# On-line Inference of Off-line Learned Operational Concepts

**Volker Klingspor**

*University of Dortmund, Computer Science Dept. VIII, D-44221 Dortmund, Germany*
*volker@ls8.informatik.uni-dortmund.de*

**Abstract.** Learning in robotics has received more and more attention in recent years. It eases bridging the gap between low-level sensor data and high-level concepts. A high-level representation language is necessary in order to support the communication between robot and user in both directions when the robot navigates in unknown environments. Controlling robots in terms of a high-level representation formalism like first-order logic is often said to be too slow. In this paper, we present a performance system capable of inferring previously learned high-level concepts from the sensory input of a mobile robot on-line in real-time. Furthermore, we describe an inference engine tailor-made to the requirements of our representation.

## 1 Learning in robotics

In recent years, machine learning becomes more and more an important topic in robotics. Learning eases adaptation of robots to different tasks, to different environments, and different robot settings. Learning can be applied in very different ways. Biologically inspired work in robotics has developed artificial beings which adapt to their environment (Brooks, 1991; Steels, 1993). This type of learning is restricted to reflex-like behavior. Learning skills by human demonstrations (Dillmann et al., 1995) eases programming of the robot's basic behavior, but does not enable the robot to report on its actions. In order to enhance a robot's high-level processing, planning can be tailored to particular robot tasks by learning techniques (Segre, 1988; Bennett, 1989). Learning can also be used to link the environment model with perceptions when executing a plan (DeJong and Bennett, 1993).

These approaches have in common, that learning takes place at one horizontal level of control. Our approach, in contrast, concerns vertical learning to close the gap between numerical data and high-level concepts. The concepts, learned at higher levels of the hierarchy will be used as items for communication between user and robot in both directions, from the user to the robot to control the system, and from the robot to the user to report the performed actions. The report will explain whether the task was performed correctly, or which circumstances, e.g. a wrong representation of some concept, have forced an error. The learning tasks and the symbol grounding aspects are described in more detail in (Klingspor and Morik, 1995). In contrast to many robot learning approaches, learning phase and performance phase are two separate steps. During the learning phase, we show the robot in a known environment, how objects look like. From this information, examples are generated, which are then used for supervised learning.

This paper will be a bit different from most papers describing robot learning. In most approaches, the representation formalism is tailored to the robotic domain. Then, a new learning method must be found and described by the author. We are going the other way around by taking a well-known representation formalism, restricted first order logic (FOL). We have defined a representation language, i.e. a signature, suited to this formalism and appropriate for the specific domain. Then, we applied existing logic based learning algorithms[1] to learn concept descriptions. Tests have shown, that most of these methods are more or less able to handle the learning problem (Klingspor et al., 1996). Now, we have to present the applicability of the learning result. In this paper, we will show how concepts represented in a restricted FOL can be used on-line with data of a real robot working in a real environment.

[1]A couple of methods for learning in logic are described in (Muggleton, 1992).

## 2 Scenario

Before describing the representation hierarchy and the performance system, we present the scenario and the robot we used for getting data and performing the experiments. PRIAMOS, developed at the University of Karlsruhe (Dillmann et al., 1993), is a mobile robot of size 90cm $\times$ 65cm $\times$ 75cm. It can move in three degrees of freedom, i.e., it can move into every direction and rotate simultaneously. It has 24 sonar sensors measuring the distance to the nearest object within the emission cone. At every side of the robot three sensors are installed, oriented parallelly. Three sensors are mounted at every corner oriented with a difference of $15^\circ$. While moving, PRIAMOS delivers for each sensor about three readings per second, consisting of the position of the sensor and the sensed distance. Errors can occur, if the sonar beam is multiply reflected or the angle between the beam and the object is inconvenient. Figure 1 shows PRIAMOS[2].

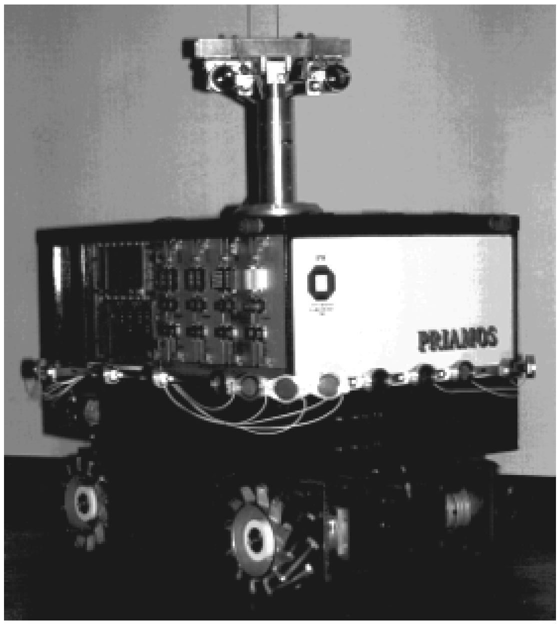


Figure 1: The mobile robot PRIAMOS.

Our goal is to enable a non-expert user to communicate with a mobile robot operating in an unknown environment. Communication has to occur in both directions, i.e. from the user to the robot to give commands and vice versa to receive protocols of actions performed by the robot. This communication has to be done in a human-adequate way. Instead of communicating in terms of real world coordinates, high-level terms should be used. An example for such a command is: "leave the room through the doorway and turn to the left". These commands require the robot to be able to find instances of the concepts used in the command; in the example, the robot must be able to detect the doorway. Since the robot has no map from which this information can be extracted, the sensor readings must be used to determine what kind of an object the robot currently perceives. This is a typical classification task. Since it is time consuming, boring, and sometimes difficult to program the classification rules by hand, we are interested in applying machine learning algorithms for this task.

In contrast to many other robot learning architectures, e.g., reinforcement learning architectures (Kaelbling, 1991; Millán and Torras, 1992), learning rules and applying these rules are two different phases. During the learning phase, rules are learned that describe (via a multi-level hierarchy of intermediate concepts) the concepts that are needed for the communication and for controlling the robot. From an abstract point of view, during the learning phase, we show the robot in a known environment, how some specific classes of objects look like when performing specific actions.

For supervised learning, examples must be provided to the learning algorithm. To calculate the examples used for learning, we need additional information about the sensor readings, e.g., which spot in the environment has been sensed by a sensor. Since this information cannot be provided by a robot operating in the real world, the data for learning is gathered by a simulator. PRIAMOS moves in the simulation through a simple structured office environment and gathers its sensor readings and the edge that is sensed by a measurement. From this information, we calculate examples at all levels of the representation hierarchy. At some level, e.g., we calculate the time intervals during which a particular constellation of edges is sensed in sequence. These examples together with the basic symbolic items calculated from the measurements are input to a standard logic-based learning algorithm. In this way, learning is a separate phase, completely performed off-line and independent from the application of the learned rules by the real robot.

During the performance phase, the learned rules are applied to classify the perceptions and actions of the robot by forward inferences. In addition the robot is controlled by triggering elementary actions via backward inferences. Before we describe the

[2]As visible in the figure, in the meantime, additional sensors like a stereo vision system has been mounted on PRIAMOS.

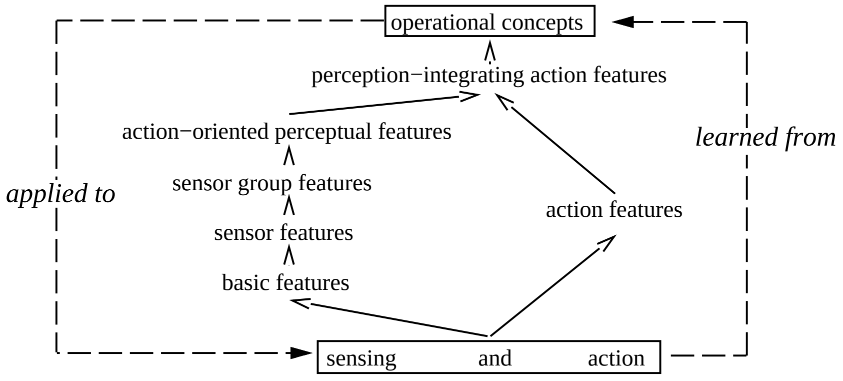


Figure 2: Representation Hierarchy

forward inference system (Sect. 4), we explain the developed representation in more detail (Sect. 3).

## 3 Representation

Figure 2 shows the representation hierarchy. On the left hand side, perceptions are placed, on the right hand side actions. At higher levels, both are integrated. At each level of the hierarchy, the concepts are defined in terms of the immediate lower level, using Horn clauses. Several levels of abstraction link the level of raw sensor and motion data with the level of operational concepts. While climbing the hierarchy and getting more abstract, the features describe more and more conceptual items. Whereas at lower levels task independent situations like the perception of a continuous surface (e.g. a wall) are represented, at higher levels these simpler features are combined to describe more complex, task dependent features like the perception when moving through a doorway. The representation as well as the motivation for operational concepts is described in more detail in (Klingspor et al., 1996) and (Sklorz, 1995).

*Action features* describe the primitives of the robot's actions, representing intervals of uniform motions. The basic action features used are move, rotate, and stand, with the arguments trace (a unique identifier for an experiment), time interval, speed, and direction (i.e., one of forward, backward, right and left).

The perceptual features at the lowest level are *basic features* BF(Trace, Or, Sensor, $\text{Time}_1$, $\text{Time}_2$, Grad) with the predicate symbol BF is one of stable, incr_peak, decr_peak, increasing, decreasing, straight_to, straight_away, no_measurement. These predicates represent, that during the time interval [$\text{Time}_1$, $\text{Time}_2$] the distance sensed by sensor Sensor, oriented in direction Or, has changed in a particular way (Wessel, 1995; Klingspor and Morik, 1995)[3]. These intervals are calculated incrementally, so that they can be easily incorporated into a performance system.

At the next higher level, situations are described where, during a time interval [$\text{Time}_1$, $\text{Time}_2$], a single sensor Sensor sensed a specific structure of the environment like a concave corner, or a jump, e.g., a situation of two parallelly shifted walls, sensed in sequence. These *sensor features* are represented by predicates SF(Trace, Or, Sensor, $\text{Time}_1$, $\text{Time}_2$, Move), where SF is one of s_jump, s_line, s_convex, s_concave. Move defines whether the sensor has been moved parallelly or diagonally along the sensed structure, or straight towards or straight away from it. The definitions of sensor features, i.e., the rules describing sensor features in terms of basic features, are learned by the inductive logic based learner GRDT (Klingspor et al., 1996). The following rule is an example of the structure of the learned rules:

stable(Trace, Or, Sensor, $\text{Time}_1$, $\text{Time}_2$, $\text{Grad}_1$) &
incr_peak(Trace, Or, Sensor, $\text{Time}_2$, $\text{Time}_3$, $\text{Grad}_2$) &
stable(Trace, Or, Sensor, $\text{Time}_3$, $\text{Time}_4$, $\text{Grad}_3$)
$\rightarrow$ s_jump(Trace, Sensor, $\text{Time}_1$, $\text{Time}_4$, parallel).

It is important to note, that the time intervals are linked via the arguments $\text{Time}_i$. The jump is sensed during the whole interval [$\text{Time}_1$, $\text{Time}_4$].

At the next level of the hierarchy, patterns sensed by different sensors of the same group of sensors are represented. A *sensor group* consists of sensors oriented to the same direction and, hence, sensing similar patterns. In this way, single sensor features are grouped together in order to increase the evidence for an edge constellation. This helps to cope with the noise of single sensor measurements by performing a kind of majority decision. E.g., it can be

[3]The last argument, Grad, denotes the gradient of the readings during the time interval. It is currently not used.

sufficient for deriving a *sensor group feature*, that only two of three sensors of a sensor group sense the same constellation. In addition, we abstract from the actual sensor numbers in this step. Instead, the orientation of the perception relative to the orientation of the motion is used.

At the next level of abstraction, the level of *action-oriented perceptual features*, perceptions at different sides of the robot will be combined, to represent the most abstract perceptual features. These features describe the perceptions of a robot in a particular situation, e.g., while moving through a doorway.

Action and perception will be integrated at the next level, the level of *perception-integrating action features*. Such a feature allows to verify the correct execution of a single action by testing its perceptual part (e.g. to perceive a doorway). In addition, the backward inference system can trigger the elementary operation defined by the feature (e.g. to start the robot moving in a particular direction), if a planner selects this feature to be the next operation to be performed.

At the highest level, the level of *operational concepts*, perception-integrating action features are used to describe complex operators with pre- and postconditions, such as: "before moving through a doorway, the robot must stand in front of a doorway, and afterwards, it must perceive the door opening at its rear".

We have chosen a hierarchical architecture for different reasons. First, the single learning tasks are easier. Less knowledge is needed for a single learning step. The hypotheses space for each single learning task is smaller, because the structure of the expected rules is simpler. Since learning in logic is subject to exponential explosion, this reduction is very helpful. On the one hand side, the simpler learning tasks permits to develop very efficient learning algorithms tailored to the individual problems. On the other hand, in contrast to the complex learning task, the simpler tasks can be handled by general learning algorithms. Because efficiency is, due to off-line learning, not most important, existing learner can be applied, avoiding effort for developing new algorithms.

A second result of a hierarchical representation is a smaller and hence better inspectable rule base. Clearly, a flat rule base defining the same goal concepts without intermediate concepts, could be found. However, then the resulting rule base consists of very many rules, because the cross product between the rules of each two levels must be build. In addition, the single rules are very large, because they contain all preconditions, otherwise gathered by climbing the hierarchy. In (Sommer, 1996)[Chapt. 4] different aspects about flattening and deepening inferential structures are discussed.

## 4 The Parallel Performance System

First order logic provides an expressive representation formalism, allowing to gather rules of different levels of abstraction and different tasks in a single inference engine, without becoming confused about the different kinds of data. Via forward chaining, such an inference engine may derive all intermediate concepts and the goal concepts from ground facts of basic features and basic actions as input.

However, using such a monolithic inference system has some disadvantages: it becomes slow, is difficult to maintain, and hard to implement. A set of restricted inference engines, each tailor-made for a specific task is much more efficient. We use several instances of a forward chaining inference engine. Each instance infers concepts of the next higher level in the abstraction hierarchy from given facts of the current level. Because of the acyclic hierarchical structure of the rule base, inference can be restricted to the depth of one. No inference from facts of another level is ever tried, but only the relevant data are known to the specific instance of the inference engine. This speeds up inference. Moreover, the instances of the inference engine can be run in parallel.

There is a second problem with using a monolithic inference engine: The complete performance system not only consists of a forward chaining part to derive concepts from sensory data, but also of a backward chaining part to control the robot, by aid of the operational concepts. Therefore, a general, monolithic inference engine has to support different control structures, namely forward and backward chaining. The backward chaining (action) part must know exactly, how far forward chaining (perception) has proceeded, in order to react to the perceptions as soon as possible. The interaction between goals that are derived by forward chaining and goals that are derived by backward chaining – which inference triggers which other inference – is a rather complex issue. Again, the overhead of controlling inference slows down the inference process. Hence, we developed three inference engines: one for inference with ordered rules by marker passing (Sect. 4.1), one for forward chaining of depth one for general Horn clauses (Sect. 4.2), and one for backward chaining (in work). These engines are

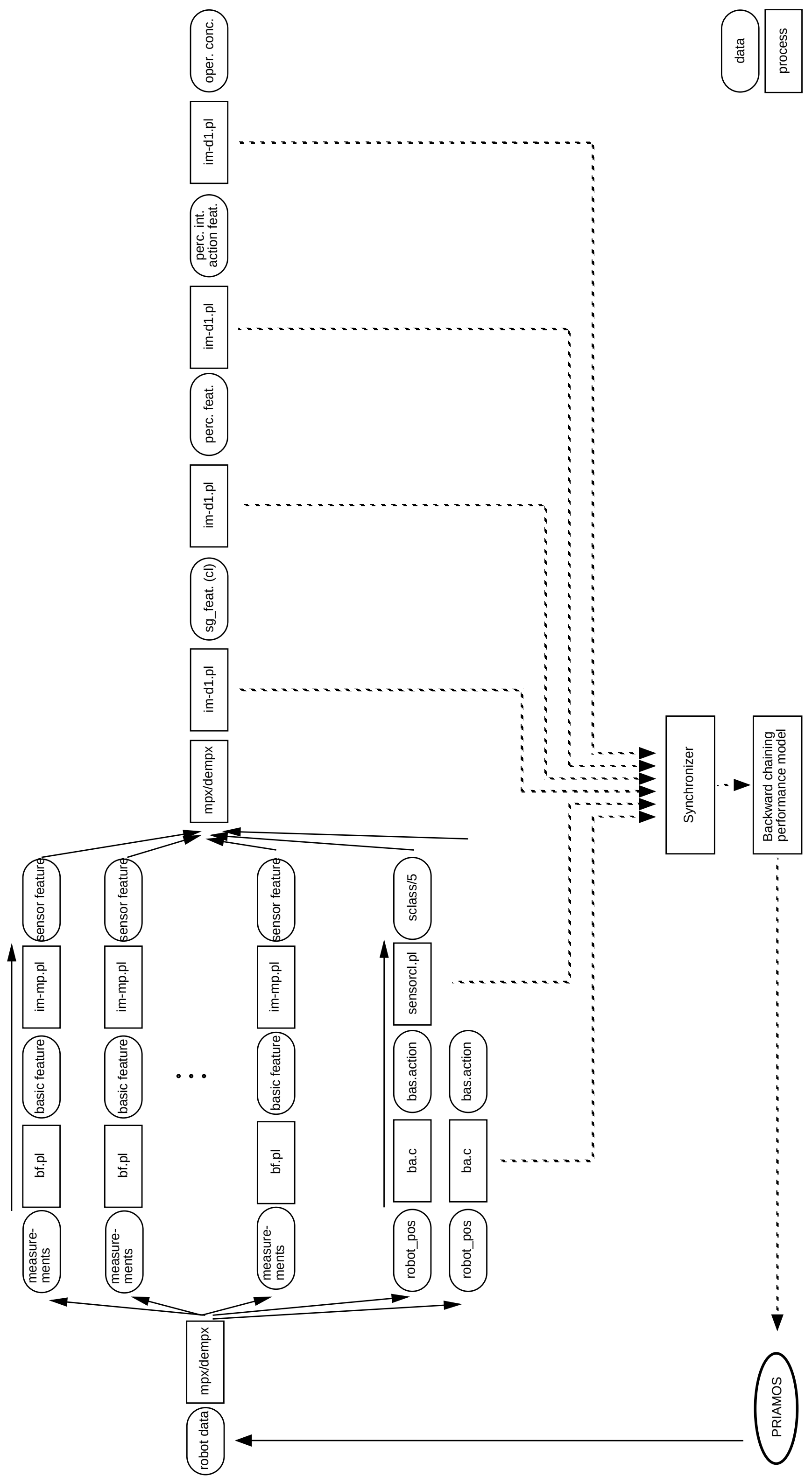


Figure 3: The forward chaining performance model
(`im-mp.pl` denotes a marker passing inference engine, `im-d1.pl` the more general inference engine for inference of depth 1)

specialized in the representation format that they use and the chaining direction. The control is now up to multiplexer and synchronizer processes. That is, control matters are not handled at the level of a knowledge base and its inference engine, but at the level of UNIX processes. This results in a speed-up that allows the overall system to be used on-line in real-time.

The different processes are organized as follows, to be integrated into a robot control system. The control system of the robot writes its data via UNIX pipes to our performance system, shown in Figure 3. A multiplexer distributes the data to 26 process chains, one for each of the 24 sensors to incrementally derive sensor features and two to generate basic actions and background knowledge about sensor classes.

After having inferred the sensor features and calculated basic actions and sensor classes, a demultiplexer gathers all the data and sends them to the next inference engine, deriving sensor group features. The inference engine used at this and all further levels is the one described in Sect. 4.2.

The different processes used by this performance system are linked by UNIX pipes. Each process reads from stdin and writes to stdout. This reduces the effort to integrate the inference chain into other systems like PRIAMOS' control system. In addition, every element of every chain can be inspected by their own, to find reasons for missing or additional derivations of features. To ease the inspection, a small program can be inserted everywhere in the chain to write transmitted data to a further shell, or to a UNIX device. This enables, e.g., the display of intermediate results.

## 4.1 Inference by Marker Passing

In this section we describe an inference technique tailored to the specific task of inferring sensor features from basic features. Therefore, we compile in a preprocessing step the learned Horn clauses into a structure facilitating incremental inference during the performance phase. This compilation is possible, because the data is ordered by time, i.e., the premises of each learned clause can be ordered by the time points and the basic features are ordered by time, too. They are calculated one after the other, without any interrupt. In (Rieger, 1995; Rieger, 1996a; Rieger, 1996c), Rieger describes a method to structure Horn clauses in a prefix (tree) acceptor to which a marker passing method for efficient forward inferences is applied. This acceptor is defined to be a finite state automata. Finite state automata, however, erroneously suggest that the acceptor is only able to handle propositional rules, because usual automata are not able to pass information from one state to the other. In first order logic, this is necessary to pass the bindings of terms to variables. In this paper, we pick up the ideas of Rieger, but we use another way to define marker passing trees, similar to those of Rieger, to overcome this deficiency of using finite state automata for definition.

### Compiling Horn clauses to marker passing trees

Let $G = (V, E, L_V, L_E)$ be a tree with $V$ as vertices of the tree and $E$ the edges of the tree. Each node is labeled by a possibly empty set of literals, the conclusions: $L_V : V \rightarrow 2^{\text{Literals}}$. Each edge is labeled by exactly one literal, a premise of a clause: $L_E : E \rightarrow \text{Literals}$. We call a path in the tree to be *complete*, iff it starts at the root node $r$ and ends in a node $v$ labeled by a nonempty set $L_V(v)$, i.e., the end node is labeled with at least one conclusion. Then, we can define the clauses, represented by a complete path $E = (E_1, \ldots, E_n)$ :

$$Clauses(E) := \{\ \{L_E(E_1), \ldots, L_E(E_n), Concl\} \mid E_n = (v_{n-1}, v_n) \text{ and } Concl \in L_V(v_n)\ \}.$$

Note, that each complete path defines exactly one clause for each conclusion attached to the end node of the path.

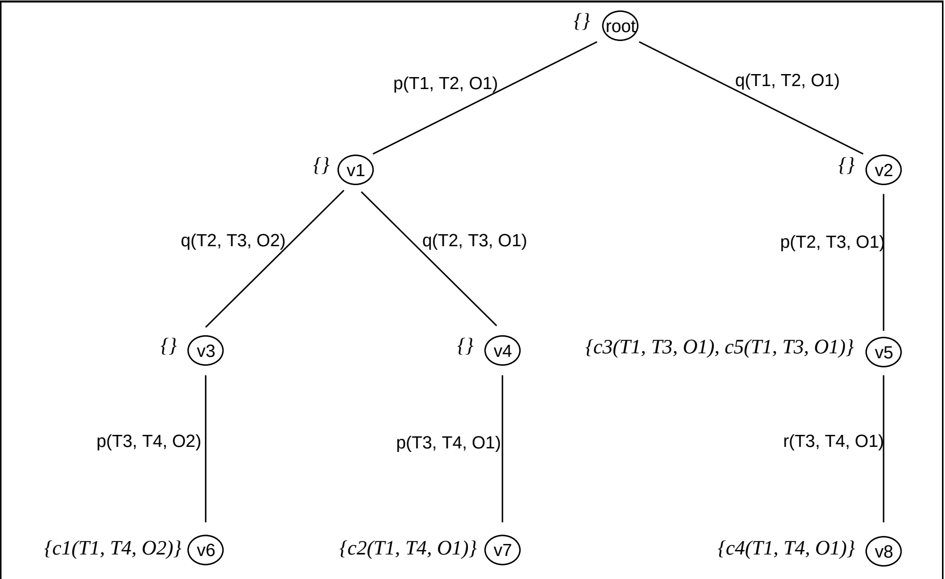
Let $K$ be a set of Horn clauses, where the premises of each clause are ordered. The marker passing tree $G$ representing the clauses $K$ is defined by the following properties:

1. $\forall$complete paths $E \in G \quad \forall C_G \in Clauses(E)$ : $\exists C_K \in K : \quad C_K \Longleftrightarrow C_G$
2. $\forall C_K \in K$ : $\exists$complete path $E \in G \quad \exists C_G \in Clauses(E)$ : $C_G \Longleftrightarrow C_K$
3. the tree has a minimal number of edges.

As a result of the third property, for two clauses $C_1$ and $C_2$, with equivalent first $i$ premises, i.e., $\{C_{1_1}, \ldots, C_{1_i}\} \Longleftrightarrow \{C_{2_1}, \ldots C_{2_i}\}$ the first $i$ edges of the paths of $C_1$ and $C_2$ are the same.

Figure 4 shows an example of a set of clauses and the resulting marker passing tree[4].

[4]We removed some arguments and simplified the predicate names to concentrate on the main aspects.



$\mathsf{p}(\mathsf{T}_1, \mathsf{T}_2, \mathsf{O}_1)\ \&\ \mathsf{q}(\mathsf{T}_2, \mathsf{T}_3, \mathsf{O}_2)\ \&\ \mathsf{p}(\mathsf{T}_3, \mathsf{T}_4, \mathsf{O}_2) \rightarrow \mathsf{c}_1(\mathsf{T}_1, \mathsf{T}_4, \mathsf{O}_2)$
$\mathsf{p}(\mathsf{T}_1, \mathsf{T}_2, \mathsf{O}_1)\ \&\ \mathsf{q}(\mathsf{T}_2, \mathsf{T}_3, \mathsf{O}_1)\ \&\ \mathsf{p}(\mathsf{T}_3, \mathsf{T}_4, \mathsf{O}_1) \rightarrow \mathsf{c}_2(\mathsf{T}_1, \mathsf{T}_4, \mathsf{O}_2)$
$\mathsf{q}(\mathsf{T}_1, \mathsf{T}_2, \mathsf{O}_1)\ \&\ \mathsf{p}(\mathsf{T}_2, \mathsf{T}_3, \mathsf{O}_1) \rightarrow \mathsf{c}_3(\mathsf{T}_1, \mathsf{T}_3, \mathsf{O}_1)$
$\mathsf{q}(\mathsf{T}_1, \mathsf{T}_2, \mathsf{O}_1)\ \&\ \mathsf{p}(\mathsf{T}_2, \mathsf{T}_3, \mathsf{O}_1)\ \&\ \mathsf{r}(\mathsf{T}_3, \mathsf{T}_4, \mathsf{O}_2) \rightarrow \mathsf{c}_4(\mathsf{T}_1, \mathsf{T}_4, \mathsf{O}_2)$
$\mathsf{q}(\mathsf{T}_1, \mathsf{T}_2, \mathsf{O}_1)\ \&\ \mathsf{p}(\mathsf{T}_2, \mathsf{T}_3, \mathsf{O}_1) \rightarrow \mathsf{c}_5(\mathsf{T}_1, \mathsf{T}_3, \mathsf{O}_1)$

Figure 4: Some rules with the corresponding marker passing tree.

## Passing markers

The marker passing method is based on the method of Rieger (Rieger, 1996b; Rieger, 1996c) and differs from her approach only in the way, variables are handled. The basic idea of marker passing is, that each marker marks for the first $i$ premises that are defined by the path from the root node to the marked node that these premises have been proven. The marker passing process reads in one basic feature after the other and incrementally infers the sensor features. Whenever the marker passing process receives a new literal, it tests for all markers whether the marker can be passed to the next node.

Let a marker be a pair ($v \in V$, substitution $\theta$). For each incoming literal $L$ we test for each marker $m = (v, \theta)$ and for all edges $(v, w) \in E$ leaving $v$, whether a substitution $\theta' \supset \theta$ exists, with $L\theta' = L_E\big((v, w)\big)$. If so, we add a new marker $(w, \theta')$. If the new marker is placed on a node labeled by a nonempty set of conclusions, these conclusions will be written to `stdout`. In each case, the old marker will be removed, because the literals must be received exactly in the order of the premises, without any further literals in between. In addition, in each iteration a new marker with an empty substitution list will be placed on the root node to start inference from the beginning.

Figure 5 shows an example of the marker passing process. It starts with the initial state, where one marker is placed on the root node. The first read item, `p(1, 5, 17)`, can be unified with the literal at the left branch of the root node, resulting in moving the marker placed on the root node to node `v1` and a corresponding substitution list. The marker at the root node will be renewed. When reading the next fact, `q(5, 12, 4)`, both current marker can be moved, the one on node `v1` to node `v3`, and the marker on the root node to node `v2`. Again, the root node marker will be renewed. After reading the third input fact, p(12, 17, 4), two markers reache nodes labeled by conclusions, namely `v6` and `v5`. The inference by marker processing can be seen as a special kind of resolution, namely *linear input ordered unit-clause resolution* (Chang and Lee, 1973).

As stated above, the marker passing algorithm of Rieger and the one described here differ in the handling of variables. In this approach, the binding of constants to variables are handled by the substitution lists, consisting of all variables that occur in the path to the current node. Rieger, in contrast, handles the variables by adding constraints

| Input literal | Marker after corresponding input | Generated output |
|---|---|---|
| *Initial state* | m(root, {}) | |
| p(1, 5, 17) | m($v_1$, {$T_1$/1, $T_2$/5, $O_1$/17}) | |
| | m(root, {}) | |
| q(5, 12, 4) | m($v_3$, {$T_1$/1, $T_2$/5, $T_3$/12, $O_1$/17, $O_2$/4}) | |
| | m($v_2$, {$T_1$/5, $T_2$/12, $O_1$/4}) | |
| | m(root, {}) | |
| p(12, 17, 4) | m($v_6$, {$T_1$/1, $T_2$/5, $T_3$/12, $T_4$/17, $O_1$/17, $O_2$/4}) | c1(1, 17, 4) |
| | m($v_5$, {$T_1$/5, $T_2$/12, $T_3$/17, $O_1$/4}) | c3(5, 12, 17), c5(5, 17, 4) |
| | m($v_1$, {$T_1$/12, $T_2$/17, $O_1$/4}) | |
| | m(root, {}) | |
| r(17, 22, 4) | m($v_8$, {$T_1$/5, $T_2$/12, $T_3$/17, $T_4$/22, $O_1$/4, $O_2$/$O_1$}) | c4(5, 22, 4) |
| | m(root, {}) | |

Figure 5: Input literals and correspondingly generated markers.

to the marker, describing the variable bindings, too. These constraints are generated and tested by domain-dependent functions. In contrast to substitution lists, where each variable can only be instantiated once, the constraints can be overwritten in further steps. In this way, the approach can be applied to more general graphs, also containing cycles. On the other hand, Rieger's approach loses the generality to be applied to different domains without adaptation.

## 4.2 Further Inferences

In contrast to the derivations of sensor features, the incoming data on all remaining levels are not ordered. Hence, we cannot remove the data just after processing it, but we have to add it to the knowledge base. This slows down inference because of the increasing size of the knowledge base, so it is important to find situations, in which some of the data can be deleted. In the case of sensor group features, e.g., all gathered data can be removed when the direction changes in which the robot moves, because the assignment of sensors to sensor groups changes.

The inference engines on all further levels are the same: Whenever a new fact is entered, all rules containing a unifyable literal are gathered and the premises of these rules will be tested against the knowledge base. If all premises of a rule can be instantiated, the conclusion will be printed out.

If the input fact is a specific synchronizing signal, all data in the knowledge base with earlier time points than the submitted time will be removed to avoid infinite growing of the knowledge base. The inference engine performs inferences of depth one, and it is able to deal with unrestricted Horn clauses. In contrast to the previously described inference system, this inference engine is not yet completely tailor-made to the task. Possibly, a RETE-like (Forgy, 1982) matcher will be more efficient.

## 4.3 Real World Test

We tested the performance system on two different SUN-clusters. The first one consists of 9 SUN-Sparc-ELC. The second one consists of a single two-processor SUN-Sparc-20. PRIAMOS performed some tasks like passing an open doorway or crossing a doorway, controlled by PRIAMOS' control system or by joy stick. It sends its measurements via radio link to its control system which writes the data to our performance system. The performance system infers concepts at all levels. To visualize the outcome, at each level some of the facts inferred are written to different windows at one of the hosts.

The concepts learned during the off-line learning phase are inferred by the performance system with only a short delay, less than half a second. The ELC-Cluster performed a bit faster than the Sparc-20. The processing will be much faster, if, instead of using a relatively slow inter-processor-connection (ethernet) and a primitive inter-computer communication (just `rsh`-commands), more efficient connections will be used. Due to the few links that are needed for each process (except for the multiplexer and the demultiplexer) and the size of each process, using a real transputer net seems to be adequate, resulting in a considerable speed up of the inference.

## 5 Conclusion

In this paper, we presented a distributed performance system for real-time inference of high level concepts, using multiple small and efficient inference engines instead of a single large one. Therefore, we presented an approach based on the work of Rieger to generate marker passing trees which corresponds to ordered Horn clauses. We sketched real world tests, to verify the efficiency of the approach.

Because of the very simple sensors used for object identification, the classification is limited to simple structured environments and concepts that are easily to discriminate. A closed door, e.g., will never be found by concepts we can learn. This, however, is not a restriction of our approach, but a consequence of the capabilities of the sensors we used. The relational representation allows the integration of further sensor systems by simply adding premises corresponding, e.g., to the results of a vision system. To verify, that the result of this further sensor system is provided just in time, the premises can be linked by the time points. This correlates to the fusion of different sensor features to sensor group features. Since the organization of the different processes in the forward inference system is open, special processes needed for specialized sensors can be problem-free added. Furthermore, time-consuming sensors like vision systems can be activated dependent on the current situation by backward chaining in the same way, as elementary actions can be activated.

As usual, a lot of further work is to be done. The backward inference system, the planning component, and the synchronization between these three systems, is still under development. In addition to object identification dependent on the actions and perceptions of the robot, the whole system should be able to control the robot aided by operational concepts. This control includes the robot's motion and the activation of time-consuming sensors, by aid of operational concepts.

Furthermore, we are interested in distributing not only the first part, up to sensor features, of the inference chain to a cluster of processes. The inferences on the levels above sensor features should be distributed, too. Since the assignment of sensors to sensor groups depends on the direction into which the robot moves, the assignment of data to processes must be dynamic. Again, we don't want to build a very general parallel inference system, but we want to find the characteristics of our representation, to build a dynamically distributed inference system specialized to the characteristics.

## Acknowledgements

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. I want to thank my advisor Katharina Morik for her ideas, suggestions, and critics and Peter Brockhausen for discussing certain topics and reading drafts. The representation and the system is a result of the whole team, namely Katharina Morik, Anke Rieger, Michael Goebel, Stefan Sklorz, Stephanie Wessel, and Eckart Zitzler. Furthermore, I want to thank Michael Kaiser and Frank Wallner from Karlsruhe for their cooperation for the real world test purposes.

## References

Bennett, S. W. (1989). 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.

Brooks, R. A. (1991). The role of learning in autonomous robots. In Valiant, L. G. and Warmuth, M. K., editors, *COLT'91. Proceedings of the fourth Annual Workshop on Computational Learning Theory.*, pages 5–10. Morgan Kaufmann, San Mateo, CA.

Chang, C.-L. and Lee, R. (1973). *Symbolic Logic and Mechanical Theorem Proving.* Academic Press.

DeJong, G. and Bennett, S. (1993). Permissive planning – a machine learning approach to linking internal and external worlds. In *AAAI-93*, pages 508–513.

Dillmann, R., Kaiser, M., and Ude, A. (1995). Acquisition of elementary robot skills from human demonstration. In *International Symposium on Intelligent Robotics Systems*, pages 185–192, Pisa, Italy.

Dillmann, R., Kreuziger, J., and Wallner, F. (1993). 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.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37.

Kaelbling, L. P. (1991). Foundations of learning in autonomous agents. *Robotics and Autonomous Systems*, 8:131–144.

Klingspor, V. and Morik, K. (1995). Towards concept formation grounded on perception and action of a mobile robot. In Rembold, U., Dillmann, R., Hertzberger, L., and Kanade, T., editors, *IAS-4, Proc. of the 4th Intern. Conference on Intelligent Autonomous Systems*, pages 271–278. IOS Press.

Klingspor, V., Morik, K., and Rieger, A. (1996). Learning concepts from sensor data of a mobile robot. *Machine Learning Journal*. to appear. Available at: ftp://ftp-ai.informatik.uni-dortmund.de/pub/publications/klingspor-etal-96.ps.Z.

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

Muggleton, S. (1992). *Inductive Logic Programming*. Number 38 in APIC series. Academic Press, London.

Rieger, A. (1995). Inferring probabilistic automata from sensor data for robot navigation. In Kaiser, M., editor, *Procs. of the 3rd European Workshop on Learning Robots*. Available at: ftp://ftp-ai.informatik.uni-dortmund.de/pub/Reports/report18.ps.Z.

Rieger, A. (1996a). Learning to guide a robot via perceptions. In Ghallab, M., editor, *Procs. of the 3rd European Workshop on Planning*. IOS Press.

Rieger, A. (1996b). MP: An efficent method for calculating the minimum Herbrand model of chain Datalog programs. In Wahlster, W., editor, *Procs. of the 12th European Conference on Artificial Intelligence*. to appear.

Rieger, A. (1996c). Optimizing chain datalog programs and their inference procedures. LS-8 Report No. 20, University of Dortmund, Lehrstuhl Informatik VIII, D-44221 Dortmund.

Segre, A. (1988). *Machine Learning of Robot Assembly Plans*. Kluwer, Boston.

Sklorz, S. (1995). Repräsentation operationaler Begriffe zum Lernen aus Roboter-Sensordaten. Master's thesis, Universität Dortmund. in German.

Sommer, E. (1996). *Theory Restructuring*. PhD thesis, Universität Dortmund.

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

Wessel, S. (1995). Lernen qualitativer Merkmale aus numerischen Robotersensordaten. Master's thesis, Universität Dortmund. in German.