Aspects of Consistency Driven Planning *

Guido Moerkotte Holger Müller Joachim Posegga

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

Abstract

A new planning paradigm called consistency driven planning was introduced in [1]. It builds on current deductive database technology, especially on consistency maintenance. This paper extends the approach and discusses several theoretical and practical aspects of consistency driven planning. More specifically, we are discussing the formal properties resulting from the combination of different planning strategies, search strategies, and pruning techniques. Especially, we are interested in the completeness of the different combinations. Some experimental results give first insight into the performance of the various techniques.

1 Introduction

We describe an approach to planning based on deduction. The advantages of deductive systems are well-known; they have clear semantics (namely the semantics of the underlying logic) and are quite well understood from a theoretical point of view. However, in practical applications deduction techniques tend to be rather inefficient, unless applied carefully.

STRIPS [4] and the situation calculus [6] show two extremes of using deduction within a planner. In STRIPS, deduction is only used for checking preconditions of actions, while situation calculus does *everything* by deduction using a theorem prover. The latter does not seem appropriate when one is interested in building usable planners, since small toy problems already overwhelm state-of-the-art provers with inferences [2].

The approach we propose lies between these two extremes. We use deduction for modeling states and transitions between states, but leave the actual search for a plan to a specialized algorithm. This avoids the semantic problems of STRIPS [9] and rules out the most important reasons for inefficiency. Additionally, incompleteness of linear planners, exemplified by the Sussman Anomaly can be avoided. The rest of the introduction informally describes the basic ideas of our approach. Subsequent sections elaborate and formalize these ideas.

*This work was partly supported by Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 314 "Artificial Intelligence and Knowledge-Based Systems", Project D5.

The starting point is a description of an initial world W_0 in a planning scenario (in a suitable first-order language), and a set of domain constraints C that must always hold. If an operator *op* is executed, we want to compute the subsequent world W_1. The first problem to be faced is the frame problem [10]. Ginsberg and Smith [5] offer the following solution: put the action's postconditions into W_1 and copy all formulas from W_0 to W_1. Then delete as few formulas with origin W_0 as possible from W_1 until it becomes consistent with C. The qualification and ramification problem can be treated in a similar way.

Our approach adopts this idea but uses a deductive database for modeling states and a repair mechanism which allows us to automatically detect and remove inconsistencies in the database (see [11, 12]). There are two important advantages over the Ginsberg and Smith approach. First, we avoid the problem that syntactically different but semantically equivalent axiomatizations yield different results [15] by a careful choice of the language's syntax. And, secondly, we not only allow formulas from W_0 to survive in W_1, but also generate new ones if necessary[1]. This is advantageous, because it allows us to compute more information than is given by the original goal *before* the actual planning starts. Thus, we perform a completion of the goal. This completed description gives valuable information for the plan search. Sussman's conjunctive goal problem (see Fig. 1) is used for illustration.

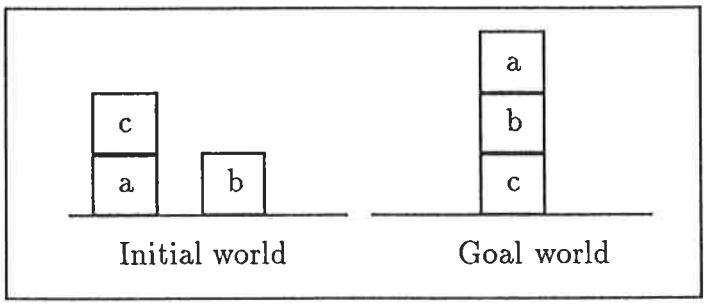


Figure 1: The conjunctive goal problem

W_0 is given by the facts $\{on(a, table).\ on(c, a),\ on(b, table)\}$, the goal G is $\{on(a, b),\ on(b, c)\}$, and C is a set of appropriate domain constraints for the blocks world (like "no object must be at two places", "blocks must not 'fly around' but rest on something that is fixed", etc.) In order to get a complete description of the goal state W_G, we add G to W_0 resulting in

$$W_G = \{on(a, table), on(c, a), on(b, table), on(a, b), on(b, c)\}.$$

This violates several constraints: block 'a' is at two places, thus '$on(a, table)$' must be deleted. For the same reason '$on(b, table)$' must go[2]. We now have $W_G = \{\ on(a,b),\ on(b,c),\ on(c,a)\}$, which is an impossible 'circular' tower, therefore '$on(c, a)$' also disappears. The result is –in this case– a world holding only the goal. However, this still violates a constraint, namely that every object must be at a certain place. Block 'c' is not, so we must find a place for it. The only choice here is *adding* '$on(c, table)$', so finally $W_G = \{on(a, b), on(b, c), on(c, table)\}$, the goal state of figure 1. As it can be seen, we arrived at a complete description of the goal state.

[1]There is of course a price to be paid for that: a *complete* description of the world is required.

[2]Recall that only formulas coming from W_0 may be deleted.

To summarize, the key idea of our approach is to borrow ideas from deductive database technology for modeling worlds in planning and to use techniques for detecting and resolving inconsistencies to model transitions from one world to another. By these means, we compute a complete description of a possible goal world and use this information to guide a dedicated (linear) planning algorithm. As the main driving force is regaining consistency, we call our approach **Consistency Driven Planning (CDP)**.

The rest of the paper is organized as follows. In section 2, we show how planning domains can be formalized using the CDP approach. The repair mechanism used to achieve consistent worlds is presented in section 3. Major properties of the repair mechanism are stated also. Section 4 includes the basic planning algorithm and several techniques which allow to prune parts of the search tree. The completeness of the planner is proven in 5. Some empirical results for different domains are presented in section 6. Section 7 concludes the paper.

2 Formal Representation of Domains

The description of domains and states of domains is based on a subset of the language of the first-order predicate calculus. Therefore, we repeat some of the basic definitions. A **signature** Σ is a pair $\langle K_\Sigma, (P_{\Sigma,i})_i \rangle$ where K_Σ is a set of constant symbols usually denoted by small letters $a, b, c, \ldots$ and $(P_{\Sigma,i})_i$ is a family of predicate symbols. V is a set of variables which are denoted by small letters from the end of the alphabet. The set of atoms $Atom_\Sigma$ is defined as $\{p(t_1, \ldots, t_n) \mid t_i \in K_\Sigma \cup V \ \& \ p \in P_{\Sigma,n}\}$. The set of literals Lit_Σ is the union of all atoms and their negations. A literal is called **ground** if it contains no variables. A **fact** is a ground atom. The complement operator ^{c} changes the signs of a set of literals $L \subset Lit_\Sigma$: $L^c = \{\neg a | a \in Lit_\Sigma \cap Atom_\Sigma\} \cup \{a | \neg a \in Lit \setminus Atom_\Sigma\}$. The set of formulas For_Σ is the smallest set obeying the following inductive definition:

- all atoms are formulas
- if f_1 and f_2 are formulas and x is a variable then $\neg f_1$, $f_1 \vee f_2$, $f_1 \wedge f_2$, $f_1 \Rightarrow f_2$, $\forall x f_1$, and $\exists x f_1$ are formulas.

Deviating from classical logic, we restrict the definition of the consequence relation $\models \subset For_\Sigma \times For_\Sigma$ to models I where every element of universe of I is named by exactly one constant $c \in K_\Sigma$. Thus, we assume that each model satisfies the domain closure axiom and the unique names assumption. In other words, in the subsequent formalization of domains each object of the domain is represented by exactly one constant.

A universally quantified formula f of the form $\forall x_1 \ldots \forall x_k (a_1, \ldots, a_n \Rightarrow a_0)$ where $a_0, \ldots, a_n$ are atoms is called a **rule**. a_0 is the **head** of the rule f and $a_1, \ldots, a_n$ is the body of f. A predicate which occurs in a head of at least one rule is a **derived** predicate otherwise it is a **base** predicate. In the following, we assume that all formulas are range-restricted[3]. Range restricted formulas are domain independent, i. e., the truth value of a formula does not depend on the selected signature Σ.

Different aspects of a planning domain can be represented by different elements of the CDP-language. Properties which should hold but which could be violated by operators

[3]The notion of range-restrictedness was introduced in [13]

are formulated by consistency constraints and properties which express universal relations are described by rules. The actual state of a domain is represented by a set of facts.

Definition 2.1 (possible worlds) *A* **possible world** *is a triple* $\langle W_{facts}, W_{rules}, W_{constr} \rangle$ *where:*

- W_{facts} *is a set of ground atoms*
- W_{rules} *is a set of range restricted rules*
- W_{constr} *is a set of range restricted formulas called* **consistency constraints**[4]

and no fact of W_{facts} *has a derived predicate.*

Obviously, a possible world can directly be represented by deductive database. In order to model constant parts of a domain, **protected predicates** are introduced whose corresponding facts may not be affected by any action. For example, information concerning the 'type' of a constant can be described by a protected predicate *is*. A fact $is(a, block)$ in the blocks world domain describes the unchangeable property of a to be a block. A possible world $W = \langle W_{facts}, W_{rules}, W_{constr} \rangle$ describes the state of a domain by its **completion** $comp(W)$:

$$comp(W) := \{a | a \text{ is a fact}, W_{facts} \cup W_{rules} \models a\} \cup \{\neg a | a \text{ is a fact}, W_{facts} \cup W_{rules} \not\models a\}$$

By incorporating the closed-world assumption, we arrive at a complete description of the state. For a formula f (or a set of formulas F), $Comp(W) \models f\ (F)$ is abbreviated by $W \models f\ (F)$.

Definition 2.2 (consistent worlds) *A world* $W = \langle W_{facts}, W_{rules}, W_{constr} \rangle$ *is* **consistent** *iff* $\text{comp}(W) \cup W_{constr}$ *is logically consistent*

Because of the modified definition of the consequence relation $\models$ the defining condition of 2.2 is equivalent to the assertion $comp(W) \models W_{constr}$.

The change of a world is described by a **world transition** $wt \subset \{add(l), del(l) \mid l \text{ ground atom}\}$:

$$wt(\langle W_{facts}, W_{rules}, W_{constr} \rangle) := \langle (W_{facts} \setminus \{l \mid del(l) \in wt\} \cup \{l \mid add(l) \in wt\}, W_{rules}, W_{constr}) \rangle$$

In the terminology of database systems world transitions are simply transactions which operate on databases. The function Γ maps a world transition wt to a set of literals $\{l \mid add(l) \in wt\} \cup \{\neg l \mid del(l) \in wt\}$. Since Γ is bijective $\Gamma(wt)$ can be seen as another representation of wt. Subsequently, we will use both forms for the representation of world transitions. The Δ-operator describes the difference between two sets of literals:

$$\Delta(L, L') := \{l \mid l \in L' \setminus L\} \cup \{l^c \mid l \in L \setminus L'\}$$

Δ_{facts} yields the difference between the set of facts of two possible worlds and Δ_{comp} between the completion of both.

World transitions are used to describe the effects which can be achieved by an agent in a domain. Since an agent cannot handle every object in every state of the domain the functionality of agents are described by operator specifications:

[4] a synonymous term would be **integrity constraints**

Definition 2.3 (operator specifications) *An* **operator specification** *is defined as*

declare $opsym(x_1,\ldots,x_n)$
Range $F_{range} \subset \mathrm{For}_\Sigma$
Pre $F_{pre} \subset \mathrm{For}_\Sigma$
Post $L_{post} \subset \mathrm{Lit}_\Sigma$
end

where opsym is the name of the operator and the parameters $x_1,\ldots,x_n$ are variables. All predicates of F_{range} have to be protected and L_{post} must not contain any derived predicate. The formulas of F_{range}, F_{pre}, and L_{post} may contain only $x_1,\ldots,x_n$ as free variables. The formulas $\exists x_1,\ldots,\exists x_n \bigwedge_{f\in F_{range}} f$ and $\bigwedge_{f\in F_{pre}} f\sigma$ where σ is a ground substitution for $x_1,\ldots,x_n$ have to be range restricted.

An **operator** *op* is a head of an operator specification, where all parameters $x_1,\ldots,x_n$ are replaced by constants. *range*(*op*), *pre*(*op*) and *post*(*op*) refer to the fully instantiated sets of F_{range}, F_{pre}, and L_{post}, respectively. An operator *op* is a **valid instantiation** of an operator specification with respect to world W if $W \models range(op)$.

The range restrictedness of the formulas guarantees that the truth value of the range and the precondition of an operator *op* for a possible world W depends only on W_{facts} and W_{rules} and not on the signature Σ. Since the range of a specification contains only protected predicates the range *range*(*op*) of an operator *op* is valid either in all possible worlds of a domain or in none. Therefore the range specifies the domain or 'type' of the parameters of an operator. Whereas the precondition specifies those possible worlds in which an operator is applicable. If we use deductive databases to represent possible worlds, we can use the answer mechanism of the deductive database system to determine all valid instantiations by the usage of queries representing the range of the specification.

$op(W) = \Gamma^{-1}(post(op))(W)$ is the world which results from the application (or execution) of an operator *op* on a world W. Since the postcondition of an operator contains no derived predicates the postcondition holds after the execution: $op(W) \models post(op)$[5] A **plan** is a sequence of operators. The application of a plan $[op_1,\ldots,op_n]$ on a world W is defined by the application of the concatenated operators $op_n \circ \cdots \circ op_1$ on W. Each plan $[op_1,\ldots,op_n]$ defines a unique sequence of possible worlds $W_1,\ldots,W_n$:

- $W_{facts,i} := (W_{facts,i-1} \setminus Post^-(op_i)) \cup Post^+(op_i)$

whereas $W_{rules,i} = W_{rules} = W_{rules,0}$ and $W_{constr,i} = W_{constr} = W_{constr,0}$ remain unchanged. An operator op_i is called an **establisher** of a literal l for a world W_j if $l \in post(op)$, $W_{i-1} \not\models l$, and $W_k \models l$ for each $k = i,\ldots,j$. In the case of $l^c \in post(op)$ and $W_{i-1} \models l$ the operator op_i is a **destroyer** of l.

Finally, with the definitions above, the two main notions of planning can be formalized: planning problems and their solutions. A **planning problem** is a triple $\langle W_0, OP, G\rangle$, where W_0 is a consistent initial world, OP a set of operator specifications, and G is

[5] If we allowed derived predicates in the postcondition, we would have to select between two difficulties. Firstly, if we had not changed the above definition of operator application, the deletion of a derived fact f might not result in a world in which $\neg f$ holds. Secondly, in order to delete f in any case several other facts which allow the derivation of f may also be deleted from W_{facts}. In this case there may be several ways to achieve this goal, i.e., the operator application would yield several possible successor worlds.

a set of closed, range-restricted formulas called the **goal**. For uniformity reasons, we assume that a goal G implicitly introduces an operator $goal()$, where $pre(goal()) := G$ and $post(goal())$ is empty.

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

1. $op_n = goal()$,
2. $\forall i \in \{1, \ldots, n\} : Comp(W_{i-1}) \models Pre(op_i)$, and
3. $\forall i \in \{1, \ldots, n\} : W_i$ is consistent.

In terms of planning, this means that before inserting an operator into an operator sequence two things have to be verified: (i) the preconditions of the operator must be satisfied, and (ii) the resulting world must be consistent. An operator which violates one of these conditions is called **critical**.

A plan $P = [op_1, \ldots, op_n]$ is a **subplan** of a plan $P' = [op'_1, \ldots, op'_{n'}]$ $(P \subset P')$ if there exists a monotonic and injective function π such that $op'_i = op_{\pi(i)}$ for each $i \in \{1, \ldots, n\}$. A solution *plan* is **optimal** if no subplan of *plan* is a solution.

The blocks world is used to illustrate the definitions above. A complete axiomatization is presented in figure 2. The rules and constraints $W_{rules,0}$ and $W_{constr,0}$ verbally read as follows:

support-rule-1:	an object which stands on another one is supported by that object
support-rule-2:	every object supports all objects that stand on it
constr-1:	everything can be at one place only
constr-2:	only one block can be on another
constr-3:	every block is supported by the table
constr-4:	only blocks can stand on other objects

3 Achieving Consistent Worlds

In this section, we mainly summarize the results of [12]. In the last part, we proof a key property of the repair system which lay the foundation for the completeness proofs of the various pruning techniques of section 4.

Definition 3.1 (symptoms and repairs) *Let* $W = \langle W_{facts}, W_{rules}, W_{constr} \rangle$ *be a possible world,* $F \in \text{For}_\Sigma$ *with* $W \not\models F$, $S \subseteq Lit_\Sigma$ *a set of ground literals, and* wt *a world transition.*

1. S *is a* **symptom**[6] *for* F *iff* $(Comp(W) \cup S) \setminus S^c \models F$ *and* S *minimal*
2. wt *is a* **repair** *for* F *iff* $\text{wt}(W) \models F$ *and* wt *minimal*

[6] In [12] the expression $(Comp(W) \setminus S) \cup S^c \models \{f\}$ is used to define symptoms. In order to represent symptoms and world transitions in a similar way, we have changed the original definition.

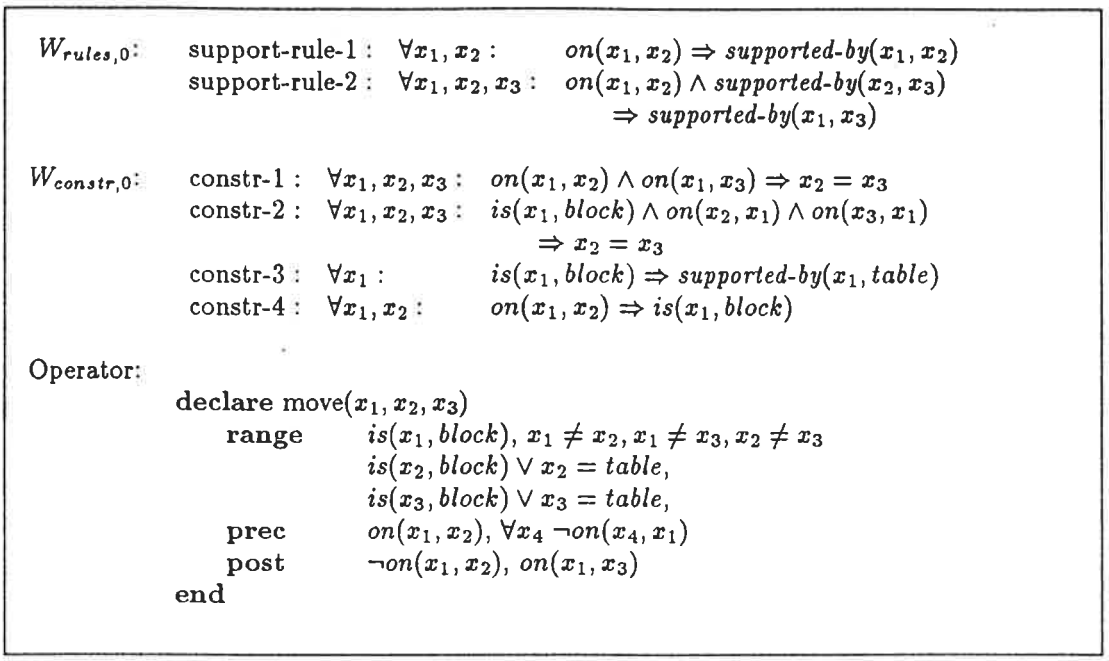


Figure 2: Formalization of the blocks world

Repairs are computed by a loop which attempts to achieve at least one invalid formula in each iteration. Repairs (or symptoms) which make a single $f \in F$ valid are called **potential**. If they make all formulas of F valid they are called **definite**. The computation of definite repairs can be divided in four levels where the higher levels depend on the result of the lower ones.

1. definite repairs
$\Uparrow$
2. potential repairs
$\Uparrow$
3. repairs for ground literals
$\Uparrow$
4. potential symptoms

Definition 3.2 (Symp(W,f)) *Let $f \in \mathrm{For}_\Sigma$ be a formula in prenex normal form and W a possible world.*

- *If $f = f_1 \wedge \ldots \wedge f_n$ with literals f_i then*
  $\mathrm{Symp}(W, f) = \{\{f_j | W \not\models f_j, 1 \leq j \leq n\}\}$;
- *If $f = f_1 \vee \ldots \vee f_k$ then*
  $\mathrm{Symp}(W, f) = \bigcup\{\mathrm{Symp}(W, f_i) | W \not\models f_i, 1 \leq i \leq k\}$;
- *If $f = \forall x f_0(x)$ then*
  $\mathrm{Symp}(W, f) = \{\bigcup\{g(c) | c \in K_\Sigma, W \not\models f_0[x \leftarrow c]\}|$ *where g is a function defined on K_Σ, such that for all $c \in K_\Sigma$, $g(c) \in \mathrm{Symp}(W, f_0(c/x))\}$;*
- *If $f = \exists x f_0(x)$ then*
  $\mathrm{Symp}(W, f) = \bigcup\{\mathrm{Symp}(W, f_0(c/x) | c \in K_\Sigma\}$.

The following theorem shows that *Symp(f)* contains all symptoms of f for a possible world W. The set of all symptoms can be obtained by considering only those elements of $Symp(W, f)$ which are minimal with respect to set-inclusion.

Theorem 3.3 *If* $(Comp(W) \cup L) \setminus L^c \models f$, *then there is an* $S \in \text{Symp}(W, f)$ *with* $S \subseteq L$.

Symptoms may contain ground literals of derived predicates. In order to insert or delete derived ground literals, we have to add or delete an appropriate set of base facts. The function G maps a ground literal l and a possible world to a set of repairs which achieve the literal. The main idea for computation of these repairs is to use a special derivation tree for l whose leaves contain the required information. A derivation tree is an AND/OR graph where the OR-nodes are ground literals and the AND-nodes are ground matrizes of the rules. The successors of an OR-node l are AND-nodes which represent the different ways how l can be derived. The successors of an AND-node r are OR-nodes containing the atoms of the body of r. Derivation trees represent all possibilities to derive l using input resolution as the only inference rule. Figure 3 shows the derivation tree of *supported-by(c,table)* for the blocks world formalization. The predicate *supported-by* is abbreviated by *s-b*.

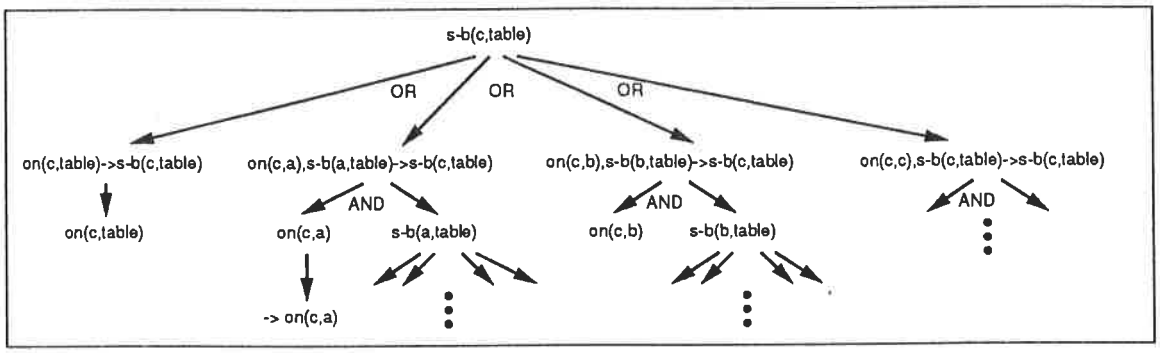


Figure 3: A derivation tree

Through the deletion of all subtrees of which the root has a predecessor containing the same element as the root, the derivation tree becomes finite. Now, the leaves of the tree contain the necessary information. The addition of a fact l requires the insertion of heads of OR-leaves (i. e., facts which do not hold in the original world). To achieve a negated fact some facts of AND-leaves must be removed from the original world. The function G shifts the extracted facts from the leaves to the root. At each inner node G combines the partial repairs of the corresponding subtree with respect to the type of the node and the sign of the root literal. The main property of G is stated by the following theorem.

Theorem 3.4 (correctness and completeness of G) *Let W be a possible world and l a ground literal with $W \not\models l$.*

$$\text{wt } \textit{is a repair of } l \textit{ for } W \quad \textit{iff} \quad \text{wt} \in G(W, l)$$

The function PR combines *Symp* and G in order to generate a set of potential repairs:

$$PR(W, f) := \{ \bigcup_{i=1,\ldots n} R_i \mid \{l_1, \ldots, l_n\} \in Symp(W, f) \;\&\; R_i \in G(W, l_i)\}$$

Definite repairs are generated by iterating over already generated potential repairs:

$$\begin{array}{lcll} repairs_0(W,F) & = & \{ \; R \in PR(W,f) & \mid \; f \in F \;\&\; R \text{ free of contradictions}\} \\ repairs_{i+1}(W,F) & = & \{ \; R_i \cup R & \mid \; R_i \in repairs_i(W,F) \;\& \\ & & & \exists f \in F : R \in PR(R_i(W),f) \;\& \\ & & & R_i \cup R \text{ free of contradictions}\} \end{array}$$

After a finite number of steps further iterations do not cause any changes, i. e., the set $repairs_i(W,F)$ contains only definite repairs.

The following theorem is essential for the completeness of the CDP-planner and its various pruning techniques. It shows that for each world W' in which $F \subset For_\Sigma$ holds, $repairs_i(W,F)$ contains a (potential or definite) repair which partly describes the difference between W and W'.

Theorem 3.5 *Let W, W' be possible worlds, $F \subset \text{For}_\Sigma$, and $W \not\models F$ and $W' \models F$.*

$$\forall i : \exists R \in \text{repairs}_i(W,F) : \quad R \subset \Delta_{facts}(W,W')$$

Proof: By induction on the number of iterations i.

We first show the theorem for $i = 0$. Let $f \in F$ be a formula not holding in W. By theorem 3.3 $Symp(W,f)$ has a symptom $S = \{l_1,\ldots,l_n\}$ which is a subset of $\Delta_{comp}(W,W')$. Since $W \models f$, there exists a repair $R_i \subset \Delta_{facts}(W,W')$ for each l_i. By theorem 3.4 $R_i \in G(W,l)$ and therefore $\cup_i R_i \in PR(W,f)$. Since $\cup_i R_i \subset \Delta_{facts}(W,W')$ the set $\cup_i R_i$ is free of contradictions. Thus $\cup_i R_i \in repairs_0(W,F)$.

If $repairs_i(W,F)$ has a definite R being a subset of $\Delta_{facts}(W,W')$ the claim is obvious for $i+1$. If R is not definite we can conclude from the base case that there exists a repair $R' \subset \Delta_{facts}(R(W),W')$ for a $f \in F$ not holding in $R(W)$. Since $\Delta_{facts}(R(W),W') \subset \Delta_{facts}(W,W')$ $R \cup R'$ is an element of $repairs_{i+1}(W,F)$ and $R \cup R' \subset \Delta_{facts}(W,W')$.

4 The CDP-Planning Algorithm

4.1 Basic CDP-Algorithm

The main difference between our approach and most planners which are based on a STRIPS-like formalism ([4]) is that the CDP planner is able to handle range-restricted formulas which occur either as preconditions or as consistency constraints of a domain. The repair system allows to map general formulas to a set of ground literals. This capacity is required during the elemination of critical operators of a plan by insertion of new ones. If op is a critical operator with respect to a world W the following set of repairs will be generated:

$$\begin{array}{lll} repairs_i(W,op) := \{R_1 \cup R_2 \mid & R_1 \in repairs_i(W,pre(op)) \\ & \& \; R_2 \in repairs_i(op(R_1(W)),W_{constr}) \;\& \\ & \& \; R_2 \cup post(op) \text{ free of contradictions} \;\& \\ & \& \; R_1 \cup R_2 \text{ free of contradictions}\} \end{array}$$

The following algorithm shows how this mechanism can be incorporated into a simple planner:

basic CDP-Planner Basic-CDP(W_0,G)
init $plan := [goal]$ where $pre(goal) := G$ and $post(goal) := \emptyset$
while $plan$ is no solution **do**
begin
(1) determine a critical operator op_l of $plan = [op_1, \ldots, op_n]$: $l := SC(plan)$
let $\forall j = 1 \ldots n : W_{j+1} = op_j(W_j)$
(2) $Rep := repairs_i(W_{l-1}, op_l)$
(3) **choose** a subgoal $g \in \bigcup_{rep \in Rep} rep$
(4) **choose** an operator op: $g \in post(op)$
(5) **choose** a position j: $j \leq l$
(6) $plan := plan[1 \ldots j-1] \circ [op] \circ plan[j \ldots n]$;
end

The lines 3, 4 and 5 are choice points, i.e. the planner has to choose one out of several possible results and has to store the remaining results in order to allow backtracking. In line 1, the selection of a critical operator is deterministic. Which critical operator is selected depends on the applied **selection function** SC. The special operator *goal* allows a uniform treatment of the goal G. If *goal* is selected as a critical operator the worlds W' in line 2 are completions of the goal, i.e., we perform a goal completion. Since the completion of the goal is done in respect to the last world W_{n-1} of the plan[7], the completed worlds always reflect the actual situation of the planning process.

In the following example we demonstrate how the basic planning algorithm generates a solution for the Sussman Anomaly. In order to keep the example short we assume that the algorithm always performs optimal choices.

Initialization

The planning algorithm starts with the plan $[goal()]$. The precondition of the special operator $goal()$ is the goal $G = \{on(a,b), on(b,c)\}$.

First execution of the while body

Step 1.1: (choose a subgoal) *To satisfy the precondition of the operator* $goal()$ *a list of repairs is computed by the repair mechanism. After that, the planner chooses a subgoal at the first choice point.*

The repair mechanism finds exactly one repair

$$\begin{aligned} R \;=\; & \{add(on(a,b)), add(on(b,c)), add(on(c,table)), \\ & del(on(c,a)), del(on(a,table)), del(on(b,table))\} \end{aligned}$$

for the world W_0 and the goal G such that $comp(R(W_0)) \cup W_{constr,0} \cup G$ is consistent. Note, that the repair mechanism derives a new fact $on(c,table)$ from W_0 and G. The fact $on(a,b)$ is chosen as a the first subgoal g.

[7] the goal operator does not change the world W_{n-1}

Step 1.2: (choose an operator) *The task for the planner is to look for an operator 'performing' the subgoal g choosen by the previous choice point. If there are several possible operators a single one is selected.*

The operators $move(a, c, b)$ and $move(a, table, b)$ can be used to achieve $g = on(a, b)$. The second operator is chosen.

Step 1.3: (choose a position) *The algorithm inserts the selected operator in the generated plan at some position before the critical operator.*

The only possible position for $move(a, table, b)$ is directly before $goal()$.

Second execution of the while body

The selection function SC determines $move(a, table, b)$ as a critical operator.

Step 2.1: (Determine repairs) The subgoal $g = \neg on(c, a)$ is chosen from the only repair $\{\neg on(c, a), on(c, table)\}$.

Step 2.2: (choose an operator) Analogously to step 1.2, the operator $move(c, a, table)$ is chosen.

Step 2.3: (choose a position) Again, the introduction of the chosen operator into the plan is deterministic. $move(c, a, table)$ is inserted in the first position of the generated plan:

$$[move(c, a, table), move(a, table, b), goal()]$$

Third execution of the while body

$goal()$ is the only critical operator.

Step 3.1: (choose a subgoal) The precondition of $goal()$ and the consistency constraints of the domain hold after the execution of the repair $\{on(b, c), \neg on(b, table)\}$. The planner choses $on(b, c)$.

Step 3.2: (choose an operator) For the subgoal $on(b, c)$ the second choice point selects $move(b, table, c)$.

Step 3.3: (choose a position) For $move(b, table, c)$ exist three possible positions. By insertion of $move(b, table, c)$ in the second position of the plan we generate a solution

$$[move(c, a, table), move(b, table, c), move(a, table, b), goal()]$$

for the planning problem, since the intermediate worlds are consistent and the preconditions of all operators are satisfied.

4.2 Enhancements of the Basic Planner

Three different planning strategies, exploiting the fact that the planner allows a deterministic selection of a critical operator of a plan, have been implemented. The 'leftmost critical operator' (LMC) strategy always selects the critical operator op_i with the smallest index i; the 'rightmost critical operator' (RMC) strategy uses the critical operator with the highest index. The third strategy GPS is borrowed from the General Problem Solver ([3]). GPS is a LMC strategy where a new operator is always inserted directly before the selected critical operator, i. e., GPS replaces the nondeterministic choice point (5) by the deterministic statement $j := l$.

On one hand, it is well-known that if the search space of an existing planning strategy is pruned, no minimal plans or — even worse — no plans at all may be found. On the other hand, the search space often has to be pruned in order to get the necessary performance to solve even simple problems. In the following we introduce three powerful pruning heuristics which preserve completeness for LMC and RMC.

The first pruning technique **hitting-set (HS)** restricts the number of alternatives at the first choice point of the algorithm above by using a minimal hitting set of all computed repairs. Note, that the set of repairs can be large. (E.g., if we extended the initial world of the conjunctive goal problem by the facts $on(d_1, table), \ldots, on(d_n, table)$ we would get $n+1$ repairs:

$$\{\{on(a,b), on(b,c), on(c,x), \neg on(c,a), \neg on(a,table), \neg on(b,table)\} \mid x \in \{table, d_1, \ldots, d_n\}, i = 1 \ldots n\}$$

The branching factor of the first choice point is decreased by restricting the subgoals to a minimal hitting set $HS(Rep)$ of all computed repairs. This idea is incorporated into the planner above by replacing line 3 by

(3.a) determine a minimal hitting set of $Rep = \{R_1, \ldots, R_k\}$: hs:=HS($Rep$);
(3.b) **choose** a subgoal $g \in hs$

The selection of a hitting set out of several possible ones in line 3.a is done deterministicly therefore no backtracking to this point is necessary.

The second pruning technique reduces the branching factor at the second choice point by rejecting operators that clobber an already achieved subgoal. As an example consider the following incomplete plan for the conjunctive goal problem:

$$[move(c, a, table), move(a, table, b), move(b, table, c)].$$

The first two operators successfully achieve the subgoals $on(c, table)$ and $on(a, b)$. The third operator tries to achieve $on(b, c)$, but is not applicable since the precondition of $move(b, table, c)$ requires that the block b is clear. The only way to achieve $\forall x_4 : \neg on(x_4, b)$ is to destroy $on(a, b)$ by inserting additional operators. This plan does not lead to an optimal solution therefore its expansion should be averted.

In order to prevent this clobbering of an already achieved subgoal we use the well-known technique of **protected subgoals**[8] **(PSG)** which only allows the insertion of

[8] see, e.g., [14]

operators into positions where they do not violate previously defined establisher-user relations. Additionally, the insertion of an operator at a position j is prevented if between j and the critical operator already lies a potential clobberer. The GPS-planner is extended by an additional variable *psg* which lists the currently protected establisher-user relations. The initialization of Basic-CDP is extended by $psg := \emptyset$ and line 5 in the planner is replaced by:

(5.a) **choose** a position j for op such that:

- $j \leq l$
- the insertion of op at position j does not violate any in psg specified establisher-user relations
- between j and l are no potential clobbers of g in *plan*

(5.b) $psg := psg \cup \{$"op establishs g for op_i"[9]$\}$;

The expansion of the above plan can only partially be prevented by this modification since the second operator $move(a, table, b)$ could be inserted in order to achieve either $on(a, b)$ or $\neg on(a, table)$. In the second case, it would be possible to remove a from b again. If we protected several subgoals for one establisher, we could also prevent this case. But so far the problem of efficiently deciding whether an operator can achieve more than one subgoal and therefore the problem of protection of more than one subgoal for an operator is unsolved. In the blocks world this is even the main problem. In [7] Gupta and Nau show that in the worst case the complexity of the computation of an optimal solution is determined by the problem of deciding whether a block has to be moved by one or two *move*-operators to its goal position.

The third pruning technique concerns the computation of repairs for a critical operator op_{crit}. The selection of a subgoal g can be interpreted as a preference of those repairs R_k of *Rep* which contain g. If during further computations op_{crit} is again selected, we should respect the former decision and try to choose an unachieved subgoal of the 'preferred' repairs R_k which contain g. This consideration is the foundation of the **postponed repair computation (PRC)** pruning technique. Each operator op of a plan has a set of associated repairs *assoc-rep*(op). Initially, this set is empty. If an operator is selected by SC, the associated set of repairs is initially used to determine a subgoal. Only if all associated repairs are already achieved or if no repair is associated a new set of repairs is computed. After the selection of a subgoal, those repairs which contain the selected subgoal are linked to the critical operator. We integrate this technique into the planner through the replacement of line 2 by

(2.a) $Rep := assoc\text{-}rep(op_l)$
(2.b) **if** $\exists R \in Rep : W_{l-1} \models R$
(2.c) **then** $Rep := repairs_i(W_{l-1}, op_l)$

and through the insertion of a 7th and 8th line:

[9] we assume that operators which occur several times in a plan can be distinguished from each other

(7) $assoc\text{-}rep(op_l) := \{R \in Rep \mid g \in R\}$
(8) $assoc\text{-}rep(op) := \emptyset$

The goal operator *goal* is initialized in line 1 by the new statement $assoc\text{-}rep(goal) := \emptyset$. Since associated repairs may no longer comply with the definition of the term 'repair' (e. g. they may not be minimal) before the application of HS, the literals already achieved have to be removed from the associated repairs.

Figure 4 shows the entire planner with all three pruning techniques incorporated.

basic CDP-Planner Basic-CDP(W_0,G)
init $plan := [goal]$ where $pre(goal) := G$ and $post(goal) := \emptyset$
$psg := \emptyset$
$assoc\text{-}rep(goal) := \emptyset$
while *plan* is no solution **do**
begin
(1) determine a critical operator op_l of $plan = [op_1, \ldots, op_n]$: $l := SC(plan)$
let $\forall j = 1 \ldots n : W_{j+1} = op_j(W_j)$
(2.a) $Rep := assoc\text{-}rep(op_l)$
(2.b) **if** $\exists R \in Rep : W_{l-1} \models R$
(2.c) **then** $Rep := repairs_i(W_{l-1}, op_l)$
(3.a) determine a minimal hitting set of $Rep = \{R_1, \ldots, R_k\}$:
$hs := HS(\{\ \{l \in R_i | W_{l-1} \not\models l\}\ \mid\ R \in Rep\})$
(3.b) **choose** a subgoal $g \in hs$
(4) **choose** an operator *op*: $g \in post(op)$
(5.a) **choose** a position j for *op* such that:

- $j \leq l$
- the insertion of *op* at position j does not violate any in *psg* specified establisher-user relations
- between j and l are no potential clobbers of g in *plan*

(5.b) $psg := psg \cup \{$"*op* establishs g for op_i"$\}$;
(6) $plan := plan[1 \ldots j-1] \circ [op] \circ plan[j \ldots n]$;
(7) $assoc\text{-}rep(op_l) := \{R \in Rep \mid g \in R\}$
(8) $assoc\text{-}rep(op) := \emptyset$
end

Figure 4: The entire CDP-planning algorithm

In the next section, we proof that for LMC and RMC the planning procedure with minimal hitting set, protected subgoals , and postponed repair computation is complete (that is, that it finds all minimal plans). We show by an example, that the basic planner is incomplete for the GPS strategy.

5 Completeness of the CDP-Planner

The basic planning algorithm *Basic-CDP* defines a search tree which depends on the planning problem, the employed hitting set function, and the selection function for critical operators. The following definition describes this search tree.

Definition 5.1 (CDP search tree) *Let $\langle W_0, OP, G\rangle$ be a planning problem, HS a function which yields a hitting set of a set of repairs, i a number, and SC a selection function.*

A CDP search tree $ST(\langle W_0, OP, G\rangle)$ via HS, SC, and i is a tree, satisfying the following restrictions:

1. *the nodes are separated into the three disjoint sets* CP-SG, CP-OP, *and* CP-POS.
2. *Each successor of a* CP-SG *(*CP-OP / CP-POS*) node is an element of* CP-OP *(*CP-POS / CP-SG*).*
3. *A node n can have associated a plan* plan(n), *an operator* op(n), *a subgoal $sg(n)$, and an index $critical(n)$.*
4. *the root n_0 is a* CP-SG *node with* plan(n_0) = [goal()]
5. $\forall n \in$CP-SG*:* critical(n) = SC(plan(n)) *and each successor n' of n has a unique* $sg(n') \in HS(\text{repairs}_i(W_{critical(n)-1}, op_{critical(n)}))$

   if plan(n) *is a solution n has no successors.*
6. $\forall n \in$CP-OP*: each successor n' of n has a unique* op(n') $\in$ {op|sg(n) $\in$ post(op)}
7. $\forall n \in$CP-POS*: for each successor n' of n exists a unique $j \in \{1, \ldots,$ critical(n)$\}$ with:* plan(n') *can be build from* plan(n) *by inserting* op(n) *at position j.*
8. *nodes from* CP-OP *and* CP-POS *inherit their* plan*-value from their predecessors.*

Definition 5.2 (Completeness) *A planning strategy is* **complete** *if for every optimal solution of a solvable planning problem there exists a sequence of choices for the choice points such that the planning strategy finds this solution.*

We will illustrate the definition 5.1 by an example which will demonstrate that the GPS strategy is incomplete.

Example 5.3 (Incompleteness of GPS Strategy) *Let $W_0 = \langle\emptyset, \emptyset, \emptyset\rangle$ be the initial world and $G = \{q, (r \Rightarrow p)\}$ the goal. The following table specifies the operators of the domain:*

goal() :	pre	$\{q, (r \Rightarrow p)\}$	op'() :	pre	$\emptyset$	op''() :	pre	$\emptyset$
	post	$\emptyset$		post	$\{q, r\}$		post	$\{p, \neg q\}$

The figure 5 shows the search tree $ST(\langle W_0, OP, G\rangle)$. As far as the search tree is depicted in 5 it is independent of the actual choice for the selection function SC, the hitting set function HS, and the iteration level i. If we employ GPS the only CP-SG *node which contains the optimal solution is pruned. If we employ GPS and PSG together no solution at all is found.*

The following lemma corresponds the theorem 3.5. It shows that the extension of *repairs*$_i$ in subsection 4.1 has not destroyed the property of *repairs*$_i$ stated in 3.5.

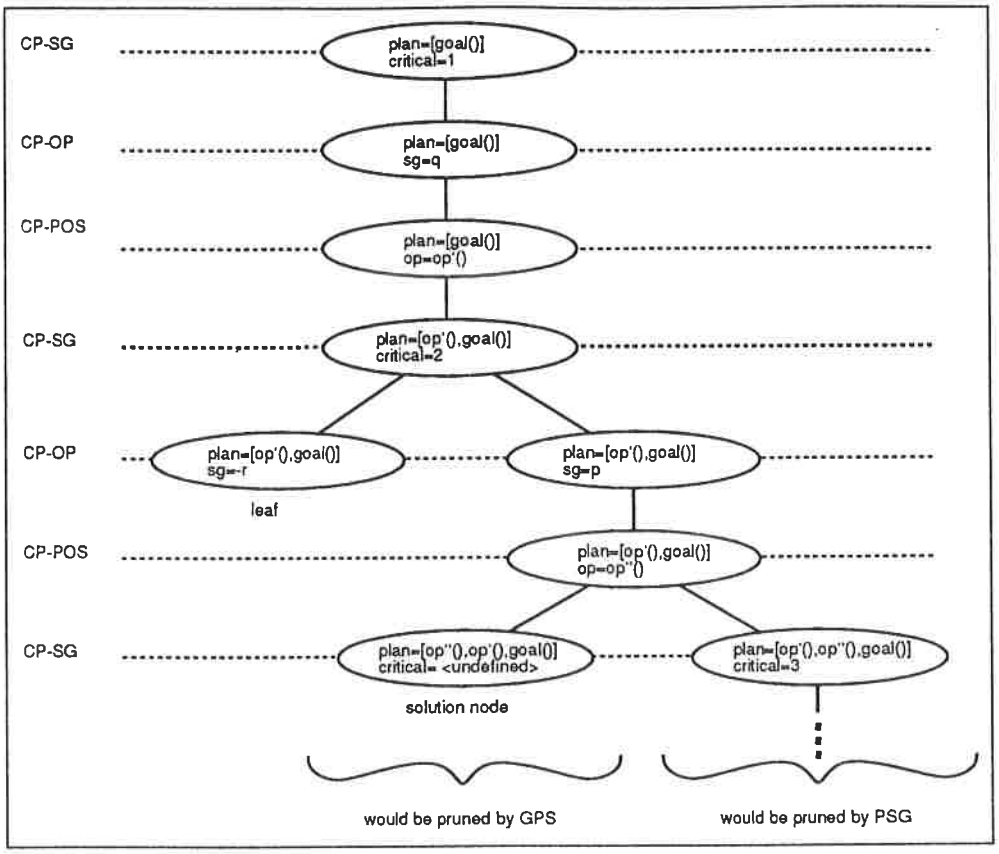


Figure 5: A search tree demonstrating the incompleteness of GPS

Lemma 5.4 *Let W, W' be possible worlds,* op *an operator which is critical for W and not critical for W'.*

$$\forall i : \exists R \in \text{repairs}_i(W, \text{op}) : \quad R \subset \Delta_{facts}(W, W')$$

Proof:

By theorem 3.5 there are R_1 and R_2 with:

$$\begin{aligned} R_1 \in repairs_i(W, pre(op)) \quad &: \quad R_1 \subset \Delta_{facts}(W, W') \\ R_2 \in repairs_i(op(R_1(W)), W_{constr}) \quad &: \quad R_2 \subset \Delta_{facts}(op(R_1(W)), op(W')) \end{aligned}$$

Since

$$\Delta_{facts}(op(R_1(W)), op(W')) \subset \Delta_{facts}(R_1(W), W')$$

and

$$\Delta_{facts}(R_1(W), W') \subset \Delta_{facts}(W, W')$$

the union $R_1 \cup R_2$ is a subset of $\Delta_{facts}(W, W')$. Therefore $R_1 \cup R_2$ is free of contradictions. $R_2 \cup post(op)$ also is free of contradictions since:

$$\Delta_{facts}(op(R_1(W)), op(W')) \cap (post(op) \cup post^c(op)) \; = \; \emptyset$$

Thus $R_1 \cup R_2$ is an element of $repairs_i(W, op)$.

Now we can proof the completeness of the basic CDP planner. For that we show that if a planning problem has a solution $plan_{sol}$ the search tree which is induced by *Basic-CDP* has a solution node containing a subplan of $plan_{sol}$.

Theorem 5.5 (Completeness of the Basic CDP-Planner) *Let HS be a hitting set function, SC a selection function, and i a number. If a planning problem $\langle W_0, OP, G\rangle$ has a solution* plan_{sol} *the search tree $ST(\langle W_0, OP, G\rangle)$ via HS, SC, and i has a* CP-SG *leaf n where* plan(n) *is a solution for $\langle W_0, OP, G\rangle$ and* plan(n) *is a subplan of* plan_{sol}.

Proof: we will use the solution $plan_{sol}$ as a guide to determine a path in the search tree which leads to a solution node

> It suffices to show, that for each *CP-SG* node n where $plan(n)$ is a subplan of $plan_{sol}$ and $plan(n)$ is not a solution, there exists a *CP-SG* (transitive) successor n_{SG} with $plan(n_{SG}) \subset plan_{sol}$.
>
> Let $plan_{sol} = [op_1, \ldots, op_l]$, $plan(n) = [op'_1, \ldots, op'_{l'}]$, W_i the intermediate worlds of $plan_{sol}$ and W'_i those of $plan(n)$. Further, let π be a monotonic mapping such that $op_i = op'_{\pi(i)}$ for all $i = 1, \ldots, l$.
>
> By lemma 5.4 there exists an $R \in repairs_i(W_{critical(n)-1}, op'_{critical(n)})$ which is a subset of $\Delta(W'_{critical(n)-1}, W_{\pi(critical(n))-1})$. By the definition of 5.1 and of HS there exists a direct successor n_{OP} of n with $sg(n_{OP}) \in R$. $sg(n_{OP})$ has an establisher op_k, with $0 < k < \pi(critical(n))$, in $plan_{sol}$ for the world $W_{\pi(critical(n))}$ since $W_{\pi(critical(n))} \models sg(n_{OP})$ and either $W_0 \not\models sg(n_{OP})$ or before op_k lies a clobberer of $sg(n_{OP})$ (otherwise $sg(n_{OP})$ could not be in repair R).
>
> Obviously, n_{OP} has a successor n_{POS} with $op(n_{POS}) = op_k$.
>
> If there is no j such that $k = \pi(j)$ a successor n_{SG} of n_{POS} exists where $plan(n_{SG}) \subset plan_{sol}$ (see figure 6). Assume there exists such a j. Then an operator $op'_{j'} = op_k$ lies before $op'_{critical(n)}$ in $plan(n)$. Since $sg(n_{OP})$ is an element of a repair for $W'_{critical(n)}$ between $op'_{j'}$ and $op'_{critical(n)}$ must lie a destroyer of $sg(n_{OP})$. This destroyer lies in $plan_{sol}$ between $op_{\pi(j')}$ and $op_{\pi(critical(n))}$. This is contrary to the fact that op_k is an establisher of $sg(n_{OP})$ for $W_{\pi(critical(n))}$.

Note that this theorem does not include the basic planner together with the GPS strategy.

Theorem 5.6 (Completeness of the Hitting Set Strategy) *The basic planner together with the minimal hitting set strategy is complete.*

Proof: This follows directly from 5.5

Theorem 5.7 (Completeness of the PSG Strategy) *The basic planner together with the protected subgoal strategy is complete.*

Proof:

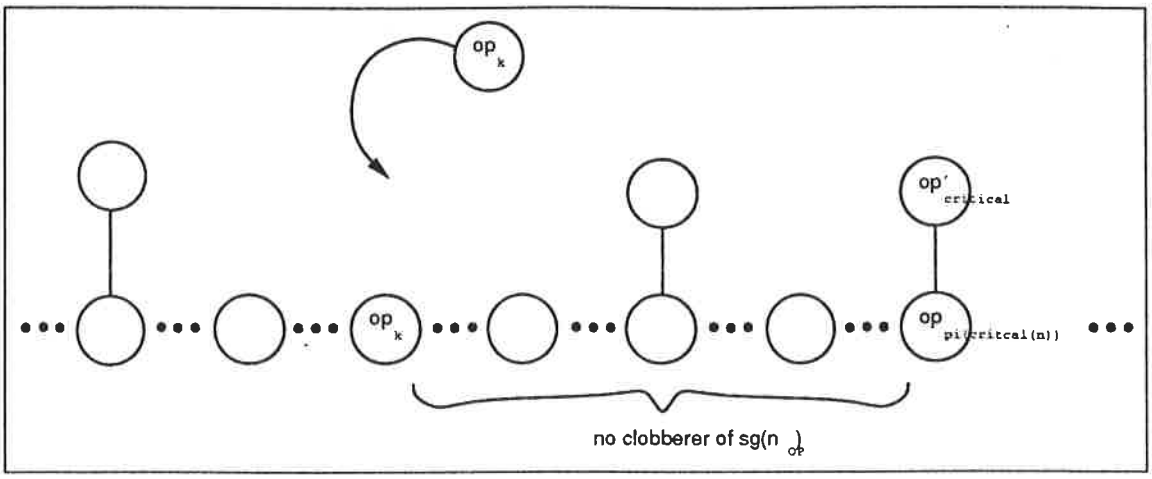


Figure 6: Insertion of an operator

Let π be the mapping considered in the proof of theorem 5.5 and $n_0, n_1, \ldots n_m$ the nodes leading from the root of the search tree to the solution node found in proof 5.5.

Assume that π is adapted to the insertion of an operator at every *CP-SG* node such that π always maps an operator of the evolving plan to the same[10] operator of the solution plan. This π could be used in the proof of 5.5 since no special assumptions about π are made in 5.5.

Let $op'_{k'}$ be inserted in $plan(n_i)$ as an establisher of the subgoal g for $op'_{l'}$. By the above considerations we can construct π such that the mapping of $op'_{k'}$ and $op'_{l'}$ onto two operators op_k and op_l of the solution plan $plan_{sol}$ by π is fixed. Therefore, on the way from n_0 to the solution node n_m, only those operators which lie between op_k and op_l in $plan_{sol}$ can be inserted between $op'_{k'}$ and $op'_{l'}$. Since op_k is an establisher of g for op_l in $plan_{sol}$, no clobberer of g is inserted.

Therefore, all paths containing a clobbering of an establisher-user relation which is explicitly introduced by the choice points, can be pruned without endangering completeness.

Theorem 5.8 (Completeness of the PRC Strategy) *The basic planner together with the postponed repair computation strategy is complete.*

Proof:

Let n be a *CP-SG* node which lies on the path from the root to a solution node and which inherits the repairs for the critical operator from a predecessor as described in the section 4.2. Let $plan_{sol} = [op_1, \ldots, op_l]$ be a solution plan, $plan(n) = [op'_1, \ldots, op'_{l'}]$, W_i the intermediate worlds of $plan_{sol}$ and W'_i those of $plan(n)$. Further, let π be a monotonic mapping such that $op_i = op'_{\pi(i)}$ for all $i = 1, \ldots, l$.

[10] operators which occur several times in a plan have to be distinguished from each other

By induction on the number of steps by which the associated repairs are inherited, we show that the following assertion holds:

$$\exists R \in \textit{assoc-rep}(op'_{critical(n)})) :$$
$$\{l \in R \mid W'_{critical(n)-1} \not\models l\} \subset \Delta_{facts}(W'_{critical(n)-1}, W_{\pi(critical(n))-1}) \quad (1)$$

(This assertion corresponds to 3.5 and 5.4.)

Assume that we have reached node n by applying the means described in the proof of 5.5 whenever possible and assume that the mapping π is constructed as in the proof of theorem 5.8. Originally the repairs of $\textit{assoc-rep}(op'_{critical(n)})$ are computed for an operator $op''_{critical(n'')}$ of a plan $plan(n'')$ of a predecessor n'' of n. Let $plan(n'') = [op''_1, \ldots, op''_{l''}]$ and $W''_1, \ldots, W''_{l''}$ the accompanying worlds.

Base Case:
By theorem 5.4 there is a repair $R \in \textit{repairs}_i(W''_{critical(n'')-1}, op_{critical(n'')})$ which is also a subset of $\Delta_{facts}(W''_{critical(n'')-1}, W_{\pi(critical(n''))-1})$. The subgoal g of the successor which leads from n'' to n lies in such a R. Thus R lies in $\textit{assoc-rep}(op'_{critical(n)})$. The construction of π guarantees that $W_{\pi(critical(n''))-1} = W_{\pi(critical(n))-1}$. Therefore $W_{\pi(critical(n))-1} \models R$ and R satisfies 1.

Induction step:
By the induction hypothesis the last time $\textit{assoc-rep}(op'_{critical(n)})$ was updated by a predecessor n' of n the set of associated repairs of the critical operator has a repair R which satisfies 1. Analogue to the base case we can conclude the existance of an $R \in \textit{assoc-rep}(op'_{critical(n)})$ which satisfies 1.

Theorem 5.9 (Completeness of Combination) *The basic planner together with all three strategies HS, PSG, and PRC is complete*

Proof: The incorporation of any strategy into the basic planner does not invalidate the completeness proof of any other strategy.

6 Empirical Evaluation of the Planner

In the last section, we have presented different planning strategies and pruning techniques. In order to evaluate the usefulness of these extensions to the basic planner, we have conducted a series of experiments with several planning domains. In this section, we present some of these results.

The tables 1, 2, 3, and 4 summarize performance results for the conjunctive goal problem, for the river crossing example, and for the railway example. In the river crossing example, a farmer has to move a goose, a dog, and bag of corn from the left side of a river to the right side. The farmer has a small boat which can carry himself and at most one further object. Neither the dog and the goose, nor the goose and the corn may stay on the same side without the farmer, since the dog would eat the goose and the goose the corn. Figure 7 in Appendix A shows a formalization of the domain.

In the railway example (Figure 8 in Appendix B) an engine has to transport a waggon from one end of a small marshalling yard to the other end. Four operators describe the

different actions. The operator *move* moves the engine whereas *transport* moves the engine and the wagon. The last two operators describe the coupling-up and decoupling of wagon and engine.

All tested configurations incorporate goal completion and pure breadth first search. No heuristic information is used in the search. Every choice at any choice point induces the generation of all successors of the corresponding node of the search tree. Note, that we count the generated nodes and not the expanded (or *closed*) nodes.

		GPS	LMC	GPS_{hs}	LMC_{hs}	$GPS_{hs,psg}$	$LMC_{hs,psg}$
conj. goal	nodes	306	287	36	30	36	30
	run time	7.5	6.4	2.8	1.9	2.7	1.9
river cross.	nodes	>2001	>2001	262	535	208	331
	run time	54	56	11	22	6.9	11
rail-way	nodes	>2001	>2001	76	326	76	326
	run time	89	92	5.5	17	5.6	17

Table 1: Number of generated search nodes and run time for different strategies. All runs include the PRC-strategy.

		GPS	LMC	GPS_{hs}	LMC_{hs}	$GPS_{hs,psg}$	$LMC_{hs,psg}$
conj. goal	nodes	316	292	36	30	36	30
	run time	9.2	8.2	3.3	2.3	3.2	2.3
river cross.	nodes	>2001	>2001	259	505	208	331
	run time	57	58	11	21	6.9	11
rail-way	nodes	>2001	>2001	76	326	76	326
	run time	90	92	5.6	19	5.4	19

Table 2: Number of generated search nodes and run time for different strategies. For all runs the PRC-strategy is switched off.

The application of the hitting set strategy results in the greatest reduction of the search space. If we look at table 3 which shows the branching factors for the three choice points, we can see why the minimal hitting set strategy leads to such an improvement. Its application reduces the branching factor of the subgoal choice point on average by a factor of 3. Table 3 also shows that the completeness of the LMC planning strategy does not increase the cost. In comparison with the GPS search strategy the LMC strategy has a branching factor of the choice point 'choose a position' which is only about 1.2 times higher than that of the (incomplete) GPS strategy.

The PSG strategy shows only for the river crossing example together with the hitting set strategy an improvement of the run time (Table 4). The performance of the PRC strategy is also modest. It results in almost no reduction of the generated nodes (Table 1 and 2). Only the run time is slightly improved since the planner has to compute fewer repairs if the PRC strategy is employed.

	choice point	GPS	LMC	GPS_{hs}	LMC_{hs}	$GPS_{hs,psg}$	$LMC_{hs,psg}$
conj. goal	subgoal	2.9	3.2	1	1	1	1
	operator	2	2	2	2	2	2
	ins. pos.	1	1.5	1	1.4	1	1.4
river cross.	subgoal	2.4	2.7	1.3	1.2	1.1	1.2
	operator	1.3	1.4	1.6	1.7	1.9	1.7
	ins. pos.	1	1.2	1	1.3	1	1.3
rail-way	subgoal	5.9	5.9	1	1	1	1
	operator	3.9	3.9	3.7	5.4	3.7	5.4
	ins. pos.	1	1.1	1	1.2	1	1.3

Table 3: Branching factors of the three choice points for different strategies. For all runs the PRC-strategy is switched on.

		conjunctive goal		river crossing		railway	
		$\neg psg$	psg	$\neg psg$	psg	$\neg psg$	psg
$\neg hs$	GPS	306/7.5	306/7.3	>2000/54	>2000/48	>2000/89	>2000/88
	LMC	287/6.4	287/6.6	>2000/56	>2000/49	>2000/92	>2000/88
	RMC	445/8.6	445/8.6	>2000/46	>2000/45	>2000/95	>2000/95
hs	GPS	36/2.8	36/2.7	262/11	208/6.7	76/5.5	76/5.6
	LMC	30/1.9	30/1.9	535/22	331/11	326/18	326/19
	RMC	38/2.2	38/1.6	516/21	203/6.7	721/41	721/36

Table 4: (generated search nodes/run time) pairs for different examples, strategies, and pruning techniques. '¬' indicates the omission of a technique. For all runs the PRC-strategy is switched on.

If additional domain independent heuristics[11] and a best-first search are applied, the number of generated nodes for the conjunctive goal problem can be reduced to 20 nodes, 52 nodes for the river crossing problem, and 39 nodes for the railway example.

7 Conclusion

The results presented in the previous section show that generally applicable pruning techniques can reduce the search space. However, they also indicate that these techniques are not strong enough to handle larger problems. At the moment, we are incorporating two further techniques: situation abstraction and the explicit treatment of the problem space. Knoblock ([8]) showed that under certain conditions situation abstraction can reduce the worst-case complexity from exponential to linear in the solution length. Although these conditions do not always hold, this result shows the power of abstraction.

[11] the heuristics include the preference of operators which achieve several subgoal or the preference of uncritical positions for the insertion of operators

References

[1] Martin Decker, Guido Moerkotte, Holger Müller, and Joachim Posegga. Consistency driven planning. In P. Barahona, L. Moniz Pereira, and A. Porto, editors, *5th Portuguese Conference on Artificial Intelligence*, Lecture Notes in Artificial Intelligence, pages 195–209, Albufeira, Portugal, October 1991.

[2] Jürgen Dix, Joachim Posegga, and Peter H. Schmitt. Modal Logic for AI Planning. In *First International Conference on Expert Planning Systems*, pages 157–162, Brighton, GB, July 1990. IEE.

[3] G. Ernst and A. Newell. *GPS: A Case Study in Generality and Problem Solving.* Academic Press, 1969.

[4] R. Fikes and N. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *Artificial Intelligence*, 2:189–205, 1971.

[5] Mattew L. Ginsberg and David E. Smith. Possible Worlds Planning I. *Artificial Intelligence*, 35:165–195, 1988.

[6] Corell Green. Application of theorem proving to problem solving. In *1st International Joint Conference on Artificial Intelligence*, pages 219–239, Washington, USA, 1969.

[7] Naresh Gupta and Dana S. Nau. Compexity results for blocks-world planning. In *Proceedings of the Ninth National Conference on Aritificial Intelligence*, pages 629–633, San Francisco, USA, 1991. American Association of Artifical Intelligence.

[8] Craig A. Knoblock. Search reduction in hierarchical problem solving. In *Proceedings of the Ninth National Conference on Aritificial Intelligence*, pages 686–691, San Francisco, USA, 1991. American Association of Artifical Intelligence.

[9] 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*, pages 523–530, Los Altos, USA, 1987. Morgan Kaufmann.

[10] John McCarthy and Pat Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, GB, 1969.

[11] Guido Moerkotte and Peter C. Lockemann. Reactive consistency control in deductive databases. *ACM Transactions on Database Systems*, 16(4):670–702, December 1991.

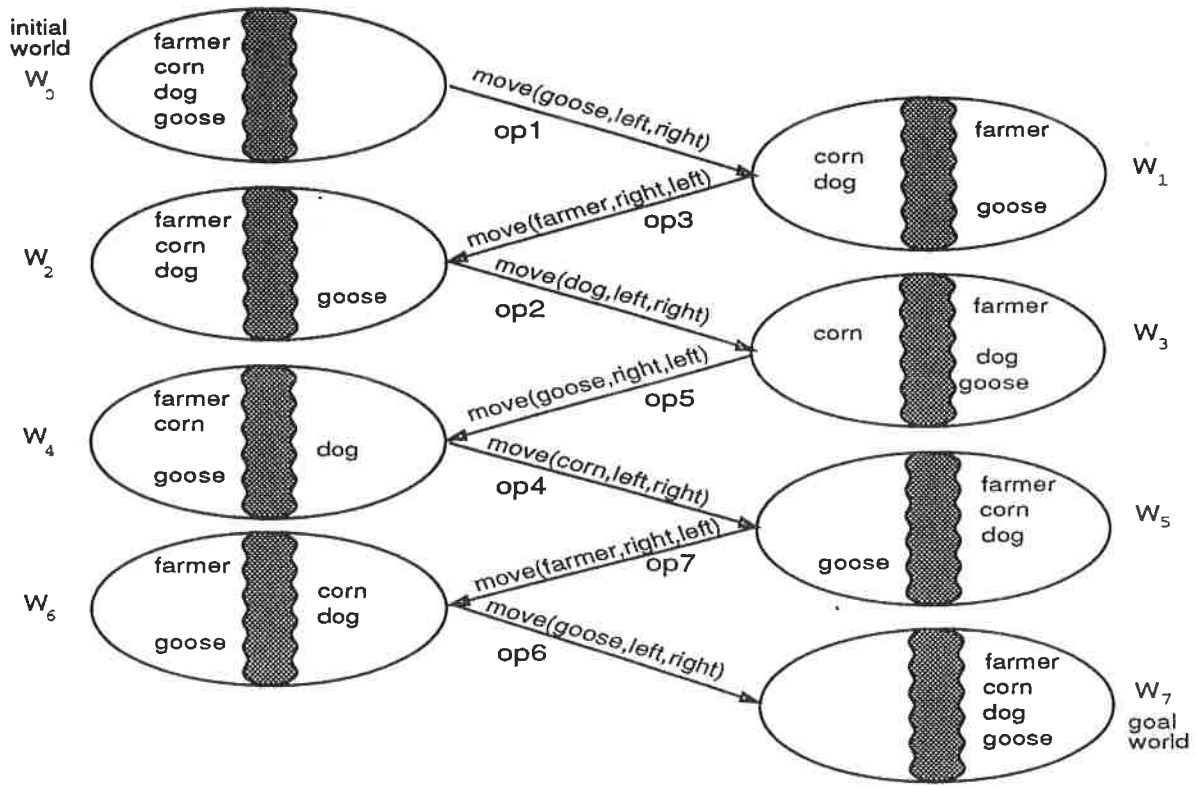
[12] Guido Moerkotte and P. H. Schmitt. Analysis and repair of inconsistencies in deductive databases. (submitted), 1992.

[13] Jean-Marie Nicolas. Logic for improving integrity checking in relational databses. *Acta Informatica*, 18:227–253, 1982.

[14] Austin Tate. Generating project networks. In *5th International Joint Conference on Artificial Intelligence*, pages 888–893, Bosten, USA, 1977.

[15] Marianne Winslett. Reasoning about Action Using a Possible Models Approach. In *Proceedings of the Seventh National Conference on Aritificial Intelligence*, pages 89–93, St. Paul, USA, 1988.

A Formalization of the River Crossing Example



$W_{facts,0}$: *is(farmer,object), is(dog,object), is(goose,object), is(corn,object), pos(farmer,left), pos(dog,left), pos(goose,left), pos(corn,left), is(left,location), is(right,location)*

$W_{constr,0}$:

constr-1 : $\forall x_1\ \exists x_2$: $is(x_1, object) \Rightarrow pos(x_1, x_2) \wedge is(x_2, location)$

constr-2 : $\forall x_1, x_2, x_3$: $pos(x_1, x_2) \wedge pos(x_1, x_3) \Rightarrow x_2 = x_3$

constr-3 : $\forall x_1$: $pos(dog, x_1) \wedge pos(goose, x_1) \Rightarrow pos(farmer, x_1)$

constr-4 : $\forall x_1$: $pos(goose, x_1) \wedge pos(corn, x_1) \Rightarrow pos(farmer, x_1)$

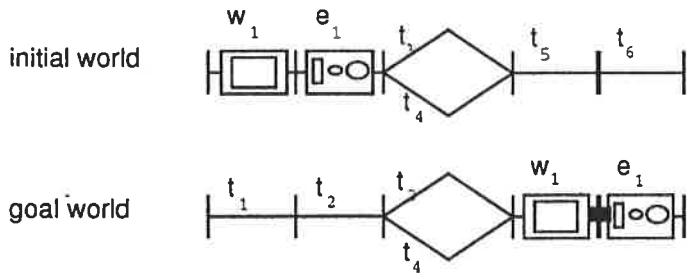
Goal G: $\{pos(farmer, right),\ pos(dog, right),\ pos(goose, right),\ pos(corn, right)\}$

Operators:

declare $move(x, l_1, l_2)$
range $is(x, object), is(l_1, location), is(l_2, location), l_1 \neq l_2$
prec $pos(farmer, l_1), pos(x, l_1)$
post $pos(farmer, l_2), pos(x, l_2), \neg pos(farmer, l_1), \neg pos(x, l_1)$
end

Figure 7: Formalization of the river crossing example

B Formalization of the Railway Example



$W_{rules,0}$:

rule-1 : $\forall Xv_1, Xv_2, Xp_1, Xp_2 :$
$pos(Xv_1, Xp_1) \wedge pos(Xv_2, Xp_2) \wedge connected(Xp_1, Xp_2)$
$\Rightarrow next\text{-}to(Xv_1, Xv_2)$

rule-2 : $\forall x_1, x_2 : \; is(x_1, x_2) \Rightarrow isa(x_1, x_2)$

rule-3 : $\forall x : \; isa(x, engine) \Rightarrow isa(x, vehicle)$

$W_{constr,0}$:

constr-1 : $\forall Xv_1, Xv_2, Xp : \; pos(Xv_1, Xpos) \wedge pos(Xv_2, Xp) \Rightarrow Xv_1 = Xv_2$

constr-2 : $\forall Xv, Xp_1, Xp_2 : \; pos(Xv, Xpos_1) \wedge pos(Xv, Xp_2) \Rightarrow Xp_1 = Xp_2$

constr-3 : $\forall Xv_1, Xv_2, Xv_3 : \; coupled(Xv_2, Xv_1) \wedge coupled(Xv_3, Xv_1) \Rightarrow Xv_2 = Xv_3 \wedge$
$coupled(Xv_1, Xv_2) \wedge coupled(Xv_1, Xv_3) \Rightarrow Xv_2 = Xv_3$

constr-4 : $\forall Xv_1, Xv_2 : \; coupled(Xv_1, Xv_2) \Rightarrow next\text{-}to(Xv_1, Xv_2)$

Operators:

declare transport$(Xv_2, Xv_1, Xfrom_2, Xform_1, Xto)$
range $is(Xv_1, vehicle), is(Xv_2, vehicle),$
$is(Xfrom_1, track), is(Xform_2, track), is(Xto, track),$
$connected(Xfrom_2, Xfrom_1) \vee connected(Xto, Xfrom_1),$
$connected(Xfrom_2, Xfrom_1) \vee connected(Xfrom_1, Xfrom_2),$
$connected(Xfrom_1, Xto) \vee connected(Xto, Xfrom_1),$
$connected(Xfrom_1, Xto) \vee connected(Xfrom_1, Xfrom_2)$
prec $pos(Xv_1, Xfrom_1), pos(Xv_2, Xfrom_2),$
$is(Xv_1, engine) \vee is(Xv_2, engine),$
$coupled(Xv_2, Xv_1) \vee coupled(Xv_1, Xv_2),$
$\forall Xv \; \neg pos(Xv, Xto)$
post $pos(Xv_2, Xfrom_1), pos(Xv_1, Xto), \neg pos(Xv_1, Xfrom_1), \neg pos(Xv_2, Xfrom_2)$
end

declare move$(Xe, Xfrom, Xto)$
range $is(Xe, engine), is(Xfrom, track), is(Xto, track),$
$connected(Xfrom, Xto) \vee connected(Xto, Xfrom)$
prec $pos(Xe, from), coupled(Xe, nil), coupled(nil, Xe)$
post $pos(Xe, Xto), \neg pos(Xe, Xfrom_1)$
end

declare couple-up(Xv_1, Xv_2)
range $is(Xv_1, vehicle), is(Xv_2, vehicle)$
prec $next\text{-}to(Xv_1, Xv_2),$
$coupled(Xv_1, nil), coupled(nil, Xv_1),$
$coupled(Xv_2, nil), coupled(nil, Xv_2)$
post $coupled(Xv_1, Xv_2), \neg coupled(Xv_1, nil), \neg coupled(nil, Xv_2)$
end

declare decouple(Xv_1, Xv_2)
range $is(Xv_1, vehicle), is(Xv_2, vehicle)$
prec $coupled(Xv_1, Xv_2)$
post $coupled(Xv_1, nil), coupled(nil, Xv_2), \neg coupled(Xv_1, Xv_2)$
end

Figure 8: Formalization of the railway example and depiction of a solution with its intermediate worlds