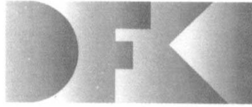

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-92-07

Decision-theoretic Transformational Planning

Michael Beetz

February 1992

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern und Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Daimler Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Krupp-Atlas, Mannesmann-Kienzle, Philips, Sema Group Systems, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- ❑ Intelligent Engineering Systems
- ❑ Intelligent User Interfaces
- ❑ Intelligent Communication Networks
- ❑ Intelligent Cooperative Systems.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Prof. Dr. Gerhard Barth
Director

Decision-theoretic Transformational Planning

Michael Beetz

DFKI-RR-92-07

Part of the work reported here was done during a 1991 summer visit at DFKI Kaiserserslautern.

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9104).

© Deutsches Forschungszentrum für Künstliche Intelligenz 1992

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, Federal Republic of Germany; an acknowledgement of the authors and individual contributors to the work; all applicable portions of this copyright notice. Copying, reproducing, or republishing for any other purpose shall require a licence with payment of fee to Deutsches Forschungszentrum für Künstliche Intelligenz.

Decision-theoretic Transformational Planning

Michael Beetz
Department of Computer Science
Yale University
P.O. Box 2158, Yale Station
New Haven, CT 06520
beetz@cs.yale.edu

February 18, 1992

Abstract

In this paper we develop decision-theoretic transformational planning as a novel computational theory for planning reactive behavior under hard time constraints. The theory is based on three main paradigms: transformational planning, decision theory, and time-dependent computations. Knowledge about goals and the robot control language is accessed through transformation rules that define semantic relationships between constructs in the plan representation language and associations between goals and canned plans. The computational theory deals with uncertainty by applying decision-theoretic methods to control the planning process. The tradeoffs between planning and acting are weighed by applying time-dependent algorithms for testing the applicability and utility of transformation rules with respect to the current situation and the preferences of the robot.

Contents

1 **Introduction** 3

2 **Representational Formalism** 4
  2.1 RPL - A Reactive Plan Language 5
  2.2 Representation of the Robot Control Language 6
    2.2.1 RPL Construct Descriptions 7
    2.2.2 Heuristic Transformation Rules 8
  2.3 Representation of Domain Knowledge 12

3 **Planning Algorithm** 14
  3.1 Generation of Plan Hypotheses 14
  3.2 Test of Plan Hypotheses 18
  3.3 Debugging Plan Hypotheses 18

4 **Conclusion** 20

1 Introduction

In order to compute useful plans for different problem-solving situations, a robot planner has to take into account different optimization criteria, the available computation time, and the uncertainty of the available information. Suppose a robot gets the task to deliver all the blue blocks in the kitchen to the livingroom. The simplest solution for this problem is to go to the kitchen, pick up a blue block, deliver it to the livingroom, and repeat this procedure until no blue block is left in the kitchen. This plan, however, is inappropriate in many cases. Assume, for instance, other robots are around which may steal blocks after their delivery. In this case the robot should lock the livingroom whenever it leaves it. If faster solutions are required, the robot could go to the kitchen first, determine all the locations of the blue blocks, and deliver them in an optimal order. While this solution is faster than others, it is also much more sensitive to changes in the locations of the blocks. This example demonstrates that different plans are preferable under different circumstances. Some plans are considered to be good if robustness is preferred over efficiency, others if reactivity is the most important feature, and still others if certain events in the environment are likely to occur.

In this paper we are concerned with representational and algorithmic problems in the design and implementation of planning systems for mobile robots that generate plans in such a flexible way. For our class of applications we consider planning to be the iterative improvement of a given plan based on expectations about the environment and given preference criteria. A planning system for robots acting in complex and changing environments has to provide solutions for three problems. Firstly, it has to deal explicitly with the uncertainty that arises due to the unpredictability of, and the incomplete information regarding, the environment in which the robot is operating [Han90]. Secondly, the robot planner has to treat planning time as a limited resource; i.e. it has to return plans for any allocation of computation time and has to reason about whether the expected gains from further planning will outweigh the costs of spending more planning time [DB88, BD89]. Finally, the planner has to be able to synthesize plans implementing any problem-solving behavior necessary to solve complex tasks in its environment—not just sequences of primitive robot actions [McD91b].

We develop decision-theoretic transformational planning as a novel computational theory for robot planning under time constraints and in uncertain environments. The theory is based on three main paradigms: transformational planning [Lin90], decision theory [HR90], and time-dependent computations [Dea91]. Knowledge about

the robot control language is accessed through transformation rules which define semantic relationships between constructs in the plan representation language.[1] The computational theory deals with uncertainty by applying decision-theoretic methods to control the planning process. The tradeoffs between planning and acting are weighed by applying time-dependent algorithms for testing the applicability and utility of transformation rules with respect to the current situation and the preferences of the robot.

The computational theory is implemented as the XFRM [McD90] planning system. XFRM consists of a reactive plan interpreter [Fir89, Fir87] and a transformational planner which, controlled by a CPU, simultaneously work on a sketchy plan. Sketchy plans have to be adapted to the current environment in order to be executable and the plan interpreter has to react to changes in the environment which are not handled by the plan. The planner improves the plan while the plan interpreter executes sketchy actions and monitors the execution [McD90, HF90]. The plan, whether it is a descriptive, abstract task or an operational, low-level control statement is formulated in the robot control language RPL [McD91a]. The plan interpreter interprets non-primitive expressions by reducing them using default methods. In this architecture the task of the planner is to replace default methods by more robust, reliable, or efficient code which the planner finds through reasoning about the consequences of actions. The planning system computes better plans when more planning time is available.

The paper is organized in two main sections. Section 2 describes and discusses the representation structures used in the XFRM system and section 3 describes the basic algorithms that interpret the representation structures.

2 Representational Formalism

XFRM has to estimate and compare the quality of alternative plans, optimize the current plan, simulate its execution in order to predict problems, decide whether a simulation corresponds to a successful task execution, and schedule the optimization given some limited computation time. In order to optimize plans it must know how RPL expressions can be transformed and how good the resulting plans are. To evaluate simulations, XFRM needs decision criteria for determining whether or not the execution of a plan achieved the toplevel task. To perform simulations descriptions of

[1] We will use the terms robot control language and plan representation language interchangably depending on whether we want to emphasize the expressiveness or the transparency of the language.

the pre- and postconditions of subtasks are necessary. In order to schedule the planning process XFRM needs methods for assigning resources to optimization problems. In this section we discuss how these different kinds of informations are represented in XFRM.

2.1 RPL - A Reactive Plan Language

RPL (Reactive Plan Language) is the robot control and plan representation language used in the XFRM system. This section is not intended to be a description of RPL, as such a description can be found in [McD91a]. Here we want to discuss the interaction between the design of the robot control language and the planning system; i.e. we discuss which requirements of the planning system are reflected by the language design and how the type of language affects the choice of planning techniques and methods.

```
(achieve-for-all (λ (x) (and (category x block)
                             (color x blue)
                             (in x kitchen)))
                 (λ (x) (in x livingroom)))
```

Figure 1: RPL code for the task *"get all the blue blocks in the kitchen to the livingroom."*

The requirements which have to be fulfilled by the language come from both the plan interpreter and the planning system. From the point of view of the plan interpreter, it is important that the language is expressive enough to describe all sorts of complex problem-solving behaviors that the robot needs to perform. From the planner's point of view, it is important that the language describes plans transparently enough to allow the planner to reason about it. Another planning requirement is that the language should provide primitives for describing robot actions as well as descriptive (predicate-calculus like) high-level tasks. This makes it possible for parts of the plan to be detailed while other parts are left sketchy. In other words, a plan representation language should provide constructs for task specification, high-level algorithmic constructs (like recursion and conditionals), as well as constructs which are closer to the machine and primitive robot actions (like variables, sequentialization, or iteration).

RPL is a Lisp-like programming language for the implementation of reactive robot control programs, which satisfies the requirements stated above. Unlike classical plan languages that only provide sequencing for the combination of robot actions, RPL, as control language for robots that sense and react, provides a rich set of language constructs for the description of problem-solving behavior. RPL allows the definition of robot control programs at different levels of abstraction. Some of its expressions such as, **achieve**, **maintain**, or **prevent** a property of the world are abstract and purely descriptive. Others such as, **wait** five seconds or **fail** are operational and primitive. RPL also provides constructs with a similar semantics, like **try-one-of**, **try-in-order**, or **try-all**; **try-one-of** randomly selects methods from a given set of methods until one of them succeeds, **try-in-order** applies the methods in the specified order, and **try-all** tries the different methods in parallel. Several high-level concepts (interrupts, monitors) are provided that can be used to synchronize parallel actions, to make plans reactive, etc. The language allows for the formulation of sketchy plans, i.e. plans which do not deterministically describe how they should be performed but can be adapted to the current situation at run time.

2.2 Representation of the Robot Control Language

A planning system which optimizes plans must know the semantics of the plan representation language it uses. In classical AI planning systems, which consider sequencing as the only way to construct complex plans from simpler ones, the semantics can be implicitly encoded into the basic planning algorithms such as plan critics or goal achievement procedures. However, it is not promising to implicitly encode the semantics of an expressive language like RPL into the planning algorithms. A planning system for expressive robot control languages needs representation structures that describe the language in an explicit and declarative way.

In XFRM semantical knowledge about the plan representation language is encoded in the form of RPL *construct descriptions* and *heuristic transformation rules*. We discuss these representational structures using the **achieve-for-all** construct as an example. The syntactic structure of the **achieve-for-all** statement is (achieve-for-all (λ (x) (d x)) (λ (x) (g x))). The first argument of the **achieve-for-all** statement specifies the set of objects for which d holds. The second argument describes the goal g that has to be achieved for each of them. The intention of the RPL construct **achieve-for-all** is that all objects which satisfied the property d at a time instant between the begin and end of the **achieve-for-all** task have to satisfy the property g at the end of the task.

2.2.1 RPL Construct Descriptions

The RPL construct description for the achieve-for-all statement is shown in figure 2. An RPL construct description specifies the postconditions that must be achieved for the action to be successful. In addition, it contains a default method that is used by the plan interpreter to reduce the expression, a set of heuristic transformation rules which can be applied in order to optimize and debug the expression, and a scheduler for the control of the rule interpretation.

RPL construct description	achieve-for-all
syntax	(achieve-for-all (λ (?x) ?desc) (λ (?y) ?goal))
postcondition	$\forall$x. $\exists$ti. start $\leq$ ti $\wedge$ ti $\leq$ end $\wedge$ holds(σ(?x$\leftarrow$x)(?desc) ti) $\rightarrow$ holds(σ(?y$\leftarrow$x)(?goal) end)
execution	achieve-for-all.default
generation	{ach-for-all.1 ach-for-all.2 ach-for-all.3}
debugging	{achieve-for-all.debug}
scheduler	achieve-for-all.scheduler

Figure 2: Representation structure for the representation of the RPL command achieve-for-all in XFRM. ?v denotes a pattern variable with name v. $\sigma(exp_1 \leftarrow exp_2)exp$ is the expression that results from the substitution of exp_2 for all occurrences of exp_1 in exp.

postconditions are formalized in a time logic. The postcondition of achieve-for-all states that any object satisfying the description ?desc at an arbitrary time instant ti between the beginning and the end of the achieve-for-all task must satisfy ?goal at the end of the task. A postcondition of an achieve-for-all expression is computed by matching the expression against the pattern in the syntax slot and instantiating the pattern in the postcondition slot using the bindings obtained by the match. The generation slot lists heuristic transformation rules that can be used to make plan hypotheses by transforming the RPL construct. A default-transformation is stored in the slot execution of each nonprimitive RPL construct and can be applied by the plan interpreter if the RPL construct is to be executed reactively. A default transformation is always applicable and should be, in general, the most reliable and robust method for the performance of the RPL construct. The debugging slot contains transformation rules for debugging plans once failures have been detected during execution and

simulation. The scheduler is a function that returns an ordered list of pairs for a given amount of computation time, where the first element in each pair denotes a transformation rule and the second element the maximal time resource for checking the applicability of this rule. The task of the scheduler is to divide the available time resources so that the biggest gains in terms of plan quality can be expected. Scheduling is often necessary because the transformations that produce better plans require more computation time.

2.2.2 Heuristic Transformation Rules

One high-level description of the semantics of RPL is a set of theorems that state the conditions under which expressions in the language are semantically equivalent to, or specializations from, other RPL expressions. Such theorems are good descriptions for a planning system that is intended to transform plans into semantically equivalent ones that are more efficient or robust [BP86]. In transformational systems, these theorems are often represented in the form of transformation rules with an input and output plan schema and an applicability condition. The operational semantics of a transformation rule states that whenever an expression matches the input plan schema and the applicability condition can be proven, then the instantiated output plan schema can be substituted for the expression. A transformation rule is correct if, for all cases for which the applicability condition holds, the input plan schema is semantically equivalent to the output plan schema. Unfortunately, correct transformation rules are of little practical use for robot planning. The information available in robot planning problems is often uncertain and cannot be formalized as premises or facts which are necessary for proving applicability conditions. However, even if we could prove the applicability conditions in principle, those proofs would typically be computationally intractable and could certainly not be derived under real time constraints.

XFRM uses heuristic transformation rules (HTRs) instead of correct transformation rules. HTRs allow a planning system to generate plan hypotheses efficiently using heuristic and associational reasoning. Plan hypotheses are flexibly generated based on given preferences, like efficiency is more important than robustness, and based on expectations about, and changes in, the environment. Since plan hypotheses are generated based on heuristics they are to be tested before being passed to the plan interpreter.

HTRs are approximations of their correct counterparts which terminate fast, or at least propose plans at any time during their interpretation, and improve them over

time. HTRs may also have probabilistic application conditions in order to decide whether or not a transformation should be performed. The problem caused by weakening application conditions in the above ways is that HTRs no longer correspond to semantic equivalences in the robot control language. The plan resulting from a transformation might be related to the original plan in several ways. It might be more or less likely to fail, likely to be slower or faster, or more or less likely to achieve certain conditions of the toplevel task. HTRs transform expressions in order to increase the overall quality of a plan. Different heuristic variants of a correct transformation rule differ in their effects on the robustness, expected runtime efficiency, or degree of correctness of the resulting plan. Consequently, the planner has to decide which of the variants to choose in order to transform a given piece of code. We propose methods which allow for the estimation of the quality of a plan under uncertainty and apply decision-theoretic techniques to choose the most promising transformation.

XFRM evaluates a plan in terms of three qualities, the *robustness*, *efficiency*, and *completeness* of a plan. The ability of the plan to allow the plan interpreter to recover from execution errors and to achieve the postconditions, despite the problems occurred, is called the *robustness* of the plan. From a practical point of view, it is useful to separate robustness into *stability* and *correctness*. Stability is the likelihood that the execution of the RPL expression does not result in an error state from which the system cannot automatically recover. Correctness is the likelihood that the postcondition of an expression will hold once the expression is executed. Since the achievement of the postcondition and the occurrence of an irrecoverable error state are defined as independent events, we can define the robustness of a rplexp as the product of stability(rplexp) and correctness(rplexp).

The robustness of a plan can be increased by constructing alternative courses of actions such that the plan interpreter can choose between them at runtime, depending on the situations the actions should be executed in. Reasoning about disjunctive plans, however, is often very expensive in terms of computational resources. Therefore, the planner should focus on relatively few alternatives and explore others only when necessary, i.e., when none of the alternative ways incorporated in its current plan succeeds. The plan quality *completeness* characterizes this trade-off, representing the likelihood that the planner might find alternative plans which succeed and achieve their postconditions, once the current plan failed. *Execution time* is the third aspect of plan quality which estimates the execution time of a plan based on the execution times of the subplans. Often plans which require less execution time should be preferred by the planner.

Figure 3 shows the default transformation rule for the achieve-for-all statement.

heuristic transformation achieve-for-all.default	
?task ← (achieve-for-all (λ (?x) ?desc) (λ (?y) ?goal)) ↓ (reduce ?task (let* ((desigs nil)) (!= desigs (perceive (λ (x) σ(?x←x) (?desc)))) (if (not (null desigs)) (seq (achieve σ(?y←(car desigs)) (?goal)) ?task) (no-op))))	true

```
?task ← (achieve-for-all (λ (?x) ?desc)
                         (λ (?y) ?goal))
                    ↓
(reduce ?task
  (let* ((desigs nil))
    (!= desigs (perceive (λ (x) σ(?x←x)
                                 (?desc))))
    (if (not (null desigs))
        (seq (achieve σ(?y←(car desigs))
                      (?goal))
             ?task)
        (no-op))))
```

declarations	?p ← (perceive (λ (x) σ(?x←x)(?desc)) ?a ← (achieve σ(?y←(car desigs))(?goal)) ?iter ← (r-how-many ?desc)
stability	$(\theta \times \text{stability}(?p) \times \text{stability}(?a) \times \text{correctness}(?p))^{?iter}$
correctness	$(\text{correctness}(?p) \times \text{correctness}(?a))^{?iter} \times \prod_{i=1}^{?iter-1} \text{Prob}(\neg\text{clipped}(g(a_i)\ [\text{end}(?a(\text{iter } i))\ \text{end}(t)]))$
completeness	completeness(?p) × completeness(?a)
exec-time	?iter × (exec-time(?p) + exec-time(?a))
to-be-optimized	{?p, ?a}

Figure 3: Default transformation rule for the RPL command achieve-for-all

A representation structure for a heuristic transformation rule consists of two major components: a representation of the transformation performed and methods for estimating the quality of the plan resulting from the plan transformation.

In the output plan schema, the perceive expression returns a set of effective designators d for which ((λ (?x) ?desc) d) holds. A designator [McD90] is a data structure which carries the information necessary to resense and manipulate a perceived object. A correct designator is required to perform actions with an object. If the set of designators returned by the perception subtask is nonempty, the task for achieving the goal for the object described by the first designator in desigs is initiated. In the case of a successful achievement, achieve-for-all is recursively called until no object with the property d can be found. Transforming achieve-for-all into a recursive solution

delays the final decision of how to implement the expression and gives the planner time to optimize the recursive call while the interpreter is achieving the goal for the first object. For the subsequent discussion let ?p be the expression (perceive (λ (x) σ(?x←x)(?desc)))) and ?a be the expression (achieve σ(?y←(car desigs))(?goal)).

The planner controls the application of transformation rules based on their expected utility. To compute the utility of a plan using decision-theoretic methods we have to consider all the different timelines that might result from the simulation of the plan and assign utilities and probabilities to them. Clearly, this is not feasible. Therefore we classify the timelines resulting from plan simulations based on whether they contain an irrecoverable error state, whether the postcondition of the plan is satisfied, and based on their average execution time and estimate the probability distribution of timelines over these categories. We can obtain such *a priori* probability distribution by running many sample projections of the plan. The utility of the plan is then defined as the sum of the probability that it succeeds times the utility of success and the average run time of the plan times the utility of efficiency.

This assessment of the plan utility, however, requires running many simulations and is therefore quite expensive in terms of computational resources. For a more efficient generation of plan hypotheses we need fast estimation methods that return rough estimations of the plan quality based the syntactic structure of the plan and prior statistical information about the environment. Our claim is that such estimation methods can be obtained based on the semantics of the plan language, the use of subexpressions, and statistical information. Since we know most of the reasons why a plan might fail we can specify a mathematical model of stability of the plans that are instances of an output plan schema. Such a model is typically a function from subexpressions in the plan, expectations about the environment, and some unknown parameters. The idea is that the unknown parameters can be estimated using the *a priori* statistics of simulations of plan instances of the output plan schema. In the following paragraph we show how such an estimation method can be derived using the stability of the transformation rule achieve-for-all.default as an example.

The subtasks of the output plan schema are ?p, ?g, and ?task. The semantics of the let expression implies that the let expression fails if a subtask for binding local variables or its body fails. Since the output plan schema does not contain code for error recovery, if one of its subtasks runs into an irrecoverable error, then it will do so as well. However, we cannot simply multiply the stabilities of the subtasks because the subtasks interact in various ways and those interactions have to be reflected in the estimation method. These interactions include that ?g almost certainly fails if ?p returns an invalid designator or if the exact location of the object which is part

of the designator changes between the computation of the designator and the use of the designator in a manipulation action. Other interactions might occur due to the effects of ?p and ?g. Often it is a reasonable assumption that execution errors are equally distributed over the iterations. We choose $(\theta \times \text{stability(?p)} \times \text{correctness(?p)} \times \text{stability(?a)})^{?iter}$ as an estimation method for plan resulting from the default transformation. In this estimation method, θ is an *a priori* estimation of the impact of these interactions on the plan stability and the number of iterations has to be estimated given prior information about the number of objects satisfying d.

2.3 Representation of Domain Knowledge

The task of the planning system is to compute plans that achieve, perceive, maintain, or prevent given aspects of the environment. For uncertain environments, it also includes the simulation of a plan execution in typical states of the environment and reasoning about interactions of environmental events with the plan execution. In order to perform these tasks, the planning system needs canned plans for specific tasks, information about the relevant aspects of the environment, and information about how control programs and aspects of the environment interact and relate with each other. An example of a canned plan is the robot control program (deliver <obj> <rm>) that achieves that the object described by the <obj> will be in the room <rm> after the delivery task.

XFRM uses statistical knowledge about the distributions of events in, and aspects of, environment for the computation of expectations such as *"there might be five blue blocks in the kitchen"*. Such knowledge might read: There are typically between four and twenty blocks in the environment. About half of them are usually blue, the rest are red and yellow (equally distributed). Normally, half of the blocks are in the kitchen; the rest are in the bathroom and the livingroom. If another transportation robot is around, blocks stay in a room, on average, about 40 minutes. In addition to statistical information, knowledge about the domain physics is necessary or at least useful. Two examples are that every object is, at any instant of time, in exactly one room and that the robot can only perceive objects which are in the same room. As a result, it is sufficient to run a filter on sensor data in order to decide whether an object is blue, while it is necessary to go into the kitchen in order to find a block that is in the kitchen.

The internal representations of aspects of the environment are predicates. Predicate descriptions are representational structures that provide the planning system with the necessary information about the relevant aspects in the world. Predicate

Predicate Description IN	
syntax	(in ?ob ?rm)
prob-density-fct	(category ?ob block) → (((?rm = kitchen) 0.5) ((?rm = livingroom) 0.3) ((?rm = bathroom) 0.2)) (category ?ob robot) → (...)
avg-lifetime	(avg-lifetime (in ?ob ?rm) ?time) ← ((category ?ob block) ∧ ?time=40min) ∨ ((category ?ob robot) ∧ ?time=2min)
predicate-features	{functional, perception-constraining}
achieve	achieve.in
perceive	perceive.in
prevent	prevent.in
maintain	maintain.in

RPL construct description	achieve.in
syntax	(achieve (in ?ob ?rm))
postcondition	holds((in ?ob ?rm) end)
execution	in-achieve.default
generation	{ach-in.1 ach-in.2}
debugging	{ach-in-dbg.1 ach-in-dbg.2}
scheduler	achieve-in.scheduler

heuristic transformation	in-achieve.default
(achieve (in ?ob ?rm)) ↓ (deliver ?ob ?rm)	true
stability	0.98
correctness	0.99
completeness	0.8
exec-time	5min

Figure 4: Representation structure for the predicate "in" in XFRM.

descriptions associate methods for achieving, perceiving, maintaining, and preventing a certain aspect of the world with the predicate formalizing that aspect. Besides this indexing information, predicate descriptions also contain the statistical information we have discussed in the last paragraph. In the case that the situation in which a plan is to be executed is uncertain, this prior information allows the planning system to construct typical situations in the environment through sampling. Also, if the domain physics is uncertain or only partially known, the planner can generate random events during the simulation given the distribution of environmental events.

Figure 4 shows the predicate description of the predicate in. Information about typical situations in the environment can be inferred from the probability density functions (slot prob-density-fct) which specify the likelihood of aspects of the environ-

ment. Information about the occurrence of events in the environment are stored in the form of average lifetimes of predicates. The slot **predicate-features** specifies that the predicate **in** is a functional predicate, i.e. at any time an object is in exactly one room. It also specifies that the predicate is perception constraining. The slots **achieve**, **perceive**, **prevent**, and **maintain** are references to RPL construct descriptions. The RPL construct description for achieving that an object will be in a certain room is shown in figure 4b and the corresponding default rule in figure 4c.

3 Planning Algorithm

The planning system is designed according to the GENERATE-TEST-DEBUG (GTD) control strategy [SD87] and consists of a PLAN HYPOTHESIZER, a PLAN TESTER, and a PLAN DEBUGGER. The PLAN HYPOTHESIZER proposes plans that are promising under heuristic evaluation. The quality of promising plan hypotheses is evaluated globally by the PLAN TESTER in order to check whether the hypotheses are still promising when the context of the plan and task interactions are considered. Finally, the PLAN DEBUGGER transforms plans in order to avoid possible problems detected by the PLAN TESTER.

3.1 Generation of Plan Hypotheses

The PLAN HYPOTHESIZER transforms plans heuristically based on their syntactic structure. The plans or subplans which would result from the application of these transformations are then rated according to a heuristic evaluation function which estimates their quality. The purpose of the PLAN HYPOTHESIZER is the efficient generation of hypothetical plans which are—with respect to the rough heuristic estimations—significantly better than the plan we started with. In order to achieve the required efficiency, the estimation methods used by the PLAN HYPOTHESIZER are based on numerical estimations of the context sensitivity of tasks. It is obvious that statistical estimation makes hypothesis generation very efficient but also may yield unreliable results.

The basic algorithm for generating plan hypotheses is straightforward. The PLAN HYPOTHESIZER keeps a list of subexpressions that are to be optimized. The list is ordered according to the expected gain in plan quality resulting from the transformation of these subexpressions. The gain in plan quality is approximated by the difference between the quality of the current plan with the fully optimized and the

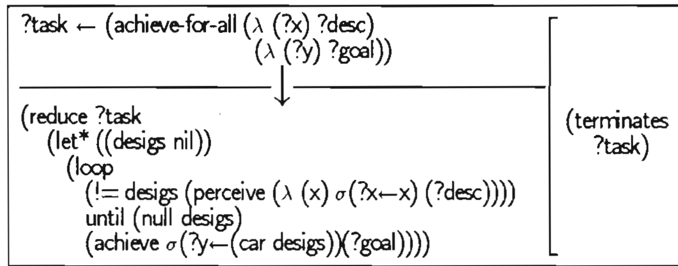

Figure 5: GENERATOR transformation ach-for-all.1

unoptimized subexpression. Ordering planning tasks according to their expected gain in plan quality prevents spending too much computation time on the optimization of unimportant subexpressions.

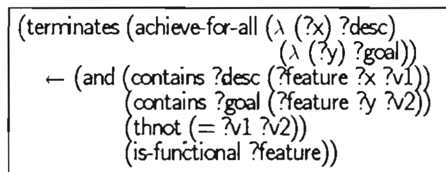

Figure 6: Procedure for proving the termination of an achieve-for-all expression

While there are expressions to be optimized, the PLAN HYPOTHESIZER takes the first one and retrieves the corresponding RPL construct description. Now the default transformation of the construct description is applied, the quality of the default plan estimated, and the result set to be the best transformation computed so far. The next step is to run the scheduler in order to determine the set of transformation rules which should be checked for applicability, their order, and the computational resources for each of them. The transformation rules are checked according to the computed schedule. Checking a transformation rule consists of matching its input plan schema against the expression, checking its applicability condition, and computing the quality of the resulting expression. If the estimated quality is higher than the quality of the best transformation, it is set to be the best transformation so far. The generation

of plan hypotheses is an algorithm that can be interrupted anytime and returns the best transformation computed. Better transformations can be expected if more computation time is given.

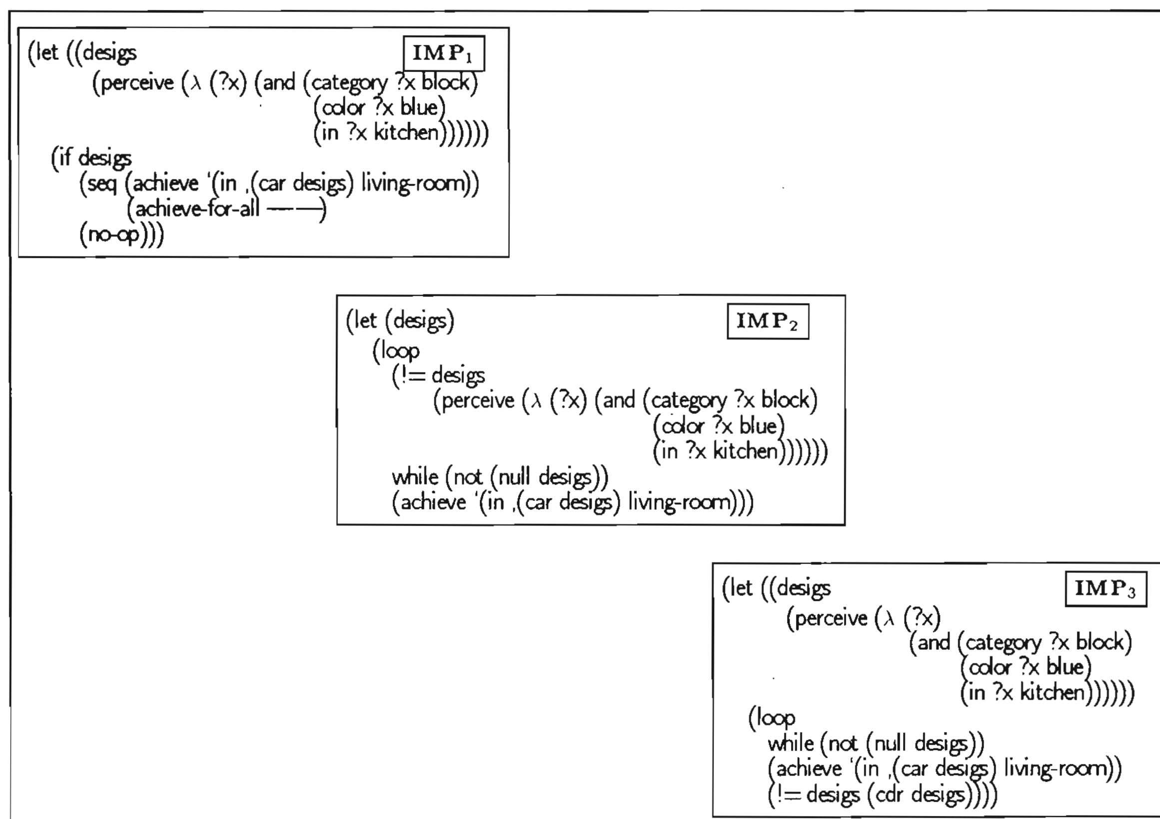

Figure 7: Different implementations for the task *"get all the blue blocks in the kitchen to the livingroom."*

Applicability conditions are checked by a collection of fast special-purpose decision procedures for termination proofs, probabilistic decision rules, and other categories of proofs useful for, and common in, plan transformation. An example for such a decision procedure is the PROLOG-like procedure **terminates** (see figure 6), which checks whether a given **achieve-for-all** expression terminates. The toplevel task in figure 1 terminates because any block that has been delivered to the livingroom

cannot be in the kitchen anymore and thus, the number of blue blocks in the kitchen is decreasing. This common type of termination proofs is implemented by the PROLOG procedure **terminates**, which checks whether there is a feature **?feature** common to all objects matching **?desc** which is functional and provably changed by achieving **?goal**. The information about whether a feature is functional, i.e. whether it can only have one value at a time, is contained in the slot **predicate-features** of predicate descriptions.

Running the default transformation and the **generator** transformation rules of the RPL construct description of **achieve-for-all** the PLAN HYPOTHESIZER proposes three different hypotheses for the implementation of the toplevel task (see figure 7). The first implementation, proposed by the transformation rule **ach-for-all.default** is a recursive solution; the robot senses the kitchen and perceives the blue blocks in the kitchen. If the set of perceived blue blocks is not empty, the robot gets a blue block into the livingroom and calls the **achieve-for-all** task recursively. The second implementation performs a loop consisting of the perception and achievement steps instead of calling **(achieve-for-all — —)** recursively. In the third solution, descriptors for all the blue blocks in the kitchen are computed in advance and then for each of these descriptors the achievement task is executed.

The **achieve** task fails if the robot cannot grasp the block it has to get into the livingroom. This happens if designators have been computed incorrectly or if the location of the block has changed between the computation of the designator and the grasp step. Based on these considerations the planner compares the quality of the different implementations; in the first and second, the robot grasps a blue block immediately after the corresponding designator has been computed and, therefore, the location of the block cannot change in the meantime. In the third implementation, all the designators are computed in advance. Knowing the location of each blue block in the kitchen gives the robot more information for scheduling the delivery. For instance, the robot can deliver the blocks arranged in stacks in a topdown order so to avoid restacking. The difference between the first and the second implementation is that the first implementation delays the decision how to perform the **achieve-for-all** task. It gives the planner time to generate a better plan for **(achieve-for-all — —)** while the robot is already delivering the first blue block. On the other hand, the second implementation does not have to be further planned which frees planning resources for the optimization of other subtasks.

The quality of the plan is the weighted sum of the different aspects of plan quality: stability, correctness, execution time, and completeness. The robot can specify preferences between aspects of plan quality by changing the weights of the aspects. IMP_2, which is proposed by transformation rule **ach-for-all.1**, is preferred over IMP_3 if

the average lifetime of the location of a block is comparatively low and/or the weight assigned to the robustness of a plan is higher than the weight assigned to the runtime efficiency. IMP_1, the plan hypothesis generated by the default transformation, can be returned by the PLAN HYPOTHESIZER if the generation of plan hypotheses has been interrupted before a better hypothesis was generated.

3.2 Test of Plan Hypotheses

The plan proposed by the PLAN HYPOTHESIZER is passed to the PLAN TESTER. The PLAN TESTER is a query component on top of a temporal projection module. It gets a plan hypothesis as its input and generates a set of simulations. Based on these simulations it estimates the stability, the correctness, and the average execution time of the plan hypothesis. In addition, it computes the postconditions that have not been achieved and the tasks that caused irrecoverable errors.

The PLAN TESTER samples projections (simulations) of the current plan hypothesis until it is interrupted or it runs out of time. It performs the following loop: (1) generate a random initial situation according to the probability density functions in the relevant predicate descriptions; (2) project the plan hypothesis for the initial situation. (During the projection, events in the environment are generated randomly according to the average lifetime defined for predicate descriptions.) (3) analyze the projection by checking whether an irrecoverable error occurred, whether the postcondition has been achieved, and how long the simulated execution took.

3.3 Debugging Plan Hypotheses

The PLAN DEBUGGER gets a set of projections of the current plan hypothesis as its input and produces a debugged plan by running the transformation rules in the **debug** slots of the relevant RPL construct descriptions (see [Sim88] for a theory of plan debugging). The debugged version of the plan is passed to the plan tester for further testing. The TEST-DEBUG cycle terminates when the PLAN TESTER module cannot find any more bugs in the hypothesis. Figure 8 shows a debug rule for the achieve-for-all construct. The applicability conditions of debug rules are formalized in XFRM-ML (XFRM Meta Language). XFRM-ML is a PROLOG-like interface to, and an explicit and declarative query language for, plans, execution protocols, and simulations. For efficiency reasons, XFRM-ML operates on basic datastructures like projections, tasknets, and timelines instead of on a PROLOG-like database. Using XFRM-ML, debug transformation rules can retrieve and reason about task failures,

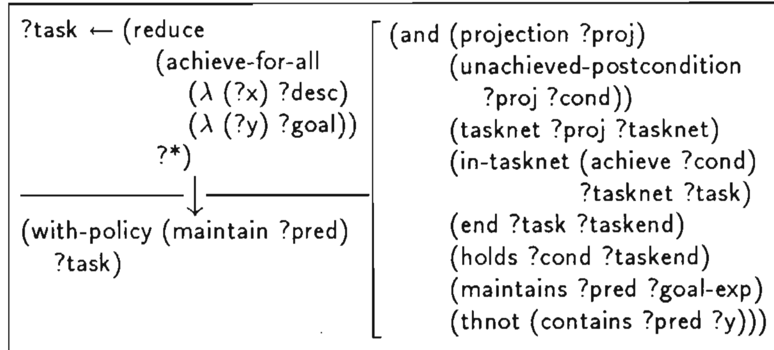

Figure 8: Debug transformation rule for the RPL command achieve-for-all

unachieved postconditions, perceptual confusions, properties of the environment at different stages of plan execution, etc.

XFRM-ML provides a set of predefined predicates on tasknets, timelines and projections. For instance, the query (prolog '(and (projection ?proj) (unachieved-postcondition ?proj ?cond))) succeeds if one of the projections in the current planning cycle did not achieve its postcondition. In this case the query returns a list of bindings, where ?proj is bound to the projection in which the plan could not achieve its postcondition and ?condis bound to the condition which could not be achieved.

Expressions in XFRM-ML are used to specify application conditions of debug rules. The applicability condition of the debug transformation in figure 8 checks first whether there is an unachieved postcondition ?cond, then tests whether the robot has tried to achieve the postcondition, i.e. whether (achieve ?cond) is in the tasknet. If the task is not in the task network the robot has not recognized that !SK318 satisfies (λ (x) (and (category x block) (color x blue) (in x kitchen))) and the perception task needs to be debugged. Next terminates checks whether ?cond holds at the end of (achieve ?cond). If not, (achieve ?cond) needs to be debugged. Otherwise after the goal is achieved it is invalidated by another event after it has been achieved. In this case we compute a predicate that maintains the goal and which is itself easy to maintain. Let us assume that the database contains the fact (maintains (locked ?rm) (in ?x ?rm)). This fact formalizes that no object can appear in, and disappear from, a locked room. The query (maintains ?pred (in !SK318 livingroom)) would succeed and bind ?pred to (locked

livingroom). The debug transformation would return `(with-policy (maintain '(locked livingroom)) ?task)` where **?task** is bound to the current plan. Note, that in order to debug the plan we apply the debug rules of the original abstract task. This is possible because RPL plans contain explicit representations of the transformations performed (**reduce** statement).

4 Conclusion

In this paper we have described a representation scheme and the basic planning algorithms for a robot planner. The planner is implemented as described this paper. Several features of the planner make it attractive for robot planning. The planning algorithm is time-dependent and proposes executable plans for any allocation of computation time. The approach is extendible for expressive robot control languages since planning knowledge is encoded explicitly in the form of transformation rules. And, finally, uncertainty is handled explicitly by associating utility functions with the heuristic transformation rules. These utility functions allow the planner to choose among alternative plans rather than refining an abstrct and provably successful plan.

The paper describes what knowledge is necessary for this category of planners and how to represent and index it. This research, however, is still in an early stage and a lot remains to be done before we have a powerful theory of planning reactive behavior. The most critical issue in this framework are the utility functions for heuristic transformation rules. The basic approach is to learn these estimation methods from running a large number of sample simulations. How to verify learn or at least verify these estimation methods is the next step towards a theory of decision-theoretic transformational plan generation. The ultimate goal is to view these estimations as probably approximate estimations for the quality of a plan computed by sampling simulations. Another important issue is that plan qualities like resource consumption and partial goal fulfillment are not yet taken into consideration.

Acknowledgements. The research reported in this paper has been done under the supervision of, and in collaboration with, with D. McDermott.

References

[BD89] M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proc. of the 11 th IJCAI*, pages 979–984, Detroit, MI, 1989.

[BP86] M. Broy and P. Pepper. Program development as a formal activity. In C. Rich and R. C. Waters, editors, *Artificial Intelligence and Software Engineering*, pages 123–131. 1986.

[DB88] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. of AAAI-88*, pages 49–54, St. Paul, MN, 1988.

[Dea91] T. Dean. Decision-theoretic control of inference for time-critical applications. *International Journal of Intelligent Systems*, 6:417–441, 1991.

[Fir87] J. Firby. An investigation into reactive planning in complex domains. In *Proc. of AAAI-87*, pages 202–206, Seattle, WA, 1987.

[Fir89] J. Firby. *Adaptive Execution in Complex Dynamic Worlds.* Technical report 672, Yale University, Department of Computer Science, 1989.

[Han90] S. Hanks. *Projecting Plans for Uncertain Worlds.* Technical report 756, Yale University, Department of Computer Science, 1990.

[HF90] S. Hanks and J. Firby. Issues and architectures for planning and execution. In K. Sycara, editor, *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 59–70, 1990.

[HR90] P. Haddawy and L. Rendell. Planning and decision theory. *The Knowledge Engineering Review*, 5:15–33, 1990.

[Lin90] T. Linden. Transformational synthesis: An approach to large-scale planning applications. In K. Sycara, editor, *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 126-136, 1990.

[McD90] D. McDermott. Planning reactive behavior: A progress report. In J. Allen, J. Hendler, and A. Tate, editors, *Innovative Approaches to Planning, Scheduling and Control*, pages 450–458, San Mateo, CA, 1990. Kaufmann.

[McD91a] D. McDermott. A reactive plan language. Technical report 864, Yale University, Department of Computer Science, 1991.

[McD91b] D. McDermott. Robot planning. Technical report 861, Yale University, Department of Computer Science, 1991.

[SD87] R. Simmons and R. Davis. Generate, test and debug: Combining associational rules and causal models. In *Proc. of the 10 th IJCAI*, pages 1071–1078, Milan, Italy, 1987.

[Sim88] R. Simmons. A theory of debugging plans and interpretations. In *Proc. of AAAI-88*, pages 94–99, St. Paul, MN, 1988.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-6750 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all publisched papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-90-16
Franz Baader, Werner Nutt: Adding Homomorphisms to Commutative/Monoidal Theories, or: How Algebra Can Help in Equational Unification
25 pages

RR-90-17
Stephan Busemann:
Generalisierte Phasenstrukturgrammatiken und ihre Verwendung zur maschinellen Sprachverarbeitung
114 Seiten

RR-91-01
Franz Baader, Hans-Jürgen Bürckert, Bernhard Nebel, Werner Nutt, Gert Smolka: On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations
20 pages

RR-91-02
Francesco Donini, Bernhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamela, Daniele Nardi, Werner Nutt: The Complexity of Existential Quantification in Concept Languages
22 pages

RR-91-03
B.Hollunder, Franz Baader: Qualifying Number Restrictions in Concept Languages
34 pages

RR-91-04
Harald Trost: X2MORF: A Morphological Component Based on Augmented Two-Level Morphology
19 pages

RR-91-05
Wolfgang Wahlster, Elisabeth André, Winfried Graf, Thomas Rist: Designing Illustrated Texts: How Language Production is Influenced by Graphics Generation.
17 pages

RR-91-06
Elisabeth André, Thomas Rist: Synthesizing Illustrated Documents: A Plan-Based Approach
11 pages

RR-91-07
Günter Neumann, Wolfgang Finkler: A Head-Driven Approach to Incremental and Parallel Generation of Syntactic Structures
13 pages

RR-91-08
Wolfgang Wahlster, Elisabeth André, Som Bandyopadhyay, Winfried Graf, Thomas Rist: WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation
23 pages

RR-91-09
Hans-Jürgen Bürckert, Jürgen Müller, Achim Schupeta: RATMAN and its Relation to Other Multi-Agent Testbeds
31 pages

RR-91-10
Franz Baader, Philipp Hanschke: A Scheme for Integrating Concrete Domains into Concept Languages
31 pages

RR-91-11
Bernhard Nebel: Belief Revision and Default Reasoning: Syntax-Based Approaches
37 pages

RR-91-12
J.Mark Gawron, John Nerbonne, Stanley Peters: The Absorption Principle and E-Type Anaphora
33 pages

RR-91-13
Gert Smolka: Residuation and Guarded Rules for Constraint Logic Programming
17 pages

DFKI Documents

D-91-01
Werner Stein , Michael Sintek: Relfun/X - An Experimental Prolog Implementation of Relfun
48 pages

D-91-02
Jörg P. Müller: Design and Implementation of a Finite Domain Constraint Logic Programming System based on PROLOG with Coroutining
127 pages

D-91-03
Harold Boley, Klaus Elsbernd, Hans-Günther Hein, Thomas Krause: RFM Manual: Compiling RELFUN into the Relational/Functional Machine
43 pages

D-91-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1990
93 Seiten

D-91-06
Gerd Kamp: Entwurf, vergleichende Beschreibung und Integration eines Arbeitsplanerstellungssystems für Drehteile
130 Seiten

D-91-07
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner
TEC-REP: Repräsentation von Geometrie- und Technologieinformationen
70 Seiten

D-91-08
Thomas Krause: Globale Datenflußanalyse und horizontale Compilation der relational-funktionalen Sprache RELFUN
137 Seiten

D-91-09
David Powers, Lary Reeker (Eds.):
Proceedings MLNLO'91 - Machine Learning of Natural Language and Ontology
211 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-91-10
Donald R. Steiner, Jürgen Müller (Eds.):
MAAMAW'91: Pre-Proceedings of the 3rd European Workshop on „Modeling Autonomous Agents and Multi-Agent Worlds"
246 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-91-11
*Thilo C. Horstmann:*Distributed Truth Maintenance
61 pages

D-91-12
Bernd Bachmann:
$H^{iera}C_{on}$ - a Knowledge Representation System with Typed Hierarchies and Constraints
75 pages

D-91-13
International Workshop on Terminological Logics
Organizers: Bernhard Nebel, Christof Peltason, Kai von Luck
131 pages

D-91-14
Erich Achilles, Bernhard Hollunder, Armin Laux, Jörg-Peter Mohren: KRIS : Knowledge Representation and Inference System
- Benutzerhandbuch -
28 Seiten

D-91-15
Harold Boley, Philipp Hanschke, Martin Harm, Knut Hinkelmann, Thomas Labisch, Manfred Meyer, Jörg Müller, Thomas Oltzen, Michael Sintek, Werner Stein, Frank Steinle:
µCAD2NC: A Declarative Lathe-Worplanning Model Transforming CAD-like Geometries into Abstract NC Programs
100 pages

D-91-16
Jörg Thoben, Franz Schmalhofer, Thomas Reinartz:
Wiederholungs-, Varianten- und Neuplanung bei der Fertigung rotationssymmetrischer Drehteile
134 Seiten

D-91-17
Andreas Becker:
Analyse der Planungsverfahren der KI im Hinblick auf ihre Eignung für die Abeitsplanung
86 Seiten

D-91-18
Thomas Reinartz: Definition von Problemklassen im Maschinenbau als eine Begriffsbildungsaufgabe
107 Seiten

D-91-19
Peter Wazinski: Objektlokalisation in graphischen Darstellungen
110 Seiten

RR-91-35
Winfried Graf, Wolfgang Maaß: Constraint-basierte Verarbeitung graphischen Wissens
14 Seiten

RR-92-02
Andreas Dengel, Rainer Bleisinger, Rainer Hoch, Frank Hönes, Frank Fein, Michael Malburg:
Π_{ODA}: The Paper Interface to ODA
53 pages

RR-92-03
Harold Boley:
Extended Logic-plus-Functional Programming
28 pages

RR-92-04
John Nerbonne: Feature-Based Lexicons: An Example and a Comparison to DATR
15 pages

RR-92-05
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Michael Schulte, Rainer Stark:
Feature based Integration of CAD and CAPP
19 pages

RR-92-07
Michael Beetz:
Decision-theoretic Transformational Planning
22 pages

RR-92-08
Gabriele Merziger: Approaches to Abductive Reasoning - An Overview -
46 pages

DFKI Technical Memos

TM-91-01
Jana Köhler: Approaches to the Reuse of Plan Schemata in Planning Formalisms
52 pages

TM-91-02
Knut Hinkelmann: Bidirectional Reasoning of Horn Clause Programs: Transformation and Compilation
20 pages

TM-91-03
Otto Kühn, Marc Linster, Gabriele Schmidt:
Clamping, COKAM, KADS, and OMOS:
The Construction and Operationalization of a KADS Conceptual Model
20 pages

TM-91-04
Harold Boley (Ed.):
A sampler of Relational/Functional Definitions
12 pages

TM-91-05
Jay C. Weber, Andreas Dengel, Rainer Bleisinger:
Theoretical Consideration of Goal Recognition Aspects for Understanding Information in Business Letters
10 pages

TM-91-06
Johannes Stein: Aspects of Cooperating Agents
22 pages

TM-91-08
Munindar P. Singh: Social and Psychological Commitments in Multiagent Systems
11 pages

TM-91-09
Munindar P. Singh: On the Semantics of Protocols Among Distributed Intelligent Agents
18 pages

TM-91-10
Béla Buschauer, Peter Poller, Anne Schauder, Karin Harbusch: Tree Adjoining Grammars mit Unifikation
149 pages

TM-91-11
Peter Wazinski: Generating Spatial Descriptions for Cross-modal References
21 pages

TM-91-12
Klaus Becker, Christoph Klauck, Johannes Schwagereit: FEAT-PATR: Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM
33 Seiten

TM-91-13
Knut Hinkelmann:
Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter
16 pages

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel:
ODA-based modeling for document analysis
14 pages

TM-91-15
Stefan Bussmann: Prototypical Concept Formation
An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang:
Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

DFKI Documents

D-91-01
Werner Stein , Michael Sintek: Relfun/X - An Experimental Prolog Implementation of Relfun
48 pages

D-91-02
Jörg P. Müller: Design and Implementation of a Finite Domain Constraint Logic Programming System based on PROLOG with Coroutining
127 pages

D-91-03
Harold Boley, Klaus Elsbernd, Hans-Günther Hein, Thomas Krause: RFM Manual: Compiling RELFUN into the Relational/Functional Machine
43 pages

D-91-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1990
93 Seiten

D-91-06
Gerd Kamp: Entwurf, vergleichende Beschreibung und Integration eines Arbeitsplanerstellungssystems für Drehteile
130 Seiten

D-91-07
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner
TEC-REP: Repräsentation von Geometrie- und Technologieinformationen
70 Seiten

D-91-08
Thomas Krause: Globale Datenflußanalyse und horizontale Compilation der relational-funktionalen Sprache RELFUN
137 Seiten

D-91-09
David Powers, Lary Reeker (Eds.):
Proceedings MLNLO'91 - Machine Learning of Natural Language and Ontology
211 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-91-10
Donald R. Steiner, Jürgen Müller (Eds.):
MAAMAW'91: Pre-Proceedings of the 3rd European Workshop on „Modeling Autonomous Agents and Multi-Agent Worlds"
246 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-91-11
*Thilo C. Horstmann:*Distributed Truth Maintenance
61 pages

D-91-12
Bernd Bachmann:
$H^{iera}C_{on}$ - a Knowledge Representation System with Typed Hierarchies and Constraints
75 pages

D-91-13
International Workshop on Terminological Logics
Organizers: Bernhard Nebel, Christof Peltason, Kai von Luck
131 pages

D-91-14
Erich Achilles, Bernhard Hollunder, Armin Laux, Jörg-Peter Mohren: KRIS : Knowledge Representation and Inference System
- Benutzerhandbuch -
28 Seiten

D-91-15
Harold Boley, Philipp Hanschke, Martin Harm, Knut Hinkelmann, Thomas Labisch, Manfred Meyer, Jörg Müller, Thomas Oltzen, Michael Sintek, Werner Stein, Frank Steinle:
µCAD2NC: A Declarative Lathe-Worplanning Model Transforming CAD-like Geometries into Abstract NC Programs
100 pages

D-91-16
Jörg Thoben, Franz Schmalhofer, Thomas Reinartz:
Wiederholungs-, Varianten- und Neuplanung bei der Fertigung rotationssymmetrischer Drehteile
134 Seiten

D-91-17
Andreas Becker:
Analyse der Planungsverfahren der KI im Hinblick auf ihre Eignung für die Abeitsplanung
86 Seiten

D-91-18
Thomas Reinartz: Definition von Problemklassen im Maschinenbau als eine Begriffsbildungsaufgabe
107 Seiten

D-91-19
Peter Wazinski: Objektlokalisation in graphischen Darstellungen
110 Seiten

Decision-theoretic Transformational Planning

Michael Beetz

RR-92-07

Research Report