Towards a Theory of Conflict Detection and Resolution in Nonlinear Plans

Joachim Hertzberg and Alexander Horz

Forschungsgruppe Expertensysteme
Gesellschaft fur Mathematik und Datenverarbeitung (GMD)
Schlofi Birlirighoveri, Postfach 1240
D-5205 Sankt Augustin 1, F.R.G.

Abstract

This paper deals with a well known problem in AI planning: detecting and resolving conflicts in nonlinear plans. We sketch a theory of restricted conflict detection and resolution that subsumes conflict handling in classical nonlinear planners. By relaxing the restrictions, we develop a more general concept of conflicts suggesting practical and theoretical limitations of conflict handling in nonlinear plans.

1 The problem

Nonlinear planning was invented [Saccrdoti, 1977, Tate, 1977] as a lea,st commitment strategy for operator ordering. A nonlinear plan specifies a strict partial ordering on operators that can be interpreted to represent every linear ordering compatible with it. (Note that NOAH [Saccrdoti, 1977], e.g., uses a specialization of this interpretation as it does not allow *every* compatible linear ordering but considers ordering whole branches only.)

While being efficient in that it allows whole areas of the space of linear plans to be searched at once, nonlinear planning poses a new problem compared to linear planning: one has to decide whether really *every* linear ordering of operators in a nonlinear plan results in a correct operator sequence to be executed. This is the well known problem of detecting and if necessary—resolving conflicts in nonlinear plans. Of course, conflicts also arise in linear plans: a STRIPS [Fikes *et.* a/., 1971] plan, e.g., containing what we call a conflict would simply be incorrect or incomplete. But as linear plans are special cases of nonlinear ones, conflict handling in linear plans proves to be less complex.

Although every nonlinear planner has to detect and resolve conflicts in one way or another, there is no coherent theory of conflict handling. Instead, this is described anew for every planner, and it seems that some planner implementors have deviated from the path of generality without mentioning that they have done so.

In this paper, we deliver a sketch of a theory of conflict detection and resolution in nonlinear plans. The emphasis is on the coherence of the theory developed; we do not claim all the ideas we present here are new.

2 The theory (restricted)

In this section, we introduce relevant concepts and give some results on conflict handling for the "ground case", where a single node in a plan produces conditions that may be destroyed by another single node. We will see how to generalize matters in the following section.

Throughout the paper, plans are partially ordered sets of operators with unique least and greatest elements called *start* and *finish,* respectively. As usual, plans will be represented as graphs; hence we will speak of start, finish, and operator *nodes* or simply nodes if the exact class of a node is of no importance. As nodes correspond to operators, we will speak of preconditions and postconditions of nodes. The precondition of *start* is empty; its postcondition describes the initial state. The precondition of *finish* describes the goal state. We assume that conditions are formalized in a first order language A, employing the concept of derivability of conditions from sets of conditions. (In the following, all conditions are tacitly assumed to be in A.)

As to the "ground case" of conflict management considered in this section, we make two assumptions:

1. The STRIPS assumption: domain conditions only change if mentioned by the postconditions of a node in the plan, and the operator description defines a function that unambiguously maps situations onto situations.
2. The *locality assumption:* every node must specify all the domain conditions it may change regardless of the domain conditions holding before it.

There are different ways to actually implement the locality assumption. One way—which is inspired by [Lifschitz, 1986]—is the following: operator postconditions are represented by add and delete lists in STRIPS [Fikes *et.* a/., 1971] fashion; they may contain only ground atomic formulas (or schemata for such formulas); every non atomic formula true at one place in the plan must be true in the domain in general and is represented outside the plan.

We do not, however, want to favor one particular interpretation of the locality assumption and therefore will speak just of positive and negative node postconditions and of derivability of conditions from the postconditions of some node in the plan.

Note that the locality assumption does not necessarily imply decidability of the postconditions of a node.

2.1 Conflicts

We now come to the definition of a conflict which, however, requires some preliminaries.

Definition 2.1 (Plan, start, finish) *The ordered pair* $\Pi=(O,<)$ *is a* plan *iff*

O *is a set of nodes containing at least the nodes* start *and* finish, *and*

$< \subseteq O \times O$ *is a strict partial ordering on* O *such that* $start<finish$ *and* $\forall N \in O \setminus \{start,finish\}: start<N<finish$.

If there is some node N with a precondition C in the plan, and there is no other node P preceding N which produces C, then the plan cannot be correct (assuming that conditions that are generally true do not appear in preconditions). Executing it will fail because the action corresponding to N will not be applicable. So, for a condition C, there may be *users* and *producers* in a plan. Using a condition does not imply deleting it, of course.

Definition 2.2 (User, Producer) *Let* N *be a node with preconditions* Pre *and postconditions* $Post$, *and* C *a condition;*

N *is a* producer *of* C *iff* $Post \vdash C$

N *is a* user *of* C *iff* $C \in Pre$

Now we are going to define well formed plans, i.e. plans that are syntactically useful. A plan is well formed if every user of a condition is preceded by a producer of the condition.

Definition 2.3 (Well Formedness) *Let* $\Pi=(O,<)$ *be a plan.* Π *is* well formed *iff*

$\forall C \in \Lambda\ \forall U \in O$: *if* U *is user of* C *then* $\exists P \in O$: P *is producer of* C *and* $P<U$

In the following, all plans are assumed to be well formed.

Given a plan, there will often be more than one producer of a condition C. The definition of dependency which follows makes explicit that a user node U of condition C possibly gets C delivered by a producer P which is closest to U respecting $<$:

Definition 2.4 (Dependency) *Let* $\Pi=(O,<)$ *be a plan,* C *a condition, and* $P,U \in O$. *Then* $D=(P,C,U,\Pi)$ *is a* dependency *between* P *and* U *concerning* C *in* Π *iff*

- P *is a producer of* C
- U *is a user of* C,
- $P<U$, *and*
- $\forall N \in O$: *if* $P<N<U$ *then* N *is not a producer of* C

The partial ordering of nodes induced by dependencies is a subset of the operator ordering, due to the third condition in the definition. There may, of course, be more than one dependency assuring the same condition for the same node, serving the need of assuring it under possibly many linearization alternatives.

In figure 1 you see a node representing the operator *stack(B,C)* with preconditions *holding(B)* and *clear(C)*. For both of the preconditions there is a relevant producer, e.g. the node *unstack(B,Table)*.

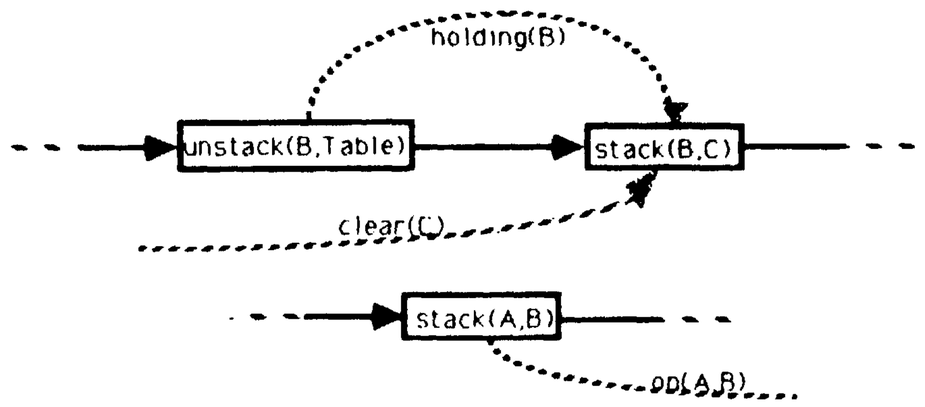

Figure 1: A partial blocks world plan (dashed lines represent dependencies; solid lines the operator ordering)

Now we are heading into the direction of defining a conflict. The idea is that we have a conflict if the condition of a dependency is destroyed by the postcondition of a node that can be ordered between the user and the producer of the dependency with no new producer restoring the predicate. In order to state this formally, we need an intermediate step.

Definition 2.5 (Enclosure) *Let* $\Pi=(O,<)$ *be a plan and* $D=(P,C,U,\Pi)$ *be a dependency between* P *and* U *concerning* C.
Then D encloses $N \in O$ *iff*
$\exists <' \subseteq O \times O : < \subseteq <'$ *and* $P <' N <' U$
$<'$ *is called an* enclosure *of* N *in* D.

The partial plan in figure 1 is intuitively not correct, because the dependency concerning *holding(B)* encloses the node *stack(A,B)* which is a producer of $\neg$*holding(B)*. (Note that this must be locally derivable from the postconditions of *stack(A,B)*!)

Definition 2.6 (Conflict) *Let* $\Pi=(O,<)$ *be a plan,* $D=(P,C,U,\Pi)$ *be a dependency between* P *and* U *concerning* C, *and* $Post$ *be the postconditions of* $N \in O$.
Then (D,N) *is a* conflict *between* D *and* N *iff*

- $\{C\} \cup Post \vdash false$
- D *encloses* N *with enclosure* $<'$
- $\forall P' \in O$: *if* $N<'P'<'U$ *then* P' *is not a producer of* C

A plan containing no conflicts is called conflict free.

Deriving a procedure for conflict detection is straightforward from the definition. For an inefficient version [Hertzberg and Horz, 1988] one may simply translate the conflict definition and the preceding ones into a Prolog program.

Applying the definition, there are four different types of conflicts depending on the relative positions of nodes P,U, and N. They are shown in figure 2.

Every other "type" of conflict, such as the double cross conflict in NOAH [Sacerdoti, 1977], is a combination of these four elementary types and can be handled by handling the elementary conflicts involved. (Special combinations may suggest special resolution strategies, of course.)

Dependencies or related concepts have long been recognized as important for planning, even before the advent of nonlinear planning as in, e.g. STRIPS' triangle tables [Fikes *et. al.*, 1972], HACKER's [Sussman, 1975]

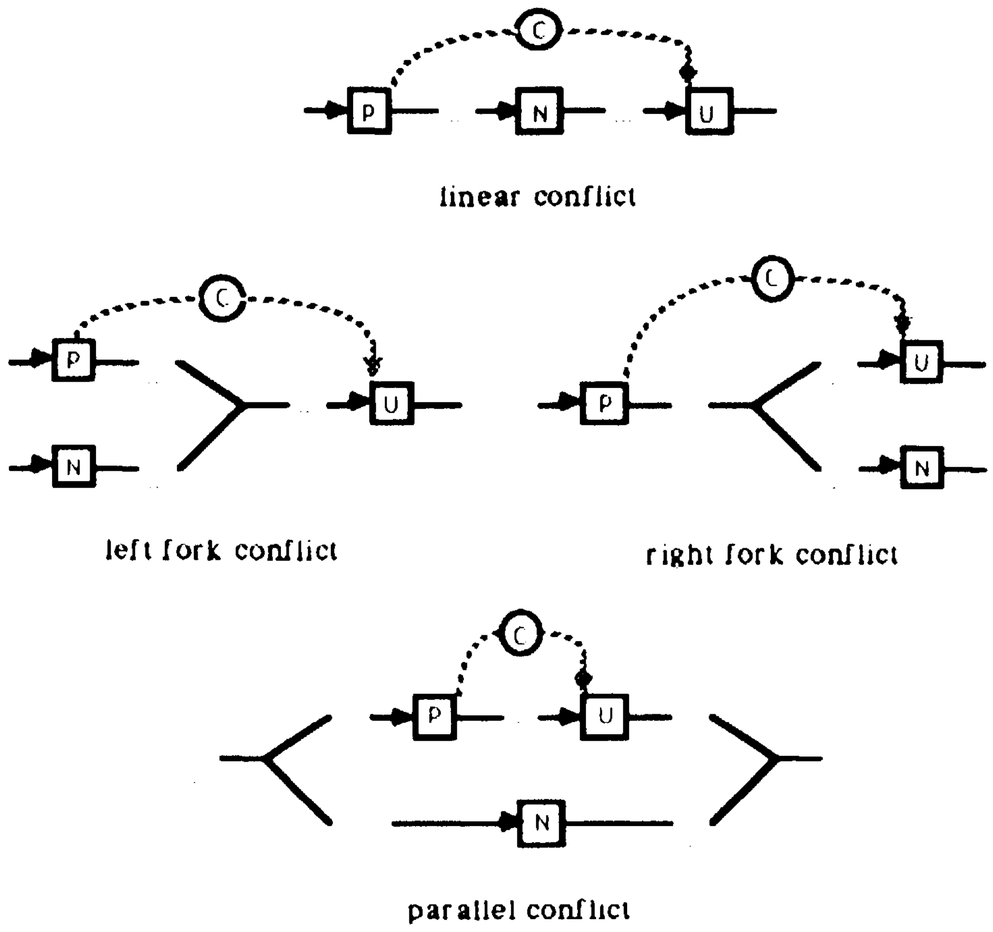

Figure 2: Types of conflicts

and Waldinger's [Waldingcr, 1977] protections, and INTERPLAN's [Tate, 1975] holding periods. In nonlinear planners, detecting conflicts by looking for some variant of dependency in one branch and a node destroying it in a parallel branch is an obvious idea and has—in one way or another -been done, e.g. in NOAH [Sacerdoti, 1977] using the TOME, NONLIN [Tate, 1977] using the COST, S1PE [Wilkins, 1984], and TWEAK [Chapman, 1987] using the necessary truth criterion. A similar concept of dependency and conflict is used in time map management as in, e.g., [Dean and McDermott, 1987]

2.2 Resolving conflicts

Detecting conflicts is one thing; resolving them is another. (Eor brevity, we will keep things mostly informal here. A more formal treatment is straightforward and can be found in [Hertzberg and Horz, 1988]. In the following, variables *D,C, P, N,U* refer to those in Definition 2.6.) Definition 2.6 tells us what to do: One can resolve a conflict bv

- constraining variables to prevent the derivation of *false,* or
- preventing the enclosure of *N* by D, or
- arranging another producer of the condition endangered.

Like Chapman [Chapman, 1987], we don't consider resolving conflicts by deleting nodes.

If the derivation *of false* from *{C}* $\cup$ *Post* has involved substituting a variable by some term, then a conflict of whichever type can be resolved by constraining the variable not to be unifiable with this term. This strategy is used in, e.g. [Chapman, 1987].

All conflicts other than the linear conflicts can also be resolved by preventing the enclosure of *N* by *D* by means of extending the relative ordering of P, *N,* and *U.* In particular, we have to assert $N <' P, U <' N$, and $N <' P$ or $U <' N$ to resolve a left fork, right fork and parallel conflict, respectively (assuming $N \neq$ *start* and $U \neq$ *finish).*

Mind that a parallel conflict is resolved by mutual exclusion of *N* and the branch within and including *P* and *U.*

Extending the operator ordering in some way has been a conflict resolution strategy in virtually every nonlinear planner. In particular, it corresponds to Chapman's [Chapman, 1987] concepts of promotion and demotion.

Every conflict of whichever type can also be solved by arranging a C-producer between *N* and *U.* This means asserting *N* <' P'<' *U* for an existing or new node P' producing *C.* Naturally, patching in a new node may result in new conflicts, leading—at the extreme—to an infinite conflict handling regress. This strategy corresponds, e.g., to Chapman's [Chapman, 1987] white knights. (Note that just ordering *existing* nodes never produces new conflicts involving existing dependencies.)

Now that we have studied the different types of conflicts in more detail, we can infer the domain dependent knowledge the conflict resolution module of a nonlinear planner must be fed with. Given a plan containing a set of conflicts, a planner has to decide

- which conflict to work on first and
- whether to solve a conflict by
  - constraining variable substitutions (if variables are involved), or
  - just ordering the nodes involved (except for linear conflicts), or
  - arranging a C-producer at the right place

Knowledge of this kind must be part of the domain representation adapting a general planner to particular planning strategies in a particular domain. Choosing among conflict resolution alternatives is a hopelessly heuristic matter.

3 Generalizing the Theory

This completes the formal reconstruction of the issues that have been discussed in the literature, at least implicitly (though mostly under a restricted view). In general, however, matters are even more complex, as conditions are not the result of just *one* operator but may be derived from the postconditions of *sets* of operators.

Consider an extended blocks world [Chapman, 1987] with different sized blocks, where two small blocks may be put on a single large target block. Imagine a plan with three unordered nodes, each of them representing stacking a different small block on one and the same target block. There is a conflict in this plan, of course, because each pair of two stacking nodes destroys a necessary precondition of the third node -there will be no more space on the target block.

The *locality assumption* does not hold in this example: the actual postconditions of operators depend on the situations in which they are applied. This is the known problem of "synergy" [Chapman, 1987] effects.

Planning without *STRIPS assumption* also enforces consideration of node sets in conflict detection. Imagine a scenario of electrical appliance repairs: first of all, you have to interrupt power supply. So, there may be an

abstract operator negating the condition that power is supplied, but giving no additional details about possible effects on the connections in the chain of power cable and different extension leads from the appliance to the wall socket. (An abstract operator is an operator that is yet to be expanded to a subplan.)

The *STRIPS assumption* does not hold: the operator description does not define a *function* that unambiguously maps a given state of application onto a resulting state.

Obviously, detecting conflicts hinges on the ability to realize what conditions are true at which points in a plan; this is impossible in plans with abstract operators [Wilkins, 1986]. So, there may be undetectable conflicts. One should, however, be able to detect those kinds of conflicts where sets of nodes produce conditions the combination of which is inconsistent with the intended effect of the abstract operator, as, e.g., a set of nodes all together asserting a perfect cable chain from the wall socket to the electrical appliance in the scenario given above.

Consequently, when planning both without the locality assumption or without the STRIPS assumption, a *general producer* of a condition is a set of nodes, and a *general dependency* leads from a set of nodes to a node. In a *general conflict,* such a general dependency *generally encloses* a general producer of the negation of a condition necessary for deriving the dependency condition.

3.1 General Conflicts

Following are the generalized definitions for producers, dependencies, enclosures, and conflicts. The generalizations are straightforward: instead of a single node producing a condition, we have to consider sets of nodes.

Definition 3.1 (General Producer)
Let $\mathcal{P} = \{P_1, \ldots, P_p\}$ *be a set of nodes; let* C *be a condition.*

$\mathcal{P}$ *is a* general producer *of* C *with* supporting conditions set $\mathcal{S} = \{c_1, \ldots, c_p\} \subseteq \Lambda$ *iff*

$\mathcal{S}$ *is a minimal consistent set such that* $\mathcal{S} \vdash C$, *and*
$\forall\, i \in \{1, \ldots, p\} : P_i$ *is a producer of* c_i

Note that one node may deliver more than one supporting condition, i.e. it is not assumed that $P_i \neq P_j$ for different $i, j \in \{1, ..., p\}$.

The definition of a dependency must be generalized accordingly from definition 2.4 by substituting the producer node P with a set of nodes $\mathcal{P}$. Furthermore, we have to extend the concept of closeness of nodes to closeness of node sets and nodes.

Definition 3.2 (General Dependency) *Let* $\Pi = (O, <)$ *be a plan,* $C, c_1, \ldots, c_p$ *conditions,* $\mathcal{P} = \{P_1, \ldots, P_p\} \subseteq O$, *and* $U \in O$. *Then* $\mathcal{D} = (\mathcal{P}, C, U, \mathcal{S}, \Pi)$ *is a* general dependency *between* $\mathcal{P}$ *and* U *concerning* C *in* Π *iff*

- $\mathcal{P}$ *is a general producer of* C *with supporting conditions set* $\mathcal{S} = \{c_1, \ldots, c_p\}$
- U *is a user of* C,
- $\forall\, \mathcal{N} \subseteq O\ \exists\, i \in \{1, \ldots, p\}\ \forall\, N \in \mathcal{N}$:
  if $P_i < N < U$ *then* $\mathcal{N}$ *is not a general producer of* c_i.

The last condition formalizes that a user node U of condition C possibly gets C delivered by a particular general producer $\mathcal{P}$ with supporting conditions set $\mathcal{S}$ which is closest to U respecting $<$. Note that there may be other general producers of C closer to U than $\mathcal{P}$, e.g. if they have different supporting condition sets.

A general dependency *generally encloses* every node enclosed by some of its branches. So, the enclosure definition 2.5 is generalized by having P existentially quantified within the producer set $\mathcal{P}$ of the general dependency $\mathcal{D}$ involved.

Definition 3.3 (General Enclosure) *Let* $\Pi = (O, <)$ *be a plan and* $\mathcal{D} = (\mathcal{P}, C, U, \mathcal{S}, \Pi)$ *be a general dependency between* $\mathcal{P}$ *and* U *concerning* C.
Then $\mathcal{D}$ generally encloses $N \in O$ *by* $P_i \in P$ *iff*

$$\exists\, <' \subseteq O \times O\ \ \exists i \in \{1, \ldots, p\} : \quad < \subseteq <' \text{ and } P_i <' N <' U$$

$<'$ *is called a* general enclosure *of* N *in* $\mathcal{D}$ *by* P_i.

Finally, we arrive at the definition of a general conflict. A general dependency concerning C with general producer $\mathcal{P} = \{P_1, \ldots, P_p\}$ and supporting conditions set $\mathcal{S} = \{c_1, \ldots, c_p\}$ is in conflict with a general producer $\mathcal{N}$ of $\neg c_i$ if there is a general enclosure $<'$ by P_i of $N \in \mathcal{N}$ such that there is a partial ordering $<'' \supseteq <'$ where c_i is deleted by $\mathcal{N}$ but not restored by any other general producer before U.

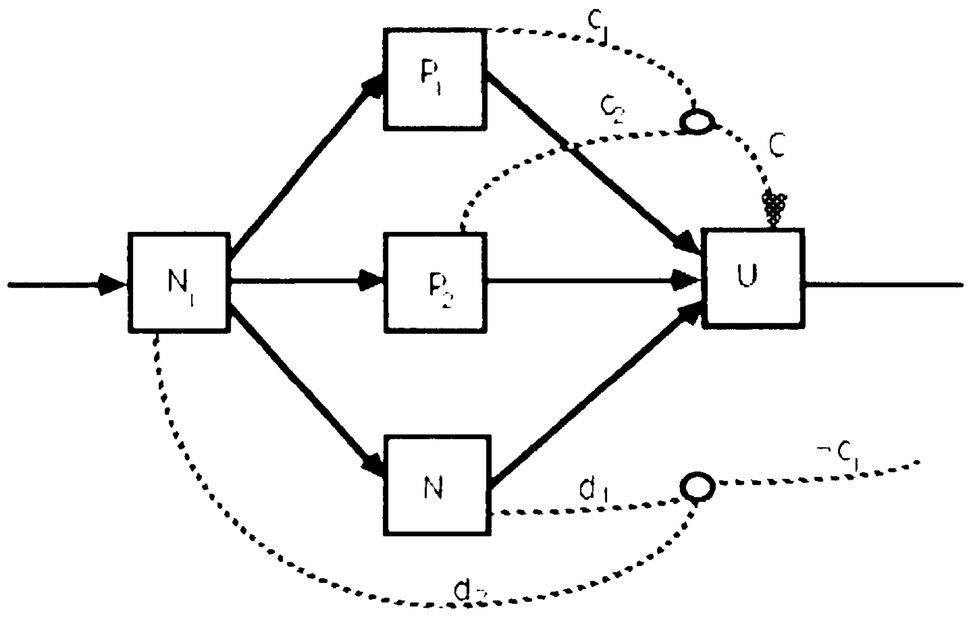

Figure 3: An example of a general conflict in $(O, <)$

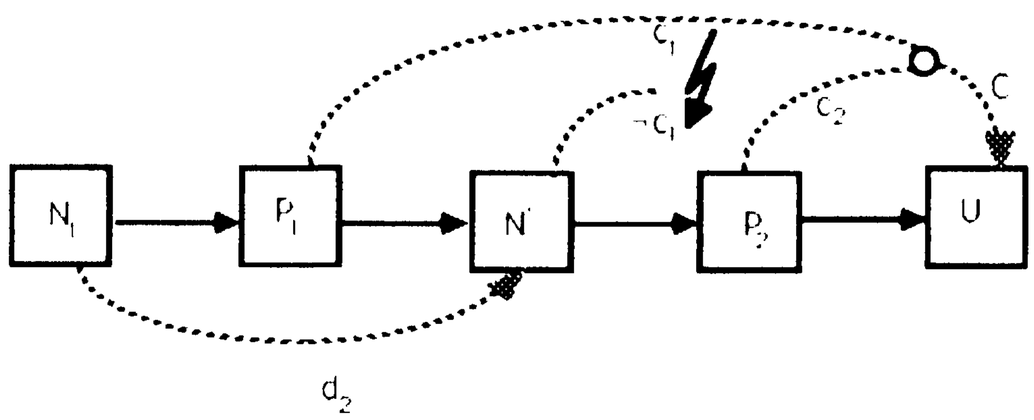

Figure 4: The corresponding conflict in $(O', <'')$

Consider as an example the plan shown in figure 3, where $\mathcal{P} = \{P_1, P_2\}$ is a general producer of C with supporting conditions set $\mathcal{S} = \{c_1, c_2\}$ and there is a general

dependency $\mathcal{D}$ between $\mathcal{P}$ and U concerning C. There is a conflict between $\mathcal{D}$ and $\mathcal{N}=\{N, N_1\}$ for the following reasons. $\mathcal{D}$ generally encloses $N \in \mathcal{N}$ by P_1 with enclosure $<'$, the transitive closure of $< \cup \{(P_1, N)\}$. Moreover, N is guaranteed to be an element of a general producer of $\neg c_1$. We can check the latter in the following way. We replace N by another node N' that is identical to N but has d_2 as an additional precondition. We then have to find an extension $<''$ of $<'$ such that the resulting dependency $\mathcal{D}'$ concerning d_2 is not in general conflict with some node, in particular not with P_1. (Of course, possibly $<''=<'$; the two step construction of $<''$ is just for clarity.) A possible transformation of the example plan is shown in figure 4.

This intuition can be generalized to the following definition.

Definition 3.4 (General Conflict) *Let $\Pi=(O,<)$ be a plan, $\mathcal{P}=\{P_1, \ldots, P_p\}$, $\mathcal{D}=(\mathcal{P}, C, U, \mathcal{S}, \Pi)$ a general dependency, and $\mathcal{N} \subseteq O$.*
Then $(\mathcal{D}, \mathcal{N})$ is a general conflict *between $\mathcal{D}$ and $\mathcal{N}$ iff*
$\exists <' \subseteq O \times O \; \exists i \in \{1, \ldots, p\} \; \exists N \in \mathcal{N}$:

- *$\mathcal{D}$ generally encloses N with $<'$ by P_i,*
- *$\mathcal{N}$ is a general producer of $\neg c_i$ with supporting conditions set $\{d_1, \ldots, d_n\}$*
- *there is $<'' \subseteq O \times O$ such that $<'' \supseteq <'$ and the general dependency*

$$\mathcal{D}'=(\mathcal{N} \setminus \{N\}, c_i, U, \{d_1, \ldots, d_n\}, (O', <''))$$

is not generally conflicted in $(O', <'')$.

where $(O', <')$ is the plan resulting from replacing the node $N \in O$ by N' with elements of $\{d_1, \ldots, d_n\} \setminus Post(N)$ as additional preconditions in $(O, <')$.

Assuming finite node sets, general conflicts are well defined.

The types of conflicts can be generalized from section 2.1 accordingly, respecting the single nodes involved in a general conflict and their relative positions in a plan. There are, of course, combination types of basic conflicts forming one single general conflict. In addition, there can be combination types of the basic conflict types with respect to some general dependency (where some nodes of $\mathcal{N}$ are ordered with respect to some of the P_i and others are not).

Note that the concept of general conflict –different from Chapman's [1987] modal truth criterion–permits handling effects of "synergy". In synergy cases one cannot assume locality, it is true, but conditions are produced or destroyed by sets of nodes working together in the way required for detecting general conflicts.

What, then, are the assumptions under which the concept of general conflict makes sense? Obviously, we give up the locality assumptions, but we still employ a *weakened version* of the STRIPS assumption, for we still have to compute all the *possible* effects of every operator:

> Operator descriptions define functions from states of application to sets of states containing at least *every* possible resulting state.

This is also true for abstract operators, and so we have rediscovered a special case of the problems [Wilkins, 1986] has reported for these. Theoretically, there is a loophole for managing conflicts in plans involving abstract operators anyway: consider every expansion into elementary operators. But this introduces just the sort of complexity one wanted to avoid by using abstract operators.

When giving up the original STRIPS assumption, we cannot detect everything one might intuitively call a conflict in some plan. This is, however, a general limitation of conflict detection: you cannot expect to know the exact effects and side effects of an operator of which you do not know what it actually does!

3.2 Handling General Conflicts

As to the practical effects of the generalized conflict definition, detecting conflicts works as before, but using definition 3.4 now; conflict resolution is analogous to the simple case, where combination types of general conflicts are resolved like combinations of elementary types. So, conflict handling remains the same, it is just more complex.

But *how much* more complex is it?

When, in the simple case of conflict, one has to check all single nodes enclosed by some dependency for possibly destroying and not restoring its single condition, then one now has to check *every subset* of the set of nodes ordered *not after the user* with *at least one* node enclosed by *at least one* branch of a general dependency, ('becking now means checking the postconditions of *all* the nodes to be consistent with *every* supporting conditions set. In short: we cannot any more restrict conflict detection to the area enclosed by the dependency involved, but have to consider the whole part of the plan not definitely after the user.

Although there may be strategies for reducing the complexity of the consistency checking (the work necessary for the different proofs is highly overlapping, suggesting the use of some context management mechanism like an ATMS) it seems obvious that handling general conflicts thoroughly is practically infeasible. So, we have identified another decision point where domain dependent knowledge must come in when giving up the locality assumption or the STRIPS assumption:

- given a plan, decide which dependencies to examine for being in general conflict; given a general dependency to examine, decide which nodes in the rest of the plan to check for being in general conflict with it.

4 Conclusion

We have sketched a theory of conflict handling in nonlinear plans. Starting from the concepts described in the literature for the simple case of single producers and single destroyers under the STRIPS and locality assumptions, we have sketched the problems arising when considering conflict handling for the general case, and we have sketched some solution ideas. But there is a general caveat: woe to the reckless conflict handler who is not equipped whith the appropriate domain knowledge!

Acknowledgements

Thanks to Torn Gordon and Uli Junker for comments on earlier versions of this paper.

References

[Chapman, 1987] Chapman, D.: Planning for Conjunctive Goals. *Art. Int.,* 32 (1987), pp. 333-377

[Dean and McDermott, 1987] Dean, T.L./ McDermott, D.V.: Temporal Data Base Management. *Art. Int.,* 32 (1987), pp. 1-55

[Fikes *et ai,* 1972] Fikes, R.E./ Hart, P./ Nilsson, N.J.: Learning and Executing Generalized Robot Plans. *Art. Int.,* 3 (1972), pp. 251-288

[Fikes *et a*l., 1971] Fikes, R.E./Nilsson, N.J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Art. Int.,* 2 (1971), pp. 189-208

[Hertzberg and Horz, 1988] Hertzberg, J./ Horz, A.: Erkennung und Auflosung von Konflikten in nichtlinearen Planen. In: Forschungsgruppe Expertensysteme (ed.): Aus der Arbeit dcr Forschungsgruppe Expertensysteme. Arbeitspapiere der GMD No. 337 (1988)

[Lifschitz, 1986] Lifschitz, V.: On the Semantics of STRIPS. In: Georgeff, MP./ Lansky, A L. (eds): *Proc. of the 1986 Workshop "Reasoning about Actions and Plans".* Los Altos: Morgan Kaufmann, 1986

[Sacerdoti, 1977] Sacerdoti, E. D.: *A Structure for Plans and Behavior.* New York: Elsevier North Holland, 1977

[Tate, 1975] Tate, A.: Interacting Goals and their Use. *Proc. IJCAI-I975,* pp. 215-218

[Tate, 1977] Tate, A.: Generating Project Networks. *Proc. IJCAI-1977,* pp. 888-893 (1977)

[Sussman, 1975] Sussman, G.J.: *A Computer Model of Skill Acquisition.* New York: Elsevier North Holland, 1975

[Waldinger, 1977] Waldinger, R.: Achieving Several Goals Simultaneously. *Mach. Int.,* 8 (1977), pp. 94 136

[Wilkins, 1984] Wilkins, D.E.: Domain-independent Planning: Representation and Plan Generation. *Art. Int.,* 22 (1984), 269-301

[Wilkins, 1986] Wilkins, D.E.: Hierarchical Planning: Definition and Implementation. *Proc. FJCAI-86,* 466-478 (1986)