, , 1?? ()


# Learning Concepts from Sensor Data of a Mobile Robot

VOLKER KLINGSPOR klingspor@ls8.informatik.uni-dortmund.de
KATHARINA J. MORIK morik@ls8.informatik.uni-dortmund.de
ANKE D. RIEGER rieger@ls8.informatik.uni-dortmund.de
*Univ. Dortmund, Computer Science Dept. LS VIII, 44221 Dortmund, Germany*

**Abstract.** Machine learning can be a most valuable tool for improving the flexibility and efficiency of robot applications. Many approaches to applying machine learning to robotics are known. Some approaches enhance the robot's high-level processing, the planning capabilities. Other approaches enhance the low-level processing, the control of basic actions. In contrast, the approach presented in this paper uses machine learning for enhancing the link between the low-level representations of sensing and action and the high-level representation of planning. The aim is to facilitate the communication between the robot and the human user. A hierarchy of concepts is learned from route records of a mobile robot. Perception and action are combined at every level, i.e., the concepts are perceptually anchored. The relational learning algorithm GRDT has been developed which completely searches in a hypothesis space, that is restricted by rule schemata, which the user defines in terms of grammars.

**Keywords:** inductive logic programming, robot navigation, combining sensing and action, perceptually anchored concepts

## 1. Introduction

Robotics offers a variety of learning objectives. We mention some of the well-known applications of machine learning to robotics and contrast them to our approach, before we present our scenario for learning. In order to enhance a robot's high-level processing, planning can be tailored to particular robot tasks (Segre, 1988). Lessons learned from failures of robot actions can be used to enhance the planning capabilities of a robot (Bennett, 1989), (Zercher, 1992). Learning is used to link the environment model with the perception when executing a plan (DeJong & Bennett, 1993), (Gil, 1994). The plan or the environment model is refined to accommodate practical experience with the plan. These approaches require an almost complete description of the robot's actions and its environment. It is a time consuming task to build up such a description. Moreover, if the robot's task or environment is changing, the description has to be changed accordingly. This makes the application of robots *inflexible*. In our approach, machine learning helps to construct the description of a robot's actions and its environment.

At lower levels of a robot's processing, machine learning can be applied in order to improve its performance (Kaelbling & Rosenschein, 1990), (Millán & Torras, 1992), (Mitchell & Thrun, 1993), (Baroglio *et al.*, 1994). These approaches assume a fully automated mode of robot operation and learning where no human interaction with the robot or the learning component is intended. Therefore, the representation for learning resembles the robot's low-level representation and is *not easily understandable* to human users. It has been argued, however, that human-machine interaction is necessary in order to fully exploit the opportunities of a robot (Nilsson, 1984), (Badler *et al.*, 1991). In order to facilitate access to a robot for human users, the low-level representation has to be translated into an understandable form. We address robot applications in which the robot is not completely autonomous but is interacting with human users. In our approach, machine learning is applied to bridge the gap between low-level and high-level representations.

Biologically inspired work in robotics has developed artificial beings that adapt to their environment (Brooks, 1991), (Steels, 1993). This type of learning is restricted to reflex-like behavior. Higher levels of cognition such as reasoning and concept formation are excluded. In contrast, we are interested in the link between perception and concepts. This is the problem of *symbol grounding* (Harnad, 1990), (Wrobel, 1991). In our approach, machine learning characterizes concepts on the basis of perceptual and action features, i.e., the learned concepts are perceptually anchored.

Up to now, if a robot has to navigate in office rooms, the coordinates of the walls and doors, desks and cupboards are entered into the robot as a map – for each new set of rooms anew. However, on an abstract level, offices are looking more or less the same wherever they are. Moving in an office environment can be described by concepts such as:

move through the doorway, turn left, move until the cupboard, and stop.

These concepts combine sensing and action. They do not describe a door independent of the robot's movement, but a move through a doorway. Similarly, not the cupboard is the concept here, but approaching a cupboard. The turn implies walls to be detected. However, not the walls are the relevant concepts, but the turn with respect to the walls. We have called such concepts *operational concepts*.

Whereas the specific measurements differ in different offices (e.g., doors have different width and depth), the relations between measurements acquired by sensors during the performance of a movement (e.g., moving through a doorway) remain the same. Moving through a doorway means to first perceive the doorposts with the front sensors, then by the sensors at both sides, and then with the rear sensors. Roughly spoken, there is first a time interval in which the front sensors measure decreasing distances, then a time interval in which the sensors at both sides measure a sudden decrease of distances, and finally a time interval, during which the rear sensors measure increasing distances. The relations between sequential sensor measurements, that characterize approaching or passing a doorpost (e.g., a continuous or sudden decrease of distances) remain the same, independent of the particular distance measurements. The particular time intervals depend on the size of the door, the speed of the robot, and the frequency with which sensor measurements are delivered. However, the relations between the time intervals (*first* decreasing distance ahead, *then* ..., *then* increasing distance at the back) are independent of the particular size of the door. Also the relations between the sensor groups (*in* *front* is opposite to *in the rear, left* is opposite to *right*) are the same for all moves through a doorway, although the particular side, which is in front when approaching the door, may differ from one move through a doorway to another. Hence, the concepts can be characterized by relations between time intervals, relations between sensor measurements, and relations between sensor groups.

It is our overall learning task to characterize a concept (e.g., moving through a doorway) and distinguish it from other ones (e.g., moving along a wall) on the basis of sensor measurements and robot movements. As our learning goals are concepts that are described in terms of relations, we need a learning method that is capable of handling relational hypotheses. All learning techniques within the framework of *inductive logic programming* (Muggleton, 1992) learn relational concepts. Whereas inductive logic programming has been successfully applied to relatively small training sets, sets of sensor measurements and robot actions become very large, even if only one concept is to be learned. The application of inductive logic programming to sensor data of a mobile robot demands a method of how to cope with such huge data sets. We developed three answers to this demand. First, we did not solve the learning problem in one step, but have split the overall learning task into several learning steps where each learning step uses the results of the previous one. This decreases the size of the training set for each learning step. The lay-out of learning steps is described in Section 2. Second, we designed the representation language (i.e., the particular predicates that are used to describe a concept and those that denote a concept) such that the points in time or time intervals become *deterministic terms* for the rules to be learned. This reduces the complexity of the learning task, as *determinate rules* are easier to learn than non-determinate ones (Džeroski *et al.*, 1992). The maximal arity of predicates, as well as the maximal depth of terms, is restricted. In addition, an upper bound on the length of rules helps to make the learning task tractable. The representation language is described in Section 3. Third, we applied a learning algorithm which exploits a declarative bias: the hypothesis space can be restricted explicitly. The learning algorithm GRDT is described in Section 4. The concept of moving through a doorway and all concepts that are necessary to characterize this concept were learned from sensor data acquired during movements of a mobile robot (the training set). The learning results were evaluated by applying the learned concepts to sensor data acquired during other paths of the mobile robot (the test set). The experiments are described in Section 5.

## 2. Learning Scenario

The overall learning task of acquiring concepts from sensor data and robot actions can be modeled as learning from examples. Sensor measurements accumulated during robot routes in known environments and classified by a simulation component are the input to learning. We used the data gathered by a mobile robot which moved in a variety of simple rooms. The rooms were completely known and their description was entered into a simulation component. Also the path of the

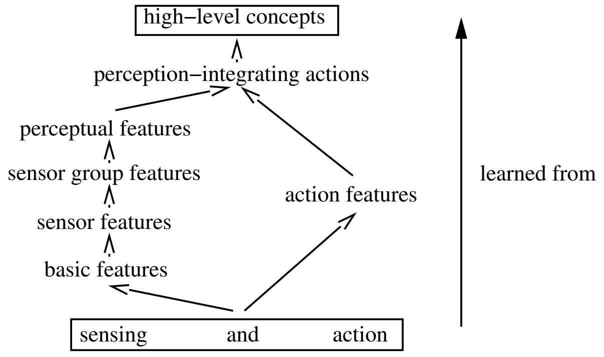


*Figure 1.* Learning steps

robot was entered. The sensor measurements were then classified by the simulation component, indicating the edge of an object that has most likely been sensed. Of course, in addition to the errors of the sensors misclassifications occur, too. Hence, the training sets always include noisy data.

Concepts, that describe a movement as well as a pattern of sensor data to be sensed during the execution of that movement are the output of learning. The learning results are applied to other route records, the test set. Then we compare the classification of the simulation component with the classification derived by learned rules. This evaluation shows the applicability of machine learning to defining higher-level descriptions of actions, perceptions, and the environment. Moreover, as both, training set and test set, are made of real-world data, it shows how well the learning algorithm can cope with noise and missing measurements. The next step is a real-world test of the learning results. This, however, requires a lot of work that does not concern machine learning (see Section 6 for our ongoing and future work).

Learning high-level concepts directly from sensor data and robot movements is not likely to produce good results. Therefore, we have designed a representation language with intermediate concepts. Five levels of abstraction bridge the gap between the raw data of the robot and the high-level concepts (see Figure 1). Each concept is defined in terms of concepts of the abstraction level below. The learning task is to find these concept definitions.

Now, we can describe the learning tasks. There are several learning steps, each corresponding to a particular level of abstraction. The first step, the learning of basic features from sensor data, is different from all consecutive steps. This learning task corresponds to the problem of signal-to-symbol conversion. At the lowest level, the input is numerical: the robot's movements and sensor measurements are represented by real numbers. The first learning task is then to adjust the calculation of qualitative basic features from the numerical data.

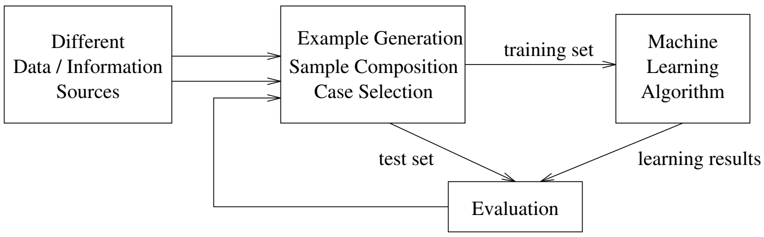


*Figure 2.* The setting for concept learning.

**Learning basic features**

**Given:** a sequence of sensor measurements, a set of parameters, and a set of functions for calculating basic features;

**Output:** the appropriate parameters for calculating basic features and the resulting sequence of basic features.

All following learning steps correspond to the same learning task. The output of one learning step forms the basis for the next one.

**Learning concepts**

**Given:** classified instances of a concept and background knowledge;

**Output:** rules that define a concept in terms of concepts of the next lower level.

A simulation component denotes for every sensor at every time point the edge, that was sensed. This information gives the classification for sensor and sensor group features. Perceptual features are classified by hand. The user of the learning system is supported by a human-computer interface which graphically shows the path of the robot so that the time interval in which sensors should sense, e.g., in_front_of_door, can easily be seen.

The general setting for solving the learning tasks is illustrated in Figure 2. For each learning step, training sets have to be set up. Given the representation, used at each level of the abstraction hierarchy in Figure 1, the respective descriptions have to be derived from the real-world data. Different information sources, such as data files and programs, are used to retrieve and generate predicates, representing classified instances of a concept and background knowledge. This time-consuming example and background knowledge *generation* process is supported and partially automated by a data preparation tool (Rieger, 1995a).

Given a set of positive and negative examples, the user has to decide, which examples are to be included in a training/test set. This decision concerns, e.g., the percentage of positive and negative examples, the construction of several training sets to be used for *cross validation*, and the specification of distributions, according to which examples for a training set are to be drawn. Besides these statistical criteria, the user also makes decisions concerning features of the data, e.g., the data of certain traces may be used for training, data of other traces for testing. Our tool allows the user to specify these decisions for *sample composition* in a uniform framework. With this specification and the given source set of examples, the tool performs the composition automatically.

A training set contains a set of *target predicates*, describing the examples of the concepts to be learned, and a set of *defining predicates*, describing the background knowledge. Target predicates may appear in the conclusion of a learned rule, defining predicates in its premise. Real-word domains, such as robotics, are characterized by an enormous amount of data. In order to apply learning successfully, it is crucial to provide the learning algorithms only with relevant background knowledge. This prevents the algorithm from being overloaded with irrelevant information, which slows down the learning process, and which may even cause the algorithm to find results of poor quality. We provide a method, *case selection*, which allows the user to specify, which defining predicates with which features should be included in the background knowledge for a given set of target predicates. The selection of the relevant defining predicates is then done automatically, yielding *cases*, which associate each target predicate instance with its relevant defining predicate instances.

Thus, the use of our tool makes the data preparation phase more transparent, revealing more clearly its effect on the learning results, and making it amenable to analysis.

## 3. Representation Hierarchy

High-level concepts integrate perceptual features and action features where the features can be regarded as lower-level concepts. They are anchored in basic features that are learned from sensing and action.

For our scenario, we have chosen the high-level concepts:

move_closer_to_wall, rotate_in_front_of_wall, move_parallel_in_corner, move_along_door, move_through_door, move_in_front_of_door, and rotate_in_front_of_door.

Rules are learned, whose conclusions represent these concepts. The premise of each rule describes the pre-condition for executing a concept, the sensing-acting loop for verifying the concept, and the resulting state (Figure 3).

standing(Trace, T1, T2, *in_front_of_door*, PDirect, small_side, *PrevP*) &
parallel_moving(Trace, T2, T3, MSpeed, PDirect, *through_door*, right_and_left) &
standing(Trace, T3, T4, *in_front_of_door*, rear, small_side, *through_door*)
→ move_through_door(Trace, T1, T4).

*Figure 3.* A high-level concept.[1]

The rule describes the high-level concept of moving through a doorway. It combines the recognition of the doorway with the action of moving through it on a parallel track. Moving diagonally through a doorway is a different concept. Action and perception are linked by the perceptual features (written in italics) occurring as arguments in the action predicates.

The first premise of the rule states, that the robot is standing in a time interval from T1 to T2 of a particular action sequence (Trace) and senses the perceptual feature in_front_of_door with the sensors of a small side after having perceived the perceptual feature *PrevP* prior to T1. This premise denotes the precondition for moving parallelly through a doorway.

The second premise states, that in the following time interval the action parallel_moving is executed, which is associated with the robot measuring the perceptual feature through_door with the sensors at its right left side. Note, that the particular values for the time intervals do not matter for the concept but are instantiated by the specifics of a particular path. Only the time relation is important. This makes the concept independent of the particular depth of a doorway and the robot's particular rate of advance.

The third premise describes the end of the movement through the doorway: the robot is standing in front of the door, now sensing with its rear sensors mounted at its small side the feature in_front_of_door after having perceived the feature through_door. PDirect, the orientation of perception (e.g., left, right, front, rear) is not fixed by the rule. It is only relevant that the orientation should not change during the parallel move and that, with respect to this orientation, the doorway is sensed in the rear after the movement. That means, whatever the orientation was at the beginning of the action, it is defined as being front, so that after completing the action the doorway is sensed by the rear sensors.

The predicates used in the concept are *perception-integrating action features*. This is the level below high-level concepts in Figure 1. The predicates are standing, moving, parallel_moving, diagonal_moving, rotating, straight_towards, and straight_away. The parallel_moving-predicate (Figure 4), e.g., states that the robot moved in direction MDirec starting at time T1 and it perceived Perc throughout the duration of that move and kept moving until the perception changes at time T2.

move(Tr, T1, T2, Speed, MDirec) &
period_of_time_perception(Tr, T1, T2, Perc, PDirect, PSide, parallel)
→ parallel_moving(Tr, T1, T2, Speed, MDirec, Perc, PDirect).

*Figure 4.* A perception-integrating action feature.

Some arguments of a high-level concept refer to *perceptual features*. These are defined by a hierarchy of rules. We may trace the feature through_door from its use as an argument in the concept move_through_door to its basic perceptual features.

A set of learned rules determines the meaning of the perceptual feature through_door (see Figure 5).[2] Other predicates are: along_door, in_front_of_door, along_wall, in_front_of_wall, and parallel_in_corner. The second rule accounts for the situation that the door post is perceived by the left hand side sensors a little later than by the right hand side ones. This is expressed by the successor relation between T1 and T3, the starting times of the sensor group feature sg_jump.

sg_jump(Tr, right_side, T1, T2, parallel) &
sg_jump(Tr, left_side, T1, T2, parallel) &
Start $\leq$ T1 & T2 $\leq$ End
$\rightarrow$ through_door (Tr, Start, End, parallel).

sg_jump(Tr, right_side, T1, T2, diagonal) &
sg_jump(Tr, left_side, T3, T4, diagonal) &
succ(T1, T3) & Start $\leq$ T1 & T2 $\leq$ End
$\rightarrow$ through_door (Tr, Start, End, Move).

*Figure 5.* A perceptual feature.

*Sensor group features* and *sensor features* describe configurations of edges as they are perceived by (a group of) sensors during a certain movement. Where, at higher levels of the representation hierarchy, concepts are denoted (e.g., wall, door, corner), here we refer to edges. Configurations of edges are jump, line, concave, convex.

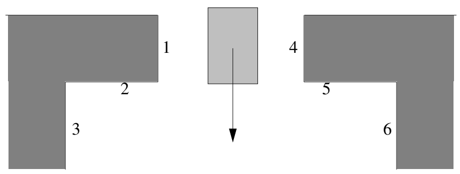


*Figure 6.* Edge configurations.

In Figure 6, the edges 1 and 3 as well as edges 4 and 6 are in the relation jump; edges 1 and 2 as well as 4 and 5 are in the relation convex.

If several sensors at the same side or corner of the robot perceive the same edge configuration, the feature predicate has the prefix sg_, denoting a sensor group feature. In Figure 7, in order for the sg_jump feature to hold, three of five sensors at the right side of the robot must perceive the jump configuration when moving parallelly along the edges.

s_jump(Tr, S1, T1, T2, parallel) & sclass(Tr, S1, T0, T7, right_side) &
s_jump(Tr, S2, T3, T4, parallel) & sclass(Tr, S2, T0, T7, right_side) &
s_jump(Tr, S3, T5, T6, parallel) & sclass(Tr, S3, T0, T7, right_side) &
succ(T1, T3) & succ(T3, T5) & T0 $\leq$ T1 & T6 $\leq$ T7
$\rightarrow$ sg_jump(Tr, right_side, T1, T6, parallel).

*Figure 7.* A sensor group feature.

The sensor feature s_jump for a single sensor is defined in terms of basic perceptual features. From sensor measurements we conclude edge groupings.

```
stable(Tr, Orien, S, T1, T2, Grad1) &
incr_peak(Tr, Orien, S, T2, T3, Grad2) &
stable(Tr, Orien, S, T3, T4, Grad3)
→ s_jump(Tr, S, T1, T4, parallel).
```

*Figure 8.* A single sensor feature.

The rule in Figure 8 states that a sensor S with a particular orientation Orien measures a sequence of distance values, that have during the time interval from T1 to T2 the gradient Grad1, from T2 to T3 the gradient Grad2, and from T3 to T4 the gradient Grad3. The end point of one time interval is the beginning of the next. An interval with a gradient approximately equal to zero is labeled stable. Grad2 is a rapid increase of measured values denoted by the basic feature incr_peak. stable and incr_peak are basic features that are incrementally calculated from the incoming sensor data (see Section 4).

```
measurement (Trace, Time, Sensor, Distance, Sx, Sy, SAngle, Object, Edge).
measurement (t35, 45, s2, 4.96, 4.84, 3.43, 353.54, 0, 5).
measurement (t35, 46, s2, 4.88, 4.92, 3.37, 353.54, 0, 5).
decreasing (t35, 353, s2, 45, 78, -38).
```

*Figure 9.* Measurements and a corresponding basic feature

Note, that the perceptual features are action-oriented. The type of movement with respect to the sensed object (e.g., parallel) is carried along the various levels of perceptual features. In this way, action and perception are integrated not only at the highest level, but also at all lower levels. Except for the basic actions, no feature refers to action or perception alone. *Basic actions* describe motions of the robot during a specific time interval. Possible predicates for actions are move, rotate, and stand, further specified by speed and direction.

The representation formalism is a restricted Horn logic. This formalism allows an elegant handling of time, time intervals, and relations between time intervals. The relational representation of time intervals makes concept definitions applicable to a variety of actual measurements. The definition of move_through_door applies to doorways of varying breadth and depth. No precise distances need to be fixed by the definition. Similarly, relations between sensors of a sensor group can easily be modeled. The fact that one sensor after the other at one side perceives a corner while the robot is moving along that corner is represented by relations between time instants independent of the particular time distance (see Figure 7).

In addition, the representation formalism allows us to exploit unification. The orientation of perception need not be fixed by the various rules but becomes instantiated when applying a rule (e.g., argument PDirect in Figure 3). The rules explicitly state, that the direction may not change or that another direction must be opposite with respect to the first direction. These abstract relations enable a compact and general representation.

## 4. Learning Methods

### 4.1. Learning Basic Features

The calculation of basic features (Wessel, 1995) constitutes the basis for learning. We approach the signal-to-symbol problem by converting numerical sonar sensor measurements to qualitative assertions over time intervals. Basic features are to represent the general tendency of change of the measurements of a sensor during a time interval.

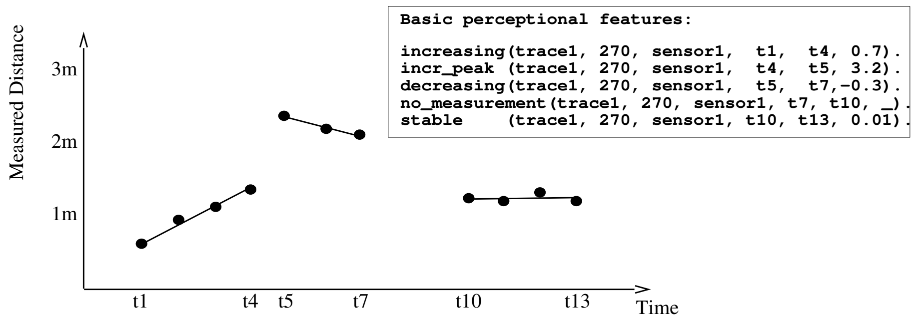


*Figure 10.* Sensor measurements and corresponding basic features.

In Figure 10, we see an example sequence of sensor measurements. The measurements of the time intervals from t1 to t4, from t5 to t7, and from t10 to t13 are grouped together, and labeled in accordance with the tendency of change as increasing, decreasing, and (more or less) stable, respectively. During the time interval from t7 to t10 the sensor does not get any echo. This situation is labeled as no_measurement. Between the consecutive time instants t4 and t5, the sensor measurements differ extremely. This situation is labeled as incr_peak. It can be characterized more precisely by considering the *gradient* of the measurements at two consecutive instants of time. It is defined as the quotient, whose numerator is the difference between the measurements at times t+1 and t, and whose denominator is the distance covered by the robot between t and t+1. In the case of incr_peak and decr_peak the gradient is greater than 1 and smaller than -1, respectively. Then, we also know, that two different edges have been perceived at the two instants of time. Therefore, it makes sense to split the time intervals at t4, instead of trying to include t5 in the increasing interval. If the robot does not move during a time interval, we associate the label no_movement with it. If the gradient is approximately equal to -1 or 1, we label the time interval as straight_to and straight_away, respectively.

The calculation of basic features is based on the *incremental* approximation of sensor measurements by (linear) functions. We have not used standard statistical approximation methods, because they don't satisfy our requirement of being applicable in an incremental way during the performance phase [3]. We provide the method with the linear functions, which are used for approximation. As the noisy real-world data cannot be fit exactly to them, we established several parameters, $P_1, \ldots, P_n$, which control the approximation. Each parameter $P_i, 1 \leq i \leq n$ has an associated set of values, $Dom(P_1), \ldots, Dom(P_n)$. We define a *parameter setting* $PS$ as a tuple $(p_1, \ldots, p_n)$, where each $p_i, 1 \leq i \leq n$ represents the value for parameter $P_i$, i.e., $p_i \in Dom(P_i)$. The parameters, which we consider at the moment, are:

1. *Interval size*, which specifies the maximal number of time instants to be included in a time interval;
2. *Representative gradient* of a time interval. It is either the first or the average gradient of the time interval. It is included as 6th argument in the predicates for basic features (see the examples in Figure 10).
3. *Tolerance* $\Delta$, which is used to compare gradients. A given gradient $gr_i$ is considered approximately equal to the representative gradient $gr$ of a time interval, if $|gr - gr_i| \leq \Delta$.
4. *Alternating sequences*, which are either taken into account or not. If they are not taken into account, an interval is closed, as soon as the current gradient has a different sign than the representative gradient and the tolerance is exceeded.
5. The set of *labels* for time intervals, which constitute the names of the basic feature predicates.

As not all possible parameter value combinations make sense, the set of parameter settings is a subset of the Cartesian product $Dom(P_1) \times \ldots \times Dom(P_n)$. Given the functions (calculating the gradient, calculating the average) and the set of parameter settings, the task is to find the setting, that calculate those basic features, leading to successful learning of concepts.

In the following we present the algorithm for calculating basic features, using a given parameter setting and incrementally reading in one sensor measurement after the other:

1. Initialization: Take the first two measurements, and calculate the gradient.
2. **If**, according to the parameter setting, the new gradient is approximately the same as the previous one
   **then** include the new time point in the current time interval and adapt the representative gradient as determined by the parameters;
   **else** close the previous interval at the previous time point and start a new one at the new time point with the new representative gradient being set to the new gradient. Label a closed interval with a symbol, depending on the representative gradient attached to the interval.
3. **If** there are further measurements **then** go to step 2.

We can define a partial order on the parameter settings with respect to the level of detail of the basic perceptional features resulting from the calculation with these parameter settings: A setting $PS_1$ is smaller or equal than a parameter setting $PS_2$, $PS_1 \leq PS_2$, if using $PS_1$ produces more and smaller time intervals than using $PS_2$. The goal of learning is to find the parameter setting, whose application results in basic features with the appropriate level of detail. In this context "appropriate" means, a sequence of time intervals of basic features corresponds to the time interval of a concept. The learning algorithm can be described by the following steps:

1. Initialization: Choose a member of the set of parameter settings.
2. Calculate the basic features based on this parameter setting.
3. Evaluate the parameter set by comparing the time intervals for basic features and those used by the classification of training instances.
4. **If** the evaluation yields that the parameters produce too few or too many time intervals, **then** choose a parameter setting which is smaller or greater according to the order defined above and go to step 2.

Ultimately, the quality of basic features is determinated by success or failure of concept learning. It is impossible to learn good concept descriptions, if the time intervals determined by the basic features do not match the time intervals used in the classification of examples (see Section 5.1).

### 4.2. Learning Concepts

The theoretical background for the task of learning concepts is *inductive logic programming* (ILP), whose aim is the induction of Horn clauses from examples of the concepts and background knowledge. Given background knowledge $B$ (here: ground facts), positive and negative examples $E = E^+ \cup E^-$ (again ground facts), and a hypothesis space $H$ (Horn clauses), the examples have to be consistent with the background knowledge (i.e., $B, E \not\models \Box$) and must not be a consequence of the background knowledge (i.e., $B \not\models E$). In ILP, two different goals for learning are considered. The first one is to find a small set of characteristic rules covering exactly the positive examples and avoiding redundancies (Plotkin, 1971). These algorithms often throw away already covered examples to focus on remaining ones. This setting is called the *definite semantics* of ILP[4]. In contrast, the goal of the *nonmonotonic semantics* is to find all rules covering some positive and no negative examples (Helft, 1989)[5]. Since the rules may cover examples multiply, the set of rules is redundant with respect to the given examples. However, the rules are not logically redundant. Whereas the latter learner usually learns more general descriptions, learning in the definite setting is easier and hence faster, because covered examples must not be further taken into account. Our learning task is to learn many logically non-redundant rules.

The goal of learning as many rules as possible is an important issue in our application. Even on an abstract level there are several ways to move through a doorway or to turn left. Therefore, the concept cannot be captured by a single rule. The variety of of concept definitions corresponds to the variety moves through a doorway.

For each of the learning steps, we have to prepare examples $E$ and the background knowledge $B$. If, for example, we want to learn discriminative rules for deriving the sensor feature s_jump from basic features, $E^+$ consists of the set of ground facts for the target predicate s_jump and $E^-$ consists of all other time intervals for which s_jump is not true. For this learning step, $B$ consists of the defining predicates, describing relevant basic features, i.e., those features, which refer to the same Trace, Sensor, and time interval. Given the example s_jump(tr10,90,s1,10,18,parallel), the data preparation tool will generate the case with this target predicate instance and the relevant defining predicate instances stable(tr10,90,s1,10,14,_), incr_peak(tr10,-90,s1,14,16,_), and stable(tr10,90,s1,16,18,_). In an analogous way cases can be constructed for the other learning steps, yielding the trainings sets for the ILP-algorithms.

### 4.3. GRDT

In principal, most of the ILP-algorithms can deal with our learning tasks. Since the hypothesis space for learning in Horn logic is huge — even if we abstain from proper functions — the different algorithms have different ways to restrict the search.

Semantically pruning the search restricts the hypothesis space dependent on the given data, leading the search into areas of special interest and pruning parts that seem to be unnecessary to characterize the goal concept. The *information gain heuristic*, used by FOIL (Quinlan, 1990), is an example for a semantic restriction.

Syntactic bias, particularly language bias, is used by many different systems to restrict the search space (e.g., (Silverstein & Pazzani, 1991), (De Raedt, 1992), (Bergadano & Gunetti, 1993), (Tausend, 1993)). For all of them, the user must be able to provide the structure of the expected rules to generate the search space description. This step depends heavily on the chosen representation language, the signature. Here, we consider two algorithms using language bias, GRENDEL (Cohen, 1994) and RDT[6] (Kietz & Wrobel, 1992). GRENDEL uses *antecedent description grammars*, a variant of context free grammars, to describe the hypothesis space. Instead of usual terminal and nonterminal symbols, terminal and nonterminal literals are used. The terminals of the grammar are the predicates of the domain. During learning, GRENDEL specializes the actual hypothesis by extending one of its nonterminals. Then it tests the new hypothesis in a similar, but more foresighted way than FOIL.

RDT uses another way to describe the hypothesis space, namely *rule schemata*. These schemata are second order rules, describing sets of learnable rules of the same structure. Their predicate variables are successively instantiated during learning. After each instantiation, RDT evaluates the possibly partially instantiated hypothesis. Whereas both, FOIL and GRENDEL, learn concepts according to the definite ILP semantic, RDT finds *all* regularities concerning the goal concept. Only if no further specialization of a hypothesis can lead to an acceptable rule, this part of the hypothesis space is pruned in a completeness preserving way.

In this section, we present a *grammar based rule discovery tool*, GRDT, combining ideas of GRENDEL and RDT in order to handle syntactic bias more flexible and more easily. Like GRENDEL, GRDT uses grammars to describe the hypothesis space. But instead of describing the learnable rules themselves, the grammar defines a set of rule schemata whose predicate variables are instantiated during learning. Like RDT, GRDT is a *nonmonotonic* learner.

### *4.3.1. Grammars defining sets of rule schemata – the hypothesis space*

We will now introduce the use of grammars for defining sets of rule schemata which establish the hypothesis space. In a first example, rules are to be learned where an object must have two specific properties to derive the goal concept. For this task the following rule schema can be used[7]:

$\mathbf{P}$ (X) & $\mathbf{Q}$ (X) $\rightarrow$ $\mathbf{Concl}$ (X).

RDT tries to find predicate names for the predicate variables $\mathbf{Q}$ and $\mathbf{P}$ such that the resulting rule satisfies the acceptance criterion, e.g., the simple rule parent(X) & female(X) $\rightarrow$ mother(X). This rule schema can be described with a single GRDT grammar rule:

$\mathbf{Concl}$ (X) $\rightsquigarrow$ $\mathbf{P}$ (X), $\mathbf{Q}$ (X).

Whereas both, RDT and GRDT, determine the predicate symbols to be used during runtime, GRENDEL needs the predicate names occurring in the grammar:

body (mother (X)) $\hookrightarrow$ *predicate* (X), *predicate* (X).
*predicate* (X) $\hookrightarrow$ parent (X).
*predicate* (X) $\hookrightarrow$ female (X).
*predicate* (X) $\hookrightarrow$ male (X).
⋮

This is a disadvantage for the user, who, dependent on the domain, may have to specify numerous predicates.

If the number of necessary properties of an object X should not be fixed, rule schemata of different length are necessary:

$\mathbf{P}_1$ (X) $\rightarrow$ $\mathbf{Concl}$ (X).
$\mathbf{P}_1$ (X) & $\mathbf{P}_2$ (X) $\rightarrow$ $\mathbf{Concl}$ (X).
$\mathbf{P}_1$ (X) & $\mathbf{P}_2$ (X) & $\mathbf{P}_3$ (X) $\rightarrow$ $\mathbf{Concl}$ (X).
⋮

The maximal necessary length of the rule schemata must be known a priori to learn all expected rules. RDT orders the different rule schemata according to an extended $\theta$-subsumption ((Kietz & Wrobel, 1992), (Plotkin, 1970)). If a specific instantiation of a general rule schema (e.g., grandparent (X) $\rightarrow$ mother (X)) covers too few positive examples, i.e., this hypothesis is too special, every more special rule schema (e.g. grandparent (X) & $\mathbf{P}_2$ (X) $\rightarrow$ mother (X)) will be pruned, because it cannot cover more positive examples than the more general hypotheses do. While the ordering of sets of hypotheses and its use for pruning is considered an advantage, the specification of premises for all lengths is a disadvantage for the user of the learning tool. In contrast to RDT, GRDT grammars allow to define *all* possible schemata with three simple grammar rules:[8]

**Concl** (X) $\leadsto$ *it* (X).
*it* (X) $\leadsto$ **P** (X).
*it* (X) $\leadsto$ **P** (X), *it* (X).

GRENDEL's grammar is similar to GRDT's, but, again, the predicate names have to occur in the grammar:

body (goal (X)) $\hookrightarrow$ *it* (X).
*it* (X) $\hookrightarrow$ *predicate* (X).
*it* (X) $\hookrightarrow$ *predicate* (X), *it* (X).
*predicate* (X) $\hookrightarrow$ ...

Let us now consider rules to be learned where a predicate occurs multiple times as a premise of a rule with different arguments:

$\mathbf{P}(X_1, X_2) \rightarrow \mathbf{Concl}(X_1, X_2).$
$\mathbf{P}(X_1, X_2)\ \&\ \mathbf{P}(X_2, X_3) \rightarrow \mathbf{Concl}(X_1, X_3).$
$\mathbf{P}(X_1, X_2)\ \&\ \mathbf{P}(X_2, X_3)\ \&\ \mathbf{P}(X_3, X_4) \rightarrow \mathbf{Concl}(X_1, X_4).$
$\vdots$

With these schemata, for instance, we are capable to learn a rule characterizing ancestor-lines of length three: ancestor(V1,V2) & ancestor(V2,V3) & ancestor(V3,V4) $\rightarrow$ ancestor_line_3(V1,V4). Unfortunately, the different rule schemata cannot be ordered along $\theta$-subsumption, because of the binding of the head-variables. So, instantiation and testing starts from scratch with each new schema. GRDT uses another way for pruning. Usually, each expansion of a nonterminal specializes the current hypothesis, because the hypothesis cannot become shorter by this step. Then, all further expansions of a hypothesis can be dropped, if the current hypothesis is already too special.

In GRENDEL, for each transitive predicate, a subgrammar must be created. If, for instance, we have the two transitive predicates male_ancestor and female_ancestor, two subgrammars must be given.

body (goal ($X_s$, $X_e$)) $\hookrightarrow$ *it* ($X_s$, $X_e$).
*it* ($X_p$, $X_e$) $\hookrightarrow$ *it_male_ancestor* ($X_p$, $X_e$).
*it* ($X_p$, $X_e$) $\hookrightarrow$ *it_female_ancestor* ($X_p$, $X_e$).
*it_male_ancestor* ($X_p$, $X_e$) $\hookrightarrow$ male_ancestor ($X_p$, $X_e$).
*it_male_ancestor* ($X_p$, $X_e$) $\hookrightarrow$ male_ancestor ($X_p$, $X_i$), *it_male_ancestor* ($X_i$, $X_e$).
*it_female_ancestor* ($X_p$, $X_e$) $\hookrightarrow$ female_ancestor ($X_p$, $X_e$).
*it_female_ancestor* ($X_p$, $X_e$) $\hookrightarrow$ female_ancestor ($X_p$, $X_i$), *it_female_ancestor* ($X_i$, $X_e$).

In contrast, GRDT's grammar allows the user to write predicate variables as arguments of nonterminal literals. Then the predicate variables of the different grammar derivations become identical by unification. The following example shows a grammar defining the previously displayed rule schemata, plus rule schemata of length greater than three premises:

$$\begin{aligned}
&\mathbf{Concl}\ (\mathsf{X}_s,\ \mathsf{X}_e) \leadsto \mathit{it}\ (\mathbf{P},\ \mathsf{X}_s,\ \mathsf{X}_e).\\
&\mathit{it}\ (\mathbf{P},\ \mathsf{X}_p,\ \mathsf{X}_e) \quad \leadsto \mathbf{P}\ (\mathsf{X}_p,\ \mathsf{X}_e).\\
&\mathit{it}\ (\mathbf{P},\ \mathsf{X}_p,\ \mathsf{X}_e) \quad \leadsto \mathbf{P}\ (\mathsf{X}_p,\ \mathsf{X}_i),\ \mathit{it}\ (\mathbf{P},\ \mathsf{X}_i,\ \mathsf{X}_e).
\end{aligned}$$

**P** can become instantiated by both predicates, male_ancestor and female_ancestor.

Let us summarize the weaknesses of GRENDEL and RDT, that are overcome by GRDT. In GRENDEL, the predicate names themselves occur in the grammar. This results in large grammars, which must be changed whenever the set of predicates of the knowledge base changes. The second problem concerns rules where the same predicate multiply occurs in the premises. For a single predicate it is easy to formulate a grammar defining multiple occurrences. If, however, more than one predicate exists, for which these sequences must be built, the grammar becomes complex: for each sequence a subgrammar has to be specified. Using a grammar for the specification of rule schemata, it is possible to maintain the language bias when the set of predicates changes. The weakness of RDT's rule schemata is their fixed length. Rules of increasing length can easily be specified, even if the transitivity of a predicate is to be expressed.

Furthermore, GRDT uses the generalization order given implicitly by the grammar instead of $\theta$-subsumption. In this way multiple testing of the same hypotheses is avoided.

### *4.3.2. Search in the hypothesis space*

The GRDT-algorithm starts with the goal predicate as conclusion and generates the first, the most general hypothesis, depending on the grammar, by choosing the grammar's starting rule and instantiating its left-hand side with the goal predicate. Then, it searches the hypothesis space in depth-first order from general to special hypotheses. Each iteration consists of two steps, the *specialization of the hypothesis* and the *test of the hypothesis*.

#### *Specialization*

For specializing a hypothesis four alternatives exist:

- **if** the hypothesis contains an uninstantiated predicate variable, try to find an admissible instantiation of this variable.
  Admissible instantiations are all predicates of the knowledge base with the right arity, right argument sorts, and corresponding to the task structure, defined by MOBAL's predicate structuring tool.

- **else if** the hypothesis contains a nonterminal symbol, try to expand the nonterminal.
  To expand the nonterminal, the next grammar rule, which has this nonterminal as left-hand side is used to replace the nonterminal with the grammar rule's right-hand side.
- **else if** the hypothesis contains a constant to be learned, try to find an admissible value for this constant.
  GRDT learns constants in exactly the same way as RDT. To learn nominal constants, GRDT tests all remaining instantiations of that variable. For ordered values, the smallest (or largest) term of the instances of this variable is chosen first. If the hypothesis is accepted or rejected, all greater (or smaller) terms, respectively, are pruned.
- **else** no further specialization is possible; start backtracking.

*Hypothesis test*

After each specialization step, the resulting hypothesis must be tested. The cardinalities of the following sets are counted, again exactly like RDT does:

> positive examples covered by the hypothesis; negative examples covered erroneously; uncovered positive examples; yet unknown facts, that can be derived by the hypothesis (predicted); total number of examples.

Testing a hypothesis means to find all solutions for the goal concept with the hypothesis — which is just a Horn clause — as concept definition and the background knowledge as theory. This test is tractable because the facts are ground (Wrobel, 1987), (Morik *et al.*, 1993). Since the knowledge is managed by MOBAL's inference engine IM-2, the test itself is slightly different to a simple PROLOG-call.

The user constructs an acceptance criterion using the cardinalities above. For instance, the user may want a hypothesis to cover more positive examples than it does not cover and a ratio of covered positive examples divided by covered negative examples greater than 2.

The result of counting the items is compared with the acceptance criterion. The evaluation is used for pruning the search safely:

- **if** the hypothesis is too special (i.e., the pruning criterion is satisfied) stop specializing and start backtracking;
- **else if** the hypothesis is partial (i.e., not all the predicate variables, constants to learn, and nonterminal symbols of the hypothesis are instantiated), 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.

### *4.3.3. Open Problems*

Open problems of GRDT should not be hidden. The specialization step does not guarantee, that the new hypothesis is really more special than the one tested previously, although this is assumed for pruning. Having a grammar defining derivations without any specialization of the derived rules, the termination of the algorithm is not assured. However, if it terminates, the algorithm has carried out a complete search through the hypothesis space. The completeness of GRDT is inherited 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.

## 5. Experiments

Our investigations are conducted in cooperation with the university of Karlsruhe. Their mobile robot PRIAMOS (Dillmann *et al.*, 1993) is used for executing the movements and gathering the sensor data. PRIAMOS is equipped with 24 sonar sensors measuring the distance to the nearest object in the range of the emission cone.

We had data from paths in a variety of simple rooms. The rooms differ in two respects. First, the doors of the rooms differ. They are wider or smaller, have thicker or thinner doorframes and the door is centered or it is near to a wall. Second, we have some obstacles in the rooms, which could easily be muddled with a doorway. 17 of the totally 72 paths are movements through a doorway, and 10 paths are along a door. Each trace is documented by 36 to 118 data sets. In total, we had 82139 measurements, each one consisting of the following entries:

> trace number and point of time; robot position; sensor number, its position and orientation; measured distance; object and edge number of what is sensed.

As mentioned in Section 4, the input to learning consists of sensor data in a known environment. Hence, the sensed point and the object to which it belongs is given. The learning task presented here is to acquire the perceptual feature of moving through a doorway, through_door.

For learning, we used three different training sets, each consisting of 26 of the 72 traces and about 2500 measurements. In two of the training sets, the robot moved 11 times, in a third training set, it moved 13 times through the doorway. For verifying the learned rules, we used the remaining 46 traces as test set, containing those examples of movements through a doorway, which were not in the respective training set (6 and 4 examples, respectively). In addition, the test sets contained 11 situations looking similar to movements through a doorway.

We now describe the results of learning three types of rules:

- Rules for patterns for single sensors (see Section 5.1)

- Rules for patterns for groups of sensors (see Section 5.2)
- Rules for moving through a doorway in terms of patterns for sensor groups (see Section 5.3)

For each type of rule, background knowledge, examples, and a grammar defining a set of rule schemata is prepared. We illustrate the learning step by step in the next sections.

### 5.1. Learning rules for single sensors

In this section, we describe how to learn rules for single sensor features. To test the influence of the choice of basic features (Section 4.1), we used three different ways of calculating basic features of different granularity.

The first variant has the least refined granularity. Basic features are intervals of a fixed length, the symbol that is attached to each interval depends on the gradient between the first two measured distances. Totally, a set of 7693 basic features was constructed.

The second variant, with a medium granularity, calculates the length of the sequence dependent on the average gradients between each pair of consecutive measurements. With this variant, we got a set of 9842 basic features.

For the variant with the finest granularity, we additionally regard the divergence of the measurements and add the new measurement to the interval only if the divergence is below a threshold. Now, we calculated a set of 14261 features.

The first learning step was to find sequences of basic features characterizing sensor features, i.e., a specific constellation of edges sensed by a single sensor. For describing the hypothesis space, the following grammar is used:

$$\begin{aligned}
&\textbf{Concl}\ (\mathsf{Tr},\ \mathsf{Se},\ \mathsf{T}_1,\ \mathsf{T}_2,\ \textbf{Movement}) \rightsquigarrow \textit{seq}\ (\mathsf{Tr},\ \mathsf{Se},\ \mathsf{T}_1,\ \mathsf{T}_2,\ \textbf{Movement}).\\
&\textit{seq}\ (\mathsf{Tr},\ \mathsf{Se},\ \mathsf{T}_i,\ \mathsf{T}_2,\ \textbf{Movement}) \rightsquigarrow \textbf{BF}\ (\mathsf{Tr},\ _,\ \mathsf{Se},\ \mathsf{T}_i,\ \mathsf{T}_2,\ _).\\
&\textit{seq}\ (\mathsf{Tr},\ \mathsf{Se},\ \mathsf{T}_i,\ \mathsf{T}_2,\ \textbf{Movement}) \rightsquigarrow \textbf{BF}\ (\mathsf{Tr},\ _,\ \mathsf{Se},\ \mathsf{T}_i,\ \mathsf{T}_j,\ _),\\
&\qquad\qquad \textit{seq}\ (\mathrm{Tr},\ \mathrm{Se},\ \mathrm{T}_j,\ \mathrm{T}_2,\ \textbf{Movement}).
\end{aligned}$$

GRDT requires the definition of an acceptance criterion. We have chosen a criterion accepting all hypotheses covering more positive examples than they derive additionally ($pos \geq pred$). For four different goal concepts (namely s_line, s_jump, s_convex, s_concave) we used the different training sets for learning resulting in different sets of rules. In Table 1, we have summarized the results of learning rules for single sensors. It shows for each basic feature set the average number of learned rules, counted for the different goal concepts separately. The next column shows the total number of positive test instances for the four concepts. The third value is the total number of instances, derived by the learned rules. From all derived instances some are derived correctly (next column), i.e., they are instances of the concept. The last two columns present the coverage and the correctness.

The results clearly indicate the choice of the basic features to be crucial. Tracing learning explains, why the first set of basic features produces poor results. The time

*Table 1.* Learning results for sensor features.

| | #rules | #given | #derived | #correctly der. | correct/given | correct/derived |
|---|---|---|---|---|---|---|
| *Basic Feature Set 1* | | | | | | |
| GRDT | 1.9 | 5113 | 541 | 279 | 5.5% | 51.6% |
| *Basic Feature Set 2* | | | | | | |
| GRDT | 12.8 | 6238 | 3278 | 1750 | 28.1% | 53.4% |
| GRENDEL | 3.5 | 3252 | 1527 | 329 | 10.1% | 21.6% |
| *Basic Feature Set 3* | | | | | | |
| GRDT | 7.6 | 6238 | 5543 | 589 | 9.4% | 10.6% |

intervals of the basic features do not often fit the time points of the examples, i.e., the sensor features. So only few of the given examples could be used for learning, resulting in a very low number of correctly derived instances. The second basic feature set seems to have the best granularity among the chosen sets.

In addition to the tests described previously, we tested RDT, FOIL, and GRENDEL. RDT learned exactly the same rules as GRDT did, because the hypothesis spaces are equal. However, because of the different pruning techniques, GRDT is up to five times faster than RDT.

To run FOIL, we prepared negative examples explicitly, because otherwise, FOIL runs out of memory, even if we restrict the automatic generation of negative examples to 1%. We provided all sensor features, that are not currently goal features as negative example. For instance, if s_jump is the goal predicate and s_concave (trace3, sensor1, 4, 19, parallel) is an instance of s_concave, we added not (s_jump (trace3, sensor1, 4, 19, parallel)). From this data, FOIL learned a single, very unintuitive rule:

no_movement (Tr, Or, Se, $T_1$, $T_2$, G) $\rightarrow$ s_jump ($Tr_2$, Se, $T_3$, $T_4$, parallel).

This rule is useless, because sensor features describe perceptions while the robot is moving.

The third, and most interesting, comparison is the one with GRENDEL. For this test, we have taken the best basic feature set, Set 2, and a grammar corresponding to GRDT's one. Table 1 summarizes the results, which clearly indicate that for our domain, a learner that just covers the goal concept learns too few rules, that are unlikely to be applicable in future scenes.[9]

### 5.2. Learning rules for groups of sensors

In the next step, we learned rules defining features for groups of sensors. Here, we used grammars focusing the search to hypotheses which constrain the number of sensors of a specific sensor class perceiving the same single sensor feature. We

*Table 2.* Learning results for sensor group features.

| | #rules | #given | #derived | #correctly der. | correct/given | correct/derived |
|---|---|---|---|---|---|---|
| CC1 | 193 | 23000 | 20642 | 19553 | 85.0% | 94.7% |
| CC2 | 152 | 30948 | 31137 | 26436 | 85.4% | 84.9% |

considered classes with one, three, and five sensors. Via the grammars, we also constrained the time intervals during which the pattern must be perceived by the sensors. The largest difference between instants of time is defined by the background knowledge predicates d*i*succ, $i = i, \ldots, 5$. Some rules of the grammar used for learning rules for groups of two sensors, belonging to a sensor class with three sensors, are

$$\begin{array}{lll}
\textbf{Concl}\ (\mathsf{Tr},\ \mathsf{SCl},\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathbf{M}) & \leadsto & \textit{sf_comb}\ (\mathsf{Tr},\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathbf{M}). \\
\textit{sf_comb}\ (\mathsf{Tr},\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathbf{M}) & \leadsto & \textbf{SF}\ (\mathsf{Tr},\ \mathsf{Se}_1,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathbf{M}), \\
 & & \textbf{SF}\ (\mathsf{Tr},\ \mathsf{Se}_2,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathbf{M}), \\
 & & \mathsf{sclass}\ (\mathsf{Tr},\ \mathsf{Se}_1,\ \mathsf{First},\ \mathsf{Last},\ \mathbf{SCl}), \\
 & & \mathsf{sclass}\ (\mathsf{Tr},\ \mathsf{Se}_2,\ \mathsf{First},\ \mathsf{Last},\ \mathbf{SCl}), \\
 & & \mathsf{First} \leq \mathsf{T}_1,\ \mathsf{T}_2 \leq \mathsf{Last}, \\
 & & \mathsf{Se}_1 < \mathsf{Se}_2. \\
\textit{sf_comb}\ (\mathsf{Tr},\ \mathsf{T}_1,\ \mathsf{T}_4,\ \mathbf{M}) & \leadsto & \textbf{SF}\ (\mathsf{Tr},\ \mathsf{Se}_1,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \mathbf{M}), \\
 & & \textbf{SF}\ (\mathsf{Tr},\ \mathsf{Se}_2,\ \mathsf{T}_3,\ \mathsf{T}_4,\ \mathbf{M}), \\
 & & \mathsf{sclass}\ (\mathsf{Tr},\ \mathsf{Se}_1,\ \mathsf{First},\ \mathsf{Last},\ \mathbf{SCl}), \\
 & & \mathsf{sclass}\ (\mathsf{Tr},\ \mathsf{Se}_2,\ \mathsf{First},\ \mathsf{Last},\ \mathbf{SCl}), \\
 & & \mathsf{First} \leq \mathsf{T}_1,\ \mathsf{T}_4 \leq \mathsf{Last}, \\
 & & \textbf{Succ}(\mathsf{T}_1,\ \mathsf{T}_3).
\end{array}$$

For learning rules describing sensor group features (i.e., sg_line, sg_jump, sg_convex, and sg_concave) in terms of sensor features, we built learn and test sets from the same traces as in the first learning step. We have tested two different acceptance criteria, the first (CC1) accepts all rules covering at least one positive example. The second (CC2) is the same as for learning sensor features, accepting rules if they cover more examples than they predict. Table 2 shows the result of learning evaluated with the test set. The columns represent the same figures as in the previous table. The correctness of the learned rules surpasses that of the first task. The number of learned rules is very high, resulting from the high number of chosen sensor classes. Because of current experiments, we are optimistic to learn the *necessary* sensor classes using conceptual clustering.

*Table 3.* Results of applying all learned rules

| | #rules | #given | #derived | #correctly der. | correct/given | correct/derived |
|---|---|---|---|---|---|---|
| Tr. Set 1 | 12 | 17 | 17 | 12 | 70.6% | 70.6% |
| Tr. Set 2 | 11 | 17 | 14 | 11 | 64.7% | 78.6% |
| Tr. Set 3 | 10 | 17 | 16 | 12 | 70.6% | 75.0% |

## 5.3. Learning perceptual features of moving through a doorway

The last learning step is to combine features sensed by different sensor groups of the robot to describe a perceptual feature like moving through a doorway. The quality of the rules learned in this step is the most relevant one, because it determines whether the whole set of rules can be applied successfully in different environments or not.

For learning, we prepared three training sets, each with 21 traces. In 13 of these traces, the robot moved through the doorway of the room, 6 times parallelly to the doorframes and 7 times diagonally. The hypothesis space is defined by a grammar specifying only a single rule schema:

$$\begin{aligned}\textbf{Concl}\ (\mathsf{Tr},\ \mathsf{T}_{start},\ \mathsf{T}_{end},\ \textbf{Movement}) \leadsto\ & \textbf{SGF}\ (\mathsf{Tr},\ \textbf{SClass}_1,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \textbf{Movement}),\\ & \textbf{SGF}\ (\mathsf{Tr},\ \textbf{SClass}_2,\ \mathsf{T}_1,\ \mathsf{T}_2,\ \textbf{Movement}),\\ & \mathsf{SClass}_1 \neq \mathsf{SClass}_2,\\ & \mathsf{T}_{start} < \mathsf{T}_1,\ \mathsf{T}_2 > \mathsf{T}_{end}.\end{aligned}$$

The acceptance criterion accepts hypotheses covering more than 3 positive examples and no negative one.

We used all 72 traces to verify the learned rules. As the test set only contains sensor measurements, learned rules from *all* levels of abstraction were applied to derive the perceptual features. Table 3 shows the results. In every test set, three situations are classified incorrectly, in which the robot moved through the doorway very close to one of the doorposts. Because of the short distance, the sensors did not got any echo and hence, it could not derive the feature through_door.

Overall, the result justifies our approach. The quality of the learned rules improves as the representation hierarchy is climbed. The rules are capable to classify many of the given examples correctly. This way to get more and more abstract concepts is robust against the noise, produced by the weak ultrasonic sensors.

As this test evaluates all rules up to perceptual features, it is the most important one. It shows the advantage of designing a sequence of learning tasks for concepts at more and more abstract levels.

## 6. Related and future work

We have presented an approach to making a complex overall learning task tractable. The learning of high-level concepts from sensor data gathered during several paths in known environments (particular office rooms) has been split into a sequence of learning steps. The results of all learning steps together form a knowledge base that derives high-level concepts from low-level data. The training phase is off-line. The results of the training phase show that inductive logic programming is capable of constructing high-level descriptions of actions, perceptions, and the environment from sensor data. Although several learning steps – particularly those at lower levels – deliver concepts that misclassify some data, the overall knowledge base achieves a high accuracy for high-level concepts. This robustness encourages us to put the learned concepts to use: concepts are to be executed in unknown environments of a certain type, e.g., office rooms. During the execution phase, the rules characterizing concepts are applied to the actual sensor measurements while performing certain moves. In the next section, we describe how we plan to apply the learned concepts to real-world missions of a mobile robot.

### 6.1. Future Work

The learned knowledge base is to be used for *task specification, execution monitoring, explanation of error recovery,* and for the *update of the environment representation.* When using the knowledge base for task specification, a planning component is needed that converts high-level concepts to actual perceptions and actions of the robot. The robot user may state a goal concept. The planner[10] then proves the premises of the rule that characterizes the concept and calls the execution of basic actions in the following way:

**Given:** State (given by an action-oriented perceptual feature), goal concept

**Planning:**

**If** the verification of the sensor pattern of the high-level concept has been achieved,
then stop with the message "success" and output the recordings of actions and perceptions in abstract terms.

**If** no planning for the current state is possible using the given rules
then stop with the message "fail" and output the recordings of actions and perceptions in abstract terms.

**Else** derive a plan for transferring the current state into a precondition of the goal concept (high-level planning), and
derive an action with its corresponding perceptual feature (deriving lower-level actions)

**Acting:** execute the action and acquire sensor data until

- either you are in danger to collide with an object – then change direction
- or you have verified a perceptual feature for an action – then proceed with **planning**.

Action and perception are closely related in this incremental planning strategy. The learned rules are used by backward and forward chaining. Backward chaining creates a plan for transferring the current state into the precondition of the high-level concept. Backward chaining also derives the low-level commands of the robot from the high-level concepts. Forward chaining derives the perceptual features from the corresponding sensor data acquired during a certain move of the robot.

When, for instance, the planner is called with the state

standing(t88, 45, 48, before_wall, right, small_side, along_door)

(standing(*Trace, Time, Time, Perception, SensorClass, RobotSide, PreviousPerception*))

and the goal concept

move_in_front_of_door(t88, 45, *T4*, *T3*),

backward chaining verifies the preconditions of the learned rule for move_in_front_of_door. The first condition is already given by the current state. The second condition (opposite(right, *X*)) is verified by matching the fact opposite(right, left), which is part of the knowledge base. The premise

parallel_moving(t88, 48, *T3*, slow, backwards, in_front_of_door, left)

requires further backward chaining in order to achieve basic robot actions. Here, the basic action is move(t88, 48, *T3*, slow, backwards)[11]. This basic action is performed until the sensor pattern in_front_of_door is verified. This is the acting step of the planner. Within this step, forward chaining is used to derive sensor patterns from incoming sensor data. It is for this use of basic features, that their calculation is already performed incrementally. As soon as the sensor group of the robot's left side perceive measurements that indicate the concept in_front_of_door, the action stops and *T3* becomes instantiated with the point in time of the last sensor data necessary to conclude in_front_of_door. The result of the acting step is

parallel_moving(t88, 48, 72, slow, backwards, in_front_of_door, left).

Planning proceeds with proving the verification condition of move_in_front_of_door,

standing(t88, 72, *T4*, in_front_of_door, left, small_side, in_front_of_door).

This evokes again the acting step. For three more points in time the sensors at the robot's left side (which is its small side) measure distances, that are classified as in_front_of_door by forward chaining. The result of the acting step is

standing(t88, 72, 75, in_front_of_door, left, small_side,in_front_of_door).

The goal concept is now verified with the fully instantiated fact

move_in_front_of_door(t88, 45, 75, 72).

Until now, the coupling between the knowledge-base and the robot is "manual", i.e., the data exchange is realized by an engineer who loads the basic actions derived by the knowledge-based system into the robot system and loads the sensor data of the robot into the knowledge-based system. We are currently working on conducting real-world experiments in collaboration with the University of Karlsruhe and their robot PRIAMOS. We are aware of problems with the time constraint of real-time processing that can hardly be met by our planner. This may well force us to distinguish in the future between planning in advance and planning at runtime as do (Gervasio & DeJong, 1994).

We explore the compilation of a rule set and facts into a structure that can more efficiently be executed in real-time. A first approach is to map rule sets to tree automata, whose final states represent concepts, which have to be recognized, in order to verify, that a goal has been achieved. In (Rieger, 1995b), we have shown, how a deterministic finite state automaton for the recognition of sensor features can be inferred. It takes as input a sequence of temporally ordered observations, i.e., basic features. Each of its final states is associated with one or possibly many sensor feature concepts. During the performance phase, we apply a marker passing method to the automaton: When the robot moves through the environment, its sensors constantly perceive observations, each of which might be the beginning of a string, which is accepted by the automaton. For each observation a marker is generated. Each marker is passed, if possible, from its current state to a successor state according to the transition function. If a marker has reached a final state, a message is produced, that an object has been recognized, which is possibly an instance of several concepts.

If a final state is associated with different concepts, we have the situation, that the sequence of observations leading to it, cannot be interpreted unambiguously by the robot. It cannot be sure, in which state it is and has difficulties in choosing the next action. In order to account for these uncertainties arising from ambiguous observations, we infer from the deterministic state automaton a hidden Markov model. This is done by splitting up the ambiguous final states into several unambiguous ones, by introducing non-deterministic transitions to the newly generated states, and by associating probabilities with the non-deterministic transitions. During learning, the training data is evaluated in order to derive relative frequencies, which are taken as estimates for the transition probabilities. During the performance phase, we use a modified version of the *Viterbi algorithm* (Jelinek, 1976), with which we determine the concept, which most probably has been perceived. In this way, the acting phase of our performance step is enhanced.

### 6.2. Related Work

There are similarities between our work and Shen's approach (Shen, 1993) to learning from the environment. In both cases the objective is to fuse application of rules with their construction, i.e., to couple learning with its evaluation. Shen also applies rule induction from examples in the context of problem-solving. With respect to the bidirectional use, our concepts are similar to the prediction rules used by Shen. The main difference concerns the domain of application, which in our case is much more demanding, as we are dealing with real-world data.

In the JPL Telerobot (Doyle *et al.*, 1986) a STRIPS-like planner is used, in which the precondition lists are amended to contain sensory requests to verify beliefs, and the postcondition lists are amended to include sensory requests that validate the effects of each action. The authors present a representation language in which these sensory requests are to be programmed. The idea being the same, our approach differs in that we represent and learn plans within the ILP-framework.

Our approach to the application of learning to robotics is closely related to approaches to learning plans for robots (Mitchell, 1990), (DeJong & Bennett, 1993), (Gervasio & DeJong, 1994), and (Gil, 1994). Some distinctions need to be mentioned. First, we do not use an a priori model of the environment. Therefore, we do not learn from mismatches between the predicted and the observed effects of actions as (DeJong & Bennett, 1993) and (Gil, 1994) do. Our learning does not refine a given plan, instead it actually constructs the rules to be used for planning. Second, our representation of planning operators is not that of classical planning with add and delete lists which makes inference non-monotonic. Instead, we attach time instants to sensor measurements and, on higher levels, we attach time intervals, so that reasoning remains monotonic. Our rules can be interpreted as operators with preconditions and postconditions, but technically they are just clauses. Third, our representation of states is different from the work cited above. A state corresponds to a time interval in which a particular sensor pattern is valid. The validation of the sensor pattern is left until actual execution. This way, we avoid committing ourselves prematurely to a firm sensor value. Our plan just states that a move has to be executed until a pattern of distance measurements is registered. This is similar to the contingent variables of (Gervasio & DeJong, 1994). Note, however, that in our case the sensor pattern is more complex than just a variable for a particular distance. A perceptual feature in our approach already abstracts beyond particular distance values and gives relations between sensor measurements instead. This abstraction allows a perceptual feature to cover many different situations. The particular time interval is instantiated in the rule by the value determined at runtime and propagated to other rules by unification. Fourth, in contrast to the mentioned planning approaches, our representation combines sensing and action at every level of detail. That is, we do not have separate sensory operators for acting in order to recognize an object, but these actions are treated in the same way as other actions. In our example, it is the along_door feature which recognizes the door so that move_through_door becomes applicable.

## 7. Conclusions

We have applied machine learning to a challenging task, namely learning a hierarchy of concepts from sensor data and robot actions. We tackled this complex overall learning task by dividing it into several learning steps that link several levels of abstraction. In order to perform learning from examples, we applied a simulation component that classifies measurements with respect to the edges of the known environment that have been sensed. An example generation tool for putting together training sets for one learning step has been developed. A comfortable human-computer interface supports the management of the various training and test sets and shows the training paths graphically.

We have designed a representation language that expresses concepts which fuse perception and action for robot navigation. Even without learning capabilities, the high-level description is useful as has been argued by (Stopp *et al.*, 1994). Their project aims at natural language access to a robot. On one hand, the user can specify the robot's task in a way appropriate for human users. On the other hand, the user can more easily understand the robot's actions. In error situations, user and robot can communicate in order to recover from the failure. We do not go as far in the high-level description as natural language does but consent to Prolog clauses which are already easily understandable for a knowledgeable user. Moreover, natural language processing requires high-level concepts that are perceptually anchored. We learn such concepts.

The application of ILP to sensor and action data goes beyond standard ILP applications. First, the application of symbolic learning to robot data requires the conversion of numerical data to qualitative terms. An algorithm for the calculation of basic features has been developed. Second, the rather large size of the training sets asks for appropriate means to restrict the hypothesis space. GRDT is an ILP-algorithm, which uses grammars in order to define rule schemata that restrict the hypothesis space. Trying to generate negative examples by the closed-world assumption (as FOIL does) leads to tremendously large data sets that cannot be handled. GRDT does not explicitly create a set of negative examples but applies the closed-world assumption implicitly, if the user defines the acceptance criterion such that no new instances of the target concept are to be predicted. Third, as there are several ways for a concept such as, e.g., move through a doorway, to be applied, as many non-redundant rules as possible should be learned. Instead of ending the search for rules as soon as the target concept is covered, GRDT stops the search only, when further hypotheses would be redundant or cannot be accepted (i.e., some of the rule premises already cover too few positive examples).

The learning results have been tested in two ways. First, a test set for the particular learning step has been formed. Here, the learned rules of this step are used independent of other learned rules. Second, the rule base that has been acquired by several learning steps was tested on sequences of basic features. It became clear, that misclassifications at lower levels of abstraction do not result in the misclassification of concepts at higher levels of abstraction. If, for instance, the measurements of one sensor are not right, the sensor group may nevertheless deliver the correct pattern. This is due to (learned) rules that demand three sensors of a group to derive the same pattern before this pattern is said to hold for the sensor group. In this way, the hierarchy of learned concepts contributes to the robustness of learning results. Experiments have shown that GRDT learns concepts from data supplied by a robot effectively and efficiently.

## Acknowledgments

The BLearn-II project is funded by the European Community under P7274. The Dortmund part of the project is additionally funded by the Ministry for Science and Research of the Federal State Nordrhein-Westfalen.

We thank Stefan Sklorz, and Stephanie Wessel, who developed parts of the system. In particular, Stephanie Wessel made the system learn basic features. Stefan Sklorz developed the action features and the experimental planner. He also provided the human-computer interface for handling training sets and ran several experiments. Moreover, in the course of many discussions both contributed ideas. Eckart Zitzler does an important job in structuring the access to robot data used by several modules. Michael Goebel has tried out alternatives (e.g., conceptual clustering) for the acquisition of sensor patterns. We also thank Peter Brockhausen for proof-reading earlier versions of this paper. The described system is the result of the collective effort of everybody involved.

We thank wholeheartedly the MOBAL team at the Gesellschaft für Mathematik und Datenverarbeitung (GMD), Werner Emde, Jörg-Uwe Kietz, Edgar Sommer, and Stefan Wrobel. We use the MOBAL-system (Morik *et al.*, 1993) for the maintenance of the knowledge base and parts of GRDT are directly taken from MOBAL's learning component RDT. In particular, Jörg-Uwe Kietz advocated the relational representation of time and assisted us with theoretical effort estimations.

## Notes

1. We follow the Prolog convention and write constant terms in small letters, variables start with a capital letter.
2. For technical reasons we had to introduce conversion rules that convert the predicate of a rule's conclusion into an argument of another predicate. An example is the rule through_door (Tr, T1, T2, PDir, Side, Orien) → period_of_time_perception(Tr, T1, T2, through_door, PDir, Side, Orien).
3. Learning an incremental function approximator definitely goes beyond the purpose of our research.
4. This and the following definition are taken from (Muggleton & De Raedt, 1993).
5. Helft's approach actually does not use positive and negative examples but completes the given observations by the generalization step, which implies a closed-world assumption. As we assume our features to be disjoint, all instances of another feature are considered negative examples for the feature currently to be learned. This approach leads to almost the same results as Helft's approach. The difference is, first, the flexible user-given acceptance criterion, applied by GRDT, and, second, the focus on one goal concept for learning in one learning run.
6. RDT is one of the learning tools of the knowledge acquisition system MOBAL (Morik *et al.*, 1993)
7. A short note about our typographical conventions: rule schemata of RDT are written with ordinary arrows ($\rightarrow$ ), GRENDEL's grammars with $\hookrightarrow$ , and GRDT's grammars with $\rightsquigarrow$ . Nonterminal literals are written *slanted*, terminals are written sans serif. Variables (both, predicate variables and argument variables) that should be learned by RDT or GRDT are written in **bold face**.
8. Variables occurring in different grammar rules are regarded to be different.
9. Because GRENDEL sometimes runs out of memory, too, we did not performed all tests, and hence, the number of given examples is smaller than with GRDT.

10. A small planning component has been implemented by Stefan Sklorz.

11. The sides of the robot are named with respect to the robot's movement such that the direction of the movement always corresponds to the front side. For execution, the relative orientation with respect to the movement is converted to the absolute orientation of the robot, where sensor s0 defines the front side of the robot. In the example here, relative and absolute orientation happen to be the same. When the robot moves backwards with respect to the previous movement, the sides change their names so that they are now named with respect to this movement. In this way, right becomes left when the robot moves backwards. Hence, the door has to be perceived by the sensors at the left side, which before were the sensors at the right side.

## References

N. I. Badler, B. L. Webber, J. Kalita, and J. Esakov. Animation from instructions. In N.I. Badler, B.A. Barsky, and D. Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 51–93. Morgan Kaufmann, San Mateo, CA, 1991.

C. Baroglio, A. Giordana, and R. Piola. Learning control functions for industrial robots. In *MLC-COLT '94 Robot Learning Workshop*, 1994.

Scott William Bennett. Learning uncertainty tolerant plans through approximation in complex domains. Technical Report UILU-ENG-89-2204, The Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, January 1989.

F. Bergadano and D. Gunetti. An interactive system to learn funcional logic programs. In *Procs. 13th IJCAI*, pages 1044–1049. Morgan Kaufmann, 1993.

R. A. Brooks. The role of learning in autonomous robots. In *Procs. COLT 91*, pages 5–10, 1991.

William W. Cohen. Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68(2):243–302, 1994.

Luc De Raedt. *Interactive Theory Revision: an Inductive Logic Programming Approach*. Academic Press, 1992.

G. DeJong and S. Bennett. Permissive planning – a machine learning approach to linking internal and external worlds. In *Procs. 11th AAAI*, pages 508–513, 1993.

Rüdiger Dillmann, Jürgen Kreuziger, and Frank Wallner. PRIAMOS – an experimental platform for reflexive navigation. In Groen, Hirose, and Thorpe, editors, *IAS-3: Intelligent Autonomous Systems*, chapter 18, pages 174–183. IOS Press, 1993.

R. J. Doyle, D. J. Atkinson, and R. S. Doshi. Generating perception requests and expectations to verify the execution of plans. In *Procs. of the AAAI*, pages 81–88, 1986.

Sašo Džeroski, Stephen Muggleton, and Stuart Russell. PAC-learnability of determinate logic programs. *Procs. of 5th COLT*, pages 128–135, 1992.

M. Gervasio and G. DeJong. An incremental learning approach to completable planning. In W. Cohen and H. Hirsh, editors, *Procs. 11th Int. Machine Learning Conf.*, pages 78–86, 1994.

Y. Gil. Learning by experimentation – incremental refinement of incomplete planning. In W. Cohen and H. Hirsh, editors, *Procs. 11th Int. Machine Learning Conf.*, pages 87–95, 1994.

Stevan Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

Nicolas Helft. Induction as nonmonotonic inference. In *Procs. of the 1st Int. Conf. on Knowledge Representation and Reasoning*, 1989.

F. Jelinek. Continuous speech recognition by statistical methods. *Procs. of the IEEE*, 64:532–556, 1976.

L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. *Robotics and Autonomous Systems*, 6:35–48, 1990.

Jörg-Uwe Kietz and Stefan Wrobel. Controlling the complexity of learning in logic through syntactic and task–oriented models. In Stephen Muggleton, editor, *Inductive Logic Programming*, chapter 16, pages 335–360. Academic Press, London, 1992.

José Millán and Carme Torras. A reinforcement connectionist approach to robot path finding in non-maze-like environments. *Machine Learning*, 8:363–395, 1992.

T. M. Mitchell and S. B. Thrun. Explanation-based neural network learning for robot control. In C. L. Giles, S. J. Hanson, and J. D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann.

T. Mitchell. Becoming increasingly reactive. In *Procs. 8th AAAI*, pages 1051–1058, 1990.

K. Morik, S. Wrobel, J.-U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning - Theory, Methods, and Applications.* Academic Press, London, 1993.

S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. Technical Report CW 178, Department of Computing Science, K.U. Leuven, May 1993.

Stephen Muggleton, editor. *Inductive Logic Programming.* Academic Press, 1992.

N. J. Nilsson. Shakey the robot. Technical Note 323, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1984.

Gordon D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, chapter 8, pages 153–163. American Elsevier, 1970.

Gordon D. Plotkin. *Automatic methods of inductive inference.* PhD thesis, University of Edinburgh, 1971.

J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

A. Rieger. Data preparation for inductive learning in robotics. In *Working Notes of the IJCAI-Workshop on Data Engineering for Inductive Learning*, 1995. also available as LS8-Report 19, University of Dortmund.

A. Rieger. Inferring probabilistic automata from sensor data for robot navigation. In *Procs. of the 3rd European Workshop on Learning Robots*, 1995. also available as LS-8 Report 18, University of Dortmund.

Alberto Segre. *Machine Learning of Robot Assembly Plans.* Kluwer, Boston, 1988.

Wei-Min Shen. Discovery as autonomous learning from the environment. *Machine Learning*, 12:143–165, 1993.

Glenn Silverstein and Michael J. Pazzani. Relational clichés: Constraining constructive induction during relational learning. In L. Birnbaum and G. Collins, editors, *Machine Learning: Procs. of the 8th Int. Workshop*, pages 298–302, San Mateo, CA, 1991. Morgan Kaufmann.

Luc Steels. Building agents out of autonomous behavior systems. In L. Steels and R. Brooks, editors, *The 'artificial life' route to 'artificial intelligence' – Building situated embodied agents.* Lawrence Erlbaum, New Haven, 1993.

Eva Stopp, Klaus-Peter Gapp, Gerd Herzog, Thomas Laengle, and Tim Lueth. Utilizing spatial relations for natural language access to an autonomous mobile robot. In *KI–94, – Procs. of the German Conference on AI*, Berlin, 1994. Springer.

Birgit Tausend. Representing biases for inductive logic programming. In Pavel Brazdil, editor, *Machine Learning - Procs. of ECML-93*, Lecture Notes in Artificial Intelligence, pages 427–430, Berlin, Heidelberg, New York, 1993. Springer.

Stefanie Wessel. Lernen qualitativer Merkmale aus numerischen Robotersensordaten. Master's thesis, Universität Dortmund, 1995.

Stefan Wrobel. Higher-order concepts in a tractable knowledge representation. In K. Morik, editor, *GWAI-87 – 11th German Workshop on Artificial Intelligence*, pages 129–138, Berlin, New York, Tokyo, October 1987. Springer Verlag.

Stefan Wrobel. Towards a model of grounded concept formation. In *Procs. 12th IJCAI*, pages 712–719, Los Altos, CA, 1991. Morgan Kaufman.

Kai Zercher. *Wissensintensives Lernen für zeitkritische technische Diagnoseaufgaben.* infix, Sankt Augustin, 1992.