# Consistency Driven Planning

Martin Decker  Guido Moerkotte  Joachim Posegga*

Universität Karlsruhe
Fakultät für Informatik
Postfach 6980, 7500 Karlsruhe, FRG
{moer|posegga}@ira.uka.de

May 7, 1996

**Abstract**

This paper describes a novel approach to linear planning. The presented algorithm is based on a consistency maintaining procedure developed for deductive databases. This allows to handle any range-restricted formula as a goal, and to avoid the well-known problems of linear planning.

The kernel of the planner consists of a deductive database system enhanced by a repair mechanism which allows to generate possible changes an inconsistent database must undergo in order to regain consistency. The generated repairs serve two purposes: as they describe possible worlds in which the goal holds, they can be used to reason about these worlds and eliminate those, which have undesired properties. Moreover, the repairs can be used for driving the actual planning procedure such that the generated plans lead to the selected possible world.

*Part of the research was performed while the author was visiting the Dept. of AI, University of Edinburgh, Scotland. This visit was partly supported by the Commission of European Communities under grant SCI*301.

# 1 Introduction

This paper describes an approach to planning based on deduction techniques. Although deductive approaches to planning have a long tradition in AI (starting with [7, 9]), they are mostly considered as "blue sky research" without practical impact. One might argue that *strips* [4, 5] *is* a successful application of theorem proving to planning, but *strips*'s efficiency relies heavily on the fact that few deduction is actually performed during planning. We would argue that deduction can contribute more to planning than *strips* suggests, without causing hopeless inefficiency.

The basic problem of a logic-based planner is to model *state changes* that result from the execution of actions. Using Situation calculus-like approaches and running a theorem prover to extract a plan from a constructive (refutation) proof is not a good solution. Even very small toy problems cause current state-of-the-art theorem provers to get bogged down in the number of inferences they have to consider. This is chiefly because the frame axioms axioms blow up the search space [2].

Switching to a different logic (e.g. [8, 3, 16]) might be reasonable if one is chiefly interested in reasoning about actions, but it doesn't contribute to the problem of finding plans.

We believe that a deduction-based approach to planning should come up with suggestions for

i) getting around the frame-, qualification and ramification problem, and

ii) providing support for the process of finding a plan.

This paper presents one possible approach based on a repair mechanism which allows to automatically back out detected inconsistencies within a deductive database (see [11]). In order to cope with i), a *decidable* language $L$ is to be used. Frame axioms and long preconditions can then be avoided by computing effects of operators by checking consistency properties of subsequent (possible) worlds. Additionally, we require the description of the initial world to be *complete*.[1] The approach persued in the paper proceeds as follows: Let $S$ be a description of the initial state of the world. Then, if a goal $g$ does not hold in $S$, $S \cup \{g\}$ is inconsistent. In a first step the minimal changes to $S$ resulting in $S'$ are computed such that $S' \cup \{g\}$ is consistent again. These minimal changes give *additional* information subsequently exploited by the planning procedure to select appropriate operators to achieve the state $S'$, and thus the specified goal. Basically, this is the idea of what we call *Consistency Driven Planning*.

Ginsberg and Smith [6] were the first who employed such techniques for planning purposes, but the principle is also well known in database technology [18]. While these approaches allow to reason about the consequences of

[1] Note, that in this context the frame- and the ramification problem are two sides of the same coin.

occuring actions they give little support in finding an actual plan. It is one goal of the paper to overcome this deficiency.

The rest of the paper is organized as follows. In sections 2 and 3 the basic definitions for representing planning domains are given. A well-known example is given in section 4. Here, we describe the household domain, and its representation within a deductive database together with the necessary operator definition and the generated plans. The planning algorithm is introduced in section 5. A short example for the ramification problem and the extended use of deduction is in section 6. Section 7 concludes the paper.

# 2 Deductive Databases

Looking at current research in databases one will notice that the problem of computing possible worlds has its parallel in the area of deductive databases: a database state directly corresponds to a possible world. Hence, the computation of updates in databases after a transaction corresponds to the task of computing the resulting state of the world in a planning scenario after the execution of an action. Usually a deductive database does not offer the expressiveness of full first-order logic, but a restricted language that can be handled efficiently. In deductive databases two major restrictions are made to the *predicate calculus*:

1. In deductive databases only ground atoms – without function symbols – are used.

2. In order to guarantee domain indepencence, the permitted formulas are required to be range-restricted. We use the common definition of range-restrictedness as introduced in [15].

Subsequently, variables are denoted by $x, y, z$, a ground atom, i.e., one which does not contain any variables, is called a *fact*, *Rules* are restricted to definite Horn-clauses. Then, a *database* is a triple $DB := (DB^a, DB^d, DB^c)$ where $DB^a$ is a set of facts, $DB^d$ is a set of closed range-restricted rules, and $DB^c$ is a set of closed range-restricted formulas called consistency constraints (for an example database see section 4). For a database $DB := (DB^a, DB^d, DB^c)$ we define the *extension* $M(DB) := \{a | a\, is\, a\, fact, DB \models a\}$, and the *complete extension* (*completion* for short) as $C(DB) := M(DB) \cup \{\neg a | a\, is\, a\, fact, DB \not\models a\}$, thus incorporating Reiter's closed-world assumption ([17]). For a formula $f$, $C(DB) \models f$ is abbreviated by $DB \models f$. A database $DB := (DB^a, DB^d, DB^c)$ is called *consistent* iff $C(DB) \cup DB^c$ is contradiction free.

For $a_i$ being facts a *transaction* is defined as a finite sequence $TA := o_1(a_1), ..., o_n(a_n)$ where $o_i \in \{add, del\}$. A transaction changes the facts in $DB^a$. In the presentation of the planning algorithm we will make use of the

mapping $\overline{\alpha}$ from the set of all transactions to a set of closed literals defined as $\overline{\alpha}(TA) := \{a | add(a) \in TA\} \cup \{\neg a | del(a) \in TA\}$.

Let TA be a transaction, $DB = (DB^a, DB^d, DB^c)$ a database and $c \in DB^c$ a consistency constraint such that $C(DB) \cup \{c\}$ is inconsistent then $TA$ is called a *repair* for $DB^c$ iff $C(TA(DB)) \cup DB^c$ is consistent and TA is minimal. The main idea for the computation of repairs is to use a derivation tree whose leaves contain the required information. For details the reader is referred to [11, 12, 14]. Concerning the integration of deductive databases and planning, we refer to [13].

# 3 Describing Operators and Goals

In this section the representations of operators and goals as utilized in our planner are described. But first some preliminaries need to be given. The database representing the initial state of the world will be denoted by $DB_0$. Since we require some facts to remain unchanged in all worlds, e.g., within the blocks world we would not allow a block to become a table, protected predicates are introduced. The set of *protected predicates* (denoted by *PP(DB$_0$)*) is defined as $\{is, \equiv\} \cup \{p \mid is(p, protected) \in DB_0^a\}$, and the set of *protected facts* (denoted by *PF(DB$_0$)*) contains all facts in $DB^a$ with a protected predicate. If a derived predicate is a protected predicate, each defining rule for this predicate is restricted to contain only protected predicates.

Now, an *operator specification* is defined as

**declare** op($x_1$,...,$x_n$)
  **range-list** *rl*
  **prec-list** *pl*
  **del-list** *dl*
  **add-list** *al*
**end**

where *op* is the name of the specified operator, the $x_i$ $(1 \leq i \leq n)$ are the variables, *rl* is a list of atoms in protected predicates, *pl* is a list of range-restricted formulas, *dl* and *al* are lists of atoms. Further, *rl*, *pl*, *dl* and *al* must contain only the $x_i$ as free variables. If deduced predicates occur within *dl*, then they must be defined by rules which have exactly one non-protected literal in the rulebody. This restriction avoids non-determinism we wanted to avoid in a first design of our planning procedure (cf. section 6). Additionally, *dl* and *al* must not contain any protected predicates. We call *op($x_1$,...,$x_n$)* the *head* of the operator specification.

An **operator** is a head of an operator specification with all Parameters replaced by constants such that the range-list is satisfied in $DB_0$, and thus in any subsequent state. Then, *P(op)*, *D(op)*, and *A(op)* refer to the instanciated sets of **prec-list**, **del-list**, and **add-list**, respectively.

We now come to the definitions of goal, planning problem, plan, and solution. A *goal* $Z$ is a set of closed range-restricted formulas. For a consistent initial database state $DB_0$, a set of operator specifications $OP$, and a goal $Z$ the triple $(DB_0, OP, Z)$ is called a *planning problem*.

For uniformity reasons, we assume that for a goal $Z$ the operator *goal()* with $P(goal()) := Z$, and with empty $D(goal())$ and $A(goal())$ is implicitly introduced. For a planning problem $(DB_0, OP, Z)$ a *plan* is any sequence $op_1, \ldots, op_n$ of operators defined for $DB_0$, $OP$, and $Z$. Then, each plan defines a unique sequence of database states $DB_0, \ldots, DB_n$:

- $DB_i^a := (DB_{i-1}^a \setminus E(op_i)) \bigcup A(op_i)$
- $DB_i^d := DB_{i-1}^d$
- $DB_i^c := DB_{i-1}^c$

with the *extended deletes E(op)* defined as:

$$E(op) := \{e \mid e \text{ is a fact, f.a. } d \in \mathrm{D(op)}\text{: } DB_0^d \cup \mathrm{PF}(DB_0) \cup \{\neg d\} \models \neg e\},$$

i.e., after deleting the facts in *E(op)* from the database, the completion of the database contains $\{\neg d \mid d \in \mathrm{D(op)}\}$. The only purpose of *E(op)* is to allow deletions not only on $DB^a$ but also on the completion of a database. For an example see section 6.

A plan is a *solution* for a planning problem, iff

- $op_n = goal()$,
- $\forall i \in \{1, \ldots, n\} : C(DB_{i-1}) \models P(op_i)$, and
- $\forall i \in \{1, \ldots, n\} : DB_i$ is consistent.

Thus, before inserting an operator into an operator sequence two things have to be verified:

1. the preconditions of the operator must be satisfied,
2. the resulting world must be consistent.

# 4 The Household Example

In this section we apply the ideas of the previous chapter to the well-known household scenario ([6]) depicted in figure 1. Figure 2 shows the corresponding facts, the rules and constraints are given in figure 3, and verbally read as follows:

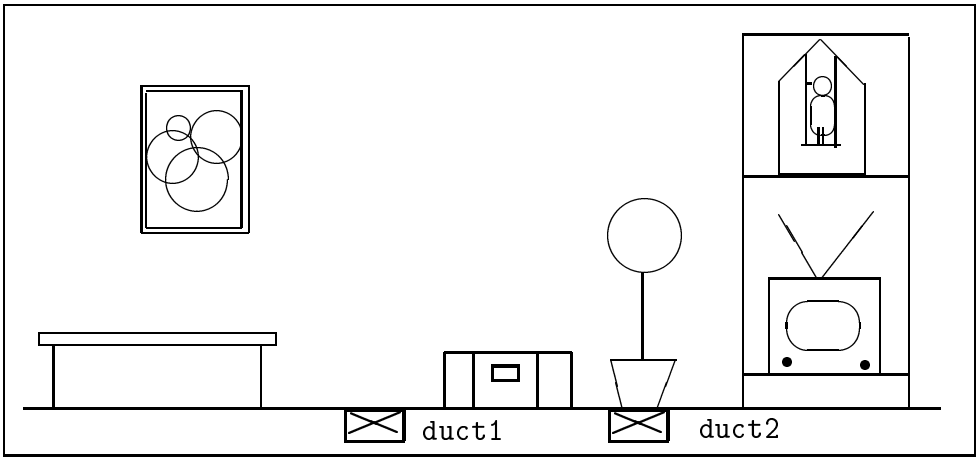


Figure 1: The Household Scenario

blocked_rule: A duct is blocked, if something is placed on it.

obscured_rule: The picture is obscured, if something stands on the table.

stuffy_rule: If both ducts are blocked, the room is stuffy.

loc_constr_1: Everything can be at one place only.

loc_constr_2: Except for the floor, everything can carry only one thing.

loc_constr_3: Nothing can be on something round.

loc_constr_4: Every movable object must be at some location.

loc_constr_5: Nothing can be placed on itself.

stuffy_contraint: A room must not be stuffy.

Assume we want to achieve that the tv is on duct1, and the picture should remain unobscured. This goal is modelled as

$$Z := \{on(tv, duct1), \forall x_1\, is(x_1, picture) \Rightarrow \neg property(x_1, obscured)\}.$$

For this goal the system first generates the repairs shown in figure 4 (cf. [13]). Each repair leads to a different possible world that is consistent and satisfies the goal. The repairs serve two purposes.

1. They give descriptions of the resulting possible states of the world, and
2. they form the basis to generate plans for achieving the goal (see section 5).

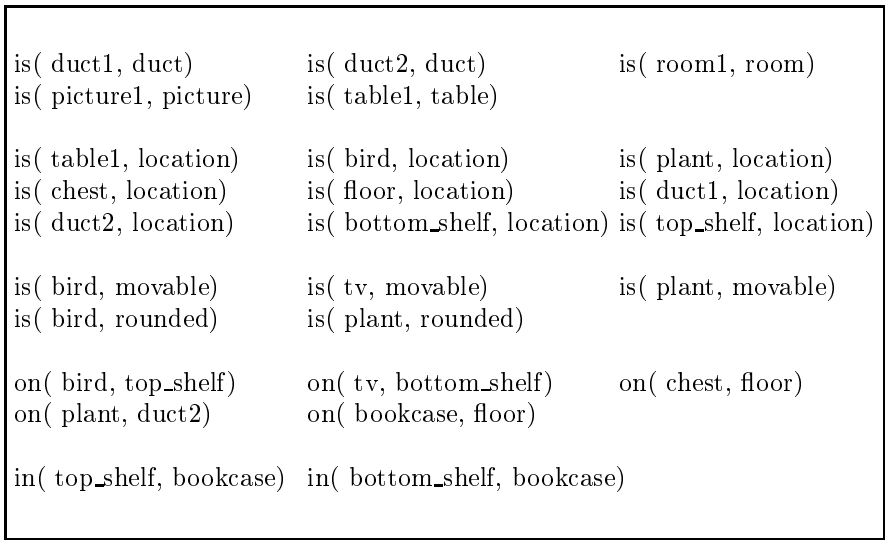


Figure 2: Facts $DB_0^a$ in the Household Domain

Overall, six possible worlds are generated. Actually, the first three repairs differ only in the choice of $l$[2].

In order to complete our example we will give examples of generated plans despite the fact that the actual planning algorithm is presented in the next section. The only operator needed within the household domain is the *move* operator specified as follows:

**declare** move($x_1$,$x_2$,$x_3$)
  **range-list** $is(x_1, movable), is(x_2, location), is(x_3, location)$
  **prec-list** $on(x_1, x_2), \forall z \neg on(z, x_1), \neg(x_2 \equiv x_3)$
  **del-list** $on(x_1, x_2)$
  **add-list** $on(x_1, x_3)$
**end**

Note, that we could replace the common precondition "$x_3$ is clear" by *loc_constr_2*. Another advantage of consistency constraints can be seen when considering the stuffiness property: in the example the room should not be stuffy in any intermediate state. This is easily expressed by the *stuffy_constraint*. Modelling this with preconditions becomes quite awkward.

In principle, the planner works as follows. Consider for instance the repair 2. Matching with the **add-** and **del-list** of the *move* operator leads easily to the following solution:

[ move( plant, duct2, chest );
  move( tv, bottom_shelf, duct1 ); goal() ].

[2]Listing all choices for $l$ explicitly in separate repairs will obviously cause problems in large domains, instead, it would be possible to hold a set of possible instantiations.

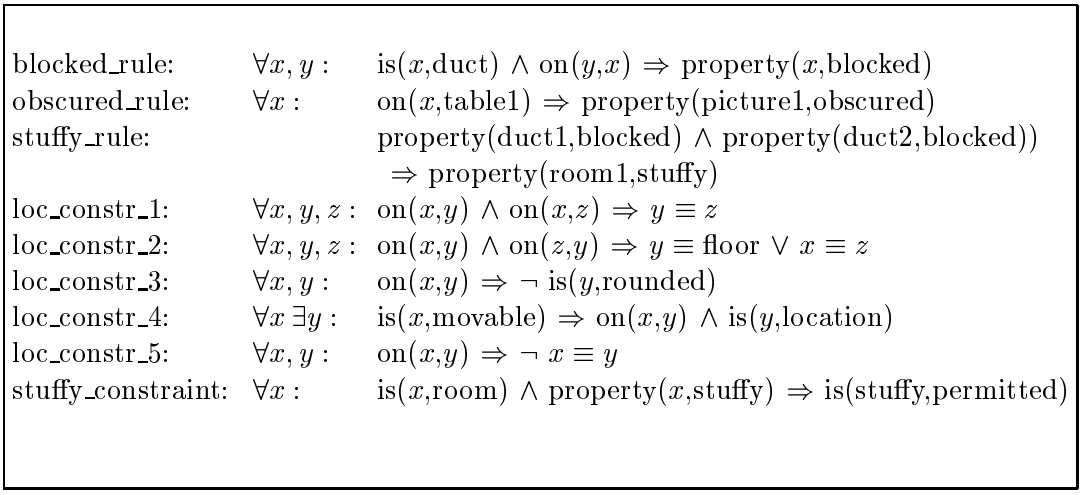


Figure 3: Rules $DB_0^d$ and constraints $DB_0^c$ in the Household Domain

However, things may be more complicated. Consider, for example, the repair 1. Here, all possible *move*-sequences lead to inconsistent intermediate worlds, e.g.,

[ move( tv, bottom_shelf, duct1 );
move( plant, duct2, bottom_shelf ); goal() ]

causes intermediate stuffiness of the world. Due to the *stuffy_constraint* this plan is not acceptable.

Our planning algorithm overcomes this problem and finds the solution

[ move( tv, bottom_shelf, table1 );
move( plant, duct2, bottom_shelf );
move( tv, table1, duct1 ); goal() ].

# 5 The Planning Algorithm

In this section we give the formal definition of our planning algorithm. First, some basic definitions are needed. Let p be a ground literal. We then say that a transaction $TA$ **denies** p, iff

$$(\exists a : add(a) \in TA \wedge p = \neg a) \ \vee \ del(p) \in TA.$$

An operator op **denies** p, iff

$$p \in \ (E(op) \setminus A(op)) \cup \{\neg a \mid a \in A(op)\}.$$

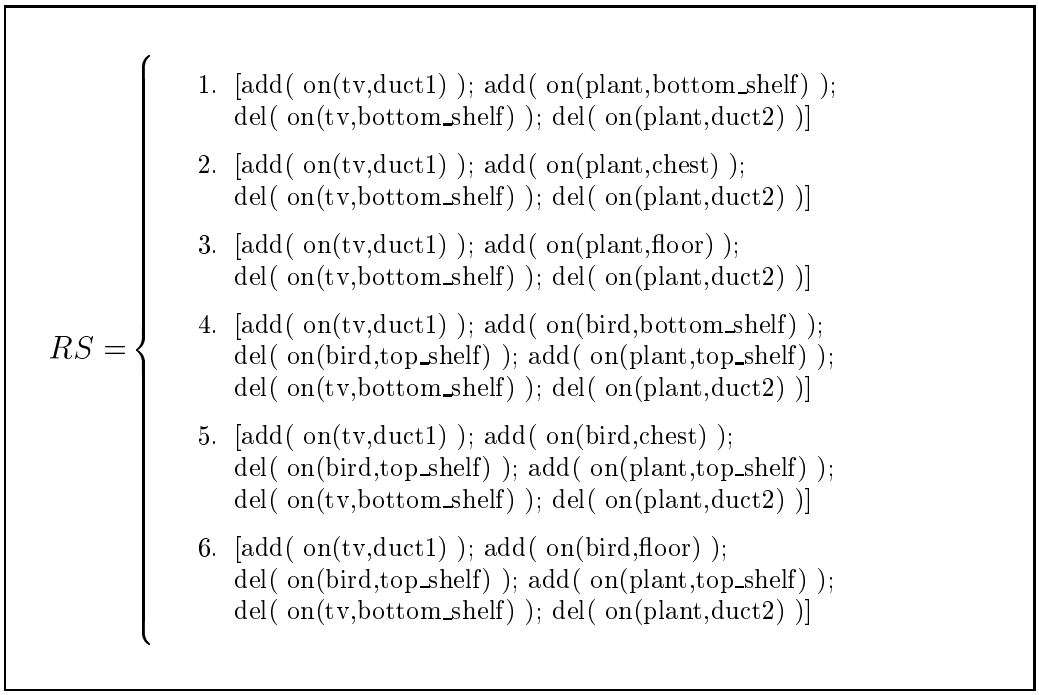


Figure 4: Repairs in the Household Domain

If $LX$ is a list then we denote by $X_i$ the i-th element of the list. In the algorithm list variables carry a capital $L$ as the first letter in their identifiers.

**input:** A planning problem $(DB_0, OP, Z)$

**init:**
1. n=1
2. $Lop = (goal())$ ;;; list of operator instances.
3. $LRS = (\emptyset)$ ;;; list of sets of repairs
4. $LPSGB = LPSGE = (\emptyset)$

The sets of protected subgoals are defined as

$$PSG(i) := \begin{cases} PSGE_n & \text{if } i = n \\ (PSG(i+1) \setminus PSGB_i) \cup PSGE_i & \text{if } 0 < i < n \end{cases}$$

**while:** $\exists f \in [1, \ldots, n]$:

- $(RS_f \neq \emptyset \wedge \forall R \in RS_f : C(DB_{f-1}) \not\models \overline{\alpha}(R))$
- $\bigvee (C(DB_{f-1}) \not\models P(op_f)) \vee$ $DB_f$ is not consistent.

**do:**
1. $f :=$ step-1-strategy( $\{f \mid f$ satisfies the while condition $\}$ ).
2. **if** $RS_f = \emptyset \vee \exists R \in RS_f : C(DB_{f-1}) \models \overline{\alpha}(R)$
   **then** $RS_f :=$ the set of all minimal repair transactions R with
   - $C(R(DB_{f-1})) \models P(op_f)$

- $((R(DB^a_{f-1}) \setminus E(op_f)) \cup A(op_f)$, $DB^d_{f-1}$, $DB^c_{f-1})$ is consistent,
- $\neg\exists g \in PSG(f) : R$ denies g,
- PF$(R(DB_{f-1}))$ =PF$(DB_{f-1})$.

**fi**

3. Let $\{R_1, \ldots, R_k\} = RS_f$.
   For $i \in \{1, \ldots, k\}$ :
   $p_i :=$ step-3-strategy( $\{a \mid a \in \overline{\alpha}(R_i), C(DB_{f-1}) \not\models a\}$ ).
4. chose(*) $p \in \{p_i \mid i \in \{1, \ldots, k\}\}$
   $RS_f := \{R_i \mid p_i = p\}$
5. chose(*) an operator $tt$ with $p \in A(tt) \cup \{\neg e \mid e \in E(tt) \setminus A(tt)\}$.
6. chose(*) a position (an index) $ps \in \{1, \ldots, f\}$ and
   - insert $tt$ as $op_{ps}$ into $Lop$, $n := n + 1$, $f := f + 1$
   - $\emptyset$ is inserted at position $ps$ into LRS, LPSGB, and LPSGE.

   such that the following conditions hold after these 2 points:
   - $ps < f$.
   - $\neg\exists g \in PSG(ps + 1) : op_{ps}$ denies $g$.
   - $\neg\exists u : (ps < u < f) \wedge op_u$ denies $p$.
7. $PSGB_{ps} := \{p\}$, $PSGE_f := PSGE_f \cup \{p\}$.

**output** : the solution $[op_1, \ldots, op_n]$.

**step-1-strategy(F)** : ;;; example for the strategy
**if** $\exists f \in F : RS_f \neq \emptyset$
**then** take the minimal such $f$
**else** take the minimal $f \in F$.
**fi** , return f.

**step-3-strategy(A)** : ;;; example for the strategy
**if** $\exists p_i \in A :$ add$(p_i)$ in $R_i$
**then** take the first add$(p_i)$ of $R_i$ with $p_i \in A$.
**else** take the first del$(q)$ of $R_i$ with $q = \neg p_i$, $p_i \in A$.
**fi**, return $p_i$.

We conclude this section by giving some additional comments on the algorithm: In order to exploit all nondeterministic(*) choices, breath-first search could be used as search strategy. However, this might not be the best choice [3]. Additionally, step-1-strategy or step-3-strategy should be further improved.

Because $E(goal()) = A(goal()) = \emptyset$ holds, the first computed repair set in step 2 are the repairs for $DB^c \cup Z$. Subsequent repairs with $f < n$ are those who

1. satisfy the precondition of the operator, and
2. guarantee the consistency of the resulting world.

[3] In our implementation we used an additional heuristic. Those operators satisfying more subgoals are prefered.

# 6 The Nextto Example

In [10] Lifschitz uses the *nextto* predicate to discuss the ramification problem. Consider a robot walking through different rooms between different locations. Then, the predicate *nextto(x,y)* expresses that some object $x$ is near object $y$. Surely, this predicate is symmetrical. Thus, the specification of a *goto* operator which lets the robot move to some other object $x$ should have $nextto(robot, x)$ and $nextto(x, robot)$ in its add-list. Of course, this is an unsatisfactory solution. We resolve this problem by expressing the symmetric closure of *nextto* by adding two rules to $DB^d$:

1. $\forall x, y : nextto(x, y) \Longrightarrow nextto^*(x, y)$
2. $\forall x, y : nextto(x, y) \Longrightarrow nextto^*(y, x)$

Now, only $nextto(\text{robot}, x)$ has to be in the add-list [4]. In preconditions, consistency constraints and del-lists the derived predicate $nextto^*$ has to be used. Remember the definition of *E(op)*. If $goto(obj1)$ is performed, and we have $nextto^*(obj1, robot)$ in $D(op_2)$ of a subsequent operator $op_2$, with the definition of *E(op)* the program has to deduce

- $\{\forall x, y : nextto(x, y) \Longrightarrow nextto^*(y, x)\ ,\ \neg nextto^*(\text{obj1,robot})\}$
  $\models \neg nextto(\text{robot,obj1})$.

Because in a first step, we wanted to avoid non-determinism of operators and thus required that the defining rules for predicates occurring in the del-list of some operator declaration must have exactly one non-protected literal in its rulebody ($nextto(x, y)$ in the above example).

# 7 Concluding Remarks

We have sketched the integration of a deductive database, together with a system for repairing inconsistencies into a planning algorithm. The database component is used for computing necessary changes to the database for preserving consistency when considering a distinct goal. These so-called repairs can be used for finding plans to achieve the goal. In some cases, the repairs enable the derivation for plans in cases where normal linare planners get struck, e.g., the Sussman Anomaly. Unfortunatly, due to space limitations, we could not demonstrate this in the current paper.

The experiences with the implementation of the introduced planning algorithm made so far are promising and the approach seems to be an encouraging direction for deriving more appropriate deduction components for future planners. Future research will concern the developement of non-linear planners based on the consistency-driven-planning paradigm. Other possible areas

[4] Note that in the add-list the original predicate *nextto* is used.

for further research concern the introduction of disjunctions or even general formulas for the add- and del-lists, and the integration of evaluable function symbols.

# References


[1] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(2):249–267, May 1987.

[2] J. Dix, J. Posegga, and P.H. Schmitt. Modal Logic for AI Planning. In *First Intern. Conf. on Expert Planning Systems*, Brighton, UK, July 1990. IEE. (to appear).

[3] L. Fariñas Del Cerro and A. Herzig. An Automated Modal Logic of Elementary Changes. In Mamdani, Smets, Dubois, and Prade, editors, *Non-standard Logics for Automated Reasoning*, chapter 2, pages 63–79. Academic Press, 1988.

[4] R. Fikes and N. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *Artificial Intelligence*, 2:189 ff., 1971.

[5] R.E. Fikes, , and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[6] M. L. Ginsberg and D. E. Smith. Possible worlds planning I + II. *Artificial Intelligence*, 35, 1988.

[7] C. Green. Application of theorem proving to problem solving. In *Proc. 1st IJCAI*, Washington D.C., 1969.

[8] Patrick J. Hayes. A logic of actions. Technical Report Memo 35, Mathematics Unit, University of Edinburgh, Edinburgh, Scotland, 1971. also in: *Machine Intelligence 6*.

[9] R. Kowalski. Logic for problem solving. DCL TechReport 75, Edinburgh, Scotland, 1974.

[10] Vladimir Lifschitz. On the semantics of STRIPS. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions and Plans, Proceedings of the 1986 Workshop*, Los Altos, CA, 1987. Morgan Kaufmann.

[11] G. Moerkotte. *Inkonsistenzen in deduktiven Datenbanken.* IFB 248. Springer, 1990.

[12] G. Moerkotte and P.C. Lockemann. Reactive consistency control in deductive databases. Internal report 3/90, Universität Karlsruhe, 1990. Also submitted.

[13] G. Moerkotte and J. Posegga. Towards the integration of deductive databases and planning systems. In *4th Workshop on Planning and Configuration*, 1990.

[14] G. Moerkotte and P. Schmitt. Analysis and repair of inconsistencies in deductive databases. *submitted*, 1989.

[15] J.-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18:227–253, 1982.

[16] Hans-Jürgen Ohlbach. A Resolution Calculus for Modal Logics. In Ewing Lusk and Ross Overbeek, editors, *Proceedings of the 9th Conference on Automated Deduction*, Argonne, Ill, May 1988. Springer Verlag.

[17] R. Reiter. On closed world data bases. *In: H. Gallaire And J. Minker (Eds.)*, Logic And Data Bases, Plenum, New York:227–253, 1978.

[18] Marianne Winslett. Theory Revision Semantics for Use in Reasoning about Actions. In *Proc. AAAI-88*, 1988.