UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

LEHRSTUHL VIII
KÜNSTLICHE INTELLIGENZ

GRDT: Enhancing Model-Based Learning for Its Application in Robot Navigation

LS–8 Report 5

Volker Klingspor

Dortmund, January 1, 1994

Universität Dortmund
Fachbereich Informatik

University of Dortmund
Computer Science Department

Forschungsberichte des Lehrstuhls VIII (KI),
Fachbereich Informatik
der Universität Dortmund

ISSN 0943-4135

Anforderungen an:

Universität Dortmund
Fachbereich Informatik
Lehrstuhl VIII
D-44221 Dortmund

Research Reports of the unit no. VIII (AI)
Computer Science Department
of the University of Dortmund

ISSN 0943-4135

Requests to:

University of Dortmund
Fachbereich Informatik
Lehrstuhl VIII
D-44221 Dortmund

e-mail: reports@ls8.informatik.uni-dortmund.de
ftp: kimo.informatik.uni-dortmund.de:pub/Reports

GRDT: Enhancing Model-Based Learning for Its Application in Robot Navigation[1]

LS–8 Report 5

Volker Klingspor

Dortmund, January 1, 1994

Universität Dortmund
Fachbereich Informatik

[1]This work is partially funded by the European Community under the project B-Learn II (P7274) and the Ministry for Sciences and Research of the German federal state Nordrhein-Westfalen

Abstract

Robotics is one of the most challenging applications for the use of machine learning. Machine learning can offer an increase in flexibility and applicability in many robotic domains. In this paper, we sketch a framework to apply *inductive logic programming (ILP)* techniques to learning tasks of autonomous mobile robots. We point out differences between three existing algorithms used within this framework and their results. Since all of these algorithms have problems in solving the tasks, we developed GRDT *(grammar based rule discovery tool)*, an algorithm combining their ideas and techniques.

Contents

1 Introduction

In contrast to most of the robot systems actually used, future systems must have more user-friendly human-system interfaces, must be adaptable and portable to new, previously unknown environments, and must be able to handle uncertain and unknown events [Dillmann, 1993]. Presently, commands are most often given on the level of real coordinates or at best by names of objects of which the positions the system knows. Flexible and user-friendly systems demand for commands in terms of *concepts*. Then, the user is not called to specify concrete objects like a specific door in the command. Instead, he can tell the robot to perform an action with any instance of a concept, e.g., he can tell the robot to go to the next a priori unknown door and enter the room. To execute this kind of commands, the robot must be able to classify the objects it perceives with its sensors, i.e., to assign them to concepts. Additionally, the system must be able to perform actions with these objects like moving through the doorway of the door found. [Morik and Rieger, 1993] have shown that this requires perceptual features and action features to be integrated, and perceptual features to be action oriented and action features to be perception oriented. They developed a representation for these *operational concepts* based on first order logic.

To simplify the adaptation and the transmission of systems to new environments and new tasks, the rules to classify the perceptions should be learned by the system. In recent years, many algorithms for machine learning were developed, some of them able to learn in restricted *first order logic*. Nevertheless, *applying* these algorithms to real systems like autonomous mobile robots is a another challenging task.

In this paper, we will first present the representation of the perceptions of a mobile robot and a scenario used for learning, both developed by [Morik and Rieger, 1993]. In Section 3, we will describe three different learning algorithms of *inductive logic programming, ILP,* the research area of learning *Horn clause programs*. We will also describe the results we got from applying these learning algorithms to our learning tasks. In Section 4, the new algorithm GRDT combining several ideas of these algorithms is presented, able to solve the learning problems we introduced. The paper ends with a short conclusion in Section 5.

2 Representation and Scenario

Suppose we have a robot moving through a known environment and measuring distances to objects while moving, using ultrasonic sensors. In our experiments, we used data of traces through a simple room (Figure 1). Most often, the robot moves along or through a doorway[1]. Figure 2 shows the measurements the robot gets from a left side sensor of the robot, called sensor 5, while entering the room diagonally through the doorway. This path is indicated as thicker line in Figure 1. The x-axis displays the time points of the measurements, the y-axis shows the measured distances. In the first part of the trace, the sensor gets no echo because the left hand side of the robot is empty. Then, the robot senses two times the door frames and three times the side of the cupboard. During the next three measurements, again the robot gets no echo, because the front side of the cupboard does

[1] The data were prepared by the University of Karlsruhe.

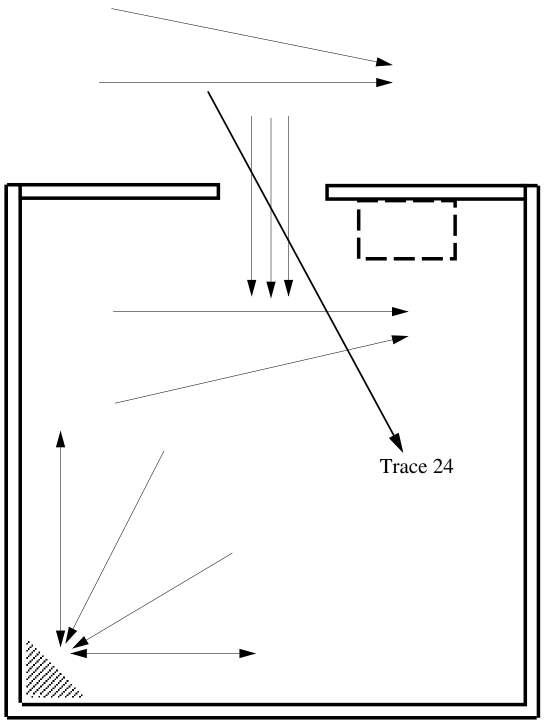

Figure 1: Room with traces

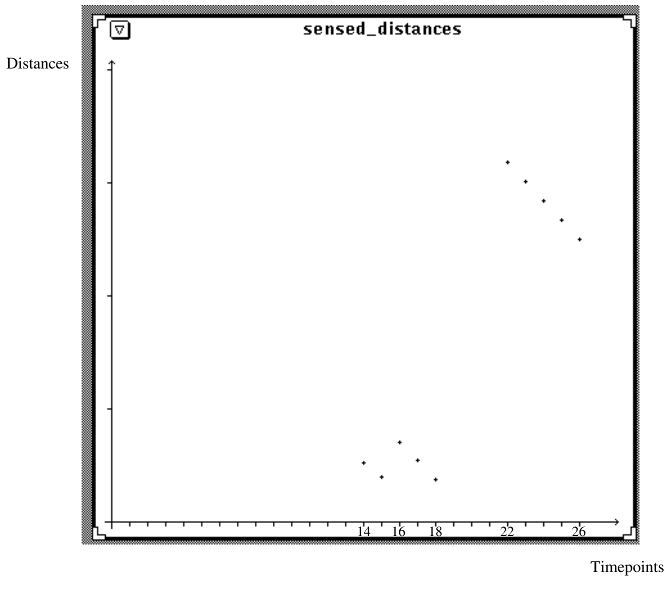

Figure 2: Distances measured by sensor 5 in trace 24

not reflect the sonar beam back to the robot. The last measurements represent the wall.

To be able to apply the sensed distances to more than this object, it is necessary to construct features from these measurements, i.e., we need a first step of abstraction. For this task, a sequence of measurements of a single sensor is divided into intervals of linear changes of measured distances. A symbol is attached to every interval dependent on the gradient of its function. The resulting sequence of these *basic features* for the example is:

```
no_measurement (trace24, Or, sensor5, 1, 14, _).
decreasing     (trace24, Or, sensor5, 14, 15, -22).
incr_peak      (trace24, Or, sensor5, 15, 16, 47).
decreasing     (trace24, Or, sensor5, 16, 18, -30).
no_measurement (trace24, Or, sensor5, 18, 22, _).
decreasing     (trace24, Or, sensor5, 22, 26, -30).
```

The arguments of these predicates are the trace number, the orientation of the sensor relative to the world coordinates, the sensor number, the start and the end time point of the sequence and a value equivalent to the gradient. For example, the last line represents a sequence of decreasing measurements, measured by sensor 5 in trace 24 from the 22th to the 26th time point. The chain of time points determines the need of first order logic as representation language[2].

Since we want to learn from classified examples, the user must classify the traces, e.g.: "in trace 24, the robot moved diagonally through the doorway". From this classification, from information about the environment, and from the measurements, examples at different representation levels are generated. The first level describes situations, in which a single sensor sensed a particular constellation of edges. For example, the fact **s_jump(trace24, sensor5, 14, 18, diagonal)** represents that sensor 5 sensed a jump in trace 24 from time point 14 to 18, i.e., two parallel walls got sensed in sequence, while the robot moved diagonally along this jump. At the next representation level, situations are gathered, in which different sensors belonging to the same group of sensors out of several given groups sensed the same single sensor features. E.g., the fact **sg_jump(trace24, left_side, 14, 18, diagonal)** expresses that the left side sensors sensed a jump, while the robot moved diagonally along the jump. At the highest representation level, we combine features of different sensor groups, e.g., the left side and the right side sensors, to describe the perceptual part of operational concepts.

Our aim is to use machine learning to find rules describing the concepts of each level in terms of the next lower level. First, we need rules describing single sensor features by patterns of basic features. Such a rule could be:

```
decreasing(Tr, _, Se, T1, T2, _) &
incr_peak(Tr, _, Se, T2, T3, _) &
decreasing(Tr, _, Se, T3, T4, _)
--> s_jump(Tr, Se, T1, T4, diagonal).
```

[2]Learning the semantic of basic features and how to compute them, is another learning task, that will be solved by the University Dortmund within the project BLEARN-II.

or

```
stable(Tr, _, Se, T1, T2, _) &
decr_peak(Tr, _, Se, T2, T3, _) &
stable(Tr, _, Se, T3, T4, _)
--> s_jump(Tr, Se, T1, T4, parallel).
```

The second kind of rules are sensor group feature descriptions. For these rules, we need additional knowledge about the class membership of the sensors in the particular traces. We try to learn rules combining single sensor features like in the following rule:

```
s_jump(Tr, Se1, T1, T2, diagonal) &
s_jump(Tr, Se2, T3, T4, diagonal) &
sclass(Tr, Se1, _, _, left_side) &
sclass(Tr, Se2, _, _, left_side) &
d1succ(T1, T3) & d1succ(T2, T4)
--> sg_jump(Tr, left_side, T1, T4, diagonal).
```

The first two premises express that two sensors perceived a jump while the robot moved diagonally along this jump. The next two premises guarantee that these two sensors belong to the sensor class left_side. The last two premises relate the start and end time points to be sure that the perceptions are chronologically close enough. These rules should classify the perceptions of the robot while it moves in new and unknown environments. Then, the robot is able to interpret commands of the conceptual level and to execute actions according to these commands. Also, it can monitor whether the actions are performed correctly, because it can compare the measurements it got with the expected perceptions and it can change the further planned actions, if it sensed unexpected values.

In this paper, two learning task are used to describe the behavior of the learning algorithms. In the first task, we try to learn descriptions for the single sensor feature s_jump. We calculated 2129 basic features from 17472 measurements in 28 traces. In these traces, 206 times a sensor perceived a jump. For the second task, we use the 206 occurrencies of s_jump to learn descriptions for sensor group features. Because of the large number of sensor classes we defined, we have 225 examples of the sensor group feature sg_jump.

3 Different ILP algorithms

Efficient machine learning in full first order logic is impossible because first order logic is undecidable. Most first order logic learners operate on more or less restricted Horn clause logic. The most frequent restriction used by these ILP-algorithms is to abstain from proper functions. There are additional syntactical restrictions like learning linked clauses only, i.e., clauses with all the arguments linked via the premises to the head arguments of the clause. The restriction used determines the complexity of the learning task and possibly the need of further semantical heuristics leading the search into areas of special interest and pruning parts that seem to be uninteresting.

I will now present three ILP-algorithms. The first one learns linked Horn clause programs using a semantical heuristic useful in many domains. The other two algorithms use different kinds of an explicit definition of the hypothesis space, so that it can be reduced. In the third algorithm, a complete search in the restricted hypothesis space will be performed.

3.1 Foil

FOIL [Quinlan, 1990] is presently one of the most acknowledged ILP-algorithms. It attracts attention because of applicability and good results in various domains without the need of additional knowledge, e.g., in form of syntactical structures as described in the next two sections.

3.1.1 General description of Foil

FOIL's hypothesis space is linked Horn clause programs. To handle this huge space, parts of it will be pruned semantically during learning by a heuristic. This heuristic leading search through the hypothesis space to find a short path to a good hypothesis, is the *information gain*, well-known from the propositional learner ID3 [Quinlan, 1986].

FOIL's algorithm can be described as follows:

- while positive examples are still uncovered by the previously learned rules
  - while negative examples are still covered by the recent hypothesis
    - build all specializations of the recent hypothesis by extending the clause by a linked literal
    - calculate the information gain for all these specializations
    - choose the specialization with the best information gain for the next iteration

The hypothesis space is searched depth-first, the choice of the next hypothesis only depends on the heuristic. Clearly, using a heuristic, it is possible that interesting parts of the hypothesis space are pruned erroneously. FOIL's heuristic evaluates a hypothesis only dependent on the newly added literal. This makes the system shortsighted, since it rejects the addition of literals of which the positive effect is seen only after more literals have been added. So FOIL cannot learn all rules of its hypothesis space: it is an incomplete algorithm.

3.1.2 Learning results in doorway domain

We applied FOIL to our two learning tasks. In the first case, learning patterns of basic features to describe single sensor features, FOIL learned a single rule:

no_measurement(Tr, _, Se, _T_k, T_1, _) &
stable(Tr, _, Se, T_1, T_2, _) &
incr_peak(Tr, _, Se, T_2, T_3, _) &
something_happened(Tr, _, Se, T_3, _T_n, _) &
straight_to(Tr, _, _Se_i, _T_i, _T_j, _)
--> s_jump(Tr, Se, T_1, T_3, M).[3]

This rule might separate the positive examples from the negative examples[4], but it is an unsatisfying rule for our application for two reasons. First, with this rule, the classification of s_jump not only depends on the measurements during the perception of the jump, but also on the measurements before and after the jump (premises 1 and 4). Second, the pattern of the specified sensor is combined with perceptions of an arbitrary other sensor possibly directed to another orientation. This constellation is only given in the environment we learn from, it cannot be expected in other environments.

FOIL was also applied to the second learning task: learning descriptions for sensor group features. The result shows that exclusively using of information gain is insufficient to lead the search. FOIL needs much time for learning, we cancelled the experiment after 26 hours with only 3 learned rules. Additionally, the learned rules cannot be used for our task. For instance, the first learned rule was:

d1succ(T_1, F) &
s_jump(Tr, _Se_1, T_1, T_2, Mov) &
sg_jump(Tr, _Sc_1, F, T_2, Mov) &
sclass(Tr, _Se_2, J, _K, _L) &
s_jump(_Tr_2, _Se_3, J, F, _O)
--> sg_jump(Tr, Sclass, T_1, T_2, Mov).

This rule relates the given situation with arbitrary sensors, traces and so on. The longer the rules become, the more d$i$succ-premises are added, relating many uninteresting time points of arbitrary sensor features.

Since FOIL, one of the best learning algorithms with heuristic search, is unable to learn the intended concept descriptions, we need algorithms restricting capable of the hypothesis space by syntactical structures to lead the learning into areas of expected rules.

3.2 GRENDEL

3.2.1 General description of GRENDEL

GRENDEL [Cohen, 1991, Cohen, 1992, Cohen, 1993] is a learning algorithm based on FOIL. Representation language, search strategy, and hypothesis evaluation are identical. The important difference is the way GRENDEL specializes hypotheses. This specialization can be determined by the user with a grammar explicitly defining the hypothesis space. This grammar is an extended context free grammar: instead of usual terminal and nonterminal symbols, terminal and nonterminal *literals* are used. In each specialization step, only

[3] Variables beginning with an underscore do not occur twice, they are anonymous.

[4] FOIL uses the closed world assumption, so examples not given are considered negative ones.

body(s_jump(Tr, Se, Beg, End, Mov))	→ *or(Mov), seq(Tr, Se, Beg, End).*
seq(Tr, Se, Beg, End)	→ *bf(Tr, Se, Beg, End).*
seq(Tr, Se, Beg, End)	→ *bf(Tr, Se, Beg, I), seq(Tr, Se, I, End).*
or(Mov)	→ m_parallel(Mov).
or(Mov)	→ m_diagonal(Mov).
or(Mov)	→ m_straight_to(Mov).
or(Mov)	→ m_straight_away(Mov).
bf(T, S, B, E)	→ stable(T, _, S, B, E, _).
bf(T, S, B, E)	→ increasing(T, _, S, B, E, _).
bf(T, S, B, E)	→ decreasing(T, _, S, B, E, _).
bf(T, S, B, E)	→ incr_peak(T, _, S, B, E, _).
bf(T, S, B, E)	→ decr_peak(T, _, S, B, E, _).
bf(T, S, B, E)	→ straight_to(T, _, S, B, E, _).
bf(T, S, B, E)	→ straight_away(T, _, S, B, E, _).
bf(T, S, B, E)	→ no_measurement(T, _, S, B, E, _).
bf(T, S, B, E)	→ no_movement(T, _, S, B, E, _).

Figure 3: GRENDEL's grammar to learn single sensor feature descriptions

those specializations will be generated and tested that can be derived by the grammar in one step. Then, all these new hypotheses will be evaluated. GRENDEL uses a heuristic similar to the information gain used by FOIL. The only difference is that this heuristic does not only depend on the part of the hypothesis generated up to now, i.e., on the terminal literals of the hypothesis. Instead, GRENDEL builds all further grammatical derivations of the hypothesis and calculates the information gain also based on the further covered and uncovered examples. So GRENDEL is more foresighted than FOIL.

3.2.2 Learning results in the doorway domain

We applied GRENDEL to our two learning tasks. We had to define grammars for both tasks. They are displayed in Figures 3 and 4 [5]. It was easy to create the grammar for this task. In short time, GRENDEL learned all patterns describing sensor features. The following rule is one example for the rules learned with GRENDEL:

diagonal(Mov) &
decreasing(Tr, Se, _, T_1, T_2, _) &
incr_peak(Tr, Se, _, T_2, T_3, _) &
decreasing(Tr, Se, _, T_3, T_4, _)
--> s_jump(Tr, Se, T_1, T_4, Mov).

But GRENDEL's grammars are very domain specific, because the predicate names occur in the grammar. For every change of a predicate name, the grammar must be changed.

[5]Terminal symbols are typed sans serif in this and all further figures, nonterminals *slanted*. Predicate and argument variables are written upper case, predicate names and constant arguments lower case. Learnable variables (RDT) are typed *italic*.

$$
\begin{array}{ll}
\textit{body(sg\_jump(Tr,SCl,}T_1\textit{,}T_2\textit{,M))} & \rightarrow \textit{sc(SCl), or(M),} \\
 & \quad \textit{sequence(SF, Tr, SCl, } T_1, T_2\textit{, M).} \\
 & \\
\textit{sequence(SF, Tr, SCl, } T_1, T_2\textit{, M)} & \rightarrow \textit{part1(SF, Tr, SCl, } T_1, T_2\textit{, M, -1, } Se_{new}\textit{),} \\
 & \quad \textit{seq1(SF, Tr, SCl, } T_1, T_2\textit{, M, } Se_{new}\textit{).} \\
\textit{sequence(SF, Tr, SCl, } T_1, T_2\textit{, M)} & \rightarrow \textit{seq2(SF, Tr, SCl, M, \_, } T_1, T_2\textit{).} \\
 & \\
\textit{seq1(SF, Tr, SCl, } T_1, T_2\textit{, M, Se)} & \rightarrow [\,] \\
\textit{seq1(SF, Tr, SCl, } T_1, T_2\textit{, M, Se)} & \rightarrow \textit{part1(SF, Tr, SCl, } T_1, T_2\textit{, M, Se, } Se_{new}\textit{),} \\
 & \quad \textit{seq1(SF, Tr, SCl, } T_1, T_2\textit{, M, } Se_{new}\textit{).} \\
 & \\
\textit{part1(SF, Tr, SCl, } T_1, T_2\textit{, M, Se, } Se_{new}\textit{)} & \rightarrow \textit{sf(SF, Tr, } Se_{new}, T_1, T_2\textit{, M),} \\
 & \quad \mathsf{sclass(Tr,\ } \mathsf{Se}_{new}\mathsf{,\ \_,\ \_,\ SCl),} \\
 & \quad \mathsf{Se} < \mathsf{Se}_{new}. \\
 & \\
\textit{seq2(SF, Tr, SCl, M, } T_{old}, T_i, T_{last}\textit{)} & \rightarrow \textit{part2(SF, Tr, SCl, M, } T_{old}, T_i, T_{last}\textit{),} \\
\textit{seq2(SF, Tr, SCl, M, } T_{old}, T_i, T_{last}\textit{)} & \rightarrow \textit{part2(SF, Tr, SCl, M, } T_{old}, T_i, T_{new}\textit{),} \\
 & \quad \textit{seq2(SF, Tr, SCl, M, } T_i, T_{new}, T_{last}\textit{).} \\
 & \\
\textit{part2(SF, Tr, SCl, M, } T_{old}, T_i, T_{new}\textit{)} & \rightarrow \textit{sf(SF, Tr, Se, } T_i, T_{new}\textit{, M),} \\
 & \quad \textit{diff(}T_{old}, T_i\textit{),} \\
 & \quad \mathsf{sclass(Tr,\ Se,\ \_,\ \_,\ SCl).} \\
 & \\
\textit{sf(SF, Tr, Se, } T_1, T_2\textit{, M)} & \rightarrow \mathsf{SFC} \quad \texttt{where}\ \text{sensor\_feature(SF),} \\
 & \qquad\qquad \text{SFC =.. [SF, Tr, Se, } \mathrm{T}_1, \mathrm{T}_2\text{, M].} \\
\textit{sc(SCl)} & \rightarrow \mathsf{SCC} \quad \texttt{where}\ \text{sensor\_class(SCl),} \\
 & \qquad\qquad \text{SCC =.. [SC, SCl].} \\
\textit{or(M)} & \rightarrow \mathsf{ORC} \quad \texttt{where}\ \text{movement(M),} \\
 & \qquad\qquad \text{ORC =.. [M, Or].} \\
 & \\
\textit{diff(}T_1, T_2\textit{)} & \rightarrow \mathsf{d1succ(T_1,T_2).} \\
\textit{diff(}T_1, T_2\textit{)} & \rightarrow \mathsf{d2succ(T_1,T_2).} \\
\textit{diff(}T_1, T_2\textit{)} & \rightarrow \mathsf{d3succ(T_1,T_2).} \\
\textit{diff(}T_1, T_2\textit{)} & \rightarrow \mathsf{d4succ(T_1,T_2).}
\end{array}
$$

Figure 4: GRENDEL's grammar to learn sensor group feature descriptions

$BF(\mathsf{Tr}, \_, \mathsf{Se}, \mathsf{T}_1, \mathsf{T}_2, \_)$
$\rightarrow \mathsf{SF}(\mathsf{Tr}, \mathsf{Se}, \mathsf{T}_1, \mathsf{T}_2, \mathit{Movement})$

$BF(\mathsf{Tr}, \_, \mathsf{Se}, \mathsf{T}_1, \mathsf{T}_2, \_)$ &
$BF(\mathsf{Tr}, \_, \mathsf{Se}, \mathsf{T}_2, \mathsf{T}_3, \_)$
$\rightarrow \mathsf{SF}(\mathsf{Tr}, \mathsf{Se}, \mathsf{T}_1, \mathsf{T}_3, \mathit{Movement})$

$\vdots$

$BF(\mathsf{Tr}, \_, \mathsf{Se}, \mathsf{T}_1, \mathsf{T}_2, \_)$ &
$BF(\mathsf{Tr}, \_, \mathsf{Se}, \mathsf{T}_2, \mathsf{T}_3, \_)$ &
$\vdots$
$BF(\mathsf{Tr}, \_, \mathsf{Se}, \mathsf{T}_5, \mathsf{T}_6, \_)$ &
$BF(\mathsf{Tr}, \_, \mathsf{Se}, \mathsf{T}_6, \mathsf{T}_7, \_)$
$\rightarrow \mathsf{SF}(\mathsf{Tr}, \mathsf{Se}, \mathsf{T}_1, \mathsf{T}_7, \mathit{Movement})$

Figure 5: RDT's rule schemata to learn single sensor feature descriptions

Moreover, the name of the goal predicate must be specified in the grammar, so a grammar is applicable to only one goal concept.

In contrast to the good result of the first task, the result of learning sensor group feature descriptions was unsatisfactory. First, it was very complicated to create a correct grammar. Only *lazy macros* (the grammar rules with the keyword "where")[6] enabled us to specify that some predicate names must be the same in different steps of specialization. Second, GRENDEL needs much time for each hypothesis test, because it calculates the information gain dependent on further derivations of the grammar and the number of these derivations is very high. We cancelled the experiment after a long time without having learned any rule. But the generated hypotheses look promising.

3.3 RDT

The last learning algorithm to be presented, RDT [Kietz and Wrobel, 1992], is based on ideas of [Emde, 1987]. RDT differs significantly from the two previously described algorithms. The representation formalism of MOBAL, into which RDT is integrated, is more powerful than FOIL's and GRENDEL's, because negated literals are allowed to occur in the clauses. Additionally, RDT is able to distinguish positive, negative, unknown, and contradictory examples.

3.3.1 General description of RDT

To restrict the hypothesis space, RDT uses rule schemata (also called rule models) describing sets of learnable rules. These rule schemata are second order logic rules. Their predicate variables are successively instantiated during learning. After each instantiation,

[6] GRENDEL expands a lazy macro $\mathsf{A} \rightarrow \mathsf{B}$ `where` P by proving P and adding a grammar rule $\mathsf{A}\sigma \rightarrow \mathsf{B}\sigma$, where σ is the used substitution, for every possible proof [Cohen, 1993]. Since only a finite set of context free rules are added, the language remains context free.

$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathit{Movement})\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \_,\ \_,\ \mathit{Class})$
$\rightarrow \mathsf{sg\_pattern}(\mathsf{Trace},\ \mathit{Class},\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathit{Movement}).$

$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \mathsf{T}_{11},\ \mathsf{T}_{12},\ \mathit{Movement})\ \&$
$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_2,\ \mathsf{T}_{21},\ \mathsf{T}_{22},\ \mathit{Movement})\ \&$
$\mathit{succ}(\mathsf{T}_{11},\ \mathsf{T}_{21})\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \_,\ \_,\ \mathit{Class})\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_2,\ \_,\ \_,\ \mathit{Class})$
$\rightarrow \mathsf{sg\_pattern}(\mathsf{Trace},\ \mathit{Class},\ \mathsf{T}_{11},\ \mathsf{T}_{22},\ \mathit{Movement}).$

$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathit{Movement})\ \&$
$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_2,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathit{Movement})\ \&$
$\mathsf{Sensor}_1 < \mathsf{Sensor}_2\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \_,\ \_,\ \mathit{Class})\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_2,\ \_,\ \_,\ \mathit{Class})$
$\rightarrow \mathsf{sg\_pattern}(\mathsf{Trace},\ \mathit{Class},\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathit{Movement}).$

$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \mathsf{T}_{11},\ \mathsf{T}_{12},\ \mathit{Movement})\ \&$
$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_2,\ \mathsf{T}_{21},\ \mathsf{T}_{22},\ \mathit{Movement})\ \&$
$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_3,\ \mathsf{T}_{31},\ \mathsf{T}_{32},\ \mathit{Movement})\ \&$
$\mathit{succ}(\mathsf{T}_{11},\ \mathsf{T}_{21})\ \&\ \mathit{succ}(\mathsf{T}_{21},\ \mathsf{T}_{31})\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \_,\ \_,\ \mathit{Class})\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_2,\ \_,\ \_,\ \mathit{Class})\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_3,\ \_,\ \_,\ \mathit{Class})$
$\rightarrow \mathsf{sg\_pattern}(\mathsf{Trace},\ \mathit{Class},\ \mathsf{T}_{11},\ \mathsf{T}_{32},\ \mathit{Movement}).$

$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathit{Movement})\ \&$
$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_2,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathit{Movement})\ \&$
$\mathit{Pattern}(\mathsf{Trace},\ \mathsf{Sensor}_3,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathit{Movement})\ \&$
$\mathsf{Sensor}_1 < \mathsf{Sensor}_2\ \&\ \mathsf{Sensor}_2 < \mathsf{Sensor}_3\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_1,\ \_,\ \_,\ \mathit{Class})\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_2,\ \_,\ \_,\ \mathit{Class})\ \&$
$\mathsf{sclass}(\mathsf{Trace},\ \mathsf{Sensor}_3,\ \_,\ \_,\ \mathit{Class})$
$\rightarrow \mathsf{sg\_pattern}(\mathsf{Trace},\ \mathit{Class},\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathit{Movement}).$

Figure 6: Some of RDT's rule schemata to learn sensor group feature descriptions

s_jump(Trace, Sensor_1, T_{11}, T_{12}, parallel) &
s_jump(Trace, Sensor_2, T_{21}, T_{22}, parallel) &
d1succ(T_{11}, T_{21}) &
sclass(Trace, Sensor_1, _, _, left_side) &
sclass(Trace, Sensor_2, _, _, left_side)
→ sg_jump(Trace, left_side, T_{11}, T_{22}, parallel).

s_jump(Trace, Sensor_1, T_1, T_2, diagonal) &
s_jump(Trace, Sensor_2, T_1, T_2, diagonal) &
s_jump(Trace, Sensor_3, T_1, T_2, diagonal) &
$\text{Sensor}_1 < \text{Sensor}_2$ & $\text{Sensor}_2 < \text{Sensor}_3$ &
sclass(Trace, Sensor_1, _, _, right_side) &
sclass(Trace, Sensor_2, _, _, right_side) &
sclass(Trace, Sensor_3, _, _, right_side)
→ sg_jump(Trace, right_side, T_1, T_2, diagonal).

Figure 7: Sensor group feature descriptions learned by RDT

RDT evaluates the new possibly partially instantiated hypothesis. This evaluation is quite different from the way the other algorithms evaluate the hypotheses. It counts the number of covered positive and negative examples, the number of uncovered positive examples, the total number of examples, and how many additional facts can be derived by this hypothesis. The user can build an arbitrarily complex expression using these items, and hence decide in which case a hypothesis will be accepted.

In contrast to the algorithms previously described, RDT searches through the whole hypothesis space. Only if no further specialization of a hypothesis leads to an acceptable rule, this part of the hypothesis space will be pruned, since completeness is preserved in this case. This case occurs, for example, if the number of covered positive examples of the tested hypothesis is less than the number required to accept the hypothesis.

3.3.2 Learning results in the doorway domain

Since the structure of the expected rules can be specified and this structure is well known in our domain, RDT learns good rules in both applications. But we need a large number of rule schemata to solve the learning task. For the first task, we have one schema for each expected pattern length (see Figure 5). RDT orders the rule schemata by θ-subsumption [Plotkin, 1970, Plotkin, 1971]. This allows pruning, if a hypothesis is already too special to be accepted. But the used schemata are not comparable by θ-subsumption, none of two schemata is more general than the other. Therefore, exactly the same partial patterns were tested for each schema. Clearly, this behavior slows RDT down. So RDT needed about twenty hours to learn exactly the same rules as GRENDELdid.

In the second learning task, we have much more complicated rule schemata; Figure 6 shows some of them. Nevertheless, learning is very fast, because the breadth of the hypothesis space, i.e., the number of instantiable predicates, is small. Figure 7 shows some of the learned rules. Unfortunately, it is again necessary to enter rule schemata of different length, thus increasing the effort of entering and decreasing the clarity of the set of schemata.

4 GRDT: Learning with rule schemata defined by grammars

Testing and comparing the different learning methods yields the following results:

- The information gain heuristic does not guide learning to the desired hypotheses. Additional knowledge about the syntactical structure of the preferred rules is necessary.
- Rule schemata and grammars can be used to describe these structures, but rule schemata are unwieldy because of the static length of each schema. Grammars of GRENDEL are unwieldy because of the predicate names to be fixed in the grammar and the problem of demanding for the same predicate names in different iterations.
- The additional information gain heuristic of GRENDEL may be very costly.
- The choice of the confirmation criterion of RDT is much more flexible than the pruning algorithm used by GRENDEL.

Why not combining ideas of GRENDEL and RDT to get a more powerful learning algorithm for our domain? The result is an algorithm using grammars to describe rule schemata instead of defining rules immediately. These rule schemata will be created successively during learning and will be used by an RDT-like algorithm instantiating the predicate variable and testing the generated partial hypotheses.

4.1 Grammars defining sets of rule schemata

We will now introduce the use of grammars defining sets of rule schemata with a sequence of examples. In our first example rules should be learned where one object must be classified to two concepts to apply the learned rule. For this task one rule schema is necessary:

$$\mathsf{P(X)} \,\&\, \mathsf{Q(X)} \rightarrow \mathsf{Concl(X)}$$

RDT tries to find predicate names for the predicate variables Q and P satisfying the confirmation criterion. This rule schema can be described with a single context free grammar rule:

$$\mathit{Concl(X)} \rightarrow \mathsf{P(X)},\ \mathsf{Q(X)}.$$

In the case that the object X is not classified to a *fixed* number of concepts to derive the conclusion, rule schemata of different length are necessary:

$$\begin{array}{ll} \mathsf{P_1(X)} & \rightarrow \mathrm{Concl(X)} \\ \mathsf{P_1(X)} \,\&\, \mathsf{P_2(X)} & \rightarrow \mathrm{Concl(X)} \\ \mathsf{P_1(X)} \,\&\, \mathsf{P_2(X)} \,\&\, \mathsf{P_3(X)} & \rightarrow \mathrm{Concl(X)} \\ \vdots & \end{array}$$

The maximal necessary length of the rule schemata must be known a priori to learn all suggested rules. The following grammar defines an infinite set of rule schemata of the previous structure:

$$
\begin{aligned}
&\mathit{Concl}(X) \rightarrow \mathit{IT}(X).\\
&\mathit{IT}(X) \quad\ \rightarrow \mathsf{P}(\mathsf{X}).\\
&\mathit{IT}(X) \quad\ \rightarrow \mathsf{P}(\mathsf{X}), \mathit{IT}(X).
\end{aligned}
$$

Both previously described learning tasks can also be easily described with GRENDEL's grammars. The only difference is that GRENDEL requires fixing the predicate names to be used in the grammar. Instead of this, GRDT determines them in runtime during learning.

In the next learning task rules describing transitivity should be learned. Therefore, an arbitrary predicate occurs multiple times as premise of the rule:

$$
\begin{array}{ll}
\mathsf{P}(\mathsf{X}_1, \mathsf{X}_2) & \rightarrow \mathsf{Concl}(\mathsf{X}_1, \mathsf{X}_2).\\
\mathsf{P}(\mathsf{X}_1, \mathsf{X}_2) \;\&\; \mathsf{P}(\mathsf{X}_2, \mathsf{X}_3) & \rightarrow \mathsf{Concl}(\mathsf{X}_1, \mathsf{X}_3).\\
\mathsf{P}(\mathsf{X}_1, \mathsf{X}_2) \;\&\; \mathsf{P}(\mathsf{X}_2, \mathsf{X}_3) \;\&\; \mathsf{P}(\mathsf{X}_3, \mathsf{X}_4) & \rightarrow \mathsf{Concl}(\mathsf{X}_1, \mathsf{X}_4).\\
\vdots &
\end{array}
$$

These rules can be described by GRENDEL's grammars only for a fixed set of predicate names, for each one an own branch must be built. GRDT's grammars allows the user to use predicate variables as arguments of nonterminal literals. Then the predicate variables of the different iterations which are normally different by definition are unified. The following example shows a grammar defining the previously displayed rule schemata:

$$
\begin{aligned}
&\mathit{Concl}(X_s, X_e) \rightarrow \mathit{IT}(P, X_s, X_e).\\
&\mathit{IT}(P, X_p, X_e) \rightarrow \mathsf{P}(\mathsf{X}_p, \mathsf{X}_e).\\
&\mathit{IT}(P, X_p, X_e) \rightarrow \mathsf{P}(\mathsf{X}_p, \mathsf{X}_2), \mathsf{IT}(\mathsf{P}, \mathsf{X}_2, \mathsf{X}_e).
\end{aligned}
$$

4.2 The algorithm of GRDT

This section briefly describes the new algorithm GRDT. The algorithm starts with the goal predicate as conclusion and generates the first hypothesis depending on the grammar. It then searches the hypothesis space in depth-first order from general to special hypotheses. Every iteration consists of three steps:

1. Specialization of the hypothesis
2. Test of the hypothesis
3. Evaluation of the test

Specialization The specialization step consists of four alternatives:

- if the hypothesis contains an uninstantiated predicate variable, try to find an admissible instantiation of this variable; (backtrackable)
- else if the hypothesis contains a nonterminal symbol, try to expand the nonterminal; (backtrackable)

Concl(Tr, Se, T_1, T_2, M)	$\rightarrow$ *seq(Tr, Se, T_1, T_2, M).*
seq(Tr, Se, T_i, T_2, M)	$\rightarrow$ BF(Tr, _, Se, T_i, T_2, _).
seq(Tr, Se, T_i, T_2, M)	$\rightarrow$ BF(Tr, _, Se, T_i, T_j, _), *seq(Tr, Se, T_j, T_2, M).*

Figure 8: A GRDT grammar to learn single sensor feature descriptions

- else if the hypothesis contains a constant to be learned, try to find an admissible value for this constant; (backtrackable)
- else no further specialization is possible, start backtracking.

Hypothesis test The following items will be counted like RDT does:

- number of covered positive examples;
- number of faultyly covered negative examples;
- number of uncovered positive examples;
- number of previously unknown facts that can be derived from the hypothesis;
- total number of examples.

If not all the predicate variables, constants to learn, and nonterminal symbols of the hypothesis are instantiated, the hypothesis is called partial.

Evaluation of the test Depending on the hypothesis test and the parameter set, the hypothesis will be evaluated. The following cases may appear:

- if the hypothesis is too special (i.e., the pruning criterion is satisfied) stop specializing and start backtracking;
- else if the hypothesis is partial, continue specialization;
- else if the hypothesis is accepted (i.e., the confirmation criterion is satisfied) store the rule and start backtracking. Further specialization is not sensible because all more special rules are subsumed by the learned one;
- else the hypothesis is too general, but cannot be expanded; start backtracking.

Concl(Tr, SCl, T_1*,* T_2*, M)*	$\rightarrow$ *sf_comb(Tr, SCl,* T_1*,* T_2*, M).*
sf_comb(Tr, SCl, T_1*,* T_2*, M)*	$\rightarrow$ SF(Tr, Se, T_1, T_2, M), sclass(Tr, Se, _,_, SCl), *sf_comb1(SF, Tr, SCl,* T_1*,* T_2*, M, Se).*
sf_comb(Tr, SCl, T_1*,* T_2*, M)*	$\rightarrow$ SF(Tr, Se, T_1, T_i, M), sclass(Tr, Se, _,_, SCl), *sf_comb2(SF, Diff, Tr, SCl, M,* T_1*,* T_2*).*
sf_comb1(SF, Tr, SCl, T_1*,* T_2*, M,* Se_{old}*)*	$\rightarrow$ [ ].
sf_comb1(SF, Tr, SCl, T_1*,* T_2*, M,* Se_{old}*)*	$\rightarrow$ SF(Tr, Se, T_1, T_2, M), sclass(Tr, Se, _,_, SCl), Se_{old} < Se, *sf_comb1(SF, Tr, SCl,* T_1*,* T_2*, M, Se).*
sf_comb2(SF, Diff, Tr, SCl, M, T_{old}*,* T_{last}*)*	$\rightarrow$ SF(Tr, Se, T_1, T_{last}, M), sclass(Tr, Se, _,_, SCl), Diff(T_{old}, T_1).
sf_comb2(SF, Diff, Tr, SCl, M, T_{old}*,* T_{last}*)*	$\rightarrow$ SF(Tr, Se, T_1, T_2, M), sclass(Tr, Se, _,_, SCl), Diff(T_{old}, T_1), *sf_comb2(SF, Diff, Tr, SCl, M,* T_1*,* T_{last}*).*

Figure 9: A GRDT grammar to learn sensor group feature descriptions

4.3 Learning results

We tested GRDT with our two learning tasks. The used grammars are displayed in Figures 8 and 9. First we will describe the way the grammar to learn single sensor features (Fig. 8) will be expanded and the predicate variables instantiated. The algorithm starts with the first grammar rule:

Concl(Tr, Se, T_1*,* T_2*, M)* $\rightarrow$ *seq(Tr, Se,* T_1*,* T_2*, M).*

Since *seq* is a nonterminal, the first specialization is the expansion of *seq*, using the second rule. This results in the rule schema:

BF(Tr, _, Se, T_1, T_2, _) $\rightarrow$ Concl(Tr, Se, T_1, T_2, M).

This rule schema contains the predicate variable BF, for which an acceptable instantiation must be found. In our domain, this may be one of the basic features, e.g., increasing.

After this instantiation, the rule schema cannot be further specialized, because no further nonterminal, no predicate variable and no constant to be learned exists. So the algorithms starts backtracking using the third rule. Applying this rule results in the rule schema:

BF(Tr, _, Se, T_1, T_2, _) & *seq(Tr, Se,* T_2*,* T_3*, M)* $\rightarrow$ Concl(Tr, Se, T_1, T_e, M).

Once again, GRDT tries to instantiate the predicate variable BF. If an acceptable predicate could be found (increasing in our example), the hypothesis will be further specialized, because the nonterminal *seq* can be expanded. The following two rule schemata will be generated in the next steps:

increasing(Tr, _, Se, T_1, T_i, _) &
BF(Tr, _, Se, T_2, T_3, _)
→ Concl(Tr, Se, T_1, T_3, M).
increasing(Tr, _, Se, T_1, T_2, _) &
BF(Tr, _, Se, T_2, T_3, _) &
seq(Tr, Se, T_3, T_e, *M)*
→ Concl(Tr, Se, T_1, T_e, M).

This specialization step iterates until a hypothesis is found that is acceptable or to special.

For testing GRDT we used a confirmation criterion accepting a hypothesis if it covers at least one positive example. RDT, GRDT and GRENDEL[7] learned exactly the same rules, only the time they needed, displayed in Table 1, differs. FOIL cannot be compared with the other algorithms. Because of its huge effort of memory, we had to restrict the learning to only one of the 28 traces to get a result at all.

The grammar used for the second learning task (Fig. 9) must be explained, too. It consists of two branches, *sf_comb1* and *sf_comb2*, alternatively generated by the nonterminal *sf_comb*. SF and DIFF are predicate variables, which should be equal in each iteration step. To reach this goal, they are passed to the next iteration as an argument of a nonterminal literal. sclass is a fixed domain predicate which have not to be learned. For the argument variables M and SCL constants must be learned. This grammar allows to learn two kinds of rules. First, cases are covered where the different sensors of a class sense the same feature at the same time, e.g.:

s_line(Tr, Se_1, T_1, T_2, parallel) & sclass(Tr, Se_1, _, _, front_right_corner) &
s_line(Tr, Se_2, T_1, T_2, parallel) & sclass(Tr, Se_2, _, _, front_right_corner) &
$Se_1 < Se_2$
--> sg_line(Tr, front_right_corner, T_1, T_2, parallel).

The second kind of rules covers cases, where the different sensors sense the features with a fixed delay, e.g.

s_jump(Tr, Se_1, T_1, T_2, diagonal) & sclass(Tr, Se_1, _, _, left_side) &
s_jump(Tr, Se_2, T_3, T_4, diagonal) & sclass(Tr, Se_2, _, _, left_side) &
s_jump(Tr, Se_2, T_5, T_6, diagonal) & sclass(Tr, Se_3, _, _, left_side) &
d2succ(T_1, T_3) & d2succ(T_3, T_5) &
d2succ(T_2, T_4) & d2succ(T_4, T_6)
--> sg_jump(Tr, left_side, T_1, T_6, diagonal).

Again, we compared the learning results of the algorithms and again we used a confirmation criterion accepting hypotheses, if they cover one positive example for RDT. GRENDEL is applied without use of the noise handlings parts, also accepting hypothesis covering one example. FOIL learned rules senseless for our domain and GRENDEL does not terminate within two days. The time used by the algorithms for solving this task is also displayed in Table 1.

[7] GRENDEL was applied without use of its abilities to handle noise

	GRENDEL	RDT	GRDT	# Examples	# Facts
s_line	35 min	8 h 42 min	1 h 42 min	718	2847
s_jump	24 min	7 h 10 min	1 h 37 min	206	2335
sg_line	> 2 days	3 h 9 min	1 h 13 min	567	3488
sg_jump	> 2 days	30 min	9 min	225	2534

Table 1: Time used for learning

	FOIL	GRENDEL	RDT	GRDT
repr. formalism	function free Horn clause logic	see FOIL	function free Horn clause logic (neg. literals allowed)	see RDT
hypoth. space restrictions	linked clauses	antecedent descr. grammars	rule schemata	grammars defining rule schemata
search control	top-down depth first controlled by heuristic	top-down depth first contr. by grammar and heuristic	top-down breadth first contr. by rule schemata, compl.	top-down depth first contr. by grammar, complete
learning task 1	insufficient result	good result, fast	good result, slow	good result, fast, simple grammar
learning task 2	insufficient result	no result, complex grammar	good result, fast, large set of rule schemata	good result, fast, grammar simpler than GRENDEL's

Table 2: Comparison of the different learning algorithms

5 Conclusion

In this paper, we have presented a learning scenario in the domain of navigation of autonomous mobile robots. We have compared the characteristics of three learning algorithms, FOIL, GRENDEL and RDT (see also Table 2). Each of these algorithms lacks some properties useful for our tasks. So we combined ideas of the algorithms and built a new one, GRDT. We tested GRDT in our doorway domain and indicated that it is able to learn efficiently in large domains, if the user can determine the syntactical structure of the expected rules by a grammar. Thus, the algorithm can be applied to real domains like learning descriptions for operational concepts.

The advantages of GRDT in contrast to GRENDEL are that the syntactical structure can be defined more easily and the confirmation criterion be chosen more flexibly. Nevertheless, there are problems left.

The specialization step, presented in Section 4.2, does not guarantee that the new hypothesis is really more special than the one before. Having grammars defining derivations without any specialization of the derived rules, the algorithm not necessarily terminates. If it terminates, it fulfills a complete search through the hypothesis space defined by the grammar. This behavior of GRDT is derived from the completeness of RDT.

Another problem concerns the way how to exclude multiple search of the same part of the hypothesis space, if different derivations of the grammar yield the same hypotheses.

References

[Cohen, 1991] Cohen, W. W. (1991). Grammatically biased learning: Learning horn theories using an explicit antecedent description language. Technical report, AT & T Bell Laboratories. (submitted to AIJ).

[Cohen, 1992] Cohen, W. W. (1992). The Grendel learning system. Technical report, AT & T Bell Laboratories.

[Cohen, 1993] Cohen, W. W. (1993). Rapid prototyping of ILP systems using explicit bias. Submitted to the 1993 IJCAI workshop on ILP.

[Dillmann, 1993] Dillmann, R. (1993). Entwicklungstendenzen und Anwendungen symbolischer Lernverfahren in der Robotik. In Herzog, O., Christaller, T., and Schütt, D., editors, *Grundlagen und Anwendungen der Künstlichen Intelligenz – 17. Fachtagung für KI*, Informatik aktuell, pages 28 – 43, Berlin. Springer-Verlag.

[Emde, 1987] Emde, W. (1987). Non-cumulative learning in METAXA.3. In *Proceedings of IJCAI-87*, pages 208–210, Los Altos, CA. Morgan Kaufman. An extended version appeared as KIT-Report 56, Techn. Univ. Berlin.

[Giordana and Saitta, 1990] Giordana, A. and Saitta, L. (1990). Abstraction – a general framework for learning. In *Procs. AAAI–Workshop on Automatic Generation of Approximations and Abstractions*, pages 245 – 256.

[Kaiser et al., 1993a] Kaiser, M., Klingspor, V., Millán, J., Moneta, C., Morik, K., and Rieger, A. (1993a). B-Learn II: Deliverable 401. Behavioural Learning II, ESPRIT Project 7274, University of Karlsruhe.

[Kaiser et al., 1993b] Kaiser, M., Klingspor, V., Millán, J., Moneta, C., Morik, K., and Rieger, A. (1993b). B-Learn II: Deliverable 402. Behavioural Learning II, ESPRIT Project 7274, University of Karlsruhe.

[Kietz, 1993] Kietz, J.-U. (1993). Some lower bounds for the computational complexity of inductive logic programming. In Brazdil, P., editor, *Proceedings of the European Conference on Machine Learning*, pages 115 – 123. Springer-Verlag.

[Kietz and Wrobel, 1992] Kietz, J.-U. and Wrobel, S. (1992). Controlling the complexity of learning in logic through syntactic and task-oriented models. In Muggleton, S., editor, *Inductive Logic Programming*, chapter 16, pages 335 – 360. Academic Press, London. Also available as Arbeitspapiere der GMD No. 503, 1991.

[Knieriemen, 1991] Knieriemen, T. (1991). *Autonome Mobile Roboter – Sensordateninterpretation und Weltmodellierung zur Navigation in unbekannter Umgebung.* Reihe Informatik. BI-Wissenschaftsverlag, Mannheim.

[Morik and Rieger, 1993] Morik, K. and Rieger, A. (1993). Learning action-oriented perceptual features for robot navigation. In Giordana, A., editor, *Workshop notes: Learning Robots of the ECML-93.* Also available as Research Report 3, Univ. Dortmund, Informatik VIII, D-44221 Dortmund.

[Morik et al., 1993] Morik, K., Wrobel, S., Kietz, J.-U., and Emde, W. (1993). *Knowledge Acquisition and Machine Learning – Theory, Methods, and Applications.* Academic Press, London.

[Plotkin, 1970] Plotkin, G. D. (1970). A note on inductive generalisation. *Machine Intelligence*, 5:153 – 163.

[Plotkin, 1971] Plotkin, G. D. (1971). A further note on inductive generalisation. *Machine Intelligence*, 6:101 – 124.

[Quinlan, 1986] Quinlan, R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Quinlan, 1990] Quinlan, R. (1990). Learning logical definitions from relations. *Machine Learning*, 5:239 – 266.

[Shapiro, 1981] Shapiro, E. Y. (1981). An algorithm that infers theories from facts. In *Proceedings of IJCAI-81*, pages 446 – 451, Los Altos, CA. Kaufmann.