Computer Science
University of Manchester

ConstraintLogicPlanning

YingZhang,HowardBarringer

TechnicalReportUMCS--94--9--2

Constraint Logic Planning *

Ying Zhang, howard Barringer

Department of Computer Science
University of Manchester
Oxford Rd., Manchester, U.K.
yzhang@cs.man.ac.uk, howard@cs.man.ac.uk

September 1994

Abstract

Deductive planning provides a rigorous approach for planning, such that the capabilities and limitations of the planning systems can be clearly defined. However, most of the deductive planners suffer from the frame problem and they are linear planners. This report presents a novel nonlinear deductive planning framework: *constraint logic planning*. Procedurally speaking, nonlinear planning is an activity of constraint solving. Constraint logic planning is a combination of the constraint solving for nonlinear planning with a temporal logic paradigm. In comparison with other state based deductive planning methods, constraint logic planning deals with the frame problem efficiently in a way similar to that in classical nonlinear planning, and constraint logic planning is a nonlinear deductive planning approach, it employs the least commitment strategy for both temporal relation decision and object selection. In comparison with classical nonlinear planning systems, constraint logic planning is a deductive planning approach and it achieves completeness, soundness and conflict freeness. Constraint logic planning is believed to be even more efficient than classical nonlinear planning.

Recent technical reports issued by the Department of Computer Science, Manchester University, are available by anonymous ftp from ftp.cs.man.ac.uk in the directory pub/TR. The files are stored as PostScript, in compressed form, with the report number as filename. They can also be obtained on WWW via URL http://www.cs.man.ac.uk/csonly/cstechrep/index.html. Alternatively, all reports are available by post from The Computer Library, Department of Computer Science, The University, Oxford Road, Manchester M13 9PL, UK.

1 Introduction

For deductive planners, planning is a reasoning process within some kind of formal logic. Deductive planning provides a rigorous approach for planning such that the capabilities and limitations of planning systems can be clearly defined. In particular, in a deductive planning framework, a planning problem to be solved can be examined independently from the procedural details of finding a plan. In addition, deductive planning systems are more expressive than classical planning systems because of their use of formal logics (Allen, 1990). However, most deductive planners have the frame problem and they are linear planners (Zhang, 1994). Therefore, they are less efficient than classical nonlinear planners and they are not widely accepted by planning researchers. On the other hand, without a formal framework, most efficient nonlinear planners are "complicated, heuristic, ill defined AI programs without clear conditions under which they work" (Chapman, 1987). To quote Sacerdoti:

> [The basic operations of NOAH] were developed in an ad hoc fashion. No attempt has been made to justify the transformations that they perform, or to enable them to generate all transformations. However, it should be possible to define an algebra of plan transformations ... a body of formal theory about the ways in which interacting subgoals can be dealt with (Sacerdoti, 1977).

Providing a formal framework for nonlinear planning motivates the work presented in this report.

Procedurally speaking, nonlinear planning is an activity of constraint solving. In a nonlinear planning system, generally, plans are partially described at least until their execution. A constraint is essentially a partial description of an object. A nonlinear planner solves a problem by building up progressively the constraints that partially describe the plan and the world it represents(Stefik, 1981; Chapman, 1987; Zhang, 1994).

Constraint logic programming schema is a formal framework for a class of programming languages which share the same essential semantics (Jaffar & Lassez, 1987b; Jaffar & Lassez, 1987a; Colmerauer, 1987; Colmerauer, 1990). Constraint logic programming combines constraint solving in a specific problem domain with the logic programming paradigm. This combination provides a overall rule based framework for reasoning about constraints.

Borrowing the basic ideas of the constraint logic programming, we combine constraint solving for nonlinear planning with a temporal logic programming paradigm. This combination provides a promising formal framework, *constraint logic planning*, for nonlinear planning. In constraint logic programming, "the basic components of a problem are stated as constraints. The problem as a whole is then represented by putting the various constraints together by means of rules" (Jaffar & Lassez, 1987b). In our constraint logic planning, the basic components of nonlinear plans, i.e., the actions and the temporal relations among them, are represented as constraints. To solve a planning problem is to progressively construct the nonlinear plan by putting the related constraints together by means of reasoning about *planning rules*, which are specification of actions in the temporal logic. In constraint logic programming, the central elements in the semantics are a many sorted structure and the corresponding theory. In constraint logic planning, a temporal model of the temporal logic corresponds to a many sorted structure and a set of specific temporal formulas: *action constraints* introduced in (Zhang & Barringer, 1994b) gives the central part of the corresponding theory. In constraint logic programming, a constraint is satisfiable in a many sorted structure if and only if it is satisfiable in the corresponding theory. In constraint logic planning, to check if a constraint is consistent is to check if it is satisfiable in the corresponding theory. In (Zhang & Barringer, 1994b), we proved that a constraint is consistent with the corresponding theory if and only if the nonlinear plan that the constraint represents is conflict free. This result lays down the foundation for soundness and completeness of constraint logic planning.

In comparison with other state based deductive planning methods, constraint logic planning deals with the frame problem efficiently in a way similar to that in classical nonlinear planning, and constraint logic planning is a nonlinear deductive planning approach, it employs the least commitment strategy for both temporal relation decision and object selection. In comparison with the classical nonlinear planning systems, constraint logic planning is a deductive planning approach and it achieves completeness, soundness and conflict freeness. Constraint logic planning is believed to be even more efficient than classical nonlinear planning.

2 Preliminaries

2.1 A Reified Temporal Logic for Nonlinear Planning

Deductive planning is a reasoning process within some formal logic. In (Zhang & Barringer, 1994a), we presented a point based, reified interval temporal logic for nonlinear planning. In the temporal logic, time is discrete and is in a partial order structure; time points are the only temporal primitives and, therefore, time intervals are referred to by their end-points; assertions of proposition types are interpreted over maximal time intervals; a proposition type is associated with a time interval through a global predicate; there are two proposition types, property and action, in the logic; to distinguish between these two types, we use two global predicates $Hold$ and $Exec$. The atomic formula $Hold(P, t_i, t_j)$ expresses that the property P holds true over the maximal time interval (t_i, t_j) and the atomic formula $Exec(A, t_i, t_j)$ expresses that the action A is executed over the maximal time interval (t_i, t_j). An interpretation of the temporal logic is a tuple $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \mathcal{D}, \pi_F, \pi_P \rangle$, where $\mathcal{W}$ is a set of time points; $\mathcal{R}$ is a partial order relation among the time points in $\mathcal{W}$; $\mathcal{D}$ is the domain of non-temporal variables; π_F is an interpretation of the function symbols and π_P is an interpretation of the proposition types. In appendix B, we give the formal definition of the temporal logic.

In planning, a given problem domain can be described by a set of actions Γ and a set of domain constraints Σ_D. In this temporal language, each action in Γ can be formally specified by a set of formulas in Horn clause form.

Example 1 *In the blocks world, let $Puton(x, y, z)$ be an action that moves block x from y to z, as shown in Figure 1a*[1]*. This action can be specified by the following two clauses.*

$$Hold(On(x, z), t_1, SKF1(t_1, x, y, z)) :- \\ Hold(Clear(x), t_2, t_2'), Hold(Clear(z), t_3, t_1), Hold(On(x, y), t_4, t_1),$$

[1]In this definition, x is not equal to the table. If y is equal to the table, then $Clear(y)$ is not an asserted property. If z is equal to the table, then $Clear(z)$ is not a deleted property.

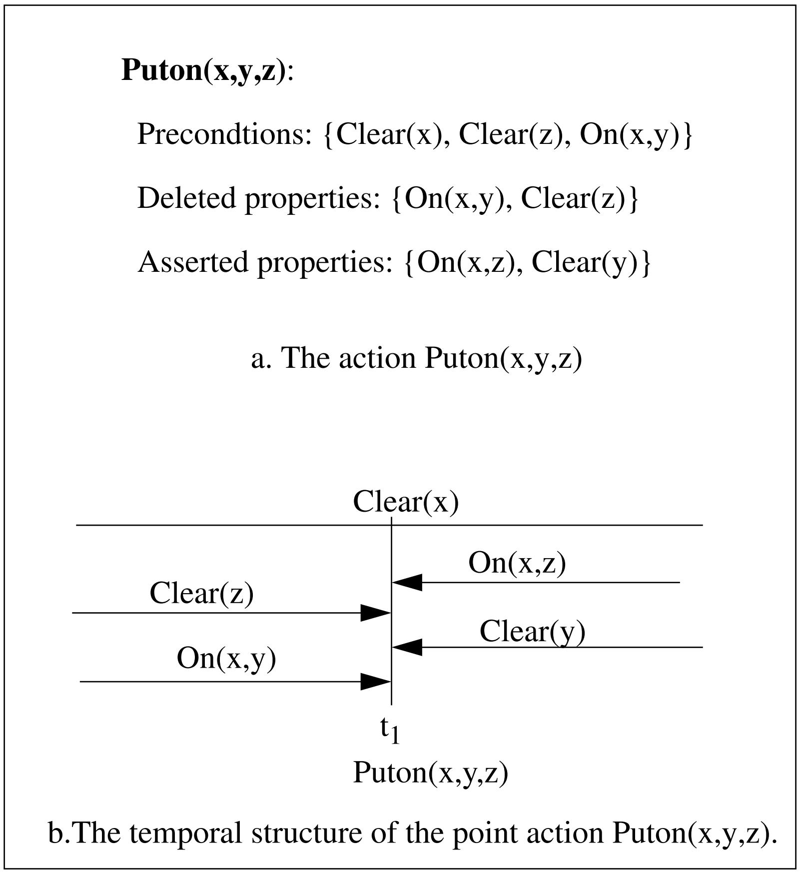

Figure 1: The point action Puton(x,y,z).

$$Exec(Puton(x,y,z),t_1,t_1), t_2 \prec t_1 \prec t_2';$$

$$\begin{aligned}&Hold(Clear(y),t_1,SKF2(t_1,x,y,z)) :-\\ &\quad Hold(Clear(x),t_2,t_2'), Hold(Clear(z),t_3,t_1), Hold(On(x,y),t_4,t_1),\\ &\quad Exec(Puton(x,y,z),t_1,t_1), t_2 \prec t_1 \prec t_2';\end{aligned}$$

The causal relations and the temporal relations of the properties and the action involved are clearly specified in the formulas. For example, the first formula expresses that if the blocks x and z are clear, x is on y and the action $Puton(x,y,z)$ is executed, then x will be on z. The formula also expresses that the temporal intervals over which the deleted properties $Clear(z)$ and $On(x,y)$ hold true are finished at the time point t_1 at which the action is executed, the intervals over which the undeleted precondition $Clear(x)$ holds overrids the time point t_1, and the interval over which the asserted property $On(x,z)$ holds is started from t_1. SKF1 and SKF2 are Skolem functions.

In our temporal logic, assertions of proposition types are interpreted exclusively over the necessarily maximal intervals. The fact that a property p is asserted over an interval (t_i,t_j), $Hold(p,t_i,t_j)$, implies that (t_i,t_j) is the maximal interval for p; i.e. p is asserted at t_i and deleted at t_j. Therefore, causal relations and temporal relations of actions can be easily, clearly and directly expressed.

In general, in a given planning problem domain, the set of (point) actions Γ is a set of planning rules of the form

$$Hold(p, t, SKF(t, v_1, \ldots, v_m)) : -B_1, \ldots, B_n, Exec(A, t, t), C$$

where B_i, $i = 1, \ldots n$, the bodies of the planning rule are of the form $Hold(q_i, t_i, t_i')$, A is the action of the rule and C is a formula consists of the temporal constraints. If the above planning rule is in Γ, then p is said to be *an asserted property of* A, p is also said to be a *postcondition of* A. q_i, $i = 1 \ldots n$, are said to be the *preconditions of* A. For any B_i, if it is of the form $Hold(q_i, t_i, t)$, then it is said to be *a deleted property of* A. Otherwise, it is also said to be a postcondition of A. SKF is a Skolem function and $v_1, \ldots, v_m$ are all the universally quantified data variables that appear in p.

In a state-based language, like situation logics, a domain constraint expresses that two properties which are defined to be mutually exclusive are not allowed to hold true at the same state or time point. In our nonlinear interval temporal logic, if two properties are defined to be mutually exclusive, then the temporal intervals over which they hold true respectively must not *possibly* and properly overlap.

In our temporal logic, a set of domain constraints Σ_D for a problem domain is a set of formulas of the form:

$$\forall t_1, t_2, t_3, t_4.(Hold(p, t_1, t_2) \land Hold(q, t_3, t_4) \Longrightarrow t_2 \preccurlyeq t_3 \lor t_4 \preccurlyeq t_1)$$

If Σ_D contains the above formula, the properties p and q are said to be *mutually exclusive* according to Σ_D.

2.2 Action Constraint Maintenance

In our temporal logic, a nonlinear plan can be represented as a temporal formula. For example, in the blocks world, let P be a nonlinear plan consisting of two actions $Exec(Puton(A, T, B), t_1, t_1)$, which moves block A from the table T to the top of block B at the time point t_1, and $Exec(Puton(B, T, C), t_2, t_2)$, which moves block B from the table T to the top of block C at the time point t_2. In P, both time points t_1 and t_2 are after the initial time point T_0 and there is no direct temporal relation between t_1 and t_2. This nonlinear plan can be expressed as a well formed formula in our temporal logic as follows.

$$\begin{aligned} P \;=\; & Exec(Puton(A, T, B), t_1, t_1) \land T_0 \prec t_1 \land \\ & Exec(Puton(B, T, C), t_2, t_2) \land T_0 \prec t_2 \end{aligned}$$

Syntactically, a nonlinear plan can be expressed as a well formed temporal formula. Semantically, a nonlinear plan describes a nonlinear temporal model. That is, under the *closed world assumption* and the specification of actions, a temporal model can be constructed for a given nonlinear plan. For example, based on the action $Puton(x, y, z)$ given in Figure 1, the temporal model for the above nonlinear plan is shown in Figure 2. In this figure, T_∞ represents the universal ending point. A bi-directed line of a property indicates a time interval over which the property hold true. For example, the property $On(B, T)$ holds true over the interval that is started from T_0 and is ended at t_2.

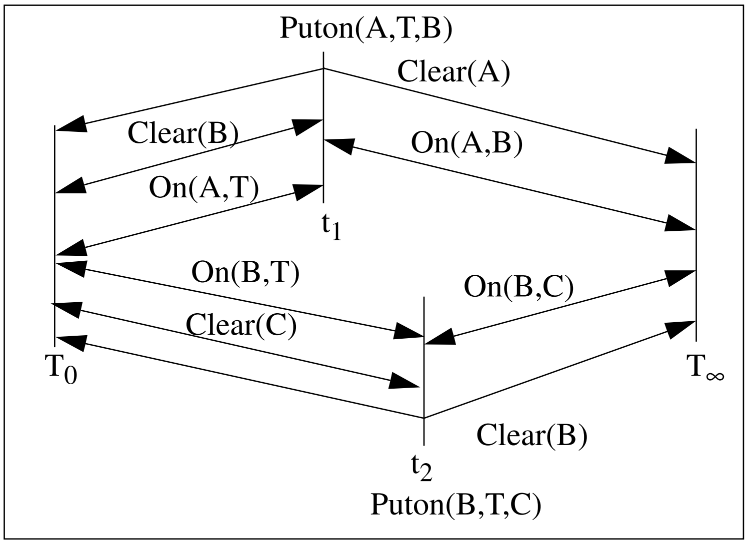

Figure 2: The temporal model of a nonlinear plan which contains conflict.

In (Zhang *et al.*, 1994), we proposed a set of restrictions on actions and domain constraints. We showed that the set of restrictions can prevent actions from being improperly defined. We also argued that actions in most of the problem domains either satisfy the restrictions or can be reasonably modified to satisfy the restrictions. We proved that, under the set of restrictions on actions, a nonlinear plan is conflict free if and only if its temporal model satisfies the corresponding set of domain constraints. For example, the nonlinear plan given above contains conflict. The action $Puton(A, T, B)$ possibly deletes one of the preconditions $Clear(B)$ of the action $Puton(B, T, C)$. This conflict in P can be detected by the fact that the domain constraint

$$Hold(On(A, B), t_1, T_\infty) \wedge Hold(Clear(B), T_0, T_\infty) \Longrightarrow T_\infty \preccurlyeq t_1 \vee T_\infty \preccurlyeq T_0$$

is not satisfied in the temporal model of P, shown in Figure 2.

In (Zhang & Barringer, 1994b), we showed that, from a set of domain constraints Σ_D, we can obtained a set of temporal formulas Σ_Γ, which we termed *action constraints*. For example, one such action constraint in the blocks world expresses that the actions $Puton(x, y, z)$ and $Puton(z, y', x)$ should not properly overlap. Suppose both the two actions are point actions, the action constraint can be expressed as:

$$\forall t_i, t_j, x, y, z, y'.(Exec(Puton(x, y, z), t_i, t_i) \wedge Exec(Puton(z, y', x), t_j, t_j) \Longrightarrow t_i \neq t_j)$$

We proved that nonlinear plans that satisfy a certain property are conflict free if and only if they satisfy the corresponding set of action constraints (we showed that all nonlinear plans constructed from our constraint logic planning satisfy this property and we also proved that all the nonlinear plans that do not satisfy this property contain conflicts). Therefore, under certain conditions, to verify the conflict freeness of a nonlinear plan, we need only check that the set of action constraints is satisfied by the nonlinear plan itself. We termed this conflict detecting method *action constraint maintenance*. We showed that action constraint maintenance is much more efficient than conventional conflict detecting methods for nonlinear plans. We also showed that, if a nonlinear plan is not consistent with an action constraint, all its possible expansions will not be consistent with the action constraint.

In the following discussion, when we mention a set of actions Γ (or a definite planning program P) and the corresponding action constraints Σ_Γ, we mean that the Σ_Γ is constructed from a set of domain constraints Σ_D for Γ (resp. P) and the Σ_D and Γ (resp. P) satisfy the restrictions.

3 Constraint Logic Planning

Constraint logic planning is directly built upon the action constraint maintenance. Planning in our framework, we simply continuously introduce actions with the corresponding temporal relations to achieve goals by resolving the goals with the corresponding planning rules and check if the plan under construction is consistent with the corresponding set of action constraints. The soundness and completeness of the action constraint maintenance guarantee that a constructed plan is conflict free. The property that if a nonlinear plan is not consistent with an action constraint, all the possible expansions of the plan will not be consistent with the constraint guarantees that backtracking from an inconsistency will not lose possible solutions. This lays down the foundation for the completeness of constraint logic planning.

3.1 Definite Constraint Logic Planning Programs

We have presented an interval temporal logic for nonlinear planning. We are not interested in the general decision problem of the temporal logic. Our purpose is to use the language to provide a formal framework for nonlinear planning. To obtain the efficiency for this restricted application, we distinguish the interval temporal logic from the planning language: definite constraint logic planning language. The planning language is a special version of a subset of the temporal logic.

The basic components and semantics of the definite planning language is the same as that of the interval temporal logic. However, in this planning language, the atomic formulas are divided into two disjoint sets, the *properties* and the *plan constraints*. Our intention is to use plan constraints to characterize the nonlinear plans and use property predicates to reason about the plan construction.

The atomic properties are of the form: $Hold(p, t_i, t_j)$ where p is a property.

There are three kinds of atomic plan constraints.

1. The atomic formulas which are of the form $t_i \prec t_j$ where t_i and t_j are temporal terms. Such an atomic constraint gives the temporal relation between two time points.

2. The atomic formulas which are of the form $Exec(a, t_i, t_j)$ where t_i and t_j are temporal terms and a is an action. Such an atomic constraint binds an action to a time interval.

3. The atomic formulas which are of the form $t_i = t_j$ where t_i and t_j are temporal terms.

A *plan constraint* is a conjunction of atomic plan constraints.

Definition 1 *A* definite constraint logic planning program *consists of a set of* definite constraint planning rules. *A definite constraint planning rule is a modified notation of the Horn clause of the form:*

$$H : -B_1, \ldots, B_m | C$$

where the head of the rule, H, and the bodies of the rule, B_i, $i = 1, \ldots, m$, are atomic properties. The constraint of the rule, C, is a plan constraint.

For example, in the blocks world, the two definite constraint planning rules which specify the point action $Puton(x, y, z)$ are as follows.

$$\begin{aligned}&Hold(On(x,z), t_1, SKF1(t_1, x, y, z)) : -\\&\quad Hold(Clear(x), t_2, t_2'), Hold(Clear(z), t_3, t_1), Hold(On(x,y), t_4, t_1)\\&\quad |Exec(Puton(x,y,z), t_1, t_1) \wedge t_2 \prec t_1 \prec t_2';\end{aligned}$$

$$\begin{aligned}&Hold(Clear(y), t_1, SKF2(t_1, x, y, z)) : -\\&\quad Hold(Clear(x), t_2, t_2'), Hold(Clear(z), t_3, t_1), Hold(On(x,y), t_4, t_1)\\&\quad |Exec(Puton(x,y,z), t_1, t_1) \wedge t_2 \prec t_1 \prec t_2';\end{aligned}$$

There is no substantial difference between the definition of definite constraint planning rules and that of planning rules. Therefore, a definite constraint planning program can also be regarded as a set of actions. The restrictions on actions introduced in (Zhang *et al.*, 1994) are added to a definite planning program in the same way as to a set of actions.

In this planning language, the goals for a given planning problem can be expressed as a definite constraint planning rule with an empty head: $: -G_1, \ldots, G_n | C_0$ where $G_1, \ldots, G_n$ are the goals to be achieved and C_0 describes the temporal relation between the individual goals.

For example, in the blocks world, the goal that block A is on block B and B is on block C should be precisely described as follows:

> There exists a time point t such that the two time intervals over which the properties $On(A, B)$ and $On(B, C)$ hold respectively properly override t.

This goal can be expressed as a definite constraint planning rule:

$$: -Hold(On(A,B), t_1, t_2), Hold(On(B,C), t_3, t_4) | t_1 \prec t \prec t_2 \wedge t_3 \prec t \prec t_4$$

3.2 Constraint Logic Planning

Suppose that definite constraint planning rules are represented in the form:

$$Con\_Rule(Hold(p, t_1, t_2{}^2), Body, Constraint)$$

Then the basic interpreter for the planning language is given in Fig. 3, in a PROLOG-like notation, as a procedure called Solve.

The procedure Solve has three parameters:

[2] It should be noticed that t_2 is not a universally quantified variable, but a temporal term (Skolem function).

```
Solve([],C,C);

Solve([Hold(p,t1,t2)|Restgoal],Pre_C, New_C):-
      Achieve(Hold(p,t1,t2),Pre_C, Temp_C),
      Solve(Restgoal, Temp_C, New_C);

Achieve(Hold(p,t1,t2), Pre_C, Temp_C):-
      Con_Rule(Hold(p',t1',t2'), Body, Constraint),
      Unifiable(p,p',θ),
      Consistent((Pre_C ∧ Constraint ∧ t1=t1' ∧ t2=t2')θ),
      Solve(Bodyθ, (Pre_C ∧ Constraint ∧ t1=t1' ∧ t2=t2')θ, Temp_C);
```

Figure 3: Basic interpreter of definite constraint logic planning programs.

1. the list of the properties to be achieved,
2. the previous constraint and
3. the new constraint obtained from the previous constraint by solving the goals.

One of the subprocedures of Solve is Achieve, which achieves one of the properties. The parameters of Achieve are the same as those of Solve except that Achieve has only one property as its first parameter rather than a list of properties. The procedure $Unifiable(p, p', \theta)$ returns a most general (data) unifier θ for the properties p and p' if it exists.

The heart of the interpreter is the procedure Consistent which checks if the previous constraint Pre_C is consistent with the constraint of the selected planning rule Constraint and the new temporal bindings $t_1 = t_1' \wedge t_2 = t_2'$ under the data mgu θ. It should be noticed that both Pre_C and $Constraint$ are consistent themselves. If they are consistent, the procedure will stop and return the conjunction as the new constraint. Otherwise, the procedure will stop and report an inconsistency. In this case, the selected goal fails to be resolved with the current planning rule.

A plan constraint represents a nonlinear plan. Consistency checking of a plan constraint C is to check if C is consistent with the corresponding set of action constraints Σ_Γ, the axioms of partial order relation and the equality theory for time points. Because of the soundness and completeness of the action constraint maintenance, if all the goals are achieved, the constraint left describes a conflict free nonlinear plan.

Suppose that P is a definite planning program and G is a set of goals. For a planning problem, initially, G contains the given goals to be achieved. Planning in this framework, the interpreter described above resolves the goals in G with planning rules in P. If all the goals are achieved successfully, the constraint left describes a desired plan for the given problem. During this planning process, temporal constraints between actions (e.g. atomic constraints of the form $t_i \prec t_j$) are added to the plan only when necessary, i.e.

only when the corresponding actions are introduced. The temporal relation among the actions of the constructed plan remains in partial order during the planning process. Therefore, the planning process is nonlinear. Later, we will prove that, this deductive nonlinear planning approach is sound and complete.

4 Declarative Semantics of Planning Program

In first order logic, for a given language, the Herbrand base is the set of all the ground atoms with ground terms from the Herbrand universe as arguments. A Herbrand model and a Herbrand interpretation for a set of clauses is a subset of Herbrand base. A set of clauses has a model if and only if it has a Herbrand model.

In logic programming, a definite logic program P is a set of Horn clauses of the form

$$p : -q_1, \ldots, q_n; (n \geq 0)$$

A definite goal G is a clause of the form $: -B_1, \ldots, B_m,\ m \geq 1$. Firstly, we assume that all literals of the goal are ground. The first theoretical result of logic programming connects the logical and operational meaning of logic programs. This result is due to Hill (Hill, 1974) and it can be summarized as follows.

> A SLD breadth-first theorem prover will provide a yes answer to the query $Q = \neg G$ if and only if $P \cup G$ is unsatisfiable.

The second main theoretical result, which is due to van Emden and Kowalski (van Emden & Kowalski, 1976), provides a fixed point characterization of Least Herbrand Model of a definite program. Given a definite program P, 2^{B_P}, which is the set of all Herbrand interpretations of P (or the set of all the subsets of the Herbrand base of P), is a complete lattice under the set inclusion relation $\subseteq$. The top element of this lattice is the Herbrand Base B_P and the bottom element is ϕ. Emden and Kowalski defined a mapping $T_P : 2^{B_P} \longrightarrow 2^{B_P}$, which provides the link between the declarative semantics and the operational semantics of a logic program. Suppose that P is a definite program and $S \subseteq B_P$ is a Herbrand interpretation, then

$$\begin{aligned} T_P(S) \quad = \quad & \{A \in B_P :\ A : -B_1, \ldots, B_n \text{ is a ground instance} \\ & \text{of a clause in } P \text{ and } \{B_1, \ldots, B_n\} \in S\} \end{aligned}$$

Let the ordinal power of the mapping T_P be defined as follows:

$$\begin{aligned} T_P \uparrow 0 &= \perp \\ T_P \uparrow \alpha &= T_P(T_P \uparrow (\alpha - 1)) \text{ if } \alpha \text{ is a successor ordinal} \\ T_P \uparrow \alpha &= lub\{T_P \uparrow \beta : \beta < \alpha\}, \text{ if } \alpha \text{ is a limit ordinal} \end{aligned}$$

Then the second main theoretical result of logic programming can be state as follows.

> $P \cup G$ is unsatisfiable if and only if the literals in the body of G are members of the least Herbrand models which equals to the least fixed point set $T_P \uparrow \omega$[3] of P.

[3] $\omega = \{0, 1, 2, \ldots\}$ is the smallest limit ordinal.

The above two results can be generalized to include goals containing variables. The term *correct answer* provides a declarative description of the desired output from a definite program and a definite goal. A substitution θ for variables in G is defined to be a correct answer for $P \cup G$ if $\forall((B_1, \ldots, B_m)\theta)$ is a logical consequence of P. The procedural counterpart of a correct answer for $P \cup G$ is a *computed answer* which is obtained from a SLD refutation of $P \cup G$. Then, corresponding to the first result, for a definite program and a definite goal G, every computed answer is a correct answer and every correct answer is an instance of a computed answer (Lloyd, 1987). Corresponding to the second result, a substitution θ is a correct answer for P and G if and only if $(B_1, \ldots, B_m)\theta$ is true with respect to the least Herbrand model of P.

Jaffar and Lassez (Jaffar & Lassez, 1987b) proved that the main theoretical results in logic programming remain applicable in constraint logic programming, provided that the domains being considered satisfy certain conditions. The central element in the semantics of constraint logic programming is a many sorted structure $\mathcal{A}$, which plays the same role as the Herbrand Universe does in the semantics of logic programming. Apart from $\mathcal{A}$, the other component is a theory $\mathcal{T}$. A theory $\mathcal{T}$ is said to be *corresponding to* $\mathcal{A}$ if the following conditions hold.

1. $\mathcal{A}$ models $\mathcal{T}$, i.e., $\mathcal{A} \models \mathcal{T}$ and

2. for any constraint C, $\mathcal{A} \models \exists C$ implies $\mathcal{T} \models \exists C$.

For each CLP program P, corresponding to the mapping T_P given by Emden and Kowalski (76), Jaffar and Lassez define a function from and into $\mathcal{A}$ base, the "cross product between the predicate symbols in P and $\mathcal{A}$", as:

$$\begin{aligned} T(S) \quad = \quad & \{d \in \mathcal{A} : \text{ if there is a } \mathcal{A} \text{ evaluation } \theta \text{ and} \\ & \text{a rule in P, } A : -\tilde{c}, \tilde{B} \text{ such that} \\ & 1.\ \mathcal{A} \models A\theta = d \text{ and} \\ & 2.\ \mathcal{A} \models \tilde{c}\theta \text{ and} \\ & 3.\ \{\tilde{B}\theta\} \in S.\} \end{aligned}$$

This mapping is based upon $\mathcal{A}$ solvability, not unifiability in a Herbrand Universe. The two conditions that ensure the results in logic programming remain true in constraint logic programming can be stated as follows.

1. The many sorted structure $\mathcal{A}$ is *solution-compacted*, i.e., each element of the domain can be specified by a (possibly infinite) number of constraints, and the complement of any constraint can be specified by a (possibly infinite) number of constraints.

2. The theory $\mathcal{T}$ is *satisfaction-complete*, i.e., every constraint is either provably satisfiable or provably unsatisfiable.

Superficially, our constraint logic planning fits constraint logic programming well. The temporal model is a two sorted structure. The set of action constraints complemented with the equality and partial order relation axioms is the theory corresponding to the temporal model. However, the semantics of constraint logic programs are not applicable to our planning programs because of the first of the above conditions. Later in this chapter, we will show that the theory consisting of a set of action constraints and the equality and partial order axioms is satisfaction-complete, i.e., for any plan constraint C, the problem whether C is consistent with the theory or not is decidable. But our temporal model is not solution compact. A time point can't be specified by a finite or infinite set of constraints for the set of possible time points is infinite. For a plan constraint of the form $t_i \prec t_j$, its complement cannot be specified for the relation between time points is a partial order. If our temporal model were linear, the complement of $t_i \prec t_j$ would be $t_j \preccurlyeq t_i$.

Consequently, we have to introduce new concepts and techniques in order to obtain the corresponding results for constraint logic planning.

In constraint logic planning, data terms and temporal terms are treated differently. The data terms are interpreted over the classical Herbrand universe and the temporal terms are treated with constraint solving. These different treatments for data terms and temporal terms make it easier to define the semantics for constraint logic planning programs.

Later, we will show that it is natural to extend the method to the one in which both data terms and temporal terms are treated with constraint solving. In our reified temporal logic, temporal terms and proposition types are arguments of global predicates (Hold, Exec). This distinguishes temporal terms from other arguments of proposition types (data terms) and mades it possible to treat temporal terms and data terms by different mechanisms.

In this planning language, a plan constraint describes a nonlinear plan. Firstly, because our temporal models are nonlinear, a plan constraint C is required to be consistent with the axioms of the partial order relation for the set of time points, $T_{<_T}$. Secondly, because temporal terms are not treated by unification and equalities of temporal terms are atomic constraints, therefore, a plan constraint C is also required to be consistent with the equality theory for temporal terms, $T_{=_T}$. We have shown that a nonlinear plan is conflict free if and only if the plan satisfies the corresponding set of action constraints Σ_Γ (Zhang & Barringer, 1994b). In planning, we are only interested in the plans that are conflict free. Therefore, finally, C is also required to be consistent with Σ_Γ. Consequently, a plan constraint C is satisfiable by a temporal model of a conflict free nonlinear plan if it is consistent under the theory:

$$\mathcal{T} = \Sigma_\Gamma \cup T_{<_T} \cup T_{=_T}$$

Adopting the terminology of Jaffar and Lassez, $\mathcal{T}$ is the theory corresponding to the temporal model.

Now, we can define the concept *Herbrand property constraint base* for a given planning program and the corresponding action constraints.

Definition 2 *Given a definite planning program P and the corresponding set of action constraints Σ_Γ, the* Herbrand property constraint base for P and Σ_Γ, *$\mathcal{B}_{P\cup\Sigma_\Gamma}$, is the set of all the property constraint pairs of the form $Hold(p(dt_1, \ldots, dt_n), tt_1, tt_2)|C$, where p is a property symbol of arity n, $dt_1, \ldots, dt_n$ are ground data terms over the ordinary Herbrand universe, tt_1 and tt_2 are two ground temporal terms representing two time points, and C is a data ground plan constraint which is consistent under the theory $\mathcal{T} = \Sigma_\Gamma \cup T_{<_T} \cup T_{=_T}$.*

For a given planning program P and the corresponding set of action constraints Σ_Γ, the Herbrand property constraint base $\mathcal{B}_{P\cup\Sigma_\Gamma}$ plays the same role as the Herbrand base does in logic programming. In order to obtain the similar results in logic programming, we wish to investigate the deeper characteristics of $\mathcal{B}_{P\cup\Sigma_\Gamma}$ using fixed point theory. To achieve this, we need associate a complete lattice with $\mathcal{B}_{P\cup\Sigma_\Gamma}$.

The set of all the subsets of $\mathcal{B}_{P\cup\Sigma_\Gamma}$, $2^{\mathcal{B}_{P\cup\Sigma_\Gamma}}$, is a complete lattice under the set inclusion relation $\subseteq$. The top element of the lattice is $\mathcal{B}_{P\cup\Sigma_\Gamma}$, and the bottom element is ϕ. The least upper bound of any set of subsets of $\mathcal{B}_{P\cup\Sigma_\Gamma}$ is the union of all the elements in the set, and the largest lower bound is the intersection of all the elements in the set.

In logic programming, the mapping given by Emden and Kowalski $T_P : 2^{B_P} \longrightarrow 2^{B_P}$ plays the key role in linking the declarative semantics and the procedural semantics of logic programs. The mapping has been extended for constraint logic programming by Jaffar and Lassez. Different from constraint logic programming, our system is a hybrid one, that is, data terms and temporal terms are treated differently. We define a mapping which is applicable in constraint planning as follows.

Definition 3 *Let P be a definite planning program and Σ_Γ be the corresponding set of action constraints. The mapping from and to the Herbrand property constraint base of $P \cup \Sigma_\Gamma$, $T_{P\cup\Sigma_\Gamma} : 2^{\mathcal{B}_{P\cup\Sigma_\Gamma}} \longrightarrow 2^{\mathcal{B}_{P\cup\Sigma_\Gamma}}$, is defined as follows. Let S be a subset of $\mathcal{B}_{P\cup\Sigma_\Gamma}$, then*

$T_{P\cup\Sigma_\Gamma}(S) = \{Hold(p, t_1, t_2)|C \in \mathcal{B}_{P\cup\Sigma_\Gamma} :$
if there is a data ground instance of a planning rule

$$Hold(p, t'_1, t'_2) : -Hold(q_1, t'_3, t'_4), \ldots, Hold(q_m, t'_{2m+1}, t'_{2m+2})|C_0$$

such that

1. $Hold(q_i, t_{2i+1}, t_{2i+2})|C_i \in S$, $i = 1, \ldots, m$,
2. $C = \bigwedge_{i=0}^{m} C_i \wedge \bigwedge_{j=1}^{2m+2}(t_j = t'_j)$ *and* $(\Sigma_\Gamma \cup T_{<_T} \cup T_{=_T}) \models \exists C$

Like the corresponding mapping in logic programming, the mapping is *monotonic* and *continuous*.

Proposition 1 *Let P be a definite planning program and Σ_Γ be the corresponding set of action constraints. The mapping $T_{P\cup\Sigma_\Gamma}$ is* monotonic *with respect to the set inclusion relation $\subseteq$, that is, for any two subsets S_1 and S_2 of $\mathcal{B}_{P\cup\Sigma_\Gamma}$,*

$$S_1 \subseteq S_2 \Longrightarrow T_{P\cup\Sigma_\Gamma}(S_1) \subseteq T_{P\cup\Sigma_\Gamma}(S_2)$$

The proof of this proposition is straightforward.

Proposition 2 *Let P be a definite planning program and Σ_Γ be the corresponding set of action constraints. The mapping $T_{P\cup\Sigma_\Gamma}$ is* continuous*, that is, for any directed subset*[4] *X of $2^{\mathcal{B}_{P\cup\Sigma_\Gamma}}$,*

$$T_{P\cup\Sigma_\Gamma}(lub(X)) = lub(T_{P\cup\Sigma_\Gamma}(X))$$

[4] A set X is *directed* if every finite subset of X has an upper bound in X.

Proof: Let X be a directed subset of $2^{\mathcal{B}_{P\cup\Sigma_\Gamma}}$. First we proof that

$$\{e_1, \ldots, e_n\} \subseteq lub(X) \text{ iff } \{e_1, \ldots, e_n\} \subseteq S \text{ for some } S \in X$$

Without loss of generality, we can assume that $e_1 \in S_1, \ldots, e_n \in S_n$ and $S_1, \ldots, S_n \in X$. Then $\{e_1, \ldots, e_n\} \subseteq lub(X)$ if and only if there exist $S_1, \ldots, S_n \in X$ such that $e_1 \in S_1, \ldots, e_n \in S_n$. As X is a directed subset of $2^{\mathcal{B}_{P\cup\Sigma_\Gamma}}$, it must has an upper bound S for the finite subset $\{S_1., \ldots, S_n\}$. Therefore, $\{e_1, \ldots, e_n\} \subseteq lub(X)$ iff $\{e_1, \ldots, e_n\} \subseteq S$ for some $S \in X$.

Next, we prove that $T_{P\cup\Sigma_\Gamma}(lub(X)) = lub(T_{P\cup\Sigma_\Gamma}(X))$ for every directed subset X. According to the definition of $T_{P\cup\Sigma_\Gamma}$, $Hold(p, t_1, t_2)|C \in T_{P\cup\Sigma_\Gamma}(lub(X))$ if and only if there is a data ground instance of a planning rule:

$$Hold(p, t_1', t_2') : -Hold(q_1, t_3', t_4'), \ldots, Hold(q_m, t_{2m+1}', t_{2m+2}')|C_0$$

such that

1. $(Hold(q_i, t_{2i+1}, t_{2i+2})|C_i) \in lub(X)$, $i = 1, \ldots, m$. From the above result, $(Hold(q_i, t_{2i+1}, t_{2i+2})|C_i) \in S$, $i = 1, \ldots, m$, for some $S \in X$, and
2. $C = \bigwedge_{i=0}^{m} C_i \wedge \bigwedge_{j=1}^{2m+2} t_j = t_j'$ and $\Sigma_\Gamma \cup T_{<_T} \cup T_{=_T} \models \exists C$

if and only if $Hold(p, t_1, t_2)|C \in T_{P\cup\Sigma_\Gamma}(S)$ for some $S \in X$, if and only if $Hold(p, t_1, t_2)|C \in lub(T_{P\cup\Sigma_\Gamma}(X))$. ■

Because $2^{\mathcal{B}_{P\cup\Sigma_\Gamma}}$ is described as a complete lattice and the mapping $T_{P\cup\Sigma_\Gamma} : 2^{\mathcal{B}_{P\cup\Sigma_\Gamma}} \longrightarrow 2^{\mathcal{B}_{P\cup\Sigma_\Gamma}}$ is monotonic, we can define the ordinal power of the mapping as follows.

Definition 4 *Let P be a definite constraint planning program and Σ_Γ be the corresponding set of action constraints. The ordinal power of the mapping $T_{P\cup\Sigma_\Gamma}$ is defined as follows.*

$$\begin{array}{rcl}
T_{P\cup\Sigma_\Gamma} \uparrow 0 & = & \bot \\
T_{P\cup\Sigma_\Gamma} \uparrow \alpha & = & T_{P\cup\Sigma_\Gamma}(T_{P\cup\Sigma_\Gamma} \uparrow (\alpha - 1)) \textit{ if } \alpha \textit{ is a successor ordinal} \\
T_{P\cup\Sigma_\Gamma} \uparrow \alpha & = & lub\{T_{P\cup\Sigma_\Gamma} \uparrow \beta : \beta < \alpha\}, \textit{ if } \alpha \textit{ is a limit ordinal} \\
T_{P\cup\Sigma_\Gamma} \downarrow 0 & = & \top \\
T_{P\cup\Sigma_\Gamma} \downarrow \alpha & = & T_{P\cup\Sigma_\Gamma}(T_{P\cup\Sigma_\Gamma} \downarrow (\alpha - 1)) \textit{ if } \alpha \textit{ is a successor ordinal} \\
T_{P\cup\Sigma_\Gamma} \downarrow \alpha & = & glb\{T_{P\cup\Sigma_\Gamma} \downarrow \beta : \beta < \alpha\} \textit{ if } \alpha \textit{ is a limit ordinal.}
\end{array}$$

Now, we give the fixed point theorem for constraint logic planning programs.

Theorem 1 *Let P be a definite planning program and Σ_Γ be the corresponding set of action constraints. Then*

$$\begin{array}{l}
glb\{S \subseteq \mathcal{B}_{P\cup\Sigma_\Gamma} : T_{P\cup\Sigma_\Gamma}(S) \subseteq S\} \\
\quad = \ lfp(T_{P\cup\Sigma_\Gamma}) \\
\quad = \ T_{P\cup\Sigma_\Gamma} \uparrow \omega.
\end{array}$$

Proof: By Proposition 2, $T_{P\cup\Sigma_\Gamma}$ is monotonic. Then $T_{P\cup\Sigma_\Gamma}$ has a least fixed point $lfp(T_{P\cup\Sigma_\Gamma})$[5]. By the fixed point theory[6], $lfp(T_{P\cup\Sigma_\Gamma}) = glb\{S : T_{P\cup\Sigma_\Gamma}(S) \subseteq S\}$. By Proposition 1, $T_{P\cup\Sigma_\Gamma}$ is continuous. Then[7] $T_{P\cup\Sigma_\Gamma} \uparrow \omega = lfp(T_{P\cup\Sigma_\Gamma})$. ■

In logic programming, every definite program has a least Herbrand model. This least Herbrand model is the conjunction of all the Herbrand models of the definite program and it equals to the least fixed point of the mapping. However, in this planning language, the least fixed point of the mapping does not represent a model. There may be two elements in the least fixed point of the mapping such that the conjunction of the constraints is not consistent. For example, $Hold(On(A,B),T_1,T_2)|Exec(Puton(A,T,B),T_1,T_1) \wedge T_0 \prec T_1$ and $Hold(On(B,A),T_1,T_2)|Exec(Puton(B,T,A),T_1,T_1) \wedge T_0 \prec T_1$ may be two elements of the least fixed point of some definite planning program in the blocks world. The conjunction of the two constraints is not consistent according to the corresponding set of action constraints.

In logic programming, the least fixed point of the mapping is precisely the set of all the ground atoms which are the logical consequences of a definite program. In a similar way, in our planning language, the least fixed point of the mapping of a planning program can be regarded as the set of all the property-constraint pairs in $\mathcal{B}_{P\cup\Sigma_\Gamma}$ which are logical consequences of the planning program and the corresponding set of action constraints.

In this planning language, neither a property nor a plan constraint can be a logical consequence of some definite planning program and the corresponding action constraints. For example, suppose that P is a planning program that contains the initial conditions that blocks A and B are clear and are on the table at the initial time point. The property $Hold(On(A,B),T_1,T_2)$ is not a logical consequence of the corresponding planning program and neither is $Hold(On(B,A),T_1,T_2)$. The properties may be made true by some nonlinear plans which are constructed from the planning program and may be made false by some other plans of the planning program. Also a plan constraint such as $Exec(Puton(A,B),T_1,T_1)$ is not a logical consequence of the corresponding planning program. It may be part of a plan constructed out of the planning program and it may be inconsistent with other plans according to the corresponding set of action constraints.

The logical consequences of a planning program and the set of action constraints should be defined as property-constraint pairs. Let $A|C$ be a property constraint pair. If the nonlinear plan described by C achieves the property A according to P, then $A|C$ is a logical consequence of $P \cup \Sigma_\Gamma$.

Definition 5 *A property-constraint pair $Hold(p,t_i,t_j)|C$ is said to be a logical consequence of a planning program P and the corresponding set of action constraints Σ_Γ if*

1. $(\Sigma_\Gamma \cup T_{<_T} \cup T_{=_T}) \models \exists C$ *and*
2. *for all temporal models M of $P \cup \Sigma_\Gamma$,* $M \models \forall(\exists t.(C \wedge t = t_j \Longrightarrow Hold(p,t_i,t_j)))$.

In the above example, the property-constraint pairs
$Hold(On(A,B),T_1,T_2)|Exec(Puton(A,T,B),T_1,T_1) \wedge T_0 \prec T_1$ and
$Hold(On(B,C),T_1,T_2)|Exec(Puton(B,T,A),T_1,T_1) \wedge T_0 \prec T_1$
are both logical consequences of the planning program and the set of action constraints.

[5]Refer to Proposition 5.1 in (Lloyd, 1987).
[6]Refer to Proposition 5.1 in (Lloyd, 1987).
[7]Refer to Proposition 5.4 in (Lloyd, 1987).

With this definition, we can give the theorems about the declarative semantics of planning programs. The following theorem states that every element in $T_{P\cup\Sigma_\Gamma} \uparrow \omega$ is a logical consequence of $P \cup \Sigma_\Gamma$.

Theorem 2 *Let P be a definite planning program and Σ_Γ be the corresponding set of action constraints. If $A|C \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$, then $A|C$ is a logical consequence of $P \cup \Sigma_\Gamma$.*

Proof: By induction on the number of actions in C. First, suppose that there is only one pseudo action at the initial time point in C. Then, according to the definition of the mapping, $A|C = Hold(p, t_1, t_2)| \bigwedge_{i=1}^{2}(t_i = t_i') \in T_{P\cup\Sigma_\Gamma} \uparrow 1$ and there is an data instance of a unit planning rule $Hold(p, t_1', t_2') : -$. Then, $A|C$ is a logical consequence of $P \cup \Sigma_\Gamma$.

Suppose the result holds if there are $n-1$ actions in C. Let C be a constraint which contains n actions. $A|C = Hold(p, t_1, t_2)|C \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$ implies that there is an data instance of a planning rule of the form

$$Hold(p, t_1', t_2') : -Hold(q_1, t_3', t_4'), \ldots, Hold(q_m, t_{2m+1}', t_{2m+2}')|C_0$$

such that

1. $(Hold(q_i, t_{2i+1}, t_{2i+2})|C_i) \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$, $i = 1, \ldots, m$, and
2. $C = \bigwedge_{i=0}^{m} C_i \wedge \bigwedge_{j=1}^{2m+2}(t_j = t_j')$ and $\Sigma_\Gamma \cup T_{<_T} \cup T_{=_T} \models \exists C$.

By induction hypothesis, C implies $Hold(q_1, t_3, t_4) \wedge \ldots \wedge Hold(q_m, t_{2m+1}, t_{2m+2})$. C also implies $Hold(q_1, t_3', t_4') \wedge \ldots \wedge Hold(q_m, t_{2m+1}', t_{2m+2}')$ for it contains $\bigwedge_{i=3}^{2m+2}(t_i = t_i')$. From the data instance of the planning rule and the fact that C contains $\bigwedge_{i=1}^{2}(t_i = t_i')$, C implies $Hold(p, t_1, t_2)$. ∎

Unlike in logic programming, not every the property-constraint pair $A|C \in \mathcal{B}_{P\cup\Sigma_\Gamma}$ which is a logical consequence of $P \cup \Sigma_\Gamma$ is in the set $T_{P\cup\Sigma_\Gamma} \uparrow \omega$. For example, in the blocks world, the property constraint pair

$$\begin{aligned} Hold(On(A, B), T_1, T_2)|Exec(Puton(A, T, B), T_1, T_1) \wedge \\ T_0 \prec T_1 \wedge Exec(Puton(C, T, D), T_3, T_3) \wedge T_0 \prec T_3 \end{aligned}$$

may be a logical consequence of some planning programs. However, it may not be an element of the least fixed point for the planning program according to the definition of the mapping.

A property constraint pair is in the least fixed point of a planning program only if the constraint is *necessary* for the property according to the planning program. If a property constraint pair is a logical consequence of a planning program, part of the constraint must be necessary for the property. In the above example, part of the constraint $Exec(Puton(A, B), T_1, T_1) \wedge T_0 \prec T_1$ is necessary for the property $Hold(On(A, B), T_1, T_2)$.

Definition 6 *A nonlinear plan is said to be* necessary *for a property if the following conditions hold.*

1. *The property is asserted by an action A in the plan and all the other actions in the plan precede A directly or indirectly.*

2. *For any two actions A_1 and A_2 in the plan, if A_1 directly precedes A_2, then A_1 asserts at least one of the preconditions of A_2.*

That a plan is necessary for a property does not imply that the plan is the most efficient for the property. For example, suppose that a_1, a_2, a_1 is a conflict free linear plan. Action a_1 deletes property q and asserts p. Action a_2 deletes p and asserts q. Then the linear plan is necessary for p according to the above definition, but not the most efficient for p.

A plan constraint C is necessary for a property if the nonlinear plan described by C is necessary for the property.

Proposition 3 *Let P be a definite planning program and Σ_Γ be the corresponding set of action constraints. Suppose that $\mathcal{P}$ is a conflict free nonlinear plan of $P \cup \Sigma_\Gamma$ and $\mathcal{P}$ is necessary for a ground property $A = Hold(p, t_1, t_2)$. Then there is a plan constraint C such that $A|C \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$ and C describes $\mathcal{P}$.*

Proof By induction on the number of actions in $\mathcal{P}$. First, suppose that there is only one pseudo action at the initial point in the nonlinear plan. Then $Hold(p, t_1, t_2)$ must be an instance of an unit planning rule of the form $Hold(p, t'_1, t'_2) :-$. Let $C = \bigwedge_{i=1}^{2}(t_i = t'_i)$. Obviously, C describes the zero action plan. According to the definition of the mapping and the monotonic property of the mapping, $A|C \in T_{P\cup\Sigma_\Gamma} \uparrow 1 \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$.

Suppose the result holds if there are $n-1$ actions. We prove it holds when there are n actions in $\mathcal{P}$. Because $\mathcal{P}$ is necessary for $Hold(p, t_1, t_2)$, p must be asserted by an action a at t_1 in $\mathcal{P}$. Suppose that $q_1, \ldots, q_m$ are all the preconditions of a. Then $\mathcal{P}$ consists of necessary subplans for $q_1, \ldots, q_m$ and the action a for $\mathcal{P}$ is conflict free and necessary for p. See Fig. 4. By the induction hypothesis, there exist plan constraints $C_1, \ldots, C_m$ such that

$$Hold(q_1, t_3, t_4)|C_1, \ldots, Hold(q_m, t_{2m+1}, t_{2m+2})|C_m \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$$

and C_i describe the necessary subplan for q_i, $i = 1, \ldots, m$.

Corresponding to the action a and the property p, there is a data instance of a planning rule in P: $Hold(p, t'_1, t'_2) : -Hold(q_1, t'_3, t'_4), \ldots, Hold(q_m, t'_{2m+1}, t'_{2m+2})|C_0$ where C_0 consists of the action $Exec(a, t'_1, t'_1)$ and the temporal constraint between the action and its preconditions. We can construct a plan constraint $C = \bigwedge_{i=0}^{m} C_i \wedge \bigwedge_{j=1}^{2m+2}(t_j = t'_j)$. It is straightforward that $Hold(p, t_1, t_2)|C \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$ and C describes $\mathcal{P}$. ∎

For any conflict free nonlinear plan which achieves a property, it must contains an action which asserts the property. From that action, a subplan which is necessary for the property can be obtained. Therefore, Proposition 3 can be extended as follows.

Theorem 3 *Let P be a definite planning program and Σ_Γ be the corresponding set of action constraints. Suppose that $\mathcal{P}$ is a conflict free nonlinear plan of $P \cup \Sigma_\Gamma$ which achieves a ground property $A = Hold(p, t_1, t_2)$. Then there exist plan constraints C and C_0 such that $Hold(p, t_1, t_2)|C \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$ and $C \wedge C_0$ is consistent and describes $\mathcal{P}$.*

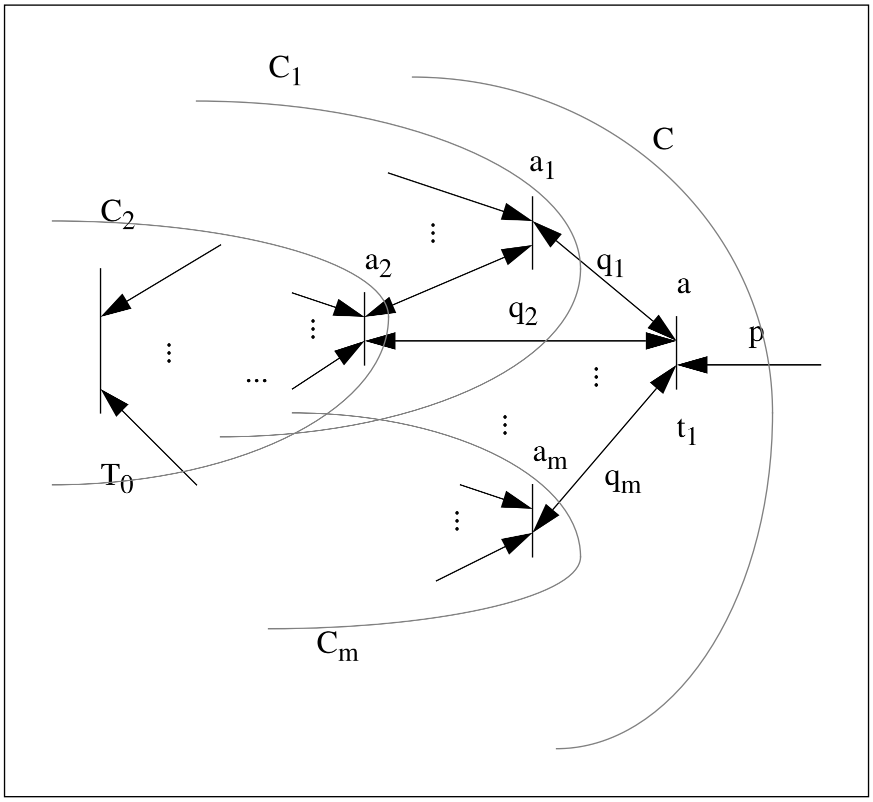

Figure 4: The nonlinear plan which is necessary for $Hold(p, t_1, t_2)$.

Proof Because $\mathcal{P}$ is a conflict free plan which achieves A, it contains a subplan which is conflict free and is necessary for A. By Proposition 3, the result is proved. ■

Theorem 4 *Let P be a definite planning program and Σ_Γ be the corresponding set of action constraints. Suppose that $\mathcal{P}$ is a conflict free nonlinear plan of $P \cup \Sigma_\Gamma$ which achieves the ground properties $A_1, \ldots, A_n$. Then there exist constraints $C_0, C_1, \ldots, C_n$ such that $A_1|C_1, \ldots, A_n|C_n \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$ and $C = \bigwedge_{i=0}^{n} C_i$ is consistent and describes $\mathcal{P}$.*

The proof of this is straightforward based on Theorem 3.

Theorem. 3 and Theorem. 4 cannot be strengthened further such that, for any property constraint pair $A|C \in \mathcal{B}_{P\cup\Sigma_\Gamma}$ that is a logical consequence of $P \cup \Sigma_\Gamma$, C contains a subconstraint C' such that $A|C' \in T_{P\cup\Sigma_\Gamma} \uparrow \omega$. For any plan, the plan constraints which describe it may be in different forms (for example, differently naming for the corresponding time points). However, these theorems are strong enough for soundness and completeness of our constraint logic planning, for, in planning, we are only interested in the plans for the given problems, not the forms of constraints that represent the plans.

5 Operational Semantics of Planning Program

This section defines an operational model for planning programs. We extend the SLD resolution in logic programming (Apt & van Emden, 1982; Lloyd, 1987) and the extended

SLD resolution in constraint logic programming (Jaffar & Lassez, 1987b; Sakai & Aiba, 1990) to ESLD resolution in our constraint logic planning. The theoretical results in this section establish the soundness and completeness of logic planning in this framework.

Definition 7 *Let G be a property constraint pair $A_1, \ldots, A_m | C$ and R be a planning rule of the form:*

$$Hold(p', t'_i, t'_j) :- B_1, \ldots, B_n | C_0.$$

A new property constraint pair G' is said to be derived from G and R using the data mgu θ by ESLD resolution if the following conditions hold.

1. *A_k, $1 \leq k \leq m$, called the selected property, is an atom property in G of the form $Hold(p, t_i, t_j)$;*
2. *θ is a data mgu for p and p';*
3. *$C' = (C \wedge C_0 \wedge (t_i = t'_i) \wedge (t_j = t'_j))\theta$ is consistent with $\Sigma_\Gamma \cup T_{<_T} \cup T_{=_T}$;*
4. *$G' = (A_1, \ldots, A_{k-1}, B_1, \ldots, B_n, A_{k+1}, \ldots, A_m)\theta | C'$.*

G' is called a resolvent of G and R.

Definition 8 *Let P be a definite planning program and G be a property constraint pair. An ESLD derivation of P and G consists of a finite (or infinite) sequence of property constraint pairs $G_0 = G, G_1, \ldots$, a sequence of variants of planning rules of P $R_1, R_2, \ldots$ and a sequence of data mgu's $\theta_1, \theta_2, \ldots$ such that each G_{i+1} is derived from G_i and R_{i+1} using θ_{i+1} by ESLD resolution.*

Definition 9 *Given a definite planning program P and property constraint pair $G_0 = A_1, \ldots, A_n | C_0$. The properties constraint pair*

$$G = (A_1, \ldots, A_n | C_0 \wedge C_1 \wedge \ldots \wedge C_n)\theta_1, \ldots, \theta_n$$

is called a successful ESLD derivation of P and G_0 *if there exists an ESLD derivation of P and G_0 of length n such that $G_n = \_|(C_0 \wedge C_1 \wedge \ldots \wedge C_n)\theta_1, \ldots, \theta_n$.*

As with a SLD derivation in logic programming, an ESLD derivation from a planning program P and a property constraint pair G_0 may be finite or infinite. A finite ESLD may be a success or a failure. A successful ESLD derivation is one which terminates with a property constraint pair with empty properties. A failed ESLD derivation is one which terminates with a property constraint pair that is not resolvable with any planning rule in P and has non-empty properties.

The following theorem gives the first soundness result for ESLD resolution.

Theorem 5 *Let P be a definite planning program, $G_0 = A_1, \ldots, A_m | C_0$ be a property constraint pair, $G = (A_1, \ldots, A_m | \bigwedge_{l=0}^{n} C_l)\theta_1, \ldots, \theta_n$ be a successful ESLD derivation of P and G_0 of length n and $\theta_1, \ldots, \theta_n$ be the sequence of data mgu's of the derivation. Then $((A_1, \ldots, A_m | \bigwedge_{l=0}^{n} C_l)\theta_1, \ldots, \theta_n)$ is a logical consequence of $P \cup \Sigma_D$.*

Proof We prove the result by induction on the length of the derivation. First, suppose that the length of the derivation equals one. Then G_0 is of the form $Hold(p, t_i, t_j)|C_0$, R_1 is a unit planning rule $Hold(p', t'_i, t'_j) : -|C'_1$, θ is the data mgu of p and p' and $G = (Hold(p, t_i, t_j)|C_0 \wedge C'_1 \wedge (t_i = t'_i) \wedge (t_j = t'_j))\theta$. Since $(Hold(p', t'_i, t'_j)|C'_1)\theta$ corresponds to a data instance of a unit planning rule in P and C is consistent, $(Hold(p', t'_i, t'_j)|C_0 \wedge C'_1 \wedge (t_i = t'_i) \wedge (t_j = t'_j))\theta$ is a logical consequence of $P \cup \Sigma_\Gamma$ and so is $(Hold(p, t_i, t_j)|C_0 \wedge C'_1 \wedge (t_i = t'_i) \wedge (t_j = t'_j))\theta$. Next, suppose that the result holds when the length of the derivation is $n-1$. For a derivation of length n, let $\theta_1, \ldots, \theta_n$ be the sequence of data mgu's used in the derivation, $A_k = Hold(p, t_i, t_j)$ be the first selected property in the derivation and $Hold(p', t'_i, t'_j) : -B_1, \ldots B_s|C'_k$ be the corresponding planning rule. By the induction hypothesis,

$$((A_1, \ldots, A_{k-1}, B_1, \ldots, B_s, A_{k+1}, \ldots, A_m | \bigwedge_{l=0}^{n} C_l)\theta_1, \ldots, \theta_n)$$

is a logical consequence of $P \cup \Sigma_D$. Thus, if $s > 0$, $(B_1, \ldots B_s| \bigwedge_{l=0}^{n} C_l)\theta_1, \ldots, \theta_n$ is a logical consequence of $P \cup \Sigma_D$. Consequently, $(Hold(p', t'_i, t'_j)| \bigwedge_{l=0}^{n} C_l)\theta_1, \ldots, \theta_n$ is a logical consequence of $P \cup \Sigma_D$ and so is $(A_k| \bigwedge_{l=0}^{n} C_l)\theta_1, \ldots, \theta_n$ for $C_k = C'_k \wedge (t_i = t'_i) \wedge (t_j = t'_j)$. As a result, $(A_1, \ldots, A_n| \bigwedge_{l=0}^{n} C_l)\theta_1, \ldots, \theta_n$ is a logical consequence of $P \cup \Sigma_D$. ∎

Analogous to the corresponding result in logic programming given by Apt and Emden (82), we can prove that if a ground property constraint pair $A|C \in \mathcal{B}_{P \cup \Sigma_\Gamma}$ and $A|C$ is a successful derivation of P and $A|C_0$ of length n, then $A|C \in T_{P \cup \Sigma_\Gamma} \uparrow n$. In the following discussion, $[A|C]$ is used to denote the set $\{A'|C' \in \mathcal{B}_{P \cup \Sigma_\Gamma} : A'|C' = (A|C)\theta$ for some substitution θ and $C\theta$ is consistent$\}$.

Theorem 6 *Let P be a definite planning program, $(A_1, \ldots, A_m|C)\theta_1, \ldots, \theta_n$ be a successful derivation from P and a property constraint pair G_0 of length n where $\theta_1, \ldots, \theta_n$ are the mgu's of the derivation. Then C contains subconstraints $C_1, \ldots, C_m$ such that*

$$[(A_1|C_1)\theta_1, \ldots, \theta_n], \ldots, [(A_m|C_m)\theta_1, \ldots, \theta_n] \subseteq T_{P \cup \Sigma_\Gamma} \uparrow n$$

.

Proof By induction on the length of derivation. First, suppose that the length is one. Then, there is a unit planning rule of the form $Hold(p', t'_i, t'_j) : -|C'_1$ and $G = (Hold(p, t_i, t_j)|C_0 \wedge C'_1 \wedge (t_i = t'_i) \wedge (t_j = t'_j))\theta$ where θ is a mgu of p and p'. $(Hold(p', t'_i, t'_j)\theta : -C'_1\theta)$ is a data instance of the unit planning rule. According to the definition of the mapping (Definition 3 in page 12),

$$[(Hold(p, t_i, t_j)|C'_1 \wedge (t_i = t'_i) \wedge (t_j = t'_j))\theta] \in T_{P \cup \Sigma_\Gamma} \uparrow 1.$$

Suppose the result holds true for the derivation of length $n-1$. Let $\theta_1, \ldots, \theta_n$ be the sequence of data mgu's used in the derivation of length n. For any atomic property A_k in G_0, if A_k is not the first selected property of G_0, then $A_k\theta_1$ is a property of G_1, the second properties constraint pair of the derivation. By the induction hypothesis, C contains a subconstraint C_k such that $[(A_k|C_k)\theta_1, \ldots, \theta_n] \subseteq T_{P \cup \Sigma_\Gamma} \uparrow (n-1)$.

By the monotonic property of $T_{P\cup\Sigma_\Gamma}$, $[(A_k|C_k)\theta_1,\ldots,\theta_n] \subseteq T_{P\cup\Sigma_\Gamma}\uparrow n$.
Otherwise, $A_k = Hold(p,t_i,t_j)$ is the first selected property of G_0. Suppose that the corresponding planning rule R is $Hold(p',t'_i,t'_j) - : B_1,\ldots,B_l|C'_k$ and θ_1 is the data mgu for p and p'. If $l=0$, then
$[(A_k|C'_k \wedge (t_i = t'_i) \wedge (t_j = t'_j))\theta_1,\ldots,\theta_n] \subseteq [(A_k|C'_k \wedge (t_i = t'_i) \wedge (t_j = t'_j))\theta_1] \subseteq$
$T_{P\cup\Sigma_\Gamma}\uparrow 1 \subseteq T_{P\cup\Sigma_\Gamma}\uparrow n$. If $l>0$, by induction hypothesis, C contains $C_{l_1},\ldots,C_{l_l}$ and

$$[(B_1|C_{l_1})\theta_1,\ldots\theta_n],\ldots,[(B_l|C_{l_l})\theta_1,\ldots\theta_n] \subseteq T_{P\cup\Sigma_\Gamma}\uparrow(n-1).$$

By the definition of the mapping,

$$[(A_k|C'_k \wedge (t_i = t'_i) \wedge (t_j = t'_j) \wedge \bigwedge_{s=1}^{l} C_{l_s})\theta_1,\ldots,\theta_n] \subseteq T_{P\cup\Sigma_\Gamma}\uparrow n$$

and by the definition of ESLD derivation, C contains $C'_k \wedge (t_i = t'_i) \wedge (t_j = t'_j) \wedge \bigwedge_{s=1}^{l} C_{l_s}$. ■

The next theorem is the first completeness result for ESLD resolution.

Theorem 7 *For any plan constraint C_0 and any property constraint pair $A|C \in T_{P\cup\Sigma_\Gamma}\uparrow n$, if $C \wedge C_0$ is consistent, then $A|C \wedge C_0$ is a successful derivation of P and $A|C_0$.*

Proof By induction on the ordinal power. $A|C = Hold(p,t_1,t_2)|C \in T_{P\cup\Sigma_\Gamma}\uparrow 1$ implies that there is a data substitution θ and a unit planning rule $Hold(p',t'_1,t'_2) : -|C'$ such that $p = p'\theta$ and $C = C'\theta \wedge (t_1 = t'_1) \wedge (t_2 = t'_2)$. Then $A|C_0 \wedge C$ is a success derivation of P and $A|C_0$. Suppose the result holds for $n-1$. Now, let $A|C \in T_{P\cup\Sigma_\Gamma}\uparrow n$. By the definition of $T_{P\cup\Sigma_\Gamma}$, there exists an instance of a planning rule $Hold(p',t'_1,t'_2) : -Hold(q'_1,t'_3,t'_4),\ldots,Hold(q'_m,t'_{2m+1},t'_{2m+2})|C'$ such that

1. $p = p'\theta$,
2. $(Hold(q'_i,t_{2i+1},t_{2i+2})\theta|C_i) \in T_{P\cup\Sigma_\Gamma}\uparrow(n-1)$, $i=1,\ldots,m$,
3. $C = (C'\theta \wedge \bigwedge_{i=1}^{m} C_i) \wedge \bigwedge_{j=1}^{2m+2}(t_j = t'_j)$ is consistent

for some data substitution θ. By the induction hypothesis, $Hold(q'_i,t_{2i+1},t_{2i+2})\theta|C_i \wedge C_0$, $i=1,\ldots,m$, are successful ESLD derivations. Because $Hold(q'_i,t_{2i+1},t_{2i+2})\theta$ is ground and $C = (C'\theta \wedge \bigwedge_{i=1}^{m} C_i) \wedge \bigwedge_{j=1}^{2m+2}(t_j = t'_j)$ is consistent, these success derivations can be combined into one unrestricted success derivation[8]. Thus $A|C \wedge C_0\theta_1,\ldots,\theta_n$ is an unrestricted success derivation for some sequence of data substitutions $\theta_1,\ldots,\theta_n$.

The Mgu Lemma in logic programming given in [Lloyd 84] (Lemma 8.1) can be directly extended for our ESLD derivation. By this lemma, if $A|C \wedge C_0\theta_1,\ldots,\theta_n$ is an unrestricted success derivation, then there exist a sequence of data mgus $\beta_1,\ldots,\beta_n$ such that $A|C \wedge C_0\beta_1,\ldots,\beta_n$ is a success derivation.

[8]An unrestricted ESLD derivation is an ESLD derivation except that the data substitution θ is not required to be the most general unifier.
If a nonlinear plan does not satisfy Σ_Γ, all its possible expansions will not satisfy Σ_Γ (Zhang & Barringer, 1994b). Therefore, the fact that C is consistent implies that any subconstraint of C is consistent. This implies that the success derivations can be combined in an arbitrary order into one unrestricted success derivation.

Because $A|C \in T_{P \cup \Sigma_\Gamma} \uparrow n$ is data ground, by properly naming the data variables in C_0, the sequence of data mgu's has no effect on $A|C_0 \wedge C$. Therefore $A|C_0 \wedge C$ is a successful ESLD derivation. ■

Theorem 7 concerns only the goals that contain one property. This theorem can be naturally extended to the one which can deal with the goals that contain more than one property.

Theorem 8 *Let $G_0 = A_1, \ldots, A_m|C_0$ be a property constraint pair where $A_1, \ldots, A_n$ are ground. If there exists a conflict free nonlinear plan $\mathcal{P}$ which satisfies C_0 and achieves $A_1, \ldots, A_n$, then there exists a successful derivation*

$$G = (A_1, \ldots, A_m | \bigwedge_{i=0}^{n} C_i)\theta_1, \ldots, \theta_n$$

out of P and G_0 such that $C = (\bigwedge_{i=0}^{n} C_i)\theta_1, \ldots, \theta_n$ describes $\mathcal{P}$.

Proof Because $\mathcal{P}$ satisfies C_0 and achieves $A_1, \ldots, A_m$, by Theorem 4 (in page 17), there exist constraints $C'_0, C'_1, \ldots, C'_m$ such that $A_1|C'_1, \ldots, A_m|C'_m \in T_{P \cup \Sigma_\Gamma} \uparrow \omega$ and $C' = \bigwedge_{i=0}^{m} C'_i$ describes $\mathcal{P}$. By Theorem 7, $A_i|C''_0 \wedge C'_i$ are success derivations of P and $A_i|C''_0$ for any C''_0 such that $C''_0 \wedge C_i$, $i = 1, \ldots, m$ is consistent. Let C''_0 consists of C_0 and C'_0. Because $A'_i s$ are ground and C' is consistent, these derivations can be combined into a success derivation $G = (A_1, \ldots, A_m | \bigwedge_{i=0}^{n} C_i)\theta_1, \ldots, \theta_n$ such that $(\bigwedge_{i=0}^{n} C_i)\theta_1, \ldots, \theta_n = C''_0 \wedge \bigwedge_{i=1}^{m} C'_i$ describes $\mathcal{P}$. ■

This completeness result can be extended to property constraint pairs which contain variables.

Theorem 9 *Let $G_0 = A_1, \ldots, A_m|C_0$ be a property constraint pair. If there exists a conflict free nonlinear plan $\mathcal{P}$ which satisfies C_0 and achieves $A_1, \ldots, A_n$, then there exists a successful derivation $G = (A_1, \ldots, A_m | \bigwedge_{i=0}^{n} C_i)\theta_1, \ldots, \theta_n$ from P and G_0 such that $C = (\bigwedge_{i=0}^{n} C_i)\theta_1, \ldots, \theta_n$ describes $\mathcal{P}$.*

Proof Suppose that $v_1, \ldots, v_l$ are all the variables in the atomic properties $A_1, \ldots, A_m$ (both temporal and data). Let $d_1, \ldots, d_l$ be distinct constants of the corresponding types not appearing in $P \cup \Sigma_\Gamma$ and G_0 and let θ be the substitution $\{v_1/d_1, \ldots, v_l/d_l\}$. Then, by Theorem 8, there is a success ESLD derivation from P and $G_0\theta$ which gives the description of $\mathcal{P}$, because $G_0\theta$ is ground. Since d_i does not appear in $P \cup \Sigma_\Gamma$ and G_0, by replacing d_i by v_i in the derivation, G_0 has a success ESLD derivation. ■

This completeness theorem is strong enough to guarantee that if there exists a conflict free nonlinear plan for a given planning problem, then there exists a success derivation such that the plan constraint of the derivation describes the plan.

6 Justification of Constraint Logic Planning

To guarantee the soundness and completeness of ESLD resolution, the theory $\mathcal{T}$ is required to be *satisfaction-complete*, i.e. for any plan constraint C, the problem whether

C is consistent with $\mathcal{T}$ should be decidable. The central part of $\mathcal{T}$ is a set of action constraints Σ_Γ

An action constraint has been defined to be of the form

$$\forall t_1, t_2.(Exec(a_1, t_1, t_1) \wedge Exec(a_2, t_2, t_2) \Longrightarrow t_1 \neq t_2)$$

Then, to check if a plan constraint satisfy an action constraint, we simply check if the two actions (or corresponding change points for interval actions) of the action constraint are at the same time point in the plan constraint. As the result, to test whether a finite plan constraint C is or is not consistent with Σ_Γ is a simple problem and is decidable. Also the problem whether a finite plan constraint C is consistent with $\mathcal{T} = \Sigma_\Gamma \cup T_{<_T} \cup T_{=_T}$ is decidable. The proof of the following theorem is routine.

Theorem 10 *For finite plan constraints, the theory $\mathcal{T} = \Sigma_\Gamma \cup T_{<_T} \cup T_{=_T}$ is* satisfaction-complete.

In any planning process, finite or infinite, constraint consistency checking is always carried upon finite plan constraints. From the results of the last two sections, if a breadth first search strategy is adopted, planning in this framework is sound and complete.

Theorem 11 *Suppose that a breadth first search strategy is adopted. Given a planning problem, under the closed world assumption, if a conflict free nonlinear plan exists for the problem, then the ESLD derivation will terminate successfully within a limited time and return a plan constraint which describes a conflict free nonlinear plan for the problem; if the ESLD derivation stops and fails, or it does not halt, then there is no conflict free plan for the given problem.*

Based on Theorem 5, Theorem 9, Theorem 10 and the soundness and completeness of action constraint maintenance, the proof of this theorem is routine.

7 Least Commitment strategy for Object Selection

In constraint logic planning presented above, actions and temporal relations among actions are constraints and are handled by constraint solving while ordinary data terms are handled by unification. This is in contrast to that of Stefik's MOLGEN in which only object (i.e. data term) selection is handled by constraint solving. In the constraint logic planning, the least commitment strategy is not fully employed for object selection.

However, as we mentioned earlier, the main purpose of handling data terms by unification is to make it easier to define the declarative and operational semantics for constraint logic programs. A deductive planner that can handle object selection by constraint solving is a natural extension of the one given above.

To achieve this extension, the atomic constraints should be expanded to include two kinds of "codesignation" constraint of the forms:

1. $dt_i = dt_j$, and
2. $dt_k \neq dt_l$

where dt_i, dt_j, dt_k and dt_l are data terms. The corresponding theory $\mathcal{T}$ should be expanded to include equality theory $T_{=_D}$ and inequality theory $T_{\neq_D}$ for data terms. i.e.

$$\mathcal{T} = \Sigma_\Gamma \cup T_{=_T} \cup T_{<_T} \cup T_{=_D} \cup T_{\neq_D}$$

Then the interpreter for constraint logic planning programs which handles object selection by constraint solving is shown in Fig. 5.

```
Solve([], C, C);

Solve([Hold(p(dt1, ..,dtn), t1, t2) | Restgoal], Pre_C, New_C):-
   Achieve(Hold(p(dt1, ...,dtn), t1, t2), Pre_C, Temp_C),
   Solve(Restgoal, Temp_C, New_C);

Achieve(Hold(p(dt1, ...,dtn),t1,t2), Pre_C, Temp_C) :-
   Con_Rule(Hold(p(dt1',...,dtn'), t1', t2'), Body, Constraint),
   Consistent(Pre_C ∧ Constraint  ∧ (∧ dti = dti') ∧t1= t1' ∧ t2 = t2'),
   Solve(Body, Pre_C∧Constraint∧ (∧ dti = dti') ∧t1= t1' ∧ t2 = t2', Temp_C);
```

Figure 5: The interpreter which handles object selection by constraint solving.

This interpreter does not employ unification any more. According to this interpreter, one of the current goals of the form $Hold(p(dt_1, \ldots, dt_n), t_1, t_2)$ can be resolved with a planning rule of the form

$$Hold(p(dt'_1, \ldots, dt'_n), t'_1, t'_2) : -B_1, \ldots, B_m | C'$$

if the constraint of the current goals C is consistent with

$$C'' = C' \wedge (\bigwedge_{i=1}^{n} dt_i = dt'_i) \wedge (t_1 = t'_1) \wedge (t_2 = t'_2)$$

and the body $B_1, \ldots, B_m$ can be achieved with the new constraint $C \wedge C''$.

For example, in the definitions of the action $Puton(x, y, z)$ in the blocks world given Fig. 1, the inequalities $x \neq y$, $y \neq z$ and $z \neq x$ should be added as preconditions of the action. Using the interpreter given in Fig. 3, such inequalities are treated as properties. To achieve such a property, the interpreter tries to resolve it with a given rule. For example, resolving a property $x \neq B$ will lead to prematurely binding a value to the variable x. This premature binding may be found to be incorrect at a later stage and therefore may cause unnecessary backtracking.

Using the interpreter defined in Fig. 5, equalities and inequalities of data terms are constraints, rather than properties, and premature commitment of bindings for data variables can be avoided. The choice of a particular value for a data variable can be delayed for as long as possible. For example, when the interpreter meets the inequality $x \neq B$, it adds the inequality as a constraint if it does not cause inconsistency and leaves the value of x undecided. Suppose that in the later planning stages, the constraints $x = z$ and $z \neq A$ are added. Then, if, in the problem domain, there are only three blocks A, B and C, x will be bound to C. In this way, the least commitment strategy is achieved for object selection. The deductive planner described here achieves nonlinearity for both temporal relation decision and object selection.

The definitions of declarative semantics and operational semantics for this extended planner can be obtained directly from the ones given in the previous sections. We will not give any formal definitions for these semantics and reprove the results. For any problem domain, the structure which consists of a finite set of values with equality and inequality relations is trivially *solution-compact.* Also, the corresponding theory of such a structure, i.e. $T_{=_D}$ and $T_{\neq_D}$, is trivially *satisfaction-complete.* That is Jaffar and Lassez's conditions are satisfied for handling (data) object selection by constraint solving. Based on the results in (Jaffar & Lassez, 1987b), all the results achieved in previous sections remain applicable for this extension, i.e. the planner is sound and complete.

8 Discussion

In this paper, we propose a nonlinear deductive planning approach, constraint logic planning, and we prove that constraint logic planning is sound and complete.

In comparison with other deductive planning methods, our constraint logic planning is a nonlinear deductive planning approach. Constraint logic planning employs the least commitment strategy for both temporal relation decision and object selection. Also constraint logic planning deals with the frame problem efficiently in a way similar to that in classical nonlinear planning.

In comparison with classical nonlinear planning, our constraint logic planning is a deductive planning approach, it has all the advantages of deductive planners over classical planners. That is, the abilities and limitations of constraint planning are clearly defined; planning problems can be examined independently from the procedural details of finding a plan; constraint logic planning is more expressive. Also, our constraint logic planning is proved to be sound and complete, and is even more efficient than classical nonlinear planning.

Few classical nonlinears planner can be claimed to be sound. The problem is that, soundness implies conflict freeness, and, in conventional methods, conflict freeness is too expansive for nonlinear planning. TWEAK was claimed to be sound. However, we believe that TWEAK is not sound in considering that soundness should imply conflict freeness. To achieve a new goal using the *Modal Truth Criterion* in TWEAK, the constructed plan may be modified. However, any modification of the existing plan may destroy the previously achieved goals. In TWEAK, no measure is employed to prevent this from happening. In fact, what TWEAK did is to prevent the most recently achieved goal from being destroyed by the actions introduced earlier in the plan. It seems that, to prevent the previously achieved goals from being destroyed, the only way is to check that every property previously achieved is still necessarily true using the Modal Truth Criterion whenever the plan is modified. Obviously, this method is too expensive to be adopted. But, without adopting any approach to protect the previously achieved goals, the plans constructed may contain conflicts; i.e., the soundness can't be guaranteeed.

In classical nonlinear planning, to "necessarily" achieve a goal, a planner needs to introduce actions and the related temporal relations to assert the goal and to take measures to prevent the goal from being possibly denied by other actions in the plan; i.e., to implement various modal truth criteria (Chapman, 1987; Christensen, 1990; Pednault, 1991) to achieve "necessary truth". As in TWEAK, by implementing modal truth criteria, what classical nonlinear planners can achieve is only to prevent the most recently achieved goal from being destroyed by the previously introduced actions. The classical nonlinear planners can not prevent previously achieved goals from being destroied by newly introduced actions or by modification of the existed plan.

Constraint logic planning avoids the computation for modal truth criteria. In our constraint logic planning, what we need to do is to introduce actions and the related temporal relations to achieve goals and to check if the constructed plan satisfies the corresponding set of action constraints. Consistency checking for a nonlinear plan and the corresponding action constraints is, at least, not more computationally complicated than preventing the most recently achieved goal from being destroyed by the previously introduced actions using modal truth criteria (Zhang & Barringer, 1994b). However, constraint logic planning achieves and maintains the necessary truth for all the goals involved in planning processes and, therefore, guarantees conflict freeness. In constraint logic planning, for a given problem, the planning process is needed only once, for the obtained plan, if exists, is guaranteed to be a correct solution for the problem. In this meaning, our constraint logic planning is even more efficient than classical nonlinear planning.

Furthermore our constraint logic planning inherently embeds a domain independent search heuristic. We showed that a nonlinear plan is inconsistent with an action constraint implies that the temporal model of the plan is not consistent with a domain constraint (Zhang, 1994). By backtracking from inconsistencies of a constructed plan with the corresponding set of action constraints, constraint logic planning can prune the search space to avoid some of the branches which will lead to impossible states specified by the domain constraints.

9 Acknowledgements

We would like to thank Behnam Bani-Eqbal, David Colisle, Clare Dixon, Ian Pratt and Rajeev Goré who read an early version of this paper and gave valuable comments.

This paper is a modified version of part of Zhang's PhD thesis (Zhang, 1994). The work was partially supported by the British Council and the State Education Commission of China.

References

Allen, James F. 1990. Formal Models of Planning. *Pages 50–54 of:* Allen, J., Hendler, J., & Tate, A. (eds), *Readings in Planning.* Morgan Kaufmann.

Apt, K. R., & van Emden, M. H. 1982. Contributions to the Theory of Logic Programming. *J. ACM*, **29**(3), 841–862.

Chapman, David. 1987. Planning for Conjunctive Goals. *Artificial Intelligence*, **32**, 333–377.

Christensen, Jens. 1990. *Automatic Abstraction in Planning.* Ph.D. thesis, Stanford University.

Colmerauer, A. 1987. Opening the Prolog-III Universe: A New Generation of Prolog Promises Some Powerful Capabilities . *BYTE*, **12**(9), 177–182.

Colmerauer, A. 1990. An Introduction to Prolog III. *Communications of the ACM*, **33**(7), 69– 90.

Hill, R. 1974. *LUSH-Resolution and Its Completeness.* Tech. rept. DCL Memo 78. Dept. of Artificial Intelligence, Univ. of Edinburgh.

Jaffar, J., & Lassez, J-L. 1987a. Constraint Logic Programming. *Pages 111–119 of: Proceedings of the Fourteenth ACM Symposium of the Principles of Programming Languages.*

Jaffar, J., & Lassez, J-L. 1987b. From Unification to Constraints. *Pages 1 – 18 of: Proceedings of the 6th Conference on Logic Programming.*

Lloyd, J. W. 1987. *Foundations of Logic Programming.* 2nd edn. Springer-Verlag.

Pednault, E. P. D. 1991. Generalizing Nonlinear Planning to Handle Complex Goals and Actions with Context-Dependent Effects. *In: Proceedings 12th International Joint Conference on Artificial Intelligence.*

Sacerdoti, Earl D. 1977. *A Structure for Plans and Behaviour.* Elsevier North-Holland, Inc.

Sakai, K., & Aiba, A. 1990 (Feb). *CAL: A Theoretical Background of Constraint Logic Programming and its Applications.* Tech. rept. TR-537. ICOT.

Stefik, M. J. 1981. Planning with Constraints. *Artificial Intelligence*, **16**, 111–140.

van Emden, M. H., & Kowalski, R. A. 1976. The Semantics of Predicate Logic as a Programming Language. *J. ACM*, Oct., 733–742.

Zhang, Ying. 1994 (May). *Constraint Logic Planning: A Formal Framework for Nonlinear Planning.* Ph.D. thesis, Manchester Univ., Computer Science Dept.

Zhang, Ying, & Barringer, Howard. 1994a (July). *A Temporal Logic For Nonlinear Planning.* Tech. rept. MUCS-94-7-1. Department of Computer Science, University of Manchester, Manchester, M13 9PL.

Zhang, Ying, & Barringer, Howard. 1994b (September). *Action Constraint Maintenance.* Tech. rept. MUCS-94-9-1. Department of Computer Science, University of Manchester, Manchester, M13 9PL.

Zhang, Ying, Barringer, Howard, & Carlisle, David. 1994 (August). *Domain Constraint Maintenance.* Tech. rept. MUCS-94-8-1. Department of Computer Science, University of Manchester, Manchester, M13 9PL.

A An Example

This example shows how a nonlinear plan can be nonlinearly constructed through constraint logic planning. The initial conditions and the goals to be achieved are shown in Fig. 6. The main part of the corresponding planning program P which contains one action schemata $Puton(x, y, z)$[9] is listed as follows.

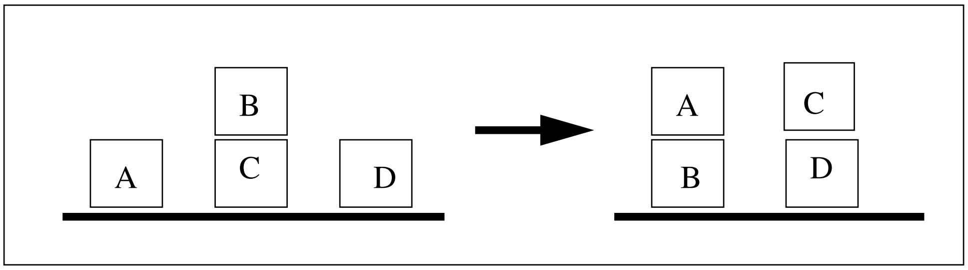

Figure 6: A planning problem.

$$
\begin{aligned}
& Hold(On(x, z), t_1, SKF1(t_1, x, y, z)) :- \\
& \quad Hold(Clear(x), t_2, t_2'), Hold(Clear(z), t_3, t_1), Hold(On(x, y), t_4, t_1) \\
& \quad |Exec(Puton(x, y, z), t_1, t_1) \wedge t_2 \prec t_1 \prec t_2'; \qquad (1) \\
& Hold(Clear(y), t_1, SKF2(t_1, x, y, z)) :- \\
& \quad Hold(Clear(x), t_2, t_2'), Hold(Clear(z), t_3, t_1), Hold(On(x, y), t_4, t_1) \\
& \quad |Exec(Puton(x, y, z), t_1, t_1) \wedge t_2 \prec t_1 \prec t_2'; \qquad (2) \\
& Hold(Clear(A), T_0, SKC3) :- ; \qquad (3) \\
& Hold(Clear(B), T_0, SKC4) :- ; \qquad (4) \\
& Hold(Clear(D), T_0, SKC5) :- ; \qquad (5) \\
& Hold(On(B, C), T_0, SKC6) :- ; \qquad (6)
\end{aligned}
$$

The goals to be achieved are expressed as:

$$:- Hold(On(A, B), t_1, t_1'), Hold(On(C, D), t_2, t_2') | t_1 \prec t \prec t_1' \wedge t_2 \prec t \prec t_2' \qquad (7)$$

The planning process can be briefly described as follows (also see Fig. 7).

[9] In this definition, if y is bound to table T, then $Clear(y)$ is not an asserted property; if z is bound to table T, then $Clear(z)$ is not a precondition for the action.

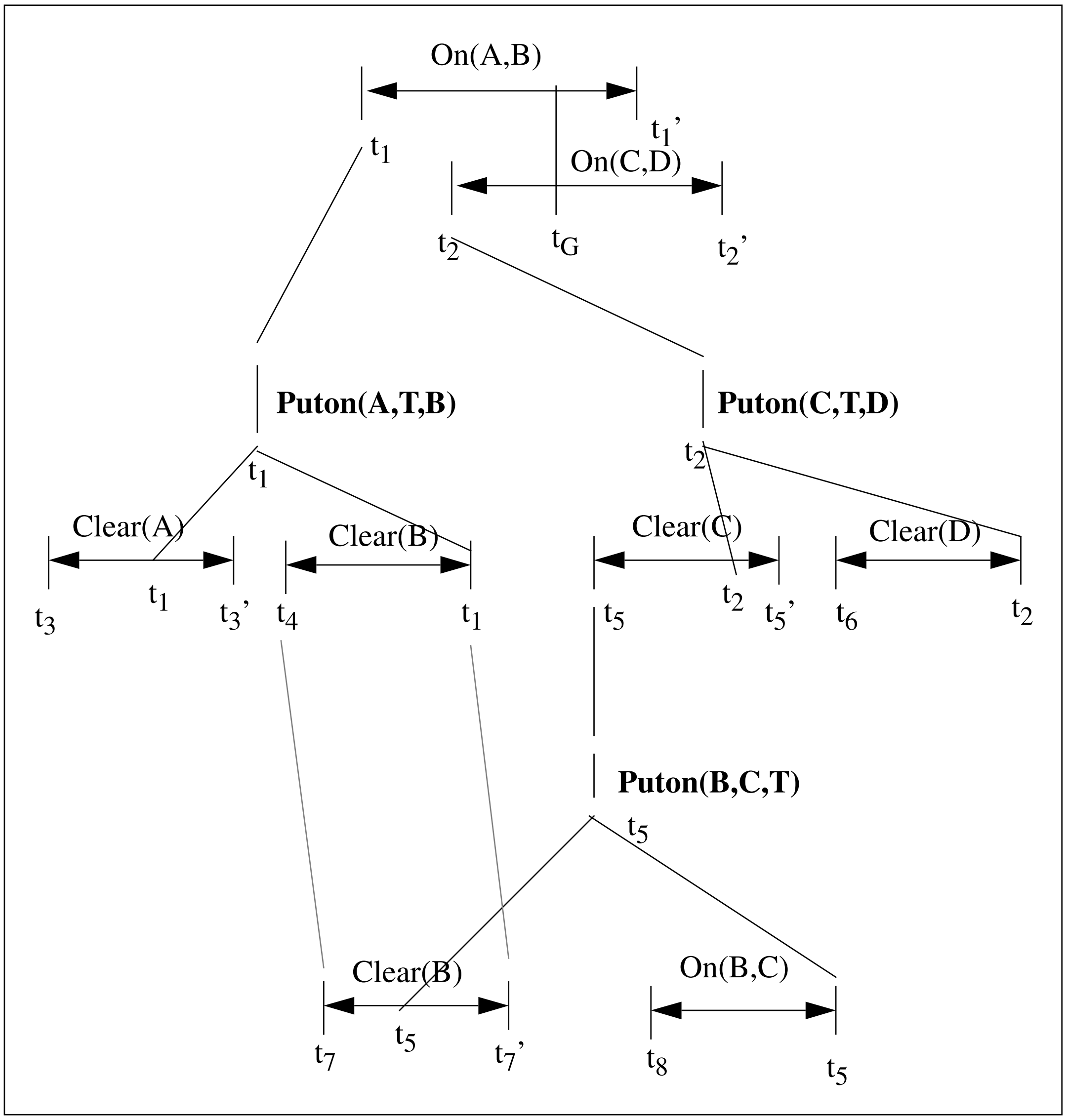

Figure 7: An ideal nonlinear planning process for the given problem.

To achieve the two goals, resolve them with the planning rule (6.1). Then two actions and the corresponding temporal relations are added to the current plan constraint.

$$Ex(Puton(A,T,B),t_1,t_1) \wedge Ex(Puton(C,T,D),t_2,t_2) \wedge \\ t_3 \prec t_1 \prec t'_3 \wedge t_5 \prec t_2 \prec t'_5 \quad (8)$$

At this stage, there is no direct relation between the two actions.

The preconditions of the two actions, then, are introduced as new goals to be achieved.

$$Hold(Clear(A),t_3,t'_3), Hold(Clear(B),t_4,t_1) \\ Hold(Clear(D),t_6,t_2), Hold(Clear(C),t_5,t'_5) \quad (9)$$

All these new goals can be resolved with the corresponding initial conditions (e.g. (6.3), (6.4), (6.5)) except $Hold(Clear(C),t_5,t'_5)$.

We try to resolve $Hold(Clear(C), t_5, t'_5)$ with a planning rule. At first, we assume that the proper planning rule is selected (a variant of (6.2)) to achieve the goal. Then the action $Exec(Puton(B, C, T), t_5, t_5)$ and the corresponding temporal relation are added to the current plan constraint.

$$Exec(Puton(B, C, T), t_5, t_5) \wedge t_7 \prec t_5 \prec t'_7 \tag{10}$$

The preconditions of this action, $Hold(Clear(B), t_7, t'_7)$ and $Hold(On(B, C), t_8, t_5)$ can be resolved with the initial conditions ((6.4), (6.6)).

Because $t_5 \prec t_2$ has been introduced (see (6.8)), this temporal constraint restricts that the action $Puton(B, C, T)$ be before the action $Puton(C, T, D)$.

Because both the preconditions $Clear(B)$ of $Puton(B, C, T)$ and $Clear(B)$ of $Puton(A, T, B)$ are resolved with the initial condition (6.4), the Skolem constant $SKC4$ binds time point t_1, at which $Puton(A, T, B)$ is executed, with time point t'_7, which is restricted to be after t_5 by the temporal constraint introduced in (6.10). Therefore, $t_5 \prec t_1$ and the action $Puton(B, C, T)$ is restricted to be before the action $Puton(A, T, B)$. The final nonlinear plan constructed for this problem can be expressed as follows and it is shown in Fig. 8.

$$\begin{aligned}
&Exec(Puton(B, C, T), t_5, t_5) \wedge T_0 \prec t_5 \\
&\quad Exec(Puton(A, T, B), t_1, t_1) \wedge t_5 \prec t_1 \prec t \\
&\quad Exec(Puton(C, T, D), t_2, t_2) \wedge t_5 \prec t_2 \prec t
\end{aligned} \tag{11}$$

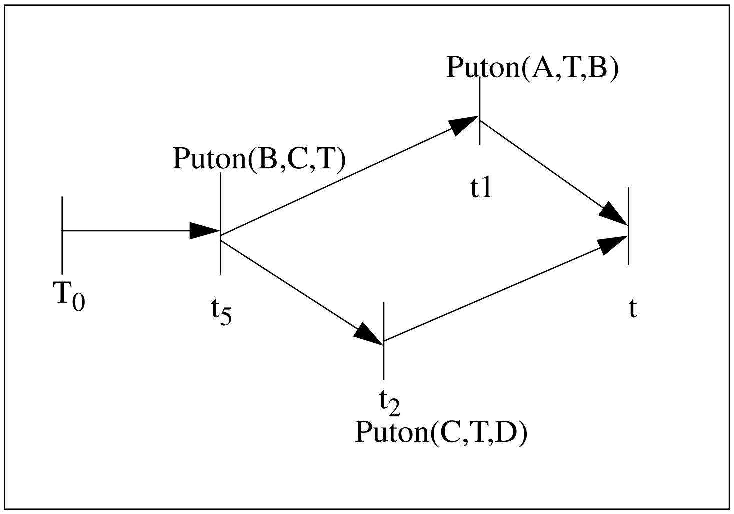

Figure 8: The nonlinear plan for the problem.

In the above planning process, suppose that, to achieve the goal $Clear(C)$, an improper action $Puton(B, C, A)$ is selected. Then a conflict free nonlinear plan can still be obtained. This time, instead of the action $Puton(B, C, T)$ and the corresponding temporal relation in (6.10), the action $Puton(B, C, A)$ and the related temporal relation are added to the current plan constraint.

$$Exec(Puton(B, C, A), t_5, t_5) \wedge t_8 \prec t_5 \prec t'_8 \tag{12}$$

See Fig. 9. The preconditions of this action are introduced as new goals:

$$Hold(Clear(A), t_1, t_5), Hold(Clear(B), t_8, t'_8), Hold(On(B, C), t_9, t_5) \quad (13)$$

All these preconditions can be resolved with the corresponding initial conditions. Resolving $Hold(Clear(A), t_1, t_5)$ with (6.3) will make t_5 bound with t'_3 by Skolem constant $SKC3$ and resolving $Hold(Clear(B), t_8, t'_8)$ with (6.4) will make t'_8 bound with t_1 by $SKC4$ (See Fig. 9). The first binding leads to $t_1 \prec t_5$ because of the temporal constraint $t_1 \prec t'_3$ in (6.8) and the second binding leads to $t_5 \prec t_1$ because of the temporal constraint $t_5 \prec t'_8$ in (6.12).

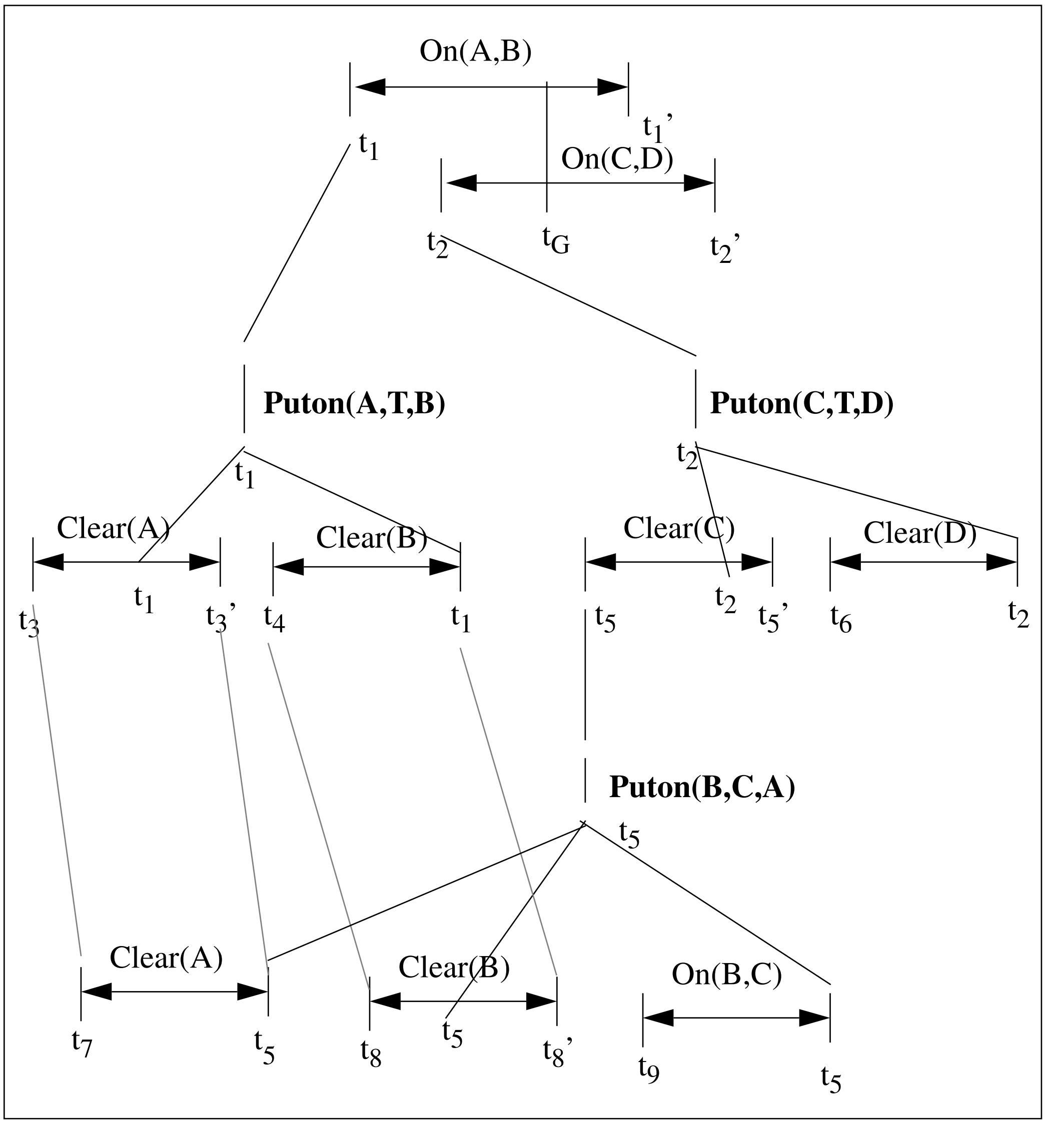

Figure 9: One possible planning process for the problem.

These two temporal relation contradict each other, so the set of temporal constraints is not consistent with $T_{<_T}$. The planner has to backtrack. One possible way to backtrack is to choose a different action $Puton(B, C, T)$ to achieve the intermediate goal $Clear(C)$ and the plan constructed will be the same as the one given before.

In this example, the constructed plan in Figure 8 satisfies the corresponding set of action constraints (Zhang, 1994). Therefore, the nonlinear plan is conflict free.

B A Temporal Logic for Nonlinear Planning

B.1 Syntax

Given

DV: a set of data variables,

TV: a set of temporal variables which is disjoint from DV,

DF: a set of data function symbols,

TF: a set of temporal function symbols which is disjoint from DF,

$\mathcal{A}$: a set of primitive action symbols,

$\mathcal{P}$: the set of primitive property symbols which is disjoint from $\mathcal{A}$,

the set of data terms, T_D, is defined recursively as follows

1. if $t \in DV$, then $t \in T_D$,
2. if $f \in DF$, $arity(f) = n$ and $t_1, \ldots, t_n \in T_D$, then $f(t_1, \ldots, t_n) \in T_D$.

The definition of the set of *temporal terms*, T_T, is similar to that of data terms except that the arguments of temporal functions are not restricted to be temporal terms. The set of temporal terms, T_T, is defined as follows:

1. if $t \in TV$, then $t \in T_T$,
2. if $f \in TF$, $arity(f) = n$, and $t_1, \ldots, t_n \in T_T \bigcup T_D$, then $f(t_1, \ldots, t_n) \in T_T$.

The set of *well-formed formulas(wffs)* is defined inductively as follows:

1. If $dt_1, \ldots, dt_n \in T_D$, $tt_1, tt_2 \in T_T$, $P \in \mathcal{P}$, $arity(P) = n$, then $Hold(P(dt_1, \ldots, dt_n), tt_1, tt_2)$ is a wff,
2. if $dt_1, \ldots, dt_n \in T_D$, $tt_1, tt_2 \in T_T$, $A \in \mathcal{A}$, $arity(A) = n$, then $Exec(A(dt_1, \ldots, dt_n), tt_1, tt_2)$ is a wff,
3. if $tt_1, tt_2 \in T_T$, then $tt_1 \prec tt_2$ and $tt_1 \preccurlyeq tt_2$ are wffs,
4. if φ and ψ are wffs, then so are $\neg\varphi$, $\varphi \wedge \psi$,
5. if φ is a wff and $v \in DV \bigcup TV$ is a free variable of φ, then $\forall v.\varphi[v]$ is a wff.

We assume the usual definitions of $\vee$, $\Longrightarrow$, $\Longleftrightarrow$ and $\exists$.

B.2 Semantics

Definition 10 *A* partial order time structure *is a pair* $(\mathcal{W}, \mathcal{R})$ *where* $\mathcal{W}$ *is a non-empty universe of time points and* $\mathcal{R}$ *is a* partial order relation *between the time points in* $\mathcal{W}$.

Definition 11 *An* interpretation *of a temporal language is a tuple :* $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \mathcal{D}, \pi_F, \pi_P \rangle$ *where*

$(\mathcal{W}, \mathcal{R})$ *is a partial order time structure,*

$\mathcal{D}$ *is the domain of* data *for all the time points in* $\mathcal{W}$.

π_F *is a interpretation of the function symbols. If* $f \in DF \bigcup TF$ *and* $arity(f) = n$, *then* $\pi_F(f) \in (\mathcal{D} \bigcup \mathcal{W})^n \longrightarrow \mathcal{D} \bigcup \mathcal{W}$.

π_P *is a interpretation of the proposition types (actions and properties). If* $P \in \mathcal{P} \bigcup \mathcal{A}$ *and* $arity(P) = n$, *then* $\pi_P(P) \in \mathcal{D}^n \longrightarrow 2^{\mathcal{W} \times \mathcal{W}}$. *We require that if* $(t_1, t_2) \in \pi_P(P)(dt_1, \ldots, dt_n)$, *then* $(t_1, t_2) \in \mathcal{R}$. *To characterize that assertions are interpreted over the necessarily maximal intervals,* π_P *has the property that, if* $(t_1, t_2), (t_3, t_4) \in \pi_P(P)(dt_1, \ldots, dt_n)$ *and* (t_1, t_2) *and* (t_3, t_4) *do not refer to the same time interval, then either* $(t_2, t_3) \in \mathcal{R}$ *or* $(t_4, t_1) \in \mathcal{R}$.

A *variable assignment* ν for a given interpretation $\mathcal{M}$, is a mapping from variable symbols to the elements of the domains, $\nu : (DV \bigcup TV) \longrightarrow (\mathcal{D} \bigcup \mathcal{W})$ which satisfies sort restriction, i.e. $\nu(x) \in \mathcal{D}$ if $x \in DV$ and $\nu(x) \in \mathcal{W}$ if $x \in TV$.

For a given interpretation $\mathcal{M}$ and variable assignment ν, a *term assignment* τ is a mapping, $\tau : (T_D \bigcup T_T) \longrightarrow (\mathcal{D} \bigcup \mathcal{W})$ which can be defined as follows.

1. If $t \in DV \bigcup TV$, then $\tau(t) = \nu(t)$,
2. if $f \in DF \bigcup TF$ and $arity(f) = n$, then $\tau(f(t_1, \ldots, t_n)) = \pi_F(f)(\tau(t_1), \ldots, \tau(t_n))$.

The semantics of the wffs is given with respect to an interpretation $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, \mathcal{D}, \pi_F, \pi_P, \rangle$ and a variable assignment ν. The *satisfaction relation* ($\models$)of the well formed formulas under such a pair is defined recursively as follows.

$$
\begin{array}{lll}
\langle \mathcal{M}, \nu \rangle & \models & Hold(P(dt_1, \ldots, dt_n), tt_1, tt_2) \text{ iff} \\
& & (\tau(tt_1), \tau(tt_2)) \in \pi_P(P)(\tau(t_1), \ldots \tau(t_n)) \\
\langle \mathcal{M}, \nu \rangle & \models & Exec(A(dt_1, \ldots, dt_n), tt_1, tt_2) \\
& & \text{ iff } (\tau(tt_1), \tau(tt_2)) \in \pi_P(A)(\tau(t_1), \ldots \tau(t_n)) \\
\langle \mathcal{M}, \nu \rangle & \models & t_1 \prec t_2 \text{ iff } (\tau(t_1), \tau(t_2)) \in \mathcal{R} \\
\langle \mathcal{M}, \nu \rangle & \models & t_1 \preccurlyeq t_2 \text{ iff } (\tau(t_1), \tau(t_2)) \in \mathcal{R} \text{ or } \tau(t_1) = \tau(t_2) \\
\langle \mathcal{M}, \nu \rangle & \models & \neg\varphi \text{ iff } \langle \mathcal{M}, \nu \rangle \models \varphi \text{ is false} \\
\langle \mathcal{M}, \nu \rangle & \models & (\varphi \wedge \psi) \text{ iff } \langle \mathcal{M}, \nu \rangle \models \varphi \text{ and } \langle \mathcal{M}, \nu \rangle \models \psi \\
\langle \mathcal{M}, \nu \rangle & \models & \forall v.\varphi \text{ iff } \langle \mathcal{M}, \nu' \rangle \models \varphi \text{ for all } \nu' \text{ which} \\
& & \text{agree with } \nu \text{ except possibly on } v.
\end{array}
$$

An interpretation $\mathcal{M}$ is a *model* for a wff φ (written as $\mathcal{M} \models \varphi$) if $\langle \mathcal{M}, \nu \rangle \models \varphi$ for all variable assignments ν. A wff is a *closed formula* if it contains no free variables. If a closed formula φ is satisfied by a interpretation $\mathcal{M}$ and a variable assignment, then $\mathcal{M}$ is a model of φ. A wff φ is said to be *satisfiable* if it has a model. A wff φ is said to be *valid* (written as $\models \varphi$) if its negation is not satisfiable.