help2Out;
               xExprList:help2In :- xExpr2:help2Out;
               help2Out :- xExprList:help2Out;}.

xStmtSignal     = { help2Out :- help2In;}.

new1xStmtSignal = { xLValue:help2In :- help2In;
                    xId:help2In :- xLValue:help2Out;
                    xActuList:help2In :- xId:help2Out;
                    help2Out :- xActuList:help2Out;}.

new2xStmtSignal = { xLValue:help2In :- help2In;
                    xActuList:help2In :- xLValue:help2Out;
                    help2Out :- xActuList:help2Out;}.

xStmtNotify     = { help2Out :- help2In;}.

new1xStmtNotify = { xLValue:help2In :- help2In;
                    xId:help2In :- xLValue:help2Out;
                    xActuList:help2In :- xId:help2Out;
                    help2Out :- xActuList:help2Out;}.

new2xStmtNotify = { xLValue:help2In :- help2In;
                    xActuList:help2In :- xLValue:help2Out;
```

```
                    help2Out :- xActuList:help2Out;}.

xStdIO      = { help2Out :- help2In;}.

new1xStdIO  = { help2Out :- help2In;}.

new2xStdIO  = { help2Out :- help2In;}.

new3xStdIO  = { help2Out :- help2In;}.

new4xStdIO  = { help2Out :- help2In;}.

new5xStdIO  = { help2Out :- help2In;}.

new6xStdIO  = { help2Out :- help2In;}.

new7xStdIO  = { help2Out :- help2In;}.

/* new8xStdIO  = { help2Out :- help2In;}. */
/* new9xStdIO  = { help2Out :- help2In;}. */
/* new10xStdIO = { help2Out :- help2In;}. */

new11xStdIO = { help2Out :- help2In;}.

new12xStdIO = { help2Out :- help2In;}.

xExpr      = { help2Out :- help2In;}.

new1xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new2xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new3xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new4xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new5xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.
```

```
new6xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new7xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new8xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new9xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new10xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new11xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new12xExpr  = { xLValue:help2In :- help2In;
               xExpr:help2In :- xLValue:help2Out;
               help2Out :- xExpr:help2Out;}.

new13xExpr = { intlit:help2In :- help2In;
               help2Out :- intlit:help2Out;}.

new14xExpr = { floatlit:help2In :- help2In;
               help2Out :- floatlit:help2Out;}.

new15xExpr = { help2Out :- help2In;}.

new16xExpr = { help2Out :- help2In;}.

new17xExpr = { help2Out :- help2In;}.

new18xExpr = { help2Out :- help2In;}.

new19xExpr = { help2Out :- help2In;}.

new20xExpr = { help2Out :- help2In;}.
```

```
new21xExpr = { help2Out :- help2In;}.

new22xExpr = { help2Out :- help2In;}.

new23xExpr = { help2Out :- help2In;}.

new24xExpr = { help2Out :- help2In;}.

new25xExpr = { help2Out :- help2In;}.

new26xExpr = { help2Out :- help2In;}.

new27xExpr = { help2Out :- help2In;}.

new28xExpr = { help2Out :- help2In;}.

new29xExpr = { help2Out :- help2In;}.

new30xExpr = { help2Out :- help2In;}.

new31xExpr = { xInstantiate:help2In :- help2In;
               help2Out :- xInstantiate:help2Out;}.

new32xExpr = { help2Out :- help2In;}.

new33xExpr = { str:help2In :- help2In;
               help2Out :- str:help2Out;}.

new34xExpr = { help2Out :- help2In;}.

new35xExpr = { help2Out :- help2In;}.

new36xExpr = { xFormer:help2In :- help2In;
               help2Out :- xFormer:help2Out;}.

new37xExpr = { xFormer:help2In :- help2In;
               help2Out :- xFormer:help2Out;}.

new38xExpr = { xQualId:help2In :- help2In;
               help2Out :- xQualId:help2Out;}.

new39xExpr = { xLambda:help2In :- help2In;
              xActuList:help2In :- xLambda:help2Out;
              help2Out :- xActuList:help2Out;}.

new40xExpr = { xActuList:help2In :- help2In;
               help2Out :- xActuList:help2Out;}.
```

```
new41xExpr = { help2Out :- help2In;}.

new42xExpr = { xLambda:help2In :- help2In;
               help2Out :- xLambda:help2Out;}.

new43xExpr = { xQualId:help2In :- help2In;
               help2Out :- xQualId:help2Out;}.

new44xExpr = { xExpr:help2In :- help2In;
               xSelector:help2In :- xExpr:help2Out;}.

new45xExpr = { xExpr:help2In :- help2In;
               help2Out :- xExpr:help2Out;}.

new46xExpr = { xExpr:help2In :- help2In;
               xHandAsso:help2In :- xExpr:help2Out;
               help2Out :- xHandAsso:help2Out;}.

new47xExpr = { Expr1:help2In :- help2In;
               Expr2:help2In :- Expr1:help2Out;
               xElIfExprs:help2In :- Expr2:help2Out;
               xElseExpr:help2In :- xElIfExprs:help2Out;
               help2Out :- xElseExpr:help2Out;}.

new48xExpr = { xExpr:help2In :- help2In;
               xCaseExprs:help2In :- xExpr:help2Out;
               xElseExpr:help2In :- xCaseExprs:help2Out;
               help2Out :- xElseExpr:help2Out;}.

new49xExpr = { xExpr:help2In :- help2In;
               help2Out :- xExpr:help2Out;}.

new50xExpr = { Expr1:help2In :- help2In;
               xBinOp:help2In :- Expr1:help2Out;
               Expr2:help2In :- xBinOp:help2Out;
               help2Out :- Expr2:help2Out;}.

new51xExpr = { xQuantifier:help2In :- help2In;
               help2Out :- xQuantifier:help2Out;}.

new52xExpr = { xUnOp:help2In :- help2In;
               xExpr:help2In :- xUnOp:help2Out;
               help2Out :- xExpr:help2Out;}.

new53xExpr = { Id1:help2In :- help2In;
               xTypeList:help2In :- Id1:help2Out;
```

```
               Id2:help2In :- xTypeList:help2Out;
               help2Out :- Id2:help2Out;}.

xFrom     = { help2Out :- help2In;}.

new1xFrom = { help2Out :- help2In;}.

new2xFrom = { help2Out :- help2In;}.

new3xFrom = { help2Out :- help2In;}.

xActuList     = { help2Out :- help2In;}.

new1xActuList = { xExprList:help2In :- help2In;
                  help2Out :- xExprList:help2Out;}.

new2xActuList = { help2Out :- help2In;}.

xElIfStmt = { xExpr:help2In :- help2In;
              xStmts:help2In :- xExpr:help2Out;
              help2Out :- xStmts:help2Out;}.

xElIfStmts     = { help2Out :- help2In;}.

new1xElIfStmts = { xElIfStmts:help2In :- help2In;
                   xElIfStmt:help2In :- xElIfStmts:help2Out;
                   help2Out :- xElIfStmt:help2Out;}.

new2xElIfStmts = { help2Out :- help2In;}.

xElseStmts     = { help2Out :- help2In;}.

new1xElseStmts = { xStmts:help2In :- help2In;
                   help2Out :- xStmts:help2Out;}.

new2xElseStmts = { help2Out :- help2In;}.

xCaseStmts     = { help2Out :- help2In;}.

new1xCaseStmts = { No1:help2In :- help2In;
                   No2:help2In :- No1:help2Out;
                   help2Out :- No2:help2Out;}.

new2xCaseStmts = { xCaseStmts:help2In :- help2In;
                   help2Out :- xCaseStmts:help2Out;}.

new3xCaseStmts = { xExprList:help2In :- help2In;
```

```
                  xStmts:help2In :- xExprList:help2Out;
                  help2Out :- xStmts:help2Out;}.

xLabel = { xId:help2In :- help2In;
           help2Out :- xId:help2Out;}.

xTemplate     = { help2Out :- help2In;}.

new1xTemplate = { xExprFormal:help2In :- help2In;
                  xTempCond:help2In :- xExprFormal:help2Out;
                  xStmts:help2In :- xTempCond:help2Out;
                  help2Out :- xStmts:help2Out;}.

new2xTemplate = { xExprFormal:help2In :- help2In;
                  xTempCond:help2In :- xExprFormal:help2Out;
                  help2Out :- xTempCond:help2Out;}.

new3xTemplate = { No1:help2In :- help2In;
                  No2:help2In :- No1:help2Out;
                  help2Out :- No2:help2Out;}.


xTempCond     = { help2Out :- help2In;}.

new1xTempCond = { xExpr:help2In :- help2In;
                  help2Out :- xExpr:help2Out;}.

new2xTempCond = { help2Out :- help2In;}.

xExprFormal     = { help2Out :- help2In;}.

new1xExprFormal = { xExpr:help2In :- help2In;
                    help2Out :- xExpr:help2Out;}.

new2xExprFormal = { xFormal:help2In :- help2In;
                    xInto:help2In :- xFormal:help2Out;
                    help2Out :- xInto:help2Out;}.

new3xExprFormal = { help2Out :- help2In;}.

new4xExprFormal = { No1:help2In :- help2In;
                    No2:help2In :- No1:help2Out;
                    help2Out :- No2:help2Out;}.

xInto     = { help2Out :- help2In;}.

new1xInto = { xExpr:help2In :- help2In;
```

```
              help2Out :- xExpr:help2Out;}.

new2xInto = { help2Out :- help2In;}.

xFormal     = { help2Out :- help2In;}.

new1xFormal = { xLValue:help2In :- help2In;
                help2Out :- xLValue:help2Out;}.

new2xFormal = { help2Out :- help2In;}.

xIterator     = { help2Out :- help2In;}.

new1xIterator = { xSimpleIts:help2In :- help2In;
                  xExpr:help2In :- xSimpleIts:help2Out;
                  help2Out :- xExpr:help2Out;}.

new2xIterator = { xSimpleIts:help2In :- help2In;
                  help2Out :- xSimpleIts:help2Out;}.

xSimpleIts     = { help2Out :- help2In;}.

new1xSimpleIts = { xSimpleIts:help2In :- help2In;
                   xSimpleIt:help2In :- xSimpleIts:help2Out;
                   help2Out :- xSimpleIt:help2Out;}.

new2xSimpleIts = { xSimpleIt:help2In :- help2In;
                   help2Out :- xSimpleIt:help2Out;}.

xSimpleIt = { help2Out :- help2In;}.

new2xSimpleIt = { xLValue:help2In :- help2In;
                  xId:help2In :- xLValue:help2Out;
                  xMapSel:help2In :- xId:help2Out;
                  help2Out :- xMapSel:help2Out;}.

xMapSel     = { help2Out :- help2In;}.

new1xMapSel = { xLValue:help2In :- help2In;
                help2Out :- xLValue:help2Out;}.

new2xMapSel = { xLValue:help2In :- help2In;
                help2Out :- xLValue:help2Out;}.

xSelector     = { help2Out :- help2In;}.

new1xSelector = { xExprList:help2In :- help2In;
```

```
                  help2Out :- xExprList:help2Out;}.

new2xSelector = { xExprList:help2In :- help2In;
                  help2Out :- xExprList:help2Out;}.

new3xSelector = { xExpr:help2In :- help2In;
                  help2Out :-  xExpr:help2Out;}.

new4xSelector = { Expr1:help2In :- help2In;
                  Expr2:help2In :- Expr1:help2Out;
                  help2Out :- Expr2:help2Out;}.

xFormer     = { help2Out :- help2In;}.

new1xFormer = { xExpr:help2In :- help2In;
                help2Out :- xExpr:help2Out;}.

new2xFormer = { Expr1:help2In :- help2In;
                Expr2:help2In :- Expr1:help2Out;
                help2Out :- Expr2:help2Out;}.

new3xFormer = { xExpr:help2In :- help2In;
                xIterator:help2In :- xExpr:help2Out;
                help2Out :- xIterator:help2Out;}.

new4xFormer = { xExpr:help2In :- help2In;
                xExprList:help2In :- xExpr:help2Out;
                help2Out :- xExprList:help2Out;}.

new5xFormer = { Expr1:help2In :- help2In;
                Expr2:help2In :- Expr1:help2Out;
                Expr3:help2In :- Expr2:help2Out;
                help2Out :- Expr3:help2Out;}.

xExprList     = { help2Out :- help2In;}.

new1xExprList = { xExprList:help2In :- help2In;
                  xExpr:help2In :- xExprList:help2Out;
                  help2Out :- xExpr:help2Out;}.

new2xExprList = { xExpr:help2In :- help2In;
                  help2Out :- xExpr:help2Out;}.

new3xExprList = { xExprList:help2In :- help2In;
                  help2Out :- xExprList:help2Out;}.

xInstantiate = { xExpr:help2In :- help2In;
```

```
                  xInstExport:help2In :- xExpr:help2Out;
                  xInstImport:help2In :- xInstExport:help2Out;
                  help2Out :- xInstImport:help2Out;}.

xInstExport     = { help2Out :- help2In;}.

new1xInstExport = { xExprList:help2In :- help2In;
                    help2Out :- xExprList:help2Out;}.

new2xInstExport = { help2Out :- help2In;}.

xInstImport     = { help2Out :- help2In;}.

new1xInstImport = { xIdList:help2In :- help2In;
                    help2Out :- xIdList:help2Out;}.

new2xInstImport = { help2Out :- help2In;}.

xQualId     = { help2Out :- help2In;}.

new1xQualId = { Id1:help2In :- help2In;
                Id2:help2In :- Id1:help2Out;
                help2Out :- Id2:help2Out;}.

new2xQualId = { xId:help2In :- help2In;
                help2Out :- xId:help2Out;}.

xLambda = { xParamList:help2In :- help2In;
            xProgBody:help2In :- xParamList:help2Out;
            help2Out :- xProgBody:help2Out;}.

xQuantifier = { xQualifier:help2In :- help2In;
                xSimpleIts:help2In :- xQualifier:help2Out;
                xExpr:help2In :- xSimpleIts:help2Out;
                help2Out :- xExpr:help2Out;}.

xQualifier     = { help2Out :- help2In;}.

new1xQualifier = { help2Out :- help2In;}.

new2xQualifier = { help2Out :- help2In;}.

xElIfExprs     = { help2Out :- help2In;}.

new1xElIfExprs = { xElIfExprs:help2In :- help2In;
                   xElIfExpr:help2In :- xElIfExprs:help2Out;
                   help2Out :- xElIfExpr:help2Out;}.
```

```
new2xElIfExprs = { help2Out :- help2In;}.

xElIfExpr = { Expr1:help2In :- help2In;
              Expr2:help2In :- Expr1:help2Out;
              help2Out :- Expr2:help2Out;}.

xElseExpr     = { help2Out :- help2In;}.

new1xElseExpr = { xExpr:help2In :- help2In;
                  help2Out :- xExpr:help2Out;}.

new2xElseExpr = { help2Out :- help2In;}.

xCaseExprs     = { help2Out :- help2In;}.

new1xCaseExprs = { No1:help2In :- help2In;
                   No2:help2In :- No1:help2Out;
                   help2Out :- No2:help2Out;}.

new2xCaseExprs = { xCaseExprs:help2In :- help2In;
                   help2Out :- xCaseExprs:help2Out;}.

new3xCaseExprs = { xExprList:help2In :- help2In;
                   xExpr:help2In :- xExprList:help2Out;
                   help2Out :- xExpr:help2Out;}.

xBinOp      = { help2Out :- help2In;}.

new1xBinOp  = { help2Out :- help2In;}.

new2xBinOp  = { help2Out :- help2In;}.

new3xBinOp  = { help2Out :- help2In;}.

new4xBinOp  = { help2Out :- help2In;}.

new5xBinOp  = { help2Out :- help2In;}.

new6xBinOp  = { help2Out :- help2In;}.

new7xBinOp  = { help2Out :- help2In;}.

new8xBinOp  = { help2Out :- help2In;}.

new9xBinOp  = { help2Out :- help2In;}.
```

```
new10xBinOp = { help2Out :- help2In;}.

new11xBinOp = { help2Out :- help2In;}.

new12xBinOp = { help2Out :- help2In;}.

new13xBinOp = { help2Out :- help2In;}.

new14xBinOp = { help2Out :- help2In;}.

new15xBinOp = { help2Out :- help2In;}.

new16xBinOp = { help2Out :- help2In;}.

new17xBinOp = { help2Out :- help2In;}.

new18xBinOp = { help2Out :- help2In;}.

new19xBinOp = { help2Out :- help2In;}.

new20xBinOp = { help2Out :- help2In;}.

new21xBinOp = { help2Out :- help2In;}.

new22xBinOp = { help2Out :- help2In;}.

new23xBinOp = { help2Out :- help2In;}.

new24xBinOp = { help2Out :- help2In;}.

new25xBinOp = { help2Out :- help2In;}.

new26xBinOp = { xQualId:help2In :- help2In;
                help2Out :- xQualId:help2Out;}.

new27xBinOp = { help2Out :- help2In;}.

new28xBinOp = { help2Out :- help2In;}.

new29xBinOp = { help2Out :- help2In;}.

new30xBinOp = { xQualId:help2In :- help2In;
                help2Out :- xQualId:help2Out;}.

new31xBinOp = { help2Out :- help2In;}.

new32xBinOp = { help2Out :- help2In;}.
```

```
new33xBinOp = { help2Out :- help2In;}.

new34xBinOp = { help2Out :- help2In;}.

new35xBinOp = { help2Out :- help2In;}.

new36xBinOp = { help2Out :- help2In;}.

new37xBinOp = { help2Out :- help2In;}.

new38xBinOp = { help2Out :- help2In;}.

new39xBinOp = { help2Out :- help2In;}.

new40xBinOp = { help2Out :- help2In;}.

new41xBinOp = { help2Out :- help2In;}.

new42xBinOp = { help2Out :- help2In;}.

new43xBinOp = { help2Out :- help2In;}.

new44xBinOp = { help2Out :- help2In;}.

new45xBinOp = { help2Out :- help2In;}.

new46xBinOp = { help2Out :- help2In;}.

new47xBinOp = { help2Out :- help2In;}.

xUnOp      = { help2Out :- help2In;}.

new1xUnOp  = { help2Out :- help2In;}.

new2xUnOp  = { help2Out :- help2In;}.

new3xUnOp  = { help2Out :- help2In;}.

new4xUnOp  = { help2Out :- help2In;}.

new5xUnOp  = { help2Out :- help2In;}.

new6xUnOp  = { help2Out :- help2In;}.

new7xUnOp  = { help2Out :- help2In;}.
```

```
new8xUnOp  = { help2Out :- help2In;}.

new9xUnOp  = { help2Out :- help2In;}.

new10xUnOp = { help2Out :- help2In;}.

new11xUnOp = { help2Out :- help2In;}.

new12xUnOp = { help2Out :- help2In;}.

new13xUnOp = { help2Out :- help2In;}.

new14xUnOp = { xQualId:help2In :- help2In;
               help2Out :- xQualId:help2Out;}.

new15xUnOp = { help2Out :- help2In;}.

new16xUnOp = { help2Out :- help2In;}.

new17xUnOp = { help2Out :- help2In;}.

new18xUnOp = { help2Out :- help2In;}.

new19xUnOp = { help2Out :- help2In;}.

new20xUnOp = { help2Out :- help2In;}.

new21xUnOp = { help2Out :- help2In;}.

new22xUnOp = { help2Out :- help2In;}.

new23xUnOp = { help2Out :- help2In;}.

new24xUnOp = { help2Out :- help2In;}.

new25xUnOp = { help2Out :- help2In;}.

new26xUnOp = { help2Out :- help2In;}.

new27xUnOp = { help2Out :- help2In;}.

new28xUnOp = { help2Out :- help2In;}.

new29xUnOp = { xQualId:help2In :- help2In;
               help2Out :- xQualId:help2Out;}.

new30xUnOp = { help2Out :- help2In;}.
```

```
new31xUnOp = { help2Out :- help2In;}.

new32xUnOp = { help2Out :- help2In;}.

new33xUnOp = { help2Out :- help2In;}.

new34xUnOp = { help2Out :- help2In;}.

new35xUnOp = { help2Out :- help2In;}.

new36xUnOp = { help2Out :- help2In;}.

new37xUnOp = { help2Out :- help2In;}.

xTypeList = { help2Out :- help2In;}.

new1xTypeList = { help2Out :- help2In;}.

new2xTypeList = { Id1:help2In :- help2In;
                  Id2:help2In :- Id1:help2Out;
                  help2Out :- Id2:help2Out;}.

new3xTypeList = { xTypeList:help2In :- help2In;
                  Id1:help2In :- xTypeList:help2Out;
                  Id2:help2In :- Id1:help2Out;
                  help2Out :- Id2:help2Out;}.

intlit = { help2Out :- help2In;}.

floatlit = { help2Out :- help2In;}.

str = { help2Out :- help2In;}.
```

C.42.6 Die Fortsetzung des Attributauswerters Output

⟨*output*⟩≡

```
xProgDefn = { xId1:writeIn := putmykey("program", pos1, writeIn);
              xProgBody:writeIn := putmykey(";", pos2, xId1:writeOut);
              xId2:writeIn := putmykey("end", pos3, xProgBody:writeOut);
              writeOut := putmykey(";", pos4, xId2:writeOut);}.

xId = { id:writeIn :- writeIn;
        writeOut :- id:writeOut;}.

xProgBody = { xDecls:writeIn :- writeIn;
              xStmts:writeIn :- xDecls:writeOut;
              xPHMDefn:writeIn :- xStmts:writeOut;
              writeOut :- xPHMDefn:writeOut;}.

xPHMDefn     = { writeOut :- writeIn;}.

new1xPHMDefn = { xId1:writeIn := putmykey("procedure", pos1, writeIn);
                 xParamList:writeIn :- xId1:writeOut;
                 xProgBody:writeIn := putmykey(";", pos2,
                                        xParamList:writeOut);
                 xId2:writeIn := putmykey("end", pos3, xProgBody:writeOut);
                 writeOut := putmykey(";", pos4, xId2:writeOut);}.

new2xPHMDefn = { xId1:writeIn := putmykey("module", pos1, writeIn);
                 xModImport:writeIn :- xId1:writeOut;
                 xModExport:writeIn :- xModImport:writeOut ;
                 xProgBody:writeIn :- xModExport:writeOut;
                 xId2:writeIn := putmykey("end", pos2, xProgBody:writeOut);
                 writeOut := putmykey(";", pos3, xId2:writeOut);}.

new3xPHMDefn = { xId1:writeIn := putmykey("handler", pos1, writeIn);
                 xRdParamList:writeIn :- xId1:writeOut;
                 xImplAsso:writeIn :- xRdParamList:writeOut;
                 xProgBody:writeIn := putmykey(";", pos2,
                                        xImplAsso:writeOut);
                 xId2:writeIn := putmykey("end", pos3, xProgBody:writeOut);
                 writeOut := putmykey(";", pos4, xId2:writeOut);}.

new4xPHMDefn = { No1:writeIn :- writeIn;
                 No2:writeIn :- No1:writeOut;
                 writeOut :- No2:writeOut;}.

new5xPHMDefn = { writeOut :- writeIn;}.

xParamList     = { writeOut :- writeIn;}.
```

```
new1xParamList = { writeOut := { putmykey("(", pos1, writeIn);

                                 writeOut = putmykey(")", pos2, 1);};}.

new2xParamList = { xParamList:writeIn := putmykey("(", pos1, writeIn);
                   writeOut := putmykey(")", pos2, xParamList:writeOut);}.

new3xParamList = { xParamList:writeIn :- writeIn;
                   xParamMode:writeIn := putmykey(",", pos,
                                              xParamList:writeOut);
                   xId:writeIn :- xParamMode:writeOut;
                   writeOut :- xId:writeOut;}.

new4xParamList = { xParamMode:writeIn :- writeIn;
                   xId:writeIn :- xParamMode:writeOut;
                   writeOut :- xId:writeOut;}.

xParamMode     = { writeOut :- writeIn;}.

new1xParamMode = { writeOut :- writeIn;}.

new2xParamMode = { writeOut := putmykey("rd", pos, writeIn);}.

new3xParamMode = { writeOut := putmykey("rw", pos, writeIn);}.

new4xParamMode = { writeOut := putmykey("wr", pos, writeIn);}.

xModImport     = { writeOut :- writeIn;}.

new1xModImport = { xIdList:writeIn := putmykey("import", pos1, writeIn);
                   writeOut := putmykey(";", pos2, xIdList:writeOut);}.

new2xModImport = { writeOut :- writeIn;}.


xModExport     = { writeOut :- writeIn;}.

new1xModExport = { xIdList:writeIn := putmykey("export", pos1, writeIn);
                   writeOut := putmykey(";", pos2, xIdList:writeOut);}.

new2xModExport = { writeOut :- writeIn;}.

xIdList     = { writeOut :- writeIn;}.

new1xIdList = { xIdList:writeIn :- writeIn;
                xId:writeIn := putmykey(",", pos, xIdList:writeOut);
```

```
                writeOut :- xId:writeOut;}.

new2xIdList = { xId:writeIn :- writeIn;
                writeOut :- xId:writeOut;}.

xRdParamList     = { writeOut :- writeIn;}.

new1xRdParamList = { writeOut := { putmykey("(", pos1, writeIn);
                                   writeOut = putmykey(")", pos2, 1);};}.

new2xRdParamList = { xIdList:writeIn := putmykey("(", pos1, writeIn);
                     writeOut := putmykey(")", pos2, xIdList:writeOut);}.


xImplAsso     = { writeOut :- writeIn;}.

new1xImplAsso = { xIdList:writeIn := putmykey("for", pos, writeIn);
                  writeOut :- xIdList:writeOut;}.

new2xImplAsso = { writeOut := { putmykey("for", pos1, writeIn);
                                writeOut = putmykey("others", pos2, 1);};}.
new3xImplAsso = { writeOut :- writeIn;}.

xDecls     = { writeOut :- writeIn;}.

new1xDecls = { xDecls:writeIn :- writeIn;
               xDecl:writeIn :- xDecls:writeOut;
               writeOut :- xDecl:writeOut;}.

new2xDecls = { writeOut :- writeIn;}.

xDecl     = { writeOut :- writeIn;}.

new1xDecl = { xDeclKey:writeIn :- writeIn;
              xSingleVar:writeIn :- xDeclKey:writeOut;
              xExplAsso:writeIn :- xSingleVar:writeOut;
              writeOut := putmykey(";", pos, xExplAsso:writeOut);}.

new2xDecl = { xPersDecl:writeIn :- writeIn;
              xIdList:writeIn :- xPersDecl:writeOut;
              xExpr:writeIn := putmykey(":", pos1, xIdList:writeOut);
              xExplAsso:writeIn :- xExpr:writeOut;
              writeOut := putmykey(";", pos2, xExplAsso:writeOut);}.

xSingleVar     = { writeOut :- writeIn;}.

new1xSingleVar = { xId:writeIn :- writeIn;
```

```
                   xExpr:writeIn := putmykey(":=", pos, xId:writeOut);
                   writeOut :- xExpr:writeOut; } .

new2xSingleVar = { xId:writeIn :- writeIn;
                   writeOut :- xId:writeOut;}.

new3xSingleVar = { No1:writeIn :- writeIn;
                   No2:writeIn := putmykey(",",pos,No1:writeOut);
                   writeOut :- No2:writeOut;}.

xDeclKey     = { writeOut :- writeIn;}.

new1xDeclKey = { writeOut := putmykey("visible", pos, writeIn);}.

new2xDeclKey = { writeOut := putmykey("hidden", pos, writeIn);}.

new3xDeclKey = { writeOut := { putmykey("visible", pos1, writeIn);
                               writeOut = putmykey("constant", pos2, 1);};}.

new4xDeclKey = { writeOut := { putmykey("hidden", pos1, writeIn);
                               writeOut = putmykey("constant", pos2, 1);};}.

new5xDeclKey = { writeOut := putmykey("constant", pos, writeIn);}.

xPersDecl     = { writeOut :- writeIn;}.

new1xPersDecl = { writeOut := { putmykey("visible", pos1, writeIn);
                                writeOut = putmykey("persistent", pos2, 1);};}.

new2xPersDecl = { writeOut := { putmykey("hidden", pos1, writeIn);
                                writeOut = putmykey("persistent", pos2, 1);};}.

new3xPersDecl = { writeOut := putmykey("persistent", pos, writeIn);}.
new4xPersDecl = { writeOut := { putmykey("visible", pos1, writeIn);
                                putmykey("persistent", pos2, 1);
                                writeOut = putmykey("constant", pos3, 1);};}.

new5xPersDecl = { writeOut := { putmykey("hidden", pos1, writeIn);
                                putmykey("persistent", pos2, 1);
                                writeOut = putmykey("constant", pos3, 1);};}.

new6xPersDecl = { writeOut := { putmykey("persistent", pos1, writeIn);
                                writeOut = putmykey("constant", pos2, 1);};}.

xStmts      = { writeOut :- writeIn;}.

new3xStmts  = { xStmts:writeIn := putmykey("begin", pos, writeIn);
```

```
                  writeOut :- xStmts:writeIn;}.

xExplAsso = { writeOut :- writeIn;}.

new1xExplAsso = { xExplAsso:writeIn :- writeIn;
                  xHandAsso:writeIn := putmykey("when", pos,
                                                 xExplAsso:writeOut);
                  writeOut :- xHandAsso:writeOut;}.

new2xExplAsso = { writeOut :- writeIn;}.

xHandAsso = { writeOut :- writeIn;}.

new1xHandAsso = { xIdList:writeIn :- writeIn;
                  xId:writeIn := putmykey("use", pos, xIdList:writeOut);
                  writeOut :- xId:writeOut;}.

new2xHandAsso = { xId:writeIn := { putmykey("others", pos1, writeIn);
                                   xId:writeIn = putmykey("use", pos2,
                                                          writeIn);};
                  writeOut :- xId:writeOut;}.

xStmt      = { writeOut :- writeIn;}.

new1xStmt  = { writeOut := putmykey("pass", pos, writeIn);}.

new2xStmt  = { writeOut := putmykey("stop", pos, writeIn);}.

new3xStmt  = { xExpr:writeIn := putmykey("stop", pos, writeIn);
               writeOut :- xExpr:writeOut;}.

new4xStmt  = { xExpr:writeIn := putmykey("return", pos, writeIn);
               writeOut :- xExpr:writeOut;}.

new5xStmt  = { writeOut := putmykey("return", pos, writeIn);}.

new6xStmt  = { xExpr:writeIn := { putmykey("return", pos1, writeIn);
                                  xExpr:writeIn = putmykey("commit", pos2, 1);};

               writeOut :- xExpr:writeOut;}.

new7xStmt  = { writeOut := { putmykey("return", pos1, writeIn);
                             writeOut = putmykey("commit", pos2, 1);};}.

new8xStmt  = { xExpr:writeIn := putmykey("resume", pos, writeIn);
               writeOut :- xExpr:writeOut;}.
```

```
new9xStmt  = { writeOut := putmykey("resume", pos, writeIn);}.

new10xStmt = { xStmtSignal:writeIn :- writeIn ;
               writeOut :- xStmtSignal:writeOut;}.

new11xStmt = { xStmtNotify:writeIn :- writeIn;
               writeOut :- xStmtNotify:writeOut;}.

new12xStmt = { xId:writeIn := putmykey("escape", pos, writeIn);
               xActuList:writeIn :- xId:writeOut;
               writeOut :- xActuList:writeOut;}.

new13xStmt = { xStdIO:writeIn :- writeIn;
               xActuList:writeIn :- xStdIO:writeOut;
               writeOut :- xActuList:writeOut;}.

new14xStmt = { xLValue:writeIn := { putmykey("any", pos1, writeIn);
                                    xLValue:writeIn = putmykey("(",
                                                                 pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new15xStmt = { xLValue:writeIn := { putmykey("rany", pos1, writeIn);
                                    xLValue:writeIn = putmykey("(",
                                                                 pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new16xStmt = { xLValue:writeIn := { putmykey("break", pos1, writeIn);
                                    xLValue:writeIn = putmykey("(",
                                                                 pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new17xStmt = { xLValue:writeIn := { putmykey("rbreak", pos1, writeIn);
                                    xLValue:writeIn = putmykey("(",
                                                                 pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new18xStmt = { xLValue:writeIn := { putmykey("len", pos1, writeIn);
                                    xLValue:writeIn = putmykey("(",
                                                                 pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new19xStmt = { xLValue:writeIn := { putmykey("rlen", pos1, writeIn);
```

```
                                xLValue:writeIn = putmykey("(",
                                                           pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new20xStmt = { xLValue:writeIn := { putmykey("lpad", pos1, writeIn);
                                xLValue:writeIn = putmykey("(",
                                                           pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new21xStmt = { xLValue:writeIn := { putmykey("rpad", pos1, writeIn);
                                xLValue:writeIn = putmykey("(",
                                                           pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new22xStmt = { xLValue:writeIn := { putmykey("match", pos1, writeIn);
                                xLValue:writeIn = putmykey("(",
                                                           pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new23xStmt = { xLValue:writeIn := { putmykey("rmatch", pos1, writeIn);
                                xLValue:writeIn = putmykey("(",
                                                           pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new24xStmt = { xLValue:writeIn := { putmykey("notany", pos1, writeIn);
                                xLValue:writeIn = putmykey("(",
                                                           pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new25xStmt = { xLValue:writeIn := { putmykey("rnotany", pos1, writeIn);
                                xLValue:writeIn = putmykey("(",
                                                           pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new26xStmt = { xLValue:writeIn := { putmykey("span", pos1, writeIn);
                                xLValue:writeIn = putmykey("(",
                                                           pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.
```

```
new27xStmt = { xLValue:writeIn := { putmykey("rspan", pos1, writeIn);
                                    xLValue:writeIn = putmykey("(",
                                                              pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new28xStmt = { xLValue:writeIn :- writeIn;
               xExpr:writeIn := putmykey(":=", pos, xLValue:writeOut);
               writeOut :- xExpr:writeOut;}.

new29xStmt = { xLValue:writeIn :- writeIn;
               xBinOp:writeIn :- xLValue:writeOut;
               xExpr:writeIn := putmykey(":=", pos, xBinOp:writeOut);
               writeOut :- xExpr:writeOut;}.

new32xStmt = { No1:writeIn :- writeIn;
               xFrom:writeIn :- No1:writeOut;
               No2:writeIn :- xFrom:writeOut;
               writeOut :- No2:writeOut;}.

new33xStmt = { xLValue:writeIn :- writeIn;
               xActuList:writeIn :- xLValue:writeOut;
               writeOut :- xActuList:writeOut;}.

new34xStmt = { xLambda:writeIn :- writeIn;
               xActuList:writeIn :- xLambda:writeOut;
               writeOut :- xActuList:writeOut;}.

new35xStmt = { xActuList:writeIn := putmykey("self", pos, writeIn);
               writeOut :- xActuList:writeOut;}.

new36xStmt = { xExpr:writeIn := putmykey("if", pos1, writeIn);
               xStmts:writeIn := putmykey("then", pos2, xExpr:writeOut);
               xElIfStmts:writeIn :- xStmts:writeOut;
               xElseStmts:writeIn :- xElIfStmts:writeOut;
               writeOut := { putmykey("end", pos3, xElseStmts:writeOut);
                             writeOut = putmykey("if", pos4, 1);};}.

new37xStmt = { xExpr:writeIn := putmykey("case", pos1, writeIn);
               xCaseStmts:writeIn :- xExpr:writeOut;
               xElseStmts:writeIn :- xCaseStmts:writeOut;
               writeOut := { putmykey("end", pos2, xElseStmts:writeOut);
                             writeOut = putmykey("case", pos3, 1);};}.

new38xStmt = { xLoops:writeIn :- writeIn;
               writeOut := { putmykey("end", pos1, xLoops:writeOut);
                             writeOut = putmykey("loop", pos2, 1);};}.
```

```
new39xStmt = { xLoops:writeIn :- writeIn;
               writeOut := { putmykey("end", pos1, xLoops:writeOut);
                             writeOut = putmykey("for", pos2, 1);};}.

new40xStmt = { xLoops:writeIn :- writeIn;
               writeOut := { putmykey("end", pos1, xLoops:writeOut);
                             writeOut = putmykey("while", pos2, 1);};}.

new41xStmt = { xLoops:writeIn :- writeIn;
               writeOut := { putmykey("end", pos1, xLoops:writeOut);
                             writeOut = putmykey("whilefound", pos2, 1);};}.

new42xStmt = { xLoops:writeIn :- writeIn;
               writeOut := { putmykey("end", pos1, xLoops:writeOut);
                             writeOut = putmykey("repeat", pos2, 1);};}.

new43xStmt = { xLabel:writeIn :- writeIn;
               xLoops:writeIn :- xLabel:writeOut;
               xId:writeIn := putmykey("end", pos, xLoops:writeOut);
               writeOut :- xId:writeOut;}.

new44xStmt = { writeOut := putmykey("quit", pos, writeIn);}.

new45xStmt = { xId:writeIn := putmykey("quit", pos, writeIn);
               writeOut :- xId:writeOut;}.

new46xStmt = { writeOut := putmykey("continue", pos, writeIn);}.

new47xStmt = { xId:writeIn := putmykey("continue", pos, writeIn);
               writeOut :- xId:writeOut;}.

new48xStmt = { xExpr:writeIn := putmykey("||", pos, writeIn);
               writeOut :- xExpr:writeOut;}.

new49xStmt = { Expr1:writeIn := putmykey("deposit", pos1, writeIn);
               Expr2:writeIn := putmykey("at", pos2, Expr1:writeOut);
               writeOut := { putmykey("end", pos3, Expr2:writeOut);
                             writeOut = putmykey("deposit", pos4, 1);};}.

new50xStmt = { Expr1:writeIn := putmykey("deposit", pos1, writeIn);
               Expr2:writeIn := putmykey("at", pos2, Expr1:writeOut);
               writeOut := { putmykey("blockiffull", pos3, Expr2:writeOut);
                             putmykey("end", pos4, 1);
                             writeOut = putmykey("deposit", pos5, 1);};}.

new51xStmt = { xTemplate:writeIn := putmykey("fetch", pos1, writeIn);
```

```
              xExpr:writeIn := putmykey("at", pos2, xTemplate:writeOut);
              xElseStmts:writeIn :- xExpr:writeOut;
              writeOut := { putmykey("end", pos3, xElseStmts:writeOut);
                            writeOut = putmykey("fetch", pos4, 1);};}.

new52xStmt = { xTemplate:writeIn := putmykey("meet", pos1, writeIn);
              xExpr:writeIn := putmykey("at", pos2, xTemplate:writeOut);
              xElseStmts:writeIn :- xExpr:writeOut;
              writeOut := { putmykey("end", pos3, xElseStmts:writeOut);
                            writeOut = putmykey("meet", pos4, 1);};}.

new53xStmt = { xId:writeIn := putmykey("c_fct_call", pos1, writeIn);
              xTypeList:writeIn := putmykey("(", pos2, xId:writeOut);
              writeOut := putmykey(")", pos3, xTypeList:writeOut);}.

new54xStmt = { xExpr:writeIn := { putmykey("getf", pos1, writeIn);
                                  xExpr:writeIn = putmykey("(", pos2, 1);};
              xExprList:writeIn := putmykey(",", pos3, xExpr:writeOut);
              writeOut := putmykey(")", pos4, xExprList:writeOut);}.

new55xStmt = { xExpr:writeIn := { putmykey("fget", pos1, writeIn);
                                  xExpr:writeIn = putmykey("(", pos2, 1);};
              xExprList:writeIn := putmykey(",", pos3, xExpr:writeOut);
              writeOut := putmykey(")", pos4, xExprList:writeOut);}.

new56xStmt = { xExpr1:writeIn := { putmykey("fgetf", pos1, writeIn);
                                  xExpr1:writeIn = putmykey("(", pos2, 1);};
              xExpr2:writeIn := putmykey(",", pos3, xExpr1:writeOut);
              xExprList:writeIn := putmykey(",", pos4, xExpr2:writeOut);
              writeOut := putmykey(")", pos5, xExprList:writeOut);}.

xStmtSignal     = { writeOut :- writeIn;}.

new1xStmtSignal = { xLValue:writeIn := putmykey("signal", pos1, writeIn);
                    xId:writeIn := putmykey(":=", pos2, xLValue:writeOut);
                    xActuList:writeIn :- xId:writeOut;
                    writeOut :- xActuList:writeOut;}.

new2xStmtSignal = { xLValue:writeIn := putmykey("signal", pos, writeIn);
                    xActuList:writeIn :- xLValue:writeOut;
                    writeOut :- xActuList:writeOut;}.

xStmtNotify     = { writeOut :- writeIn;}.

new1xStmtNotify = { xLValue:writeIn := putmykey("notify", pos1, writeIn);
                    xId:writeIn := putmykey(":=", pos2, xLValue:writeOut);
                    xActuList:writeIn :- xId:writeOut;
```

```
                    writeOut :- xActuList:writeOut;}.

new2xStmtNotify = { xLValue:writeIn := putmykey("notify", pos, writeIn);
                    xActuList:writeIn :- xLValue:writeOut;
                    writeOut :- xActuList:writeOut;}.

xStdIO      = { writeOut :- writeIn;}.

new1xStdIO  = { writeOut := putmykey("put", pos, writeIn);}.

new2xStdIO  = { writeOut := putmykey("eput", pos, writeIn);}.

new3xStdIO  = { writeOut := putmykey("fput", pos, writeIn);}.

new4xStdIO  = { writeOut := putmykey("putf", pos, writeIn);}.

new5xStdIO  = { writeOut := putmykey("eputf", pos, writeIn);}.

new6xStdIO  = { writeOut := putmykey("fputf", pos, writeIn);}.

new7xStdIO  = { writeOut := putmykey("get", pos, writeIn);}.

/* new8xStdIO  = { writeOut := putmykey("fget", pos, writeIn);}. */
/* new9xStdIO  = { writeOut := putmykey("getf", pos, writeIn);}. */
/* new10xStdIO = { writeOut := putmykey("fgetf", pos, writeIn);}. */

new11xStdIO = { writeOut := putmykey("fopen", pos, writeIn);}.

new12xStdIO = { writeOut := putmykey("fclose", pos, writeIn);}.

xExpr      = { writeOut :- writeIn;}.

new1xExpr  = { xLValue:writeIn := { putmykey("any", pos1, writeIn);
                                    xLValue:writeIn = putmykey("(",
                                                                   pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new2xExpr  = { xLValue:writeIn := { putmykey("rany", pos1, writeIn);
                                    xLValue:writeIn = putmykey("(",
                                                                   pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new3xExpr  = { xLValue:writeIn := { putmykey("break", pos1, writeIn);
                                    xLValue:writeIn = putmykey("(",
                                                                   pos2, 1);};
```

```
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new4xExpr  = { xLValue:writeIn := { putmykey("rbreak", pos1, writeIn);
                                     xLValue:writeIn = putmykey("(",
                                                                    pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new5xExpr  = { xLValue:writeIn := { putmykey("len", pos1, writeIn);
                                     xLValue:writeIn = putmykey("(",
                                                                    pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new6xExpr  = { xLValue:writeIn := { putmykey("rlen", pos1, writeIn);
                                     xLValue:writeIn = putmykey("(",
                                                                    pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new7xExpr  = { xLValue:writeIn := { putmykey("match", pos1, writeIn);
                                     xLValue:writeIn = putmykey("(",
                                                                    pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new8xExpr  = { xLValue:writeIn := { putmykey("rmatch", pos1, writeIn);
                                     xLValue:writeIn = putmykey("(",
                                                                    pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new9xExpr  = { xLValue:writeIn := { putmykey("notany", pos1, writeIn);
                                     xLValue:writeIn = putmykey("(",
                                                                    pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new10xExpr  = { xLValue:writeIn := { putmykey("rnotany", pos1, writeIn);
                                      xLValue:writeIn = putmykey("(",
                                                                    pos2, 1);};
               xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
               writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new11xExpr  = { xLValue:writeIn := { putmykey("span", pos1, writeIn);
                                      xLValue:writeIn = putmykey("(",
```

```
                                                                  pos2, 1);};
             xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
             writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new12xExpr  = { xLValue:writeIn := { putmykey("rspan", pos1, writeIn);
                                     xLValue:writeIn = putmykey("(",
                                                                  pos2, 1);};
             xExpr:writeIn := putmykey(",", pos3, xLValue:writeOut);
             writeOut := putmykey(")", pos4, xExpr:writeOut);}.

new13xExpr = { intlit:writeIn :- writeIn;
               writeOut :- intlit:writeOut;}.

new14xExpr = { floatlit:writeIn :- writeIn;
               writeOut :- floatlit:writeOut;}.

new15xExpr = { writeOut := putmykey("true", pos, writeIn);}.

new16xExpr = { writeOut := putmykey("false", pos, writeIn);}.

new17xExpr = { writeOut := putmykey("om", pos, writeIn);}.

new18xExpr = { writeOut := putmykey("atom", pos, writeIn);}.

new19xExpr = { writeOut := putmykey("boolean", pos, writeIn);}.

new20xExpr = { writeOut := putmykey("integer", pos, writeIn);}.

new21xExpr = { writeOut := putmykey("real", pos, writeIn);}.

new22xExpr = { writeOut := putmykey("string", pos, writeIn);}.

new23xExpr = { writeOut := putmykey("tuple", pos, writeIn);}.

new24xExpr = { writeOut := putmykey("set", pos, writeIn);}.

new25xExpr = { writeOut := putmykey("function", pos, writeIn);}.

new26xExpr = { writeOut := putmykey("modtype", pos, writeIn);}.

new27xExpr = { writeOut := putmykey("instance", pos, writeIn);}.

new28xExpr = { writeOut := { putmykey("{", pos1, writeIn);
                             writeOut = putmykey("}", pos2, 1);};}.

new29xExpr = { writeOut := { putmykey("[", pos1, writeIn);
                             writeOut = putmykey("]", pos2, 1);};}.
```

```
new30xExpr = { writeOut := { putmykey("newat", pos1, writeIn);
                             putmykey("(", pos2, 1);
                             writeOut = putmykey(")", pos3, 1);};}.

new31xExpr = { xInstantiate:writeIn :- writeIn;
               writeOut :- xInstantiate:writeOut;}.

new32xExpr = { writeOut := putmykey("-", pos, writeIn);}.

new33xExpr = { str:writeIn :- writeIn;
               writeOut :- str:writeOut;}.

new34xExpr = { writeOut := putmykey("$", pos, writeIn);}.

new35xExpr = { writeOut := putmykey("argv", pos, writeIn);}.

new36xExpr = { xFormer:writeIn := putmykey("{", pos1, writeIn);
               writeOut := putmykey("}", pos2, xFormer:writeOut);}.

new37xExpr = { xFormer:writeIn := putmykey("[", pos1, writeIn);
               writeOut := putmykey("]", pos2, xFormer:writeOut);}.

new38xExpr = { xQualId:writeIn :- writeIn;
               writeOut :- xQualId:writeOut;}.

new39xExpr = { xLambda:writeIn :- writeIn;
              xActuList:writeIn :- xLambda:writeOut;
              writeOut :- xActuList:writeOut;}.

new40xExpr = { xActuList:writeIn := putmykey("self", pos, writeIn);
               writeOut :- xActuList:writeOut;}.

new41xExpr = { writeOut := { putmykey("closure", pos1, writeIn);
                             writeOut = putmykey("self", pos2, 1);};}.

new42xExpr = { xLambda:writeIn := putmykey("closure", pos, writeIn);
               writeOut :- xLambda:writeOut;}.

new43xExpr = { xQualId:writeIn := putmykey("closure", pos, writeIn);
               writeOut :- xQualId:writeOut;}.

new44xExpr = { xExpr:writeIn :- writeIn;
               xSelector:writeIn :- xExpr:writeOut;}.

new45xExpr = { xExpr:writeIn :- writeIn;
               writeOut := { putmykey("(", pos1, xExpr:writeOut);
```

```
                            writeOut = putmykey(")", pos2, 1);};}.

new46xExpr = { xExpr:writeIn :- writeIn;
               xHandAsso:writeIn := putmykey("[", pos1, xExpr:writeOut);
               writeOut := putmykey("]", pos2, xHandAsso:writeOut);}.

new47xExpr = { Expr1:writeIn := putmykey("if", pos1, writeIn);
               Expr2:writeIn := putmykey("then", pos2, Expr1:writeOut);
               xElIfExprs:writeIn :- Expr2:writeOut;
               xElseExpr:writeIn :- xElIfExprs:writeOut;
               writeOut := { putmykey("end", pos3, xElseExpr:writeOut);
                             writeOut = putmykey("if", pos4, 1);};}.

new48xExpr = { xExpr:writeIn := putmykey("case", pos1, writeIn);
               xCaseExprs:writeIn :- xExpr:writeOut;
               xElseExpr:writeIn :- xCaseExprs:writeOut;
               writeOut := { putmykey("end", pos2, xElseExpr:writeOut);
                             writeOut = putmykey("case", pos3, 1);};}.

new49xExpr = { xExpr:writeIn := putmykey("(", pos1, writeIn);
               writeOut := putmykey(")", pos2, xExpr:writeOut);}.

new50xExpr = { Expr1:writeIn :- writeIn;
               xBinOp:writeIn :- Expr1:writeOut;
               Expr2:writeIn :- xBinOp:writeOut;
               writeOut :- Expr2:writeOut;}.

new51xExpr = { xQuantifier:writeIn :- writeIn;
               writeOut :- xQuantifier:writeOut;}.

new52xExpr = { xUnOp:writeIn :- writeIn;
               xExpr:writeIn :- xUnOp:writeOut;
               writeOut :- xExpr:writeOut;}.

new53xExpr = { Id1:writeIn := putmykey("c_fct_call", pos1, writeIn);
               xTypeList:writeIn := putmykey("(", pos2, Id1:writeOut);
               Id2:writeIn := putmykey(")", pos3, xTypeList:writeOut);
               writeOut :- Id2:writeOut;}.

xFrom     = { writeOut :- writeIn;}.

new1xFrom = { writeOut := putmykey("from", pos, writeIn);}.

new2xFrom = { writeOut := putmykey("frome", pos, writeIn);}.

new3xFrom = { writeOut := putmykey("fromb", pos, writeIn);}.
```

```
xActuList     = { writeOut :- writeIn;}.

new1xActuList = { xExprList:writeIn := putmykey("(", pos1, writeIn);
                  writeOut := putmykey(")", pos2, xExprList:writeOut);}.

new2xActuList = { writeOut := { putmykey("(", pos1, writeIn);
                                writeOut = putmykey(")", pos2, 1);};}.

xElIfStmt = { xExpr:writeIn := putmykey("elseif", pos1, writeIn);
              xStmts:writeIn := putmykey("then", pos2, xExpr:writeOut);
              writeOut :- xStmts:writeOut;}.

xElIfStmts     = { writeOut :- writeIn;}.

new1xElIfStmts = { xElIfStmts:writeIn :- writeIn;
                   xElIfStmt:writeIn :- xElIfStmts:writeOut;
                   writeOut :- xElIfStmt:writeOut;}.

new2xElIfStmts = { writeOut :- writeIn;}.

xElseStmts     = { writeOut :- writeIn;}.

new1xElseStmts = { xStmts:writeIn := putmykey("else", pos, writeIn);
                   writeOut :- xStmts:writeOut;}.

new2xElseStmts = { writeOut :- writeIn;}.

xCaseStmts     = { writeOut :- writeIn;}.

new1xCaseStmts = { No1:writeIn :- writeIn;
                   No2:writeIn :- No1:writeOut;
                   writeOut :- No2:writeOut;}.

new2xCaseStmts = { xCaseStmts:writeIn :- writeIn;
                   writeOut :- xCaseStmts:writeOut;}.

new3xCaseStmts = { xExprList:writeIn :- writeIn;
                   xStmts:writeIn :- xExprList:writeOut;
                   writeOut :- xStmts:writeOut;}.

xLabel = { xId:writeIn :- writeIn;
           writeOut := putmykey(":", pos, xId:writeOut);}.

xTemplate     = { writeOut :- writeIn;}.

new1xTemplate = { xExprFormal:writeIn := putmykey("(", pos1, writeIn);
                  xTempCond:writeIn :- xExprFormal:writeOut;
```

```
                  xStmts:writeIn := { putmykey(")", pos2, xTempCond:writeOut);
                                      xStmts:writeIn = putmykey("=>",
                                                                pos3, 1);};
                  writeOut :- xStmts:writeOut;}.

new2xTemplate = { xExprFormal:writeIn := putmykey("(", pos1, writeIn);
                  xTempCond:writeIn :- xExprFormal:writeOut;
                  writeOut := putmykey(")", pos2, xTempCond:writeOut);}.

new3xTemplate = { No1:writeIn :- writeIn;
                  No2:writeIn := putmykey("xor", pos, No1:writeOut);
                  writeOut :- No2:writeOut;}.

xTempCond     = { writeOut :- writeIn;}.

new1xTempCond = { xExpr:writeIn := putmykey("|", pos, writeIn);
                  writeOut :- xExpr:writeOut;}.

new2xTempCond = { writeOut :- writeIn;}.

xExprFormal     = { writeOut :- writeIn;}.

new1xExprFormal = { xExpr:writeIn :- writeIn;
                    writeOut :- xExpr:writeOut;}.

new2xExprFormal = { xFormal:writeIn := putmykey("?", pos, writeIn);
                    xInto:writeIn :- xFormal:writeOut;
                    writeOut :- xInto:writeOut;}.

new3xExprFormal = { writeOut :- writeIn;}.

new4xExprFormal = { No1:writeIn :- writeIn;
                    No2:writeIn := putmykey(",", pos, No1:writeOut);
                    writeOut :- No2:writeOut;}.

xInto     = { writeOut :- writeIn;}.

new1xInto = { xExpr:writeIn := putmykey("into", pos, writeIn);
              writeOut :- xExpr:writeOut;}.

new2xInto = { writeOut :- writeIn;}.

xFormal     = { writeOut :- writeIn;}.

new1xFormal = { xLValue:writeIn :- writeIn;
                writeOut :- xLValue:writeOut;}.
```

```
new2xFormal = { writeOut :- writeIn;}.

xIterator     = { writeOut :- writeIn;}.

new1xIterator = { xSimpleIts:writeIn :- writeIn;
                  xExpr:writeIn := putmykey("|", pos, xSimpleIts:writeOut);
                  writeOut :- xExpr:writeOut;}.

new2xIterator = { xSimpleIts:writeIn :- writeIn;
                  writeOut :- xSimpleIts:writeOut;}.

xSimpleIts     = { writeOut :- writeIn;}.

new1xSimpleIts = { xSimpleIts:writeIn :- writeIn;
                   xSimpleIt:writeIn := putmykey(",", pos,
                                                 xSimpleIts:writeOut);
                   writeOut :- xSimpleIt:writeOut;}.

new2xSimpleIts = { xSimpleIt:writeIn :- writeIn;
                   writeOut :- xSimpleIt:writeOut;}.

xSimpleIt = { writeOut :- writeIn;}.

new1xSimpleIt = { xLValue:writeIn :- writeIn;
                  xExpr:writeIn := putmykey("in", pos, xLValue:writeOut);
                  writeOut :- xExpr:writeOut;}.

new2xSimpleIt = { xLValue:writeIn :- writeIn;
                  xId:writeIn := putmykey("=", pos, xLValue:writeOut);
                  xMapSel:writeIn :- xId:writeOut;
                  writeOut :- xMapSel:writeOut;}.

xMapSel     = { writeOut :- writeIn;}.

new1xMapSel = { xLValue:writeIn := putmykey("(", pos1, writeIn);
                writeOut := putmykey(")", pos2, xLValue:writeOut);}.

new2xMapSel = { xLValue:writeIn := putmykey("{", pos1, writeIn);
                writeOut := putmykey("}", pos2, xLValue:writeOut);}.

xLValue     = { writeOut :- writeIn;}.

new2xLValue = { xLValue:writeIn := putmykey("[", pos1, writeIn);
                writeOut := putmykey("]", pos2, xLValue:writeOut);}.

new3xLValue = { No1:writeIn :- writeIn;
                No2:writeIn := putmykey(",", pos, No1:writeOut);
```

```
                writeOut :- No2:writeOut;}.

new4xLValue = { writeOut := putmykey("-", pos, writeIn);}.

new5xLValue = { xQualId:writeIn :- writeIn;
                writeOut :- xQualId:writeOut;}.

new6xLValue = { xLValue:writeIn :- writeIn;
                xSelector:writeIn :- xLValue:writeOut;
                writeOut :- xSelector:writeOut;}.

xSelector     = { writeOut :- writeIn;}.

new1xSelector = { xExprList:writeIn := putmykey("(", pos1, writeIn);
                  writeOut := putmykey(")", pos2, xExprList:writeOut);}.

new2xSelector = { xExprList:writeIn := putmykey("{", pos1, writeIn);
                  writeOut := putmykey("}", pos2, xExprList:writeOut);}.

new3xSelector = { xExpr:writeIn := putmykey("(", pos1, writeIn);
                  writeOut := { putmykey("..", pos2, xExpr:writeOut);
                                writeOut = putmykey(")", pos3, 1);};}.

new4xSelector = { Expr1:writeIn := putmykey("(", pos1, writeIn);
                  Expr2:writeIn := putmykey("..", pos2, Expr1:writeOut);
                  writeOut := putmykey(")", pos3, Expr2:writeOut);}.

xFormer     = { writeOut :- writeIn;}.

new1xFormer = { xExpr:writeIn :- writeIn;
                writeOut :- xExpr:writeOut;}.

new2xFormer = { Expr1:writeIn :- writeIn;
                Expr2:writeIn := putmykey("..", pos, Expr1:writeOut);
                writeOut :- Expr2:writeOut;}.

new3xFormer = { xExpr:writeIn :- writeIn;
                xIterator:writeIn := putmykey(":", pos, xExpr:writeOut);
                writeOut :- xIterator:writeOut;}.

new4xFormer = { xExpr:writeIn :- writeIn;
                xExprList:writeIn := putmykey(",", pos, xExpr:writeOut);
                writeOut :- xExprList:writeOut;}.

new5xFormer = { Expr1:writeIn :- writeIn;
                Expr2:writeIn := putmykey(",", pos1, Expr1:writeOut);
                Expr3:writeIn := putmykey("..", pos2, Expr2:writeOut);
```

```
               writeOut :- Expr3:writeOut;}.

xExprList     = { writeOut :- writeIn;}.

new1xExprList = { xExprList:writeIn :- writeIn;
                  xExpr:writeIn := putmykey(",", pos, xExprList:writeOut);
                  writeOut :- xExpr:writeOut;}.

new2xExprList = { xExpr:writeIn :- writeIn;
                  writeOut :- xExpr:writeOut;}.

new3xExprList = { xExprList:writeIn := putmykey("when", pos1, writeIn);
                  writeOut := putmykey("=>", pos2, xExprList:writeOut);}.

xInstantiate = { xExpr:writeIn := putmykey("instantiate", pos1, writeIn);
                 xInstExport:writeIn :- xExpr:writeOut;
                 xInstImport:writeIn :- xInstExport:writeOut;
                 writeOut := { putmykey("end", pos2, xInstImport:writeOut);
                               writeOut = putmykey("instantiate", pos3, 1);};}.

xInstExport     = { writeOut :- writeIn;}.

new1xInstExport = { xExprList:writeIn := putmykey("export", pos1, writeIn);
                    writeOut := putmykey(";", pos2, xExprList:writeOut);}.

new2xInstExport = { writeOut :- writeIn;}.

xInstImport     = { writeOut :- writeIn;}.

new1xInstImport = { xIdList:writeIn := putmykey("import", pos1, writeIn);
                    writeOut := putmykey(";", pos2, xIdList:writeOut);}.

new2xInstImport = { writeOut :- writeIn;}.

xQualId     = { writeOut :- writeIn;}.

new1xQualId = { Id1:writeIn :- writeIn;
                Id2:writeIn := putmykey(".", pos, Id1:writeOut);
                writeOut :- Id2:writeOut;}.

new2xQualId = { xId:writeIn :- writeIn;
                writeOut :- xId:writeOut;}.

xLambda = { xParamList:writeIn := putmykey("lambda", pos1, writeIn);
            xProgBody:writeIn := putmykey(":", pos2, xParamList:writeOut);
            writeOut := { putmykey("end", pos3, xProgBody:writeOut);
                          writeOut = putmykey("lambda", pos4, 1);};}.
```

```
xQuantifier = { xQualifier:writeIn :- writeIn;
                xSimpleIts:writeIn :- xQualifier:writeOut;
                xExpr:writeIn := putmykey("|", pos, xSimpleIts:writeOut);
                writeOut :- xExpr:writeOut;}.

xQualifier     = { writeOut :- writeIn;}.

new1xQualifier = { writeOut := putmykey("exists", pos, writeIn);}.
new2xQualifier = { writeOut := putmykey("forall", pos, writeIn);}.

xElIfExprs     = { writeOut :- writeIn;}.

new1xElIfExprs = { xElIfExprs:writeIn :- writeIn;
                   xElIfExpr:writeIn :- xElIfExprs:writeOut;
                   writeOut :- xElIfExpr:writeOut;}.

new2xElIfExprs = { writeOut :- writeIn;}.

xElIfExpr = { Expr1:writeIn := putmykey("elseif", pos1, writeIn);
              Expr2:writeIn := putmykey("then", pos2, Expr1:writeOut);
              writeOut :- Expr2:writeOut;}.

xElseExpr     = { writeOut :- writeIn;}.

new1xElseExpr = { xExpr:writeIn := putmykey("else", pos, writeIn);
                  writeOut :- xExpr:writeOut;}.

new2xElseExpr = { writeOut :- writeIn;}.

xCaseExprs     = { writeOut :- writeIn;}.

new1xCaseExprs = { No1:writeIn :- writeIn;
                   No2:writeIn :- No1:writeOut;
                   writeOut :- No2:writeOut;}.

new2xCaseExprs = { xCaseExprs:writeIn :- writeIn;
                   writeOut :- xCaseExprs:writeOut;}.

new3xCaseExprs = { xExprList:writeIn :- writeIn;
                   xExpr:writeIn :- xExprList:writeOut;
                   writeOut :- xExpr:writeOut;}.

xBinOp     = { writeOut :- writeIn;}.

new1xBinOp  = { writeOut := putmykey("or", pos, writeIn);}.
```

```
new2xBinOp  = { writeOut := { putmykey("or", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new3xBinOp  = { writeOut := putmykey("and", pos, writeIn);}.

new4xBinOp  = { writeOut := { putmykey("and", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new5xBinOp  = { writeOut := putmykey("=", pos, writeIn);}.

new6xBinOp  = { writeOut := putmykey("/=", pos, writeIn);}.

new7xBinOp  = { writeOut := putmykey("<", pos, writeIn);}.

new8xBinOp  = { writeOut := putmykey("<=", pos, writeIn);}.

new9xBinOp  = { writeOut := putmykey(">", pos, writeIn);}.

new10xBinOp = { writeOut := putmykey(">=", pos, writeIn);}.

new11xBinOp = { writeOut := putmykey("in", pos, writeIn);}.

new12xBinOp = { writeOut := putmykey("notin", pos, writeIn);}.

new13xBinOp = { writeOut := putmykey("subset", pos, writeIn);}.

new14xBinOp = { writeOut := { putmykey("=", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new15xBinOp = { writeOut := { putmykey("/=", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new16xBinOp = { writeOut := { putmykey(",", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new17xBinOp = { writeOut := { putmykey("<=", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new18xBinOp = { writeOut := { putmykey(">", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new19xBinOp = { writeOut := { putmykey(">=", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new20xBinOp = { writeOut := { putmykey("in", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.
```

```
new21xBinOp = { writeOut := { putmykey("notin", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new22xBinOp = { writeOut := { putmykey("subset", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new23xBinOp = { writeOut := putmykey("with", pos, writeIn);}.

new24xBinOp = { writeOut := putmykey("less", pos, writeIn);}.

new25xBinOp = { writeOut := putmykey("lessf", pos, writeIn);}.

new26xBinOp = { xQualId:writeIn := putmykey("!", pos, writeIn);
                writeOut :- xQualId:writeOut;}.

new27xBinOp = { writeOut := { putmykey("with", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new28xBinOp = { writeOut := { putmykey("less", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new29xBinOp = { writeOut := { putmykey("lessf", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new30xBinOp = { xQualId:writeIn := putmykey("!", pos1, writeIn);
                writeOut := putmykey("%", pos2, xQualId:writeOut);}.

new31xBinOp = { writeOut := putmykey("+", pos, writeIn);}.

new32xBinOp = { writeOut := putmykey("-", pos, writeIn);}.

new33xBinOp = { writeOut := putmykey("max", pos, writeIn);}.

new34xBinOp = { writeOut := putmykey("min", pos, writeIn);}.

new35xBinOp = { writeOut := { putmykey("+", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new36xBinOp = { writeOut := { putmykey("-", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new37xBinOp = { writeOut := { putmykey("max", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new38xBinOp = { writeOut := { putmykey("min", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.
```

```
new39xBinOp = { writeOut := putmykey("*", pos, writeIn);}.

new40xBinOp = { writeOut := putmykey("/", pos, writeIn);}.

new41xBinOp = { writeOut := putmykey("mod", pos, writeIn);}.

new42xBinOp = { writeOut := { putmykey("*", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new43xBinOp = { writeOut := { putmykey("/", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new44xBinOp = { writeOut := { putmykey("mod", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new45xBinOp = { writeOut := putmykey("**", pos, writeIn);}.

new46xBinOp = { writeOut := { putmykey("**", pos1, writeIn);
                              writeOut = putmykey("%", pos2, 1);};}.

new47xBinOp = { writeOut := putmykey("npow", pos, writeIn);}.

xUnOp      = { writeOut :- writeIn;}.

new1xUnOp  = { writeOut := putmykey("+", pos, writeIn);}.

new2xUnOp  = { writeOut := putmykey("-", pos, writeIn);}.

new3xUnOp  = { writeOut := putmykey("#", pos, writeIn);}.

new4xUnOp  = { writeOut := putmykey("||", pos, writeIn);}.

new5xUnOp  = { writeOut := putmykey("not", pos, writeIn);}.

new6xUnOp  = { writeOut := putmykey("pow", pos, writeIn);}.

new7xUnOp  = { writeOut := putmykey("arb", pos, writeIn);}.

new8xUnOp  = { writeOut := putmykey("random", pos, writeIn);}.

new9xUnOp  = { writeOut := putmykey("domain", pos, writeIn);}.

new10xUnOp = { writeOut := putmykey("range", pos, writeIn);}.

new11xUnOp = { writeOut := putmykey("type", pos, writeIn);}.

new12xUnOp = { writeOut := putmykey("is_map", pos, writeIn);}.
```

```
new13xUnOp = { writeOut := putmykey("is_smap", pos, writeIn);}.

new14xUnOp = { xQualId:writeIn := putmykey("!", pos, writeIn);
               writeOut :- xQualId:writeOut;}.

new15xUnOp = { writeOut := { putmykey("or", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new16xUnOp = { writeOut := { putmykey("and", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new17xUnOp = { writeOut := { putmykey("=", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new18xUnOp = { writeOut := { putmykey("/=", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new19xUnOp = { writeOut := { putmykey("<", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new20xUnOp = { writeOut := { putmykey("<=", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new21xUnOp = { writeOut := { putmykey(">", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new22xUnOp = { writeOut := { putmykey(">=", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new23xUnOp = { writeOut := { putmykey("in", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new24xUnOp = { writeOut := { putmykey("notin", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new25xUnOp = { writeOut := { putmykey("subset", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new26xUnOp = { writeOut := { putmykey("with", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new27xUnOp = { writeOut := { putmykey("less", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new28xUnOp = { writeOut := { putmykey("lessf", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.
```

```
new29xUnOp = { xQualId:writeIn := putmykey("!", pos1, writeIn);
               writeOut := putmykey("%", pos2, xQualId:writeOut);}.

new30xUnOp = { writeOut := { putmykey("+", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new31xUnOp = { writeOut := { putmykey("-", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new32xUnOp = { writeOut := { putmykey("max", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new33xUnOp = { writeOut := { putmykey("min", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new34xUnOp = { writeOut := { putmykey("*", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new35xUnOp = { writeOut := { putmykey("/", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new36xUnOp = { writeOut := { putmykey("mod", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.

new37xUnOp = { writeOut := { putmykey("**", pos1, writeIn);
                             writeOut = putmykey("%", pos2, 1);};}.


xTypeList = { writeOut :- writeIn;}.

new1xTypeList = { writeOut :- writeIn;}.

new2xTypeList = { Id1:writeIn :- writeIn;
                  Id2:writeIn := putmykey(":", pos, Id1:writeOut);
                  writeOut :- Id2:writeOut;}.

new3xTypeList = { xTypeList:writeIn :- writeIn;
                  Id1:writeIn := putmykey(",", pos1, xTypeList:writeOut);
                  Id2:writeIn := putmykey(":", pos2, Id1:writeOut);
                  writeOut :- Id2:writeOut;}.
```

C.42.7 Die Fortsetzung des Attributauswerters `Muster`

⟨*muster*⟩≡

```
xParamList = { musterOut :- musterIn;} .

new1xParamList = { musterOut := fprintf(ofile,"new1xParamList\n", musterIn);}.

new2xParamList = { xParamList:musterIn :=
                              fprintf(ofile, "new2xParamList(", musterIn);
                   musterOut := fprintf(ofile, ")\n", xParamList:musterOut); } .

new3xParamList = { xParamList:musterIn :=
                              fprintf(ofile, "new3xParamList(", musterIn);
                   xParamMode:musterIn := fprintf(ofile, ",", xParamList:musterOut);
                   xId:musterIn := fprintf(ofile, ",", xParamMode:musterOut);
                   musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

new4xParamList = { xParamMode:musterIn :=
                              fprintf(ofile, "new4xParamList(", musterIn);
                   xId:musterIn := fprintf(ofile, ",", xParamMode:musterOut);
                   musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

xParamMode = { musterOut :- musterIn;} .

new1xParamMode = { musterOut := fprintf(ofile,"new1xParamMode \n", musterIn);}.

new2xParamMode = { musterOut := fprintf(ofile,"new2xParamMode \n", musterIn);}.

new3xParamMode = { musterOut := fprintf(ofile,"new3xParamMode \n", musterIn);}.

new4xParamMode = { musterOut := fprintf(ofile,"new4xParamMode \n", musterIn);}.

xModImport = { musterOut :- musterIn;} .

new1xModImport = { xIdList:musterIn :=
                           fprintf(ofile, "new1xModImport(", musterIn);
                   musterOut := fprintf(ofile, ")\n", xIdList:musterOut);}.

new2xModImport = { musterOut := fprintf(ofile,"new2xModImport \n", musterIn);}.

xModExport = { musterOut :- musterIn;} .

new1xModExport = { xIdList:musterIn :=
                           fprintf(ofile, "new1xModExport(", musterIn);
                   musterOut := fprintf(ofile, ")\n", xIdList:musterOut);} .

new2xModExport = { musterOut := fprintf(ofile,"new2xModExport \n", musterIn);}.
```

```
xIdList = { musterOut :- musterIn;} .

new1xIdList = { xIdList:musterIn := fprintf(ofile, "new1xIdList(", musterIn);
                xId:musterIn := fprintf(ofile, ",", xIdList:musterOut);
                musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

new2xIdList = { xId:musterIn := fprintf(ofile, "new2xIdList(", musterIn);
                musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

xRdParamList = { musterOut :- musterIn;} .

new1xRdParamList = { musterOut := fprintf(ofile, "new1xRdParamList \n", musterIn); } .

new2xRdParamList = { xIdList:musterIn :=
                                      fprintf(ofile, "new2xRdParamList(", musterIn);
                     musterOut := fprintf(ofile, ")\n", xIdList:musterOut);} .

xImplAsso = { musterOut :- musterIn;} .

new1xImplAsso = { xIdList:musterIn :=
                                   fprintf(ofile, "new1xImplAsso(", musterIn);
                  musterOut := fprintf(ofile, ")\n", xIdList:musterOut);} .

new2xImplAsso = { musterOut := fprintf(ofile, "new2xImplAsso \n", musterIn);} .

new3xImplAsso = { musterOut := fprintf(ofile, "new3xImplAsso \n", musterIn);} .

xDecls = { musterOut :- musterIn;} .

new1xDecls = { xDecls:musterIn := fprintf(ofile, "new1xDecls(", musterIn);
               xDecl:musterIn := fprintf(ofile, ",", xDecls:musterOut);
               musterOut := fprintf(ofile, ")\n", xDecl:musterOut);}.

new2xDecls = { musterOut := fprintf(ofile, "new2xDecls \n", musterIn);} .

xDecl = { musterOut :- musterIn;} .

new1xDecl = { xDeclKey:musterIn := fprintf(ofile, "new1xDecl(", musterIn);
              xSingleVar:musterIn := fprintf(ofile, ",", xDeclKey:musterOut);
              xExplAsso:musterIn := fprintf(ofile, ",", xSingleVar:musterOut);
              musterOut := fprintf(ofile, ")\n", xExplAsso:musterOut);}.

new2xDecl = { xPersDecl:musterIn := fprintf(ofile, "new2xDecl(", musterIn);
              xIdList:musterIn := fprintf(ofile, ",", xPersDecl:musterOut);
              xExpr:musterIn := fprintf(ofile, ",", xIdList:musterOut);
              xExplAsso:musterIn := fprintf(ofile, ",", xExpr:musterOut);
```

```
              musterOut := fprintf(ofile, ")\n", xExplAsso:musterOut);}.

xSingleVar = { musterOut :- musterIn;} .

new1xSingleVar = { xId:musterIn :=
                                  fprintf(ofile, "new1xSingleVar(", musterIn);
                   xExpr:musterIn := fprintf(ofile, ",", xId:musterOut);
                   musterOut := fprintf(ofile, ")\n", xExpr:musterOut); } .

new2xSingleVar = { xId:musterIn :=
                                  fprintf(ofile, "new2xSingleVar(", musterIn);
                   musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

new3xSingleVar = { No1:musterIn :=
                                  fprintf(ofile, "new3xSingleVar(", musterIn);
                   No2:musterIn := fprintf(ofile, ",", No1:musterOut);
                   musterOut := fprintf(ofile, ")\n", No2:musterOut);}.

xDeclKey = { musterOut :- musterIn;} .

new1xDeclKey = { musterOut := fprintf(ofile, "new1xDeclKey \n", musterIn);} .

new2xDeclKey = { musterOut := fprintf(ofile, "new2xDeclKey \n", musterIn);} .

new3xDeclKey = { musterOut := fprintf(ofile, "new3xDeclKey \n", musterIn);} .

new4xDeclKey = { musterOut := fprintf(ofile, "new4xDeclKey \n", musterIn);} .

new5xDeclKey = { musterOut := fprintf(ofile, "new5xDeclKey \n", musterIn);} .

xPersDecl = { musterOut :- musterIn;} .

new1xPersDecl = { musterOut := fprintf(ofile, "new1xPersDecl \n", musterIn);} .

new2xPersDecl = { musterOut := fprintf(ofile, "new2xPersDecl \n", musterIn);} .

new3xPersDecl = { musterOut := fprintf(ofile, "new3xPersDecl \n", musterIn);} .

new4xPersDecl = { musterOut := fprintf(ofile, "new4xPersDecl \n", musterIn);} .

new5xPersDecl = { musterOut := fprintf(ofile, "new5xPersDecl \n", musterIn);} .

new6xPersDecl = { musterOut := fprintf(ofile, "new6xPersDecl \n", musterIn);} .

xStmts = { musterOut :- musterIn;} .

new1xStmts = { xStmts:musterIn := fprintf(ofile, "new1xStmts(", musterIn);
```

```
               xStmt:musterIn := fprintf(ofile, ",", xStmts:musterOut);
               xExplAsso:musterIn := fprintf(ofile, ",", xStmt:musterOut);
               musterOut := fprintf(ofile, ")\n", xExplAsso:musterOut);}.

new2xStmts = { xStmt:musterIn := fprintf(ofile, "new2xStmts(", musterIn);
               xExplAsso:musterIn := fprintf(ofile, ",", xStmt:musterOut);
               musterOut := fprintf(ofile, ")\n", xExplAsso:musterOut);}.

new3xStmts = { xStmts:musterIn := fprintf(ofile, "new3xStmts(", musterIn);
               musterOut := fprintf(ofile, ")\n", xStmts:musterIn);} .

xExplAsso = { musterOut :- musterIn;} .

new1xExplAsso = { xExplAsso:musterIn :=
                                      fprintf(ofile, "new1xExplAsso(", musterIn);
                  xHandAsso:musterIn := fprintf(ofile, ",", xExplAsso:musterOut);
                  musterOut := fprintf(ofile, ")\n", xHandAsso:musterOut);}.

new2xExplAsso = { musterOut := fprintf(ofile, "new2xExplAsso \n", musterIn);} .

xHandAsso = { musterOut :- musterIn;} .

new1xHandAsso = { xIdList:musterIn := fprintf(ofile, "new1xHandAsso(", musterIn);
                  xId:musterIn := fprintf(ofile, ",", xIdList:musterOut);
                  musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

new2xHandAsso = { xId:musterIn := fprintf(ofile, "new2xHandAsso(", musterIn);
                  musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

xStmt = { musterOut :- musterIn;} .

new1xStmt = { musterOut := fprintf(ofile, "new1xStmt \n", musterIn);} .

new2xStmt = { musterOut := fprintf(ofile, "new2xStmt \n", musterIn);} .

new3xStmt = { xExpr:musterIn := fprintf(ofile, "new3xStmt(", musterIn);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new4xStmt = { xExpr:musterIn := fprintf(ofile, "new4xStmt(", musterIn);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new5xStmt = { musterOut := fprintf(ofile, "new5xStmt\n", musterIn);} .

new6xStmt = { xExpr:musterIn := fprintf(ofile, "new6xStmt(", musterIn);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new7xStmt = { musterOut := fprintf(ofile, "new7xStmt \n", musterIn);} .
```

```
new8xStmt = { xExpr:musterIn := fprintf(ofile, "new8xStmt(", musterIn);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new9xStmt = { musterOut := fprintf(ofile, "new9xStmt \n", musterIn);} .

new10xStmt = { xStmtSignal:musterIn :=
                                   fprintf(ofile, "new10xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xStmtSignal:musterOut);}.

new11xStmt = { xStmtNotify:musterIn :=
                                   fprintf(ofile, "new11xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xStmtNotify:musterOut);}.

new12xStmt = { xId:musterIn := fprintf(ofile, "new12xStmt(", musterIn);
               xActuList:musterIn := fprintf(ofile, ",", xId:musterOut);
               musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

new13xStmt = { xStdIO:musterIn := fprintf(ofile, "new13xStmt(", musterIn);
               xActuList:musterIn := fprintf(ofile, ",", xStdIO:musterOut);
               musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

new14xStmt = { xLValue:musterIn := fprintf(ofile, "new14xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new15xStmt = { xLValue:musterIn := fprintf(ofile, "new15xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new16xStmt = { xLValue:musterIn := fprintf(ofile, "new16xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new17xStmt = { xLValue:musterIn := fprintf(ofile, "new17xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new18xStmt = { xLValue:musterIn := fprintf(ofile, "new18xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new19xStmt = { xLValue:musterIn := fprintf(ofile, "new19xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new20xStmt = { xLValue:musterIn := fprintf(ofile, "new20xStmt(", musterIn);
```

```
                  xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
                  musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new21xStmt = { xLValue:musterIn := fprintf(ofile, "new21xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new22xStmt = { xLValue:musterIn := fprintf(ofile, "new22xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new23xStmt = { xLValue:musterIn := fprintf(ofile, "new23xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new24xStmt = { xLValue:musterIn := fprintf(ofile, "new24xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new25xStmt = { xLValue:musterIn := fprintf(ofile, "new25xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new26xStmt = { xLValue:musterIn := fprintf(ofile, "new26xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);} .

new27xStmt = { xLValue:musterIn := fprintf(ofile, "new27xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new28xStmt = { xLValue:musterIn := fprintf(ofile, "new28xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new29xStmt = { xLValue:musterIn := fprintf(ofile, "new29xStmt(", musterIn);
               xBinOp:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               xExpr:musterIn := fprintf(ofile, ",", xBinOp:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new32xStmt = { No1:musterIn := fprintf(ofile, "new32xStmt(", musterIn);
               xFrom:musterIn := fprintf(ofile, ",", No1:musterOut);
               No2:musterIn := fprintf(ofile, ",", xFrom:musterOut);
               musterOut := fprintf(ofile, ")\n", No2:musterOut);}.

new33xStmt = { xLValue:musterIn := fprintf(ofile, "new33xStmt(", musterIn);
               xActuList:musterIn := fprintf(ofile, ",", xLValue:musterOut);
```

```
               musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

new34xStmt = { xLambda:musterIn := fprintf(ofile, "new34xStmt(", musterIn);
               xActuList:musterIn := fprintf(ofile, ",", xLambda:musterOut);
               musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

new35xStmt = { xActuList:musterIn := fprintf(ofile, "new35xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

new36xStmt = { xExpr:musterIn := fprintf(ofile, "new36xStmt(", musterIn);
               xStmts:musterIn := fprintf(ofile, ",", xExpr:musterOut);
               xElIfStmts:musterIn := fprintf(ofile, ",", xStmts:musterOut);
               xElseStmts:musterIn := fprintf(ofile, ",", xElIfStmts:musterOut);
               musterOut := fprintf(ofile, ")\n", xElseStmts:musterOut);}.

new37xStmt = { xExpr:musterIn := fprintf(ofile, "new37xStmt(", musterIn);
               xCaseStmts:musterIn := fprintf(ofile, ",", xExpr:musterOut);
               xElseStmts:musterIn := fprintf(ofile, ",", xCaseStmts:musterOut);
               musterOut := fprintf(ofile, ")\n", xElseStmts:musterOut); }.

new38xStmt = { xLoops:musterIn := fprintf(ofile, "new38xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xLoops:musterOut); }.

new39xStmt = { xLoops:musterIn := fprintf(ofile, "new39xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xLoops:musterOut); }.

new40xStmt = { xLoops:musterIn := fprintf(ofile, "new40xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xLoops:musterOut); }.

new41xStmt = { xLoops:musterIn := fprintf(ofile, "new41xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xLoops:musterOut); }.

new42xStmt = { xLoops:musterIn := fprintf(ofile, "new42xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xLoops:musterOut); }.

new43xStmt = { xLabel:musterIn := fprintf(ofile, "new43xStmt(", musterIn);
               xLoops:musterIn := fprintf(ofile, ",", xLabel:musterOut);
               xId:musterIn := fprintf(ofile, ",", xLoops:musterOut);
               musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

new44xStmt = { musterOut := fprintf(ofile, "new44xStmt \n", musterIn);}.

new45xStmt = { xId:musterIn := fprintf(ofile, "new45xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

new46xStmt = { musterOut := fprintf(ofile, "new46xStmt(", musterIn);}.
```

```
new47xStmt = { xId:musterIn :=  fprintf(ofile, "new47xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

new48xStmt = { xExpr:musterIn := fprintf(ofile, "new48xStmt(", musterIn);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new49xStmt = { Expr1:musterIn := fprintf(ofile, "new49xStmt(", musterIn);
               Expr2:musterIn := fprintf(ofile, ",", Expr1:musterOut);
               musterOut := fprintf(ofile, ")\n", Expr2:musterOut);}.

new50xStmt = { Expr1:musterIn := fprintf(ofile, "new50xStmt(", musterIn);
               Expr2:musterIn := fprintf(ofile, ",", Expr1:musterOut);
               musterOut := fprintf(ofile, ")\n", Expr2:musterOut);}.

new51xStmt = { xTemplate:musterIn := fprintf(ofile, "new51xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xTemplate:musterOut);
               xElseStmts:musterIn := fprintf(ofile, ",", xExpr:musterOut);
               musterOut := fprintf(ofile, ")\n", xElseStmts:musterOut);}.

new52xStmt = { xTemplate:musterIn := fprintf(ofile, "new52xStmt(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xTemplate:musterOut);
               xElseStmts:musterIn := fprintf(ofile, ",", xExpr:musterOut);
               musterOut := fprintf(ofile, ")\n", xElseStmts:musterOut);}.

new53xStmt = { xId:musterIn := fprintf(ofile, "new53xStmt(", musterIn);
               xTypeList:musterIn := fprintf(ofile, ",", xId:musterOut);
               musterOut := fprintf(ofile, ")\n", xTypeList:musterOut);}.

new54xStmt = { xExpr:musterIn := fprintf(ofile, "new54xStmt(", musterIn);
               xExprList:musterIn := fprintf(ofile, ",", xExpr:musterOut);
               musterOut := fprintf(ofile, ")\n", xExprList:musterOut);}.

new55xStmt = { xExpr:musterIn := fprintf(ofile, "new55xStmt(", musterIn);
               xExprList:musterIn := fprintf(ofile, ",", xExpr:musterOut);
               musterOut := fprintf(ofile, ")\n", xExprList:musterOut);}.

new56xStmt = { xExpr1:musterIn := fprintf(ofile, "new56xStmt(", musterIn);
               xExpr2:musterIn := fprintf(ofile, ",", xExpr1:musterOut);
               xExprList:musterIn := fprintf(ofile, ",", xExpr2:musterOut);
               musterOut := fprintf(ofile, ")\n", xExprList:musterOut);}.

xStmtSignal = { musterOut :- musterIn;} .

new1xStmtSignal = { xLValue:musterIn :=
                                      fprintf(ofile, "new1xStmtSignal(", musterIn);
                    xId:musterIn := fprintf(ofile, ",", xLValue:musterOut);
                    xActuList:musterIn := fprintf(ofile, ",", xId:musterOut);
```

```
                     musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

new2xStmtSignal = { xLValue:musterIn :=
                                           fprintf(ofile, "new2xStmtSignal(", musterIn);
                     xActuList:musterIn := fprintf(ofile, ",", xLValue:musterOut);
                     musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

xStmtNotify = { musterOut :- musterIn;} .

new1xStmtNotify = { xLValue:musterIn :=
                                           fprintf(ofile, "new1xStmtNotify(", musterIn);
                     xId:musterIn := fprintf(ofile, ",", xLValue:musterOut);
                     xActuList:musterIn := fprintf(ofile, ",", xId:musterOut);
                     musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

new2xStmtNotify = { xLValue:musterIn := fprintf(ofile, "new2xStmtNotify(");
                     xActuList:musterIn := fprintf(ofile, ",", xLValue:musterOut);
                     musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

xStdIO = { musterOut :- musterIn;} .

new1xStdIO = { musterOut := fprintf(ofile, "new1xStdIO \n", musterIn);} .

new2xStdIO = { musterOut := fprintf(ofile, "new2xStdIO \n", musterIn);} .

new3xStdIO = { musterOut := fprintf(ofile, "new3xStdIO \n", musterIn);} .

new4xStdIO = { musterOut := fprintf(ofile, "new4xStdIO \n", musterIn);} .

new5xStdIO = { musterOut := fprintf(ofile, "new5xStdIO \n", musterIn);} .

new6xStdIO = { musterOut := fprintf(ofile, "new6xStdIO \n", musterIn);} .

new7xStdIO = { musterOut := fprintf(ofile, "new7xStdIO \n", musterIn);} .

/*new8xStdIO = { musterOut := fprintf(ofile, "new8xStdIO \n", musterIn);}.*/

/*new9xStdIO = { musterOut := fprintf(ofile, "new9xStdIO \n", musterIn);}.*/

/*new10xStdIO = { musterOut := fprintf(ofile, "new10xStdIO \n", musterIn);}.*/

new11xStdIO = { musterOut := fprintf(ofile, "new11xStdIO \n", musterIn);} .

new12xStdIO = { musterOut := fprintf(ofile, "new12xStdIO \n", musterIn);} .

xExpr = { musterOut :- musterIn;} .
```

```
new1xExpr = { xLValue:musterIn := fprintf(ofile, "new1xExpr(", musterIn);
              xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new2xExpr = { xLValue:musterIn := fprintf(ofile, "new2xExpr(", musterIn);
              xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new3xExpr = { xLValue:musterIn := fprintf(ofile, "new3xExpr(", musterIn);
              xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new4xExpr = { xLValue:musterIn := fprintf(ofile, "new4xExpr(", musterIn);
              xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new5xExpr = { xLValue:musterIn := fprintf(ofile, "new5xExpr(", musterIn);
              xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new6xExpr = { xLValue:musterIn :=  fprintf(ofile, "new6xExpr(", musterIn);
              xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new7xExpr = { xLValue:musterIn := fprintf(ofile, "new7xExpr(", musterIn);
              xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new8xExpr = { xLValue:musterIn := fprintf(ofile, "new8xExpr(", musterIn);
              xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new9xExpr = { xLValue:musterIn := fprintf(ofile, "new9xExpr(", musterIn);
              xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new10xExpr = { xLValue:musterIn := fprintf(ofile, "new10xExpr(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new11xExpr = { xLValue:musterIn := fprintf(ofile, "new11xExpr(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new12xExpr = { xLValue:musterIn := fprintf(ofile, "new12xExpr(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.
```

```
new13xExpr = { intlit:musterIn := fprintf(ofile, "new13xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", intlit:musterOut);}.

new14xExpr = { floatlit:musterIn := fprintf(ofile, "new14xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", floatlit:musterOut);}.

new15xExpr = { musterOut := fprintf(ofile, "new15xExpr \n", musterIn);} .

new16xExpr = { musterOut := fprintf(ofile, "new16xExpr \n", musterIn);} .

new17xExpr = { musterOut := fprintf(ofile, "new17xExpr \n", musterIn);} .

new18xExpr = { musterOut := fprintf(ofile, "new18xExpr \n", musterIn);} .

new19xExpr = { musterOut := fprintf(ofile, "new19xExpr \n", musterIn);} .

new20xExpr = { musterOut := fprintf(ofile, "new20xExpr \n", musterIn);} .

new21xExpr = { musterOut := fprintf(ofile, "new21xExpr \n", musterIn);} .

new22xExpr = { musterOut := fprintf(ofile, "new22xExpr \n", musterIn);} .

new23xExpr = { musterOut := fprintf(ofile, "new23xExpr \n", musterIn);} .

new24xExpr = { musterOut := fprintf(ofile, "new24xExpr \n", musterIn);} .

new25xExpr = { musterOut := fprintf(ofile, "new25xExpr \n", musterIn);} .

new26xExpr = { musterOut := fprintf(ofile, "new26xExpr \n", musterIn);} .

new27xExpr = { musterOut := fprintf(ofile, "new27xExpr \n", musterIn);} .

new28xExpr = { musterOut := fprintf(ofile, "new28xExpr \n", musterIn);} .

new29xExpr = { musterOut := fprintf(ofile, "new29xExpr \n", musterIn);} .

new30xExpr = { musterOut := fprintf(ofile, "new30xExpr \n", musterIn);} .

new31xExpr = { xInstantiate:musterIn :=
                         fprintf(ofile, "new31xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", xInstantiate:musterOut);}.

new32xExpr = { musterOut := fprintf(ofile, "new32xExpr \n", musterIn);} .

new33xExpr = { str:musterIn := fprintf(ofile, "new33xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", str:musterOut);}.
```

```
new34xExpr = { musterOut := fprintf(ofile, "new34xExpr \n", musterIn);} .

new35xExpr = { musterOut := fprintf(ofile, "new35xExpr \n", musterIn);} .

new36xExpr = { xFormer:musterIn := fprintf(ofile, "new36xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", xFormer:musterOut);}.

new37xExpr = { xFormer:musterIn := fprintf(ofile, "new37xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", xFormer:musterOut);}.

new38xExpr = { xQualId:musterIn := fprintf(ofile, "new38xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", xQualId:musterOut);}.

new39xExpr = { xLambda:musterIn := fprintf(ofile, "new39xExpr(", musterIn);
              xActuList:musterIn := fprintf(ofile, ",", xLambda:musterOut);
              musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

new40xExpr = { xActuList:musterIn := fprintf(ofile, "new40xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", xActuList:musterOut);}.

new41xExpr = { musterOut := fprintf(ofile, "new41xExpr \n", musterIn);} .

new42xExpr = { xLambda:musterIn := fprintf(ofile, "new42xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", xLambda:musterOut);}.

new43xExpr = { xQualId:musterIn := fprintf(ofile, "new43xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", xQualId:musterOut);}.

new44xExpr = { xExpr:musterIn := fprintf(ofile, "new44xExpr(", musterIn);
               xSelector:musterIn := fprintf(ofile, ")\n", xExpr:musterOut);}.
new45xExpr = { xExpr:musterIn := fprintf(ofile, "new45xExpr \n", musterIn);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new46xExpr = { xExpr:musterIn := fprintf(ofile, "new46xExpr(", musterIn);
               xHandAsso:musterIn := fprintf(ofile, ",", xExpr:musterOut);
               musterOut := fprintf(ofile, ")\n", xHandAsso:musterOut);}.

new47xExpr = { Expr1:musterIn := fprintf(ofile, "new47xExpr(", musterIn);
               Expr2:musterIn := fprintf(ofile, ",", Expr1:musterOut);
               xElIfExprs:musterIn := fprintf(ofile, ",", Expr2:musterOut);
               xElseExpr:musterIn := fprintf(ofile, ",", xElIfExprs:musterOut);
               musterOut := fprintf(ofile, ")\n", xElseExpr:musterOut);}.

new48xExpr = { xExpr:musterIn := fprintf(ofile, "new48xExpr(", musterIn);
               xCaseExprs:musterIn := fprintf(ofile, ",", xExpr:musterOut);
               xElseExpr:musterIn := fprintf(ofile, ",", xCaseExprs:musterOut);
```

```
               musterOut := fprintf(ofile, ")\n", xElseExpr:musterOut);}.

new49xExpr = { xExpr:musterIn := fprintf(ofile, "new49xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new50xExpr = { Expr1:musterIn := fprintf(ofile, "new50xExpr(", musterIn);
               xBinOp:musterIn := fprintf(ofile, ",", Expr1:musterOut);
               Expr2:musterIn := fprintf(ofile, ",", xBinOp:musterOut);
               musterOut := fprintf(ofile, ")\n", Expr2:musterOut);}.

new51xExpr = { xQuantifier:musterIn :=
                                   fprintf(ofile, "new51xExpr(", musterIn);
               musterOut := fprintf(ofile, ")\n", xQuantifier:musterOut);}.

new52xExpr = { xUnOp:musterIn := fprintf(ofile, "new52xExpr(", musterIn);
               xExpr:musterIn := fprintf(ofile, ",", xUnOp:musterOut);
               musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new53xExpr = { Id1:musterIn := fprintf(ofile, "new53xExpr(", musterIn);
               xTypeList:musterIn := fprintf(ofile, ",", Id1:musterOut);
               Id2:musterIn := fprintf(ofile, ",", xTypeList:musterOut);
               musterOut := fprintf(ofile, ")\n", Id2:musterOut);}.

xFrom = { musterOut :- musterIn;} .

new1xFrom = { musterOut := fprintf(ofile, "new1xFrom \n", musterIn);} .

new2xFrom = { musterOut := fprintf(ofile, "new2xFrom \n", musterIn);} .

new3xFrom = { musterOut := fprintf(ofile, "new3xFrom \n", musterIn);} .

xActuList = {  musterOut :- musterIn;} .

new1xActuList = { xExprList:musterIn :=
                                          fprintf(ofile, "new1xActuList(", musterIn);
                  musterOut := fprintf(ofile, ")\n", xExprList:musterOut);}.

new2xActuList = { musterOut := fprintf(ofile, "new2xActuList \n", musterIn);} .

xElIfStmt = { xExpr:musterIn := fprintf(ofile, "xElIfStmt(", musterIn);
              xStmts:musterIn := fprintf(ofile, ",", xExpr:musterOut);
              musterOut := fprintf(ofile, ")\n", xStmts:musterOut);}.

xElIfStmts = { musterOut :- musterIn;} .

new1xElIfStmts = { xElIfStmts:musterIn :=
                                            fprintf(ofile, "new1xElIfStmts(",musterIn);
```

```
                    xElIfStmt:musterIn := fprintf(ofile, ",", xElIfStmts:musterOut);
                    musterOut := fprintf(ofile, ")\n", xElIfStmt:musterOut);}.

new2xElIfStmts = { musterOut := fprintf(ofile,"new2xElIfStmts \n", musterIn);}.

xElseStmts = { musterOut :- musterIn;} .

new1xElseStmts = { xStmts:musterIn :=
                                     fprintf(ofile, "new1xElseStmts(", musterIn);
                    musterOut := fprintf(ofile, ")\n", xStmts:musterOut);}.

new2xElseStmts = { musterOut := fprintf(ofile, "new2xElseStmts(", musterIn);} .

xCaseStmts = { musterOut :- musterIn;} .

new1xCaseStmts = { No1:musterIn :=
                                  fprintf(ofile, "new1xCaseStmts(", musterIn);
                    No2:musterIn := fprintf(ofile, ",", No1:musterOut);
                    musterOut := fprintf(ofile, ")\n", No2:musterOut);}.

new2xCaseStmts = { xCaseStmts:musterIn :=
                                         fprintf(ofile, "new2xCaseStmts(", musterIn);
                    musterOut := fprintf(ofile, ")\n", xCaseStmts:musterOut);}.

new3xCaseStmts = { xExprList:musterIn :=
                                        fprintf(ofile, "new3xCaseStmts(", musterIn);
                    xStmts:musterIn := fprintf(ofile, ",", xExprList:musterOut);
                    musterOut := fprintf(ofile, ")\n", xStmts:musterOut);}.

xLabel = { xId:musterIn := fprintf(ofile, "xLabel(", musterIn);
            musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

xLoops = { musterOut :- musterIn;} .

new6xLoops = { xStmts:musterIn := fprintf(ofile, "new6xLoops(", musterIn);
                musterOut := fprintf(ofile, ")\n", xStmts:musterOut);}.

new7xLoops = { xIterator:musterIn := fprintf(ofile, "new7xLoops(", musterIn);
               xStmts:musterIn := fprintf(ofile, ",", xIterator:musterOut);
               musterOut := fprintf(ofile, ")\n", xStmts:musterOut);}.

new8xLoops = { xExpr:musterIn := fprintf(ofile, "new8xLoops(", musterIn);
                xStmts:musterIn := fprintf(ofile, ",", xExpr:musterOut);
                musterOut := fprintf(ofile, ")\n", xStmts:musterOut);}.

new9xLoops = { xIterator:musterIn := fprintf(ofile, "new9xLoops(", musterIn);
                xStmts:musterIn := fprintf(ofile, ",", xIterator:musterOut);
```

```
              musterOut := fprintf(ofile, ")\n", xStmts:musterOut);}.

new10xLoops = { xStmts:musterIn := fprintf(ofile, "new10xLoops(", musterIn);
                xExpr:musterIn := fprintf(ofile, ",", xStmts:musterOut);
                musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

xTemplate = { musterOut :- musterIn;} .

new1xTemplate = { xExprFormal:musterIn :=
                                    fprintf(ofile, "new1xTemplate(", musterIn);
                  xTempCond:musterIn := fprintf(ofile, ",", xExprFormal:musterOut);
                  xStmts:musterIn := fprintf(ofile, ",", xTempCond:musterOut);
                  musterOut := fprintf(ofile, ")\n", xStmts:musterOut);}.

new2xTemplate = { xExprFormal:musterIn :=
                                    fprintf(ofile, "new2xTemplate(", musterIn);
                  xTempCond:musterIn := fprintf(ofile, ",", xExprFormal:musterOut);
                  musterOut := fprintf(ofile, ")\n", xTempCond:musterOut);}.

new3xTemplate = { No1:musterIn := fprintf(ofile, "new3xTemplate(", musterIn);
                  No2:musterIn := fprintf(ofile, ",", No1:musterOut);
                  musterOut := fprintf(ofile, ")\n", No2:musterOut);}.

xTempCond = { musterOut :- musterIn;} .

new1xTempCond = { xExpr:musterIn :=
                              fprintf(ofile, "new1xTempCond(", musterIn);
                  musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new2xTempCond = { musterOut := fprintf(ofile, "new2xTempCond \n", musterIn);}.

xExprFormal = { musterOut :- musterIn;} .

new1xExprFormal = { xExpr:musterIn :=
                                fprintf(ofile, "new1xExprFormal(", musterIn);
                    musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new2xExprFormal = { xFormal:musterIn :=
                                fprintf(ofile, "new2xExprFormal(", musterIn);
                    xInto:musterIn := fprintf(ofile, ",", xFormal:musterOut);
                    musterOut := fprintf(ofile, ")\n", xInto:musterOut);}.

new3xExprFormal = { musterOut := fprintf(ofile, "new3xExprFormal \n", musterIn);} .

new4xExprFormal = { No1:musterIn :=
                                  fprintf(ofile, "new4xExprFormal(", musterIn);
                    No2:musterIn := fprintf(ofile, ",", No1:musterOut);
```

```
                    musterOut := fprintf(ofile, ")\n", No2:musterOut);}.

xInto = { musterOut := fprintf(ofile, "xInto \n", musterIn);} .

new1xInto = { xExpr:musterIn := fprintf(ofile, "new1xInto(", musterIn);
              musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new2xInto = { musterOut := fprintf(ofile, "new2xInto \n", musterIn);} .

xFormal = { musterOut :- musterIn;} .

new1xFormal = { xLValue:musterIn := fprintf(ofile, "new1xFormal(", musterIn);
                musterOut := fprintf(ofile, ")\n", xLValue:musterOut);}.

new2xFormal = { musterOut := fprintf(ofile, "new2xFormal \n", musterIn);} .

xIterator = { musterOut :- musterIn;} .

new1xIterator = { xSimpleIts:musterIn :=
                                        fprintf(ofile, "new1xIterator(", musterIn);
                  xExpr:musterIn := fprintf(ofile, ",", xSimpleIts:musterOut);
                  musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new2xIterator = { xSimpleIts:musterIn :=
                                        fprintf(ofile, "new2xIterator(", musterIn);
                  musterOut := fprintf(ofile, ")\n", xSimpleIts:musterOut);}.

xSimpleIts = { musterOut :- musterIn;} .

new1xSimpleIts = { xSimpleIts:musterIn :=
                                         fprintf(ofile, "new1xSimpleIts(", musterIn);
                   xSimpleIt:musterIn :- xSimpleIts:musterOut;
                   musterOut := fprintf(ofile, ")\n", xSimpleIt:musterOut);}.

new2xSimpleIts = { xSimpleIt:musterIn :=
                                         fprintf(ofile, "new2xSimpleIts(", musterIn);
                   musterOut := fprintf(ofile, ")\n", xSimpleIt:musterOut);}.

xSimpleIt = { musterOut :- musterIn;} .

new1xSimpleIt = { xLValue:musterIn :=
                                  fprintf(ofile, "new1xSimpleIt(", musterIn);
                  xExpr:musterIn := xLValue:musterOut;
                  musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new2xSimpleIt = { xLValue:musterIn :=
                                  fprintf(ofile, "new2xSimpleIt(", musterIn);
```

```
                 xId:musterIn := fprintf(ofile, ",", xLValue:musterOut);
                 xMapSel:musterIn := fprintf(ofile, ",", xId:musterOut);
                 musterOut := fprintf(ofile, ")\n", xMapSel:musterOut);}.

xMapSel = { musterOut :- musterIn;} .

new1xMapSel = { xLValue:musterIn := fprintf(ofile, "new1xMapSel(", musterIn);
               musterOut := fprintf(ofile, ")\n", xLValue:musterOut);}.

new2xMapSel = { xLValue:musterIn := fprintf(ofile, "new2xMapSel(", musterIn);
               musterOut := fprintf(ofile, ")\n", xLValue:musterOut);}.

xLValue = { musterOut :- musterIn;} .

new2xLValue = { xLValue:musterIn := fprintf(ofile, "new2xLValue(", musterIn);
               musterOut := fprintf(ofile, ")\n", xLValue:musterOut);}.

new3xLValue = { No1:musterIn := fprintf(ofile, "new3xLValue(", musterIn);
               No2:musterIn := fprintf(ofile, ",", No1:musterOut);
               musterOut := fprintf(ofile, ")\n", No2:musterOut);}.

new4xLValue = { musterOut := fprintf(ofile, "new4xLValue \n", musterIn);} .

new5xLValue = { xQualId:musterIn := fprintf(ofile, "new5xLValue(", musterIn);
               musterOut := fprintf(ofile, ")\n", xQualId:musterOut);}.

new6xLValue = { xLValue:musterIn := fprintf(ofile, "new6xLValue(", musterIn);
               xSelector:musterIn := fprintf(ofile, ",", xLValue:musterOut);
               musterOut := fprintf(ofile, ")\n", xSelector:musterOut);}.

xSelector = { musterOut := fprintf(ofile, "xSelector \n", musterIn);} .

new1xSelector = { xExprList:musterIn :=
                                   fprintf(ofile, "new1xSelector(", musterIn);
                 musterOut := fprintf(ofile, ")\n", xExprList:musterOut);}.

new2xSelector = { xExprList:musterIn :=
                                    fprintf(ofile, "new2xSelector(", musterIn);
                 musterOut := fprintf(ofile, ")\n", xExprList:musterOut);}.

new3xSelector = { xExpr:musterIn :=
                            fprintf(ofile, "new3xSelector(", musterIn);
                 musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new4xSelector = { Expr1:musterIn :=
                                fprintf(ofile, "new4xSelector(", musterIn);
                 Expr2:musterIn := fprintf(ofile, ",", Expr1:musterOut);
```

```
                  musterOut := fprintf(ofile, ")\n", Expr2:musterOut);}.

xFormer = { musterOut :- musterIn;} .

new1xFormer = { xExpr:musterIn := fprintf(ofile, "new1xFormer(", musterIn);
                musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new2xFormer = { Expr1:musterIn := fprintf(ofile, "new2xFormer(", musterIn);
                Expr2:musterIn := fprintf(ofile, ",", Expr1:musterOut);
                musterOut := fprintf(ofile, ")\n", Expr2:musterOut);}.

new3xFormer = { xExpr:musterIn := fprintf(ofile, "new3xFormer(", musterIn);
                xIterator:musterIn := fprintf(ofile, ",", xExpr:musterOut);
                musterOut := fprintf(ofile, ")\n", xIterator:musterOut);}.

new4xFormer = { xExpr:musterIn := fprintf(ofile, "new4xFormer(", musterIn);
                xExprList:musterIn := fprintf(ofile, ",", xExpr:musterOut);
                musterOut := fprintf(ofile, ")\n", xExprList:musterOut);}.

new5xFormer = { Expr1:musterIn := fprintf(ofile, "new5xFormer(", musterIn);
                Expr2:musterIn := fprintf(ofile, ",", Expr1:musterOut);
                Expr3:musterIn := fprintf(ofile, ",", Expr2:musterOut);
                musterOut := fprintf(ofile, ")\n", Expr3:musterOut);}.

xExprList = { musterOut :- musterIn;} .

new1xExprList = { xExprList:musterIn :=
                                 fprintf(ofile, "new1xExprList(", musterIn);
                  xExpr:musterIn := fprintf(ofile, ",", xExprList:musterOut);
                  musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new2xExprList = { xExpr:musterIn :=
                              fprintf(ofile, "new2xExprList(", musterIn);
                  musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new3xExprList = { xExprList:musterIn :=
                              fprintf(ofile, "new3xExprList(", musterIn);
                  musterOut := fprintf(ofile, ")\n", xExprList:musterOut);}.

xInstantiate = { xExpr:musterIn := fprintf(ofile, "xInstantiate(", musterIn);
                 xInstExport:musterIn := fprintf(ofile, ",", xExpr:musterOut);
                 xInstImport:musterIn := fprintf(ofile, ",", xInstExport:musterOut);
                 musterOut := fprintf(ofile, ")\n", xInstImport:musterOut);} .

xInstExport     = { musterOut :- musterIn;}.

new1xInstExport = { xExprList:musterIn :=
```

```
                                        fprintf(ofile, "new1xInstExport(", musterIn);
                    musterOut := fprintf(ofile, ")\n", xExprList:musterOut);}.

new2xInstExport = { musterOut :=
                                        fprintf(ofile, "new2xInstExport\n", musterIn);} .

xInstImport = { musterOut :- musterIn;} .

new1xInstImport = { xIdList:musterIn :=
                                    fprintf(ofile, "new1xInstImport(", musterIn);
                    musterOut := fprintf(ofile, ")\n", xIdList:musterOut);}.

new2xInstImport = { musterOut :=
                                    fprintf(ofile, "new2xInstImport(", musterIn);} .

xQualId = { musterOut :- musterIn;} .

new1xQualId = { Id1:musterIn := fprintf(ofile, "new1xQualId(", musterIn);
                Id2:musterIn := fprintf(ofile, ",", Id1:musterOut);
                musterOut := fprintf(ofile, ")\n", Id2:musterOut);}.

new2xQualId = { xId:musterIn := fprintf(ofile, "new2xQualId(", musterIn);
                musterOut := fprintf(ofile, ")\n", xId:musterOut);}.

xLambda = { xParamList:musterIn := fprintf(ofile, "xLambda(", musterIn);
            xProgBody:musterIn := fprintf(ofile, ",", xParamList:musterOut);
            musterOut := fprintf(ofile, ")\n", xProgBody:musterOut);}.

xQuantifier = { xQualifier:musterIn :=
                                fprintf(ofile, "xQuantifier(", musterIn);
                xSimpleIts:musterIn := fprintf(ofile, ",", xQualifier:musterOut);
                xExpr:musterIn := fprintf(ofile, ",", xSimpleIts:musterOut);
                musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

xQualifier = { musterOut :- musterIn;} .

new1xQualifier = { musterOut :=
                                      fprintf(ofile, "new1xQualifier \n", musterIn);} .

new2xQualifier = { musterOut :=
                                    fprintf(ofile, "new2xQualifier \n", musterIn);} .

xElIfExprs = { musterOut :- musterIn;} .

new1xElIfExprs = { xElIfExprs:musterIn :=
                                      fprintf(ofile, "new1xElIfExprs(", musterIn);
                   xElIfExpr:musterIn := fprintf(ofile, ",", xElIfExprs:musterOut);
```

```
                   musterOut := fprintf(ofile, ")\n", xElIfExpr:musterOut);}.

new2xElIfExprs = { musterOut :=
                                  fprintf(ofile, "new2xElIfExprs \n", musterIn);} .

xElIfExpr = { Expr1:musterIn := fprintf(ofile, "xElIfExpr(", musterIn);
              Expr2:musterIn := fprintf(ofile, ",", Expr1:musterOut);
              musterOut := fprintf(ofile, ")\n", Expr2:musterOut);}.

xElseExpr = { musterOut :- musterIn;} .

new1xElseExpr = { xExpr:musterIn :=
                                  fprintf(ofile, "new1xElseExpr(", musterIn);
                  musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

new2xElseExpr = { musterOut := fprintf(ofile, "new2xElseExpr \n", musterIn);} .

xCaseExprs = { musterOut :- musterIn;} .

new1xCaseExprs = { No1:musterIn :=
                                        fprintf(ofile, "new1xCaseExprs(", musterIn);
                   No2:musterIn := fprintf(ofile, ",", No1:musterOut);
                   musterOut := fprintf(ofile, ")\n", No2:musterOut);}.

new2xCaseExprs = { xCaseExprs:musterIn :=
                                      fprintf(ofile, "new2xCaseExprs(", musterIn);
                   musterOut := fprintf(ofile, ")\n", xCaseExprs:musterOut);}.

new3xCaseExprs = { xExprList:musterIn :=
                                 fprintf(ofile, "new3xCaseExprs(", musterIn);
                   xExpr:musterIn := fprintf(ofile, ",", xExprList:musterOut);
                   musterOut := fprintf(ofile, ")\n", xExpr:musterOut);}.

xBinOp = { musterOut :- musterIn;} .

new1xBinOp = { musterOut := fprintf(ofile, "new1xBinOp \n", musterIn);} .

new2xBinOp = { musterOut := fprintf(ofile, "new2xBinOp \n", musterIn);} .

new3xBinOp = { musterOut := fprintf(ofile, "new3xBinOp \n", musterIn);} .

new4xBinOp = { musterOut := fprintf(ofile, "new4xBinOp \n", musterIn);} .

new5xBinOp = { musterOut := fprintf(ofile, "new5xBinOp \n", musterIn);} .

new6xBinOp = { musterOut := fprintf(ofile, "new6xBinOp \n", musterIn);} .
```

```
new7xBinOp = { musterOut := fprintf(ofile, "new7xBinOp \n", musterIn);} .

new8xBinOp = { musterOut := fprintf(ofile, "new8xBinOp \n", musterIn);} .

new9xBinOp = { musterOut := fprintf(ofile, "new9xBinOp \n", musterIn);} .

new10xBinOp = { musterOut := fprintf(ofile, "new10xBinOp \n", musterIn);} .

new11xBinOp = { musterOut := fprintf(ofile, "new11xBinOp \n", musterIn);} .

new12xBinOp = { musterOut := fprintf(ofile, "new12xBinOp \n", musterIn);} .

new13xBinOp = { musterOut := fprintf(ofile, "new13xBinOp \n", musterIn);} .

new14xBinOp = { musterOut := fprintf(ofile, "new14xBinOp \n", musterIn);} .

new15xBinOp = { musterOut := fprintf(ofile, "new15xBinOp \n", musterIn);} .

new16xBinOp = { musterOut := fprintf(ofile, "new16xBinOp \n", musterIn);} .

new17xBinOp = { musterOut := fprintf(ofile, "new17xBinOp \n", musterIn);} .

new18xBinOp = { musterOut := fprintf(ofile, "new18xBinOp \n", musterIn);} .

new19xBinOp = { musterOut := fprintf(ofile, "new19xBinOp \n", musterIn);} .

new20xBinOp = { musterOut := fprintf(ofile, "new20xBinOp \n", musterIn);} .

new21xBinOp = { musterOut := fprintf(ofile, "new21xBinOp \n", musterIn);} .

new22xBinOp = { musterOut := fprintf(ofile, "new22xBinOp \n", musterIn);} .

new23xBinOp = { musterOut := fprintf(ofile, "new23xBinOp \n", musterIn);} .

new24xBinOp = { musterOut := fprintf(ofile, "new24xBinOp \n", musterIn);} .

new25xBinOp = { musterOut := fprintf(ofile, "new25xBinOp \n", musterIn);} .

new26xBinOp = { xQualId:musterIn := fprintf(ofile, "new26xBinOp(", musterIn);
                musterOut := fprintf(ofile, ")\n", xQualId:musterOut);}.

new27xBinOp = { musterOut := fprintf(ofile, "new27xBinOp \n", musterIn);} .

new28xBinOp = { musterOut := fprintf(ofile, "new28xBinOp \n", musterIn);} .

new29xBinOp = { musterOut := fprintf(ofile, "new29xBinOp \n", musterIn);} .
```

```
new30xBinOp = { xQualId:musterIn := fprintf(ofile, "new30xBinOp(", musterIn);
                musterOut := fprintf(ofile, ")\n", xQualId:musterOut);}.

new31xBinOp = { musterOut := fprintf(ofile, "new31xBinOp \n", musterIn);} .

new32xBinOp = { musterOut := fprintf(ofile, "new32xBinOp \n", musterIn);} .

new33xBinOp = { musterOut := fprintf(ofile, "new33xBinOp \n", musterIn);} .

new34xBinOp = { musterOut := fprintf(ofile, "new34xBinOp \n", musterIn);} .

new35xBinOp = { musterOut := fprintf(ofile, "new35xBinOp \n", musterIn);} .

new36xBinOp = { musterOut := fprintf(ofile, "new36xBinOp \n", musterIn);} .

new37xBinOp = { musterOut := fprintf(ofile, "new37xBinOp \n", musterIn);} .

new38xBinOp = { musterOut := fprintf(ofile, "new38xBinOp \n", musterIn);} .

new39xBinOp = { musterOut := fprintf(ofile, "new39xBinOp \n", musterIn);} .

new40xBinOp = { musterOut := fprintf(ofile, "new40xBinOp \n", musterIn);} .

new41xBinOp = { musterOut := fprintf(ofile, "new41xBinOp \n", musterIn);} .

new42xBinOp = { musterOut := fprintf(ofile, "new42xBinOp \n", musterIn);} .

new43xBinOp = { musterOut := fprintf(ofile, "new43xBinOp \n", musterIn);} .

new44xBinOp = { musterOut := fprintf(ofile, "new44xBinOp \n", musterIn);} .

new45xBinOp = { musterOut := fprintf(ofile, "new45xBinOp \n", musterIn);} .

new46xBinOp = { musterOut := fprintf(ofile, "new46xBinOp \n", musterIn);} .

new47xBinOp = { musterOut := fprintf(ofile, "new47xBinOp \n", musterIn);} .

xUnOp = { musterOut :- musterIn;} .

new1xUnOp = { musterOut := fprintf(ofile, "new1xUnOp \n", musterIn);} .

new2xUnOp = { musterOut := fprintf(ofile, "new2xUnOp \n", musterIn);} .

new3xUnOp = { musterOut := fprintf(ofile, "new3xUnOp \n", musterIn);} .

new4xUnOp = { musterOut := fprintf(ofile, "new4xUnOp \n", musterIn);} .
```

```
new5xUnOp = { musterOut := fprintf(ofile, "new5xUnOp \n", musterIn);} .

new6xUnOp = { musterOut := fprintf(ofile, "new6xUnOp \n", musterIn);} .

new7xUnOp = { musterOut := fprintf(ofile, "new7xUnOp \n", musterIn);} .

new8xUnOp = { musterOut := fprintf(ofile, "new8xUnOp \n", musterIn);} .

new9xUnOp = { musterOut := fprintf(ofile, "new9xUnOp \n", musterIn);} .

new10xUnOp = { musterOut := fprintf(ofile, "new10xUnOp \n", musterIn);} .

new11xUnOp = { musterOut := fprintf(ofile, "new11xUnOp \n", musterIn);} .

new12xUnOp = { musterOut := fprintf(ofile, "new12xUnOp \n", musterIn);} .

new13xUnOp = { musterOut := fprintf(ofile, "new13xUnOp \n", musterIn);} .

new14xUnOp = { xQualId:musterIn := fprintf(ofile, "new14xUnOp(", musterIn);
               musterOut := fprintf(ofile, ")\n", xQualId:musterOut);}.

new15xUnOp = { musterOut := fprintf(ofile, "new15xUnOp \n", musterIn);} .

new16xUnOp = { musterOut := fprintf(ofile, "new16xUnOp \n", musterIn);} .

new17xUnOp = { musterOut := fprintf(ofile, "new17xUnOp \n", musterIn);} .

new18xUnOp = { musterOut := fprintf(ofile, "new18xUnOp \n", musterIn);} .

new19xUnOp = { musterOut := fprintf(ofile, "new19xUnOp \n", musterIn);} .

new20xUnOp = { musterOut := fprintf(ofile, "new20xUnOp \n", musterIn);} .

new21xUnOp = { musterOut := fprintf(ofile, "new21xUnOp \n", musterIn);} .

new22xUnOp = { musterOut := fprintf(ofile, "new22xUnOp \n", musterIn);} .

new23xUnOp = { musterOut := fprintf(ofile, "new23xUnOp \n", musterIn);} .

new24xUnOp = { musterOut := fprintf(ofile, "new24xUnOp \n", musterIn);} .

new25xUnOp = { musterOut := fprintf(ofile, "new25xUnOp \n", musterIn);} .

new26xUnOp = { musterOut := fprintf(ofile, "new26xUnOp \n", musterIn);} .

new27xUnOp = { musterOut := fprintf(ofile, "new27xUnOp \n", musterIn);} .
```

```
new28xUnOp = { musterOut := fprintf(ofile, "new28xUnOp \n", musterIn);} .

new29xUnOp = { xQualId:musterIn := fprintf(ofile, "new29xUnOp(", musterIn);
               musterOut := fprintf(ofile, ")\n", xQualId:musterOut);}.

new30xUnOp = { musterOut := fprintf(ofile, "new30xUnOp \n", musterIn);} .

new31xUnOp = { musterOut := fprintf(ofile, "new31xUnOp \n", musterIn);} .

new32xUnOp = { musterOut := fprintf(ofile, "new32xUnOp \n", musterIn);} .

new33xUnOp = { musterOut := fprintf(ofile, "new33xUnOp \n", musterIn);} .

new34xUnOp = { musterOut := fprintf(ofile, "new34xUnOp \n", musterIn);} .

new35xUnOp = { musterOut := fprintf(ofile, "new35xUnOp \n", musterIn);} .

new36xUnOp = { musterOut := fprintf(ofile, "new36xUnOp \n", musterIn);} .

new37xUnOp = { musterOut := fprintf(ofile, "new37xUnOp \n", musterIn);} .

xTypeList = { musterOut :- musterIn;}.

new1xTypeList = { musterOut := fprintf(ofile, "new1xTypeList\n", musterIn);} .

new2xTypeList = { Id1:musterIn := fprintf(ofile, "new2xTypeList(", musterIn);
                  Id2:musterIn := fprintf(ofile, ",", Id1:musterOut);
                  musterOut := fprintf(ofile, ")\n", Id2:musterOut);}.

new3xTypeList = { xTypeList:musterIn := fprintf(ofile, "new3xTypeList(", musterIn);
                  Id1:musterIn := putmykey(",", pos1, xTypeList:musterOut);
                  Id2:musterIn := putmykey(":", pos2, Id1:musterOut);
                  musterOut := fprintf(ofile, ")\n", Id2:musterOut);}.

intlit = { musterOut := fprintf(ofile, "intlit(%d)\n",Integer, musterIn);} .

floatlit = { musterOut := fprintf(ofile, "floatlit(%f)\n",Float, musterIn);} .

str = { musterOut := { fprintf(ofile, "str(");
                       WriteString(ofile, Str);
                       musterOut = fprintf(ofile, ")\n");};} .
```

Nun müssen noch ein paar Dateien vervollständigt werden:

C.42.8 Die Vervollständigug der Datei `mydtypes.h`

⟨*mydtypes.h*⟩≡
```
#define MYDTYPES == true
#include "Idents.h"
#include "Positions.h"
```

⟨*mydtypesh*⟩

C.42.9 Die Vervollständigug der Datei `myset.c`

⟨*myset.c*⟩≡
```
#include "myset.h"
```

⟨*mysetc*⟩

C.42.10 Die Vervollständigug der Datei `myset.h`

⟨*myset.h*⟩≡
```
#include "Tree.h"
#include "global.h"
```
⟨*myseth*⟩

C.42.11 Die Vervollständigug der Datei `mytrans.c`

⟨*mytrans.c*⟩≡
```
#include "mytrans.h"
```

⟨*mytransc*⟩

C.42.12 Die Vervollständigug der Datei `mytrans.h`

⟨*mytrans.h*⟩≡
```
#include "Tree.h"
#include "Output.h"
#include "global.h"
```
⟨*mytransh*⟩

C.42.13 Die Vervollständigug der Datei myprint.c

⟨*myprint.c*⟩≡

```
#include <string.h>
#include "myprint.h"
#include "StringMem.h"
⟨myprintc⟩
```

C.42.14 Die Vervollständigug der Datei myprint.h

⟨*myprint.h*⟩≡

```
#include "Positions.h"
#include "Idents.h"
#include "global.h"

#if !defined(MYDTYPES)
#include "mydtypes.h"
#endif

⟨myprinth⟩
```

C.42.15 Die Vervollständigug der Datei myscope.c

⟨*myscope.c*⟩≡

```
#include "myscope.h"

⟨myscopec1⟩
⟨myscopec2⟩
⟨myscopec3⟩
```

C.42.16 Die Vervollständigug der Datei myscope.h

⟨*myscope.h*⟩≡

```
#include "Tree.h"

⟨myscopeh⟩
```

C.42.17 Die Vervollständigung der Datei `global.h`

⟨*global.h*⟩+≡

```
extern char *ifile;
extern FILE *ofile;
extern int linedir;
extern int linedirflag;
extern int TRANSFLAG;
```

C.42.18 Die Vervollständigung der Datei `trans.puma`

⟨*trans.puma*⟩≡

```
TRAFO transform
TREE Tree

PUBLIC
mygettree
getcode
CanIDelete
Pattern
Delete
Statement

IMPORT
{
#include "global.h"
#include "myscope.h"
#include "mytrans.h"
#include "myset.h"
}

⟨transpuma⟩
⟨transpuma1⟩
⟨transpuma3⟩
⟨transpuma2⟩
```