CHAPTER 14 Temporal Planning

14.1 Introduction

The previous chapter was devoted to temporal reasoning techniques for planning. This chapter builds on that material by presenting some adequate representations for planning involving explicit time and by developing two related approaches to temporal planning. Time will be dealt with mainly within point algebra (PA) (Section 13.3.1) and the simple temporal networks (Section 13.4.1). Other approaches that rely on interval algebra (IA) will be discussed. We will focus on temporal planners that extend the planning techniques developed earlier in the book. However, as explained in the introduction to Part IV, we will depart significantly from the model of state-transition systems considered so far. We will view an action not as a single state transition but as a collection of local change and persistence conditions that are spread out in time but are focused on just a few state variables or propositions.

Such a view offers several advantages in expressiveness, which have been introduced earlier. In particular, explicit time is essential for handling properly the interaction of concurrent actions. For example, consider a door with a spring lock that controls the turning of the knob. Two synchronized actions are required for opening the door: (1) pushing the spring lock and maintaining the pressure, and (2) turning the knob and pulling open the door. It is not very satisfactory to represent this domain with two operators for these two actions (which may be needed separately for other tasks) and a third specific operator corresponding to their concurrent use. An adequate representation would enable reasoning on the concurrent execution of the two actions and their joint effects.

In previous chapters we considered plans that are more general than a strict sequence of actions: a partial-order plan in plan-space and HTN planning or a sequence of subsets of actions in planning-graph approaches. However, in all the algorithms presented earlier, only *independent* actions are left unordered and can occur concurrently. Their joint effects are simply the union of their individual effects. The additional combined effects of two concurrent actions cannot be expressed in classical planning unless we allow for a separate operator to represent the combined actions. Joint effects of concurrent and interfering actions will be dealt with here.

In order to specify a temporal planning domain and a problem as input to a planner, one needs a language. The purpose of this chapter is not to formalize such a language but instead to focus on the semantics of its underlying representation and to present temporal planning techniques. Two closely related approaches will be considered. The first one relies on extending the usual planning operators to include temporal preconditions and effects. That approach has a pedagogical merit: it introduces progressively most of the concepts needed in this chapter while keeping a representation as close as possible to that of classical planning, with propositions that hold over periods of time but with no logical connectives. The second approach may appear less familiar to the reader because it does not distinguish explicitly between preconditions and effects of actions. It specifies explicitly change and persistence constraints over time.

In both approaches, the underlying representation and semantics are those of CSPs, i.e., sets of constrained variables ranging over some domains. However, while in a CSP the variables are static and get single values in a solution, here the variables get different values at different time points, i.e., they are functions of time. The second approach relies on the state-variable representation that makes these functions of time explicit.

These two approaches are presented in Sections 14.2 and 14.3, respectively. The chapter ends with a discussion that briefly introduces other temporal planning techniques, followed by exercises.

14.2 Planning with Temporal Operators

Earlier in this book we viewed the dynamics of a domain as a sequence of states, and we had a way to tell which propositions are true in a given state and which are false. Here we will qualify each proposition of the domain with respect to the time periods during which the proposition is true, and we will build up a way to tell when a proposition is true and when it ceases to be true. This will be done through *temporally qualified expressions* in *temporal databases.*

14.2.1 Temporal Expressions and Temporal Databases

Let us define a representation scheme for specifying a temporal planning domain whose building blocks are finite sets of constant symbols, variable symbols, relation symbols, and constraints as described here.

The *constant symbols* are partitioned into disjoint classes corresponding to the objects of the domain, e.g., the classes of robots, locations, cranes, containers, and piles in the DWR domain.

The *variable symbols* are either *object variables* that are typed variables ranging over the classes or union of classes of constants or *temporal variables* ranging over the reals $\Re$. Object and temporal variables are viewed as CSP variables whose sets of possible values within their respective domains are constrained.

The *relation symbols* can be of two types.

- *Rigid relation symbols* represent relations that do not vary in time for a given planning problem, e.g., adjacent(loc1,loc2), belong(pile1,loc1), attached(crane1,loc1).
- *Flexible relation symbols*, also called *fluents*, represent relations on the constants of the domain that may or may not hold at some instant for a planning problem, e.g., at(robot1,loc1), in(cont1,pile1).

The *constraints* can also be of two types.

- *Temporal constraints* reside within the symbolic *PA* algebra calculus (see Section 13.3.1). Hence temporal variables will not be instantiating into numerical values[1] but is kept as a consistent set of constrained variables.
- *Binding constraints* on object variables are expressions of the form $x = y$, $x \neq y$, and $x \in D$, D being a set of constant symbols.

This representation scheme has time-invariant expressions and time-dependent expressions. The former are rigid relations and binding constraints; both will be referred to as *object constraints*. The latter are flexible relations and temporal constraints. These relations and constraints are used to specify temporally qualified expressions and temporal databases.

A *temporally qualified expression (tqe)* is an expression of the form:

$$p(\zeta_i, \ldots, \zeta_k)@[t_s, t_e)$$

where p is a flexible relation, $\zeta_i, \ldots, \zeta_k$ are constants or object variables, and t_s, t_e are temporal variables such that for $t_s < t_e$. A *tqe* $p(\zeta_i, \ldots, \zeta_k)@[t_s, t_e)$ asserts that $\forall t$ such that for $t_s \leq t < t_e$, the relation $p(\zeta_i, \ldots, \zeta_k)$ holds at the time t. For convenience reasons, to be illustrated in the following example, the interval $[t_s, t_e)$ qualifying a *tqe* is semi-open: the relation $p(\zeta_i, \ldots, \zeta_k)$ holds at t_s but not necessarily at t_e.

A *temporal database* is a pair $\Phi = (\mathcal{F}, \mathcal{C})$, where $\mathcal{F}$ is a finite set of *tqe*s and $\mathcal{C}$ is a finite set of temporal and object constraints; $\mathcal{C}$ is required to be consistent in the CSP sense, i.e., there exist values for the variables that meet all the constraints.

Example 14.1 In the DWR domain, suppose that a robot rob1 remains in a location loc1 and another robot rob2 moves from a location loc2 to an adjacent location loc3.

1. The use of numerical temporal constraint networks is discussed in Section 13.4.

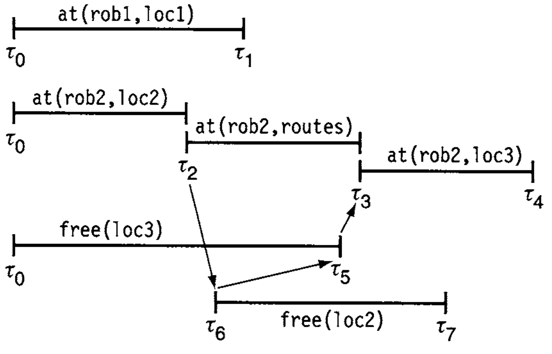

Figure 14.1 A picture of a temporal database.

We can express precisely an instance of this scenario with the following temporal database (Figure 14.1):

$$\begin{aligned}\Phi = (\{&\text{at(rob1, loc1)}@[\tau_0, \tau_1), \text{at(rob2, loc2)}@[\tau_0, \tau_2),\\ &\text{at(rob2, routes)}@[\tau_2, \tau_3), \text{at(rob2, loc3)}@[\tau_3, \tau_4),\\ &\text{free(loc3)}@[\tau_0, \tau_5), \text{free(loc2)}@[\tau_6, \tau_7)\},\\ \{&\text{adjacent(loc1, loc2), adjacent(loc2, loc3)}, \tau_2 < \tau_6 < \tau_5 < \tau_3\})\end{aligned}$$

Φ says that robot rob1 is located at loc1 from time τ_0 to τ_1. Robot rob2 is at loc2 at time τ_0; it leaves that location at time τ_2 heading for an adjacent location loc3 that it reaches at time τ_3, where it stays till τ_4. Location loc3 is free from τ_0 till rob2 reaches it; loc2 becomes free when rob2 leaves it. The constant routes refers to all possible routes between locations because we are assuming that we do not need to specify a particular route.[2] We are assuming that a location can hold only one robot at a time: the relation free means that the location is not occupied. The temporal constraints require the destination location loc3 to remain free until rob2 gets into it ($\tau_5 < \tau_3$), while loc2 becomes free when rob2 leaves it and before it reaches loc3 ($\tau_2 < \tau_6 < \tau_5$). Because of the semiopen intervals in *tqes*, there is no ambiguity about the actual location of rob2 at the time points τ_2 and τ_3. Finally, note that the variables τ_0, τ_4, and τ_7 remain unconstrained, except for the implicit constraints on the end points of intervals qualifying *tqes*, i.e., $\tau_0 < \tau_1$, $\tau_3 < \tau_4$, $\tau_6 < \tau_7$, etc. ■

A temporal database Φ represents assertions about how the world changes over time. From these assertions in Φ we are going to deduce other assertion supported by Φ. Note that the representation does not have logical connectives. In particular, we are not using negated atoms. We rely on a variant of the closed-world assumption: a flexible relation holds in a temporal database Φ only during the periods of time explicitly stated by *tqes* in Φ; a rigid relation holds iff it is in Φ.

2. This constant is like the constant pallet used in previous chapters.

The intuition for a set of *tqes* supported by a temporal database is the following: suppose we are given the database Φ illustrated in Example 14.1 and the following *tqe*: $\mathsf{free}(l)@[t, t')$. This *tqe* holds with respect to what is asserted in Φ provided that the interval $[t, t')$ can fit into one of the two intervals of Φ for which the flexible relation free is asserted, with a consistent binding for the variable l (see Figure 14.1). In other words, this *tqe* holds with respect to Φ if one of the two sets of constraints can be met: either $\{l = \mathsf{loc3}, \tau_0 \leq t, t' \leq \tau_5\}$ or $\{l = \mathsf{loc2}, \tau_6 \leq t, t' \leq \tau_7\}$. Note that we do not require the interval $[t, t')$ to be *equal* to the interval of the corresponding *tqe* in Φ, e.g., to $[\tau_0, \tau_5)$ or to $[\tau_6, \tau_7)$, because by definition a *tqe* holds for any subinterval of $[\tau_s, \tau_e)$. We will say that $\mathsf{free}(l)@[t, t')$ is *supported* by $\mathcal{F}$; the two sets of constraints are its *enabling conditions*, and one of them needs to be consistent with $\mathcal{C}$.

Definition 14.1 A set $\mathcal{F}$ of *tqes* *supports* a *tqe* $e = p(\zeta_i, \ldots, \zeta_k)@[t_1, t_2)$ iff there is in $\mathcal{F}$ a *tqe* $p(\zeta'_i, \ldots, \zeta'_k)@[\tau_1, \tau_2)$ and a substitution σ such that $\sigma(p(\zeta_i, \ldots, \zeta_k)) = \sigma(p(\zeta'_i, \ldots, \zeta'_k))$. An *enabling condition* for e in $\mathcal{F}$ is the conjunction of the two temporal constraints $\tau_1 \leq t_1$ and $t_2 \leq \tau_2$, together with the binding constraints of σ. ■

In general, an enabling condition for e in $\mathcal{F}$ is not unique because several *tqes* in $\mathcal{F}$ may be unifiable with e. The set of all possible enabling conditions is denoted by $\theta(e/\mathcal{F})$. This set is empty when $\mathcal{F}$ does not support e.

Similarly, $\mathcal{F}$ supports a set of *tqes* $\mathcal{E}$ iff there is a substitution σ that unifies every element of $\mathcal{E}$ with an element of $\mathcal{F}$. An enabling condition for $\mathcal{E}$ in $\mathcal{F}$ is the conjunction of enabling conditions for the elements of $\mathcal{E}$. The set of all possible enabling conditions for $\mathcal{E}$ in $\mathcal{F}$ is denoted by $\theta(\mathcal{E}/\mathcal{F})$. This set is empty when $\mathcal{E}$ is not supported by $\mathcal{F}$.

Definition 14.2 A temporal database $\Phi = (\mathcal{F}, \mathcal{C})$ *supports* a set of *tqes* $\mathcal{E}$ when $\mathcal{F}$ supports $\mathcal{E}$ and there is an enabling condition $c \in \theta(\mathcal{E}/\mathcal{F})$ that is consistent with $\mathcal{C}$. $\Phi = (\mathcal{F}, \mathcal{C})$ *supports* another temporal database $(\mathcal{F}', \mathcal{C}')$ when $\mathcal{F}$ supports $\mathcal{F}'$ and there is an enabling condition $c \in \theta(\mathcal{F}'/\mathcal{F})$ such that $\mathcal{C}' \cup c$ is consistent with $\mathcal{C}$. ■

The following definition generalizes to temporal databases the notion of entailed constraints, as defined in Chapter 8.

Definition 14.3 A temporal database $\Phi = (\mathcal{F}, \mathcal{C})$ *entails* another temporal database $(\mathcal{F}', \mathcal{C}')$ iff $\mathcal{F}$ supports $\mathcal{F}'$ and there is an enabling condition $c \in \theta(\mathcal{F}'/\mathcal{F})$ such that $\mathcal{C} \models \mathcal{C}' \cup c$. ■

This definition requires a set of *tqes* to be supported and one of its enabling conditions, as well as $\mathcal{C}'$, to be entailed by the constraints of Φ. Note that c is not a single constraint but a set of binding and temporal constraints.

When Φ supports a set of *tqes*, then we can consider these *tqes* to hold with respect to Φ only if we *augment* the constraints of Φ with an enabling condition consistent with $\mathcal{C}$. However, when Φ entails $(\mathcal{F}', \mathcal{C}')$, then the *tqes* of $\mathcal{F}'$ and the constraints of $\mathcal{C}'$ already hold with respect to Φ. This distinction is useful because we will be planning by performing successive refinements of a database, each refinement adding to it new *tqes* and constraints.

14.2.2 Temporal Planning Operators

We are now ready to define temporal planning operators and domains. Let us first introduce an example.

Example 14.2 In the DWR domain, we would like to specify a move operator for a robot r that leaves one location l at time t_s and reaches another adjacent location l' at time t_e. Figure 14.2 illustrates a possible definition of this operator with the intervals involved with respect to the duration of the action. When an end point of an interval is not constrained, e.g., t_1, t_5, t_3, it is left without a vertical bar (these are free variables in the operator).

The two preconditions of the operator require that the robot r should be at location l during some interval of time until t_s and that there should be free space available in location l' during some interval ending at t_e because a location can hold only one robot at a time.

The effects of the move operator are to have the robot on the way (constant routes) from t_s to t_e, to have it at l' at some interval starting in t_e, and to have space available in l starting at a time point t_4 between t_s and t_2, i.e., before space is required in l'.

The operator requires the two locations to be adjacent[3] and to have free space at the destination only after making the origin location free, i.e., $t_s < t_4 < t_2$. Note that the precondition $\text{at}(r, l)@[t_1, t_s)$ does not mean that r is not at location l outside of $[t_1, t_s)$; it only requires r to be there during this interval: any *tqe* $\text{at}(r, l)@[\tau_i, \tau_j)$ in a temporal database that may support this precondition will not be affected, at this point, by what is stated in the operator. Similarly, the precondition $\text{free}(l')@[t_2, t_e)$ does not put any requirement on the location l' outside of the interval $[t_2, t_e)$, which is after the starting point of the action. The constraints for the end points of the intervals are implicit: $t_s < t_e$, $t_1 < t_s$, $t_2 < t_e$, $t_e < t_3$, and $t_4 < t_5$. ■

A *temporal planning operator* is a tuple $o = (\text{name}(o), \text{precond}(o), \text{effects}(o), \text{const}(o))$, where:

- name(o) is an expression of the form $o(x_1, \ldots, x_k, t_s, t_e)$ such that o is an operator symbol, and $x_1, \ldots, x_k$ are all the object variables that appear in o,

3. The relations adjacent and free apply here to locations. We assume the constant routes to be always free and adjacent to and from every location.

```
move(r, l, l')@[$t_s$, $t_e$)
    precond: at(r, l)@[$t_1$, $t_s$)
             free(l')@[$t_2$, $t_e$)
    effects: at(r, routes)@[$t_s$, $t_e$)
             at(r, l')@[$t_e$, $t_3$)
             free(l)@[$t_4$, $t_5$)
    const:   $t_s < t_4 < t_2$
             adjacent(l, l')
```

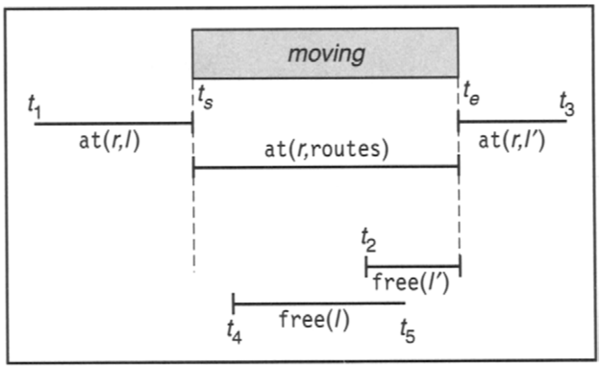

Figure 14.2 A temporal planning move operator.

together with the temporal variables in const(o). The other unconstrained temporal variables in o are *free variables.*

- precond(o) and effects(o) are *tqes*.
- const(o) is a conjunction of temporal constraints and object constraints, the latter being either rigid relations or binding constraints on object variables of the form $x = y$, $x \neq y$, or $x \in D$, with D being a set of constants.

Rather than writing an operator as a tuple, we use the format shown in Figure 14.2, where the temporal variables t_s, t_e, t_2, and t_4 are implicit parameters of the operator, while t_1, t_3, and t_5 are free variables.

An *action* is a partially instantiated planning operator $a = \sigma(o)$ for some substitution σ. The intended semantics is that if the preconditions and the constraints of an action hold with respect to some database, then the action is applicable. It will run from time t_s to t_e. The new *tqes* resulting from its execution are described by its effects. An important point to notice here is that the other changes due to the action, such as *tqes* that no longer hold, are not explicit in this representation. We have not introduced here the equivalent of effects$^-$, which would be needed in order to state explicitly, e.g., that r ceases to be at l at time t_s or that l' is not free anymore after t_e. This will become clear with the introduction of domain axioms in Section 14.2.3.

Let us now define the applicability of an action more precisely. Given a temporal database $\Phi = (\mathcal{F}, \mathcal{C})$, the preconditions of an action a are *supported* by $\mathcal{F}$ (according to Definition 14.1) iff there is a substitution σ such that for each $p@[t_1, t_2) \in$ precond(a), there is a *tqe* $q@[\tau_1, \tau_2) \in \mathcal{F}$ with $\sigma(p) = \sigma(q)$. An enabling condition for precond(a) is a conjunction, over all preconditions of a, of the two temporal constraints $\tau_1 \leq t_1$, $t_2 \leq \tau_2$, and the binding constraints in σ. This enabling condition is not unique because there can be more than one *tqe* in $\mathcal{F}$ unifiable with a condition. The set of all enabling conditions for precond(a) is denoted $\theta(a/\mathcal{F})$.

Definition 14.4 An action a is *applicable* to $\Phi = (\mathcal{F}, \mathcal{C})$ iff precond(a) is supported by $\mathcal{F}$ and there is an enabling condition c in $\theta(a/\mathcal{F})$ such that $\mathcal{C} \cup \text{const}(a) \cup c$ is a consistent set of constraints. ■

In other words, a is applicable to Φ if instances of its preconditions are supported by the database, within some consistent constraints. A result of applying a to Φ is a database *extended* with the effects of a and with other constraints. This is close to the corresponding definitions in classical planning, but the important difference compared with the state-transition function γ is that here we are not removing anything from the temporal database.

The *result* of applying an action a to a database Φ is a *set* of possible databases. Let us introduce a provisional definition of the result of a in Φ (we will refine it in Section 14.2.3):

$$\gamma_0(\Phi, a) = \{(\mathcal{F} \cup \text{effects}(a), \mathcal{C} \cup \text{const}(a) \cup c) \mid c \in \theta(a/\mathcal{F})\} \tag{14.1}$$

This result is empty whenever a is not applicable to Φ. Otherwise the set $\gamma_0(\Phi, a)$ may contain several possible databases because $\theta(a/\mathcal{F})$ is not a singleton. Each such resulting database is obtained by *consistently adding* to Φ the set of *tqes* in effects(a) and the set of constraints const(a) $\cup$ c for some $c \in \theta(a/\mathcal{F})$.

It is important to note that the result of applying an action a is defined as being a set of databases. This is not due to nondeterminism; actions here are deterministic. This is because an action a can be applied in different ways and at different times in Φ, and we want to refer to the set of all these possibilities.

Example 14.3 Consider the temporal database of Figure 14.1. The action move(rob1,loc1,loc2)@$[t_s, t_e)$ is applicable to Φ. This is because the *tqe* at(rob1,loc1)@$[\tau_0, \tau_1)$ supports at(r, l)@$[t_1, t_s)$ when r = rob1, l = loc1, $\tau_0 \leq t_1$ and $t_s \leq \tau_1$; the *tqe* free(loc2)@$[\tau_6, \tau_7)$ supports free(l')@$[t_2, t_e)$ when l' = loc2 $\tau_6 \leq t_2$, and $t_e \leq \tau_7$. An enabling condition of this move action is thus:

$$c = \{r = \text{rob1}, l = \text{loc1}, l' = \text{loc2}, \tau_0 \leq t_1, t_s \leq \tau_1, \tau_6 \leq t_2, t_e \leq \tau_7\}$$

Note that the constraints in const(o), i.e., $t_s < t_4 < t_2$, are consistent with Φ and c because, in particular, τ_6 is not constrained with respect to τ_1.

If we had in Φ another *tqe* asserting rob1 in loc1, e.g., at(rob1,loc1)@$[\tau_8, \tau_9)$, we would have another instance of move(rob1,loc1,loc2)@$[t'_s, t'_e)$ applicable to Φ at different time periods than the previous instance, provided that the constraints are consistent.

If we also assume adjacent(loc3,loc2) in Φ, then there is another enabling condition for move, corresponding to moving rob2 from loc3 to loc2. Similarly, adjacent(loc1,loc3) would allow for another way to apply this operator, by moving rob1 from loc1 to loc3. ■

At this point, let us discuss the intended semantics of the variables in a temporal planning operator. For object variables, there are no major differences between a classical planning operator and a temporal one. All such variables appearing in an operator are existentially quantified parameters of the operator, ranging over finite domains. In classical planning, these parameters are instantiated with respect to atoms in a state. Here, they are instantiated with respect to *tqes* in a temporal database. Note, however, that a temporal database corresponds to a consistent collection of states.

The temporal parameters are also existentially quantified variables, but they range over the infinite set of real numbers. Hence, an applicable action has an infinite number of instances, e.g., for the action move(rob1,loc1,loc2)@$[t_s, t_e)$ in the previous example, each consistent tuple of values of its temporal parameters t_s, t_e, t_2, and t_4 defines a different instance. In this section we are using the *PA* symbolic calculus (i.e., we are not instantiating temporal variables into numerical values), so all these instances are equivalent.[4] Note that the free variables of the operator—t_1, t_3, and t_5 in the previous example—play no particular role in the sense that an instance of the action does not depend on the values of these variables.

14.2.3 Domain Axioms

The representation described in this section has no logical connectives. The "negative" effects of an action are not made explicit in the definition of the operator. Indeed, this definition asserts only what holds as an effect of an action—it does not say what does not hold anymore. For example, the move operator in Figure 14.2 does not say that the robot is no longer at location l starting at t_s; neither does it specify that location l' is not free starting at t_e. These effects will result from domain axioms on *tqes*.

A domain axiom is a conditional expression of the form:

$$\rho = \text{cond}(\rho) \rightarrow \text{disj}(\rho)$$

where:

- $\text{cond}(\rho)$ is a set of *tqes*.
- $\text{disj}(\rho)$ is a *disjunction* of temporal and object constraints.

Example 14.4 An axiom is required to specify that an object cannot be in two distinct places at the same time. For example, for the move operator in Figure 14.2 we write:

$$\{\text{at}(r, l)@[t_s, t_e), \text{at}(r', l')@[t'_s, t'_e)\} \rightarrow (r \neq r') \vee (l = l') \vee (t_e \leq t'_s) \vee (t'_e \leq t_s)$$

If the two *tqes* in $\text{cond}(\rho)$ hold, then either we have two distinct robots, or one robot is in the same location, or the two intervals of these *tqes* are disjoint.

4. We will discuss in Section 14.3.4 the issues involved for instantiating temporal parameters.

Furthermore, we assume that there is space for just one robot at a time in a given location. However, the previous move operator does not prevent several robots from moving into the same location at the same time. The following axiom asserts this restriction:

$$\{\mathrm{at}(r,l)@[t_1,t_1'), \mathrm{free}(l')@[t_2,t_2')\} \rightarrow (l \neq l') \vee (t_1' \leq t_2) \vee (t_2' \leq t_1)$$

■

The semantics of domain axioms is given by Definition 14.5.

Definition 14.5 Let ρ be an axiom, $\Phi = (\mathcal{F}, \mathcal{C})$ be a database such that cond(ρ) is supported by $\mathcal{F}$, and $\theta(\rho/\mathcal{F})$ be the set of enabling conditions for cond(ρ) in $\mathcal{F}$. Φ is *consistent* with ρ iff for each enabling condition $c_1 \in \theta(\rho/\mathcal{F})$, there is at least one disjunct $c_2 \in$ disj(ρ) such that $\mathcal{C} \cup c_1 \cup c_2$ is a consistent set of constraints.

■

In other words, this definition requires that for every way of enabling cond(ρ), there is at least one disjunct in disj(ρ) that is consistent with Φ.

A database Φ is consistent with a set X of axioms if it is consistent with every axiom in X.

Example 14.5 Consider the database of Example14.1 augmented with the effects of move(rob1,loc1,loc2)@$[t_s, t_e)$, whose sole enabling condition, as given in Example 14.3, is $\{r = \text{rob1}, l = \text{loc1}, l' = \text{loc2}, \tau_0 \leq t_1, t_s \leq \tau_1, \tau_6 \leq t_2, t_e \leq \tau_7\}$.

The first axiom of the previous example is supported for $r = r' = \text{rob1}$, $l = \text{loc1}$, and $l' = \text{routes}$. Consequently, $(t_e \leq \tau_0) \vee (\tau_1 \leq t_s)$ holds. But $(t_e \leq \tau_0)$ entails $t_1 < t_s < t_e \leq \tau_0 < \tau_1$, which is inconsistent with $\tau_0 \leq t_1$. However, $(\tau_1 \leq t_s)$ is consistent with Φ. Furthermore, because in Φ $t_s \leq \tau_1$, then $\tau_1 = t_s$. This means that the *tqe* at(rob1,loc1)@$[\tau_0, \tau_1)$ ends exactly at t_s, the starting point of the move.[5]

The second axiom is also supported for $l = l' = \text{loc2}$. Hence $(t_3 \leq \tau_6) \vee (\tau_7 \leq t_e)$ holds. Here also $(t_3 \leq \tau_6)$ contradicts $\tau_6 \leq t_2 < t_e \leq \tau_7$. But $(\tau_7 \leq t_e)$ is consistent with Φ, and together with $t_e \leq \tau_7$, it entails $\tau_7 = t_e$. This means that loc2 ceases to be free at exactly the end of the action.

■

A *consistency condition* for Φ with respect to a set of axioms X is the following union over all axioms $\rho \in X$ and over all enabling conditions in $\theta(\rho/\mathcal{F})$:

$$\varpi(c_2) = \bigcup_{\rho \in X, c_1 \in \theta(\rho/\mathcal{F})} \{c_1 \cup c_2 \mid c_2 \in \mathrm{disj}(\rho)\} \tag{14.2}$$

5. A finer model of move may allow for some time between t_s and the moment at which rob1 leaves loc1, i.e., the time to get out of the initial location. This can easily be done with a slight modification in the definition of the operator (see Exercise 14.1).

There can be several such consistency conditions because each disj(ρ) may contain more than one constraint c_2. Let $\theta(X/\Phi) = \cup_{c_2} \varpi(c_2)$, i.e., the set of all consistency conditions of Φ with respect to X. Φ is consistent with the axioms of X iff *either* of the following holds:

- $\theta(X/\Phi)$ is empty, i.e., no axiom has its conditions supported.
- There is a consistency condition $c \in \theta(X/\Phi)$ that is consistent with $\mathcal{C}$.

In other words, Φ *satisfies* X iff either $\theta(X/\Phi)$ is empty or there is such a condition c that is *entailed* by $\mathcal{C}$.

A consistency condition gives, for all supported axioms, a set of constraints that needs to be met in order to enforce the consistency of every possible instance of Φ with the axioms. When Φ satisfies X, then this database will always remain consistent with X whatever constraints are consistently added to it. When Φ is consistent with X but does not satisfy X, then the constraints of Φ need to be augmented with a constraint $c \in \theta(X/\Phi)$ in order to maintain this consistency for every instance of Φ. The set of all databases obtained from Φ that satisfies the axioms X is thus:

$$\psi(\Phi, X) = \{(\mathcal{F}, \mathcal{C} \cup c) \mid c \in \theta(X/\Phi)\} \tag{14.3}$$

This set is empty when Φ is not consistent with X.

Having specified a set X of domain axioms, we'll restrict our interest solely to the databases that satisfy X. According to the preliminary definition of the result of an action, the set $\gamma_0(\Phi, a)$ of databases resulting from an action a may contain several databases that are not consistent with the axioms of X. Consequently, we will refine the definition of the result of applying an action a to Φ (as given by γ_0 in Equation 14.1) by restricting it to contain only databases that are consistent with X and that are augmented with the constraints needed to satisfy X. The result of applying to Φ an action a with respect to X is now the set of databases:

$$\gamma(\Phi, a) = \bigcup_i \{\psi(\Phi_i, X) \mid \Phi_i \in \gamma_0(\Phi, a)\} \tag{14.4}$$

$\gamma(\Phi, a)$ is empty whenever a is not applicable to Φ or cannot lead to a database consistent with X. Otherwise, $\gamma(\Phi, a)$ may contain several possible databases, each being a pair $(\mathcal{F} \cup \text{effects}(a), \mathcal{C} \cup \text{const}(a) \cup c \cup c')$, where $c \in \theta(a/\mathcal{F})$ is an enabling condition for precond(a) and $c' \in \theta(X/\Phi)$ is a consistency condition.

Finally, let us remark that axioms have been illustrated in this section mainly to constrain *tqe*s to intervals consistent with the specifications of the domain. But we can also use axioms to specify time-invariant properties, as in classical planning. For example, a DWR domain where the adjacency relation between locations is symmetrical may contain the axiom $\{\text{adjacent}(l, l')\} \rightarrow \text{adjacent}(l', l)$.

14.2.4 Temporal Planning Domains, Problems, and Plans

We are now ready to define temporal planning domains, problems, and their solutions.

A temporal planning *domain* is a triple $\mathcal{D} = (\Lambda_\Phi, \mathcal{O}, X)$, where:

- Λ_Φ is the set of all temporal databases that can be defined with the constraints and the constant, variable, and relation symbols in our representation.
- $\mathcal{O}$ is a set of temporal planning operators.
- X is a set of domain axioms.

A temporal planning *problem* in $\mathcal{D}$ is a tuple $\mathcal{P} = (\mathcal{D}, \Phi_0, \Phi_g)$, where:

- $\Phi_0 = (\mathcal{F}, \mathcal{C})$ is a database in Λ_Φ that satisfies the axioms of X. Φ_0 represents an initial scenario that describes not only the initial state of the domain but also the evolution predicted to take place independently of the actions to be planned.
- $\Phi_g = (\mathcal{G}, \mathcal{C}_g)$ is a database that represents the goals of the problem as a set $\mathcal{G}$ of *tqe*s together with a set $\mathcal{C}_g$ of objects and temporal constraints on variables of $\mathcal{G}$.

The *statement* of a problem $\mathcal{P}$ is given by $P = (\mathcal{O}, X, \Phi_0, \Phi_g)$.

A plan is a set $\pi = \{a_1, \ldots, a_k\}$ of actions, each being a partial instance of some operator in $\mathcal{O}$.[6]

We would like to define the result of applying a plan π to a database Φ *independently of the order* in which the actions are considered during the planning process. A tentative definition (to be refined in Section 14.2.5) of the result of π could be the following:

$$\begin{array}{lcl} \gamma(\Phi, \{\}) & = & \{\Phi\} \\ \gamma(\Phi, \pi \cup \{a\}) & = & \bigcup_i \{\gamma(\Phi_i, a) \mid \Phi_i \in \gamma(\Phi, \pi)\} \end{array} \qquad (14.5)$$

such that π is a *solution* for a problem $P = (\mathcal{O}, X, \Phi_0, \Phi_g)$ iff there is a database in $\gamma(\Phi_0, \pi)$ that entails Φ_g.

However, this definition of $\gamma(\Phi, \pi)$ is not correct because the applicability of actions, as defined earlier, may depend on the order in which actions are considered, even though the actions are situated in time. This is dealt with in the next section, which generalizes the applicability conditions of actions.

6. Note that a plan is a set, not a sequence, because the actions in π are already situated in time.

14.2.5 Concurrent Actions with Interfering Effects

In classical planning, one cannot cope with concurrent and interfering actions unless one specifies an *ad hoc* operator for each useful combination of two operators a_1 and a_2, in order to express their joint conditions and effects. In HTN planning, an explicit method is also required for decomposing a task into the joint application of the two operators a_1 and a_2. A representation that requires the explicit definition of a_1, a_2, and a joint operator $a_1 \oplus a_2$, whenever such a join is useful, is less expressive than a representation that can reason on the joint applicability and joint effects of a given set of planning operators. The latter may solve more problems than the former.

The representation described here has such a capability. For example, it can manage the case of two actions a_1 and a_2 such that neither is applicable individually to a temporal database Φ but there is a joint combination of the two that is applicable and achieves a joint effect.

Example 14.6 Consider a DWR domain with just two locations loc1 and loc2 and two robots r1 and r2 that are initially in loc1 and loc2, respectively. The action move(r1,loc1,loc2)@$[t_s, t_e)$ is not applicable any time for lack of space in loc2. Similarly, move(r2,loc2,loc1) is not applicable because loc1 is not free. However, the DWR domain puts no space restriction on the routes between locations. Consequently, these two actions can be combined in time into two applicable and synchronized moves, as shown in Figure 14.3. The constraints required to make these two moves compatible are the following.

- When r1 gets to loc2, then r2 should have already left this location, i.e., $t'_4 \leq t_2$, and when r1 finishes moving into loc2, loc2 is no longer free, i.e., $t'_5 = t_e$.
- Similarly for r2 with respect to loc1: $t_4 \leq t'_2$ and $t_5 = t'_e$.

Note that t_s may take place before or after t'_s, and similarly for t_e and t'_e. ■

It is possible to view the set $\{a_1, a_2\}$ as a single joint operator

- whose effects are the union of the effects of a_1 and a_2;
- whose preconditions are the union of precond(a_1) minus the preconditions supported by effects(a_2), and of precond(a_2) minus the preconditions supported by effects(a_1); and
- whose constraints are the union of the two sets of constraints augmented with the constraints required to support preconditions of a_1 by effects(a_2) and preconditions of a_2 by effects(a_1).

This can be achieved by defining the applicability and effects $\gamma(\Phi, \pi)$ of a set of actions π as follows.

A pair of actions $\{a_1, a_2\}$ is applicable to $\Phi = (\mathcal{F}, \mathcal{C})$ when:

- $\mathcal{F} \cup$ effects(a_2) supports precond(a_1),

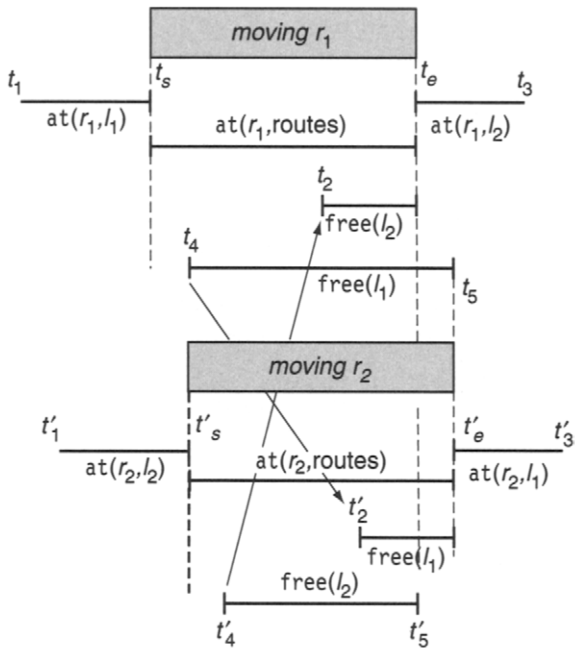

Figure 14.3 Two concurrent actions with interfering effects. Moving r_1 from l_1 to l_2 can take place only if the action is concurrent with the symmetrical move of r_2 and if the precedence constraints shown are met.

- $\mathcal{F} \cup \text{effects}(a_1)$ supports $\text{precond}(a_2)$,
- there is an enabling condition $c_1 \in C(a_1/\mathcal{F} \cup \text{effects}(a_2))$, and
- there is an enabling condition $c_2 \in C(a_2/\mathcal{F} \cup \text{effects}(a_1))$,

such that $\mathcal{C} \cup \text{const}(a_1) \cup c_1 \cup \text{const}(a_2) \cup c_2$ is a consistent set.[7]

The set of databases resulting from the application of the pair $\{a_1, a_2\}$ is:

$$\gamma(\Phi, \{a_1, a_2\}) = \bigcup_i \{\psi(\Phi_i, X) | \Phi_i \in \gamma_0(\Phi, \{a_1, a_2\})\},$$

where:

$$\begin{aligned}\gamma_0(\Phi, \{a_1, a_2\}) = \quad & \{(\mathcal{F} \cup \text{effects}(a_1) \cup \text{effects}(a_2), \\ & \quad \mathcal{C} \cup \text{const}(a_1) \cup c_1 \cup \text{const}(a_2) \cup c_2 \\ & \mid c_1 \in \theta(a_1/\mathcal{F} \cup \text{effects}(a_2)), c_2 \in \theta(a_2/\mathcal{F} \cup \text{effects}(a_1))\}.\end{aligned}$$

7. This explains the advantage of distinguishing between $\mathcal{F}$ and $\mathcal{C}$ and of defining the enabling condition to be a function only of $\mathcal{F}$, not of $\mathcal{C}$.

This definition is easily generalized to any finite set of actions. Let π be a set of actions and let us denote effects$(\pi_{-a})= \bigcup_{a' \in \pi - \{a\}}$ effects(a').

The set π is applicable to Φ iff the following conditions hold for $\forall a \in \pi$:

- $\mathcal{F} \cup$ effects(π_{-a}) supports precond(a), and
- there is an enabling condition $c_a \in \theta(a/\mathcal{F} \cup \text{effects}(\pi_{-a}))$,

such that $\mathcal{C} \bigcup_{a \in \pi}(\text{const}(a) \cup c_a)$ is a consistent set.

The sets $\gamma_0(\Phi, \pi)$ and $\gamma(\Phi, \pi)$ are now defined as follows:

$$\gamma_0(\Phi, \pi) = \begin{array}{l} \{(\mathcal{F} \bigcup_{a \in \pi} \text{effects}(a),\ \mathcal{C} \bigcup_{a \in \pi}(\text{const}(a) \cup c_a)) \mid \\ c_a \in \theta(a/\mathcal{F} \cup \text{effects}(\pi_{-a}))\} \end{array} \tag{14.6}$$

and

$$\gamma(\Phi, \pi) = \bigcup_i \{\psi(\Phi_i, X) \mid \Phi_i \in \gamma_0(\Phi, \pi)\}. \tag{14.7}$$

To summarize, the applicability of an action a is defined with respect to $\mathcal{F}$ augmented with the effects of *all* other actions in the plan. This is an essential feature of the approach that needs to be taken into account by planning algorithms, as illustrated by the procedure of the next section.

14.2.6 A Temporal Planning Procedure

Given a planning problem $P = (\mathcal{O}, X, \Phi_0, \Phi_g)$, we now concentrate on finding a set π of actions that are instances of operators in $\mathcal{O}$, such that there is a database $(\mathcal{F}, \mathcal{C}) \in \gamma(\Phi_0, \pi)$ that entails $\Phi_g = (\mathcal{G}, \mathcal{C}_g)$, i.e., such that every element of $\mathcal{G}$ is supported by $(\mathcal{F}, \mathcal{C} \cup \mathcal{C}_g)$. For any practical purpose, a planner has to exhibit not only the set π but also a database $(\mathcal{F}, \mathcal{C})$ in $\gamma(\Phi_0, \pi)$ that entails Φ_g. A natural approach for addressing this problem relies on

- a search space similar to that of partial plans (Chapter 5) that is naturally close to this representation and
- the CSP-based techniques and algorithms (Chapter 8)needed to handle the temporal and domain constraints that are essential in this representation.

A nondeterministic planning procedure called TPS implements such an approach. It generalizes the PSP procedure developed earlier (Figures 5.6 and 5.7). TPS proceeds as follows (see Figure 14.4).

```
TPS($\Omega$)
    $flaws \leftarrow$ OpenGoals($\Omega$) $\cup$ UnsatisfiedAxioms($\Omega$) $\cup$ Threats($\Omega$)
    if $flaws = \emptyset$ then return($\Omega$)
    select any flaw $\phi \in flaws$
    $resolvers \leftarrow$ Resolve($\phi, \Omega$)
    if $resolvers = \emptyset$ then return(failure)
    nondeterministically choose a resolver $\rho \in resolvers$
    $\Omega' \leftarrow$ Refine($\rho, \Omega$)
    return(TPS($\Omega'$))
end
```

Figure 14.4 TPS, a temporal planning procedure.

- It maintains a data structure, Ω, which defines the current *processing stage* of the planning problem. Ω contains *flaws* as long as a solution plan has not been found.
- It selects a flaw and finds all its resolvers (i.e., all ways to resolve the flaw), if there are any.
- It nondeterministically chooses one or several resolvers for that flaw. If needed, it eventually backtracks on this choice.
- It refines Ω with respect to the chosen resolvers.

This schema will be made precise by specifying the set of flaws (as computed by the OpenGoals, UnsatisfiedAxioms, and Threats subroutines) and how flaws are handled by the Resolve and Refine subroutines. The processing stage of the planning problem is a 4-tuple $\Omega = (\Phi, G, \mathcal{K}, \pi)$, where:

- $\Phi = (\mathcal{F}, \mathcal{C})$ is the current temporal database, as refined with the effects and constraints of planned actions.
- G is a set of *tqes* corresponding to the current open goals.
- $\mathcal{K} = \{C_1, \dots, C_i\}$ is a set of pending sets of enabling conditions of actions and pending sets of consistency conditions of axioms.
- π is a set of actions corresponding to the current plan.

Given the problem $P = (\mathcal{O}, X, \Phi_0, \Phi_g)$, where $\Phi_0 = (\mathcal{F}_0, \mathcal{C}_0)$, and $\Phi_g = (\mathcal{G}, \mathcal{C}_g)$, we first check whether the set of the goal constraints $\mathcal{C}_g$ is consistent with $\mathcal{C}_0$. If $\mathcal{C}_0 \cup \mathcal{C}_g$ is inconsistent, then the problem has no solution. Otherwise, the initial resolution stage Ω has the following components: $\Phi = (\mathcal{F}_0, \mathcal{C}_0 \cup \mathcal{C}_g)$, $G = \mathcal{G}$, $\pi = \emptyset$, and $\mathcal{K} = \emptyset$.

If the problem P is successfully solved, then the final values in Ω will be the following: $G = \mathcal{K} = \emptyset$, π is a set of actions that are instances of operators in $\mathcal{O}$, and $\Phi \in \gamma(\Phi_0, \pi)$ is such that Φ entails Φ_g.

The temporal planning procedure TPS (Figure 14.4) recursively refines Ω. TPS selects a flaw and finds its resolvers (branching step). It nondeterministically chooses a resolver and updates Ω with respect to the chosen resolver (refinement step). This is performed for three types of flaws: *open goals*, *unsatisfied axioms*, and *threats*, according to the following definitions of the resolvers of these flaws and the refinement operations.

Open Goals. If $G \neq \emptyset$, then every element $e \in G$ is a goal expression that is not yet supported by Φ. A resolver for such a flaw is either of the following.

- A *tqe* in $\mathcal{F}$ that can support e. Such a resolver will exist if $\theta(e/\mathcal{F})$ is not empty and contains at least one enabling condition that is consistent with $\mathcal{C}$. The refinement of Ω in this case is the following updates:

$$\mathcal{K} \leftarrow \mathcal{K} \cup \{\theta(e/\mathcal{F})\}$$
$$G \leftarrow G - \{e\}.$$

- An action a that is an instance of some operator in o, such that effects(a) supports e and const(a) is consistent with $\mathcal{C}$. In this case, the refinement of Ω consists of the following updates:

$$\pi \leftarrow \pi \cup \{a\}$$
$$\mathcal{F} \leftarrow \mathcal{F} \cup \text{effects}(a)$$
$$\mathcal{C} \leftarrow \mathcal{C} \cup \text{const}(a)$$
$$G \leftarrow (G - \{e\}) \cup \text{precond}(a)$$
$$\mathcal{K} \leftarrow \mathcal{K} \cup \{\theta(a/\Phi)\}$$

Note that we are requesting a to be relevant for e but not necessarily to be supported by the current stage of the database. Hence, in the last update listed, $\theta(a/\Phi)$ is a set of pending consistency conditions.

Unsatisfied Axioms. These flaws are possible inconsistencies of instances of Φ with respect to the axioms of X. That is, a flaw here is any axiom ρ such that cond(ρ) is supported by $\mathcal{F}$. A resolver is any consistency condition in $\theta(X/\Phi)$. The refinement of Ω is the update of $\mathcal{K}$ with this set of consistency conditions:

$$\mathcal{K} \leftarrow \mathcal{K} \cup \{\theta(X/\Phi)\}$$

Threats. If $\mathcal{K} \neq \emptyset$, then every $C_i \in \mathcal{K}$ is a pending set of consistency conditions or enabling conditions that are not yet entailed by the database Φ. A resolver for this flaw is a constraint $c \in C_i$ that is consistent with $\mathcal{C}$. The refinement of Ω consists of the following updates:

$$\mathcal{C} \leftarrow \mathcal{C} \cup c, \text{ for some } c \in C_i \text{ that is consistent with } \mathcal{C}$$
$$\mathcal{K} \leftarrow \mathcal{K} - \{C_i\}$$

The TPS procedure is a general schema. It is able to handle concurrent actions with interfering effects (see Exercise 14.6). It can be instantiated into a broad class of planning algorithms. For example, one may consider flaws in some particular order, either systematic and/or according to heuristics for guiding the search. One may work on several resolvers at a time by updating Ω with a *disjunction* of resolvers along the disjunctive-refinement framework. Finally, one should devise efficient ways to handle the various types of constraints involved in the TPS procedure. For example, the handling of threats may benefit from a forward-checking type of constraint management (see Figure 8.3). That is, at each time a resolver $c \in C_i$ is added to $\mathcal{C}$, the corresponding constraints are propagated and the remaining C_j in $\mathcal{K}$ are reduced to conditions that are consistent with Φ. If a C_j is reduced to an empty set, then this is a backtrack point; if a C_j is reduced to a single condition, then this condition is further propagated.

We will present such algorithms in Section 14.3, within a different and somewhat simpler representation.

14.3 Planning with Chronicles

This section presents another approach to temporal planning that also follows the time-oriented view of a dynamic system. We will rely on the state-variable representation, in which a domain is described by a set of state variables, each being a function of time. This representation extends the state-variable representation introduced earlier for classical planning (Sections 2.5 and 8.3).

The chronicle approach differs from the temporal database approach mainly because of the use of the state-variable representation, which offers some advantages in expressiveness and conciseness. For example, an axiom such as "an object is at a single place at one time," as in Example 14.4, is no longer needed because of the use of state variables, i.e., functions of time instead of relations. Another difference with respect to Section 14.2 is the lack of an explicit separation between preconditions and effects in operators: a condition and an effect of an action can be represented through a single expression. But here, as in the previous section, we will use the *PA* symbolic calculus without instantiating temporal variables into numerical values.

14.3.1 State Variables, Timelines, and Chronicles

Recall that some relations of a planning domain are *rigid* and invariant in time, while others are *flexible* and vary over time. Because time is now an explicit variable of the representation, it can be convenient to rely on the usual mathematical representation for handling a variation that is dependent on some variable, i.e., as a function of that variable. The representation of this section relies on a finite set of *state variables*, each being a function from time into some finite domain.

Example 14.7 In the DWR domain, the position of a container c1 ranges over the union of the sets of piles, cranes, and robots. Similarly, the location of a robot r1 ranges over the set of locations (for simplicity we assume that the constant routes belongs to this set). These two state variables can be described by the two function symbols

position-c1 : time $\rightarrow$ piles $\cup$ cranes $\cup$ robots and
location-r1 : time $\rightarrow$ locations,

where time is the interpretation domain for temporal variables, taken here to be the set of real numbers $\Re$.

It can be convenient to use function symbols of more than one argument, i.e., of object variables and time. In order to refer to any container or to any robot in the domain, we can have

cpos : containers $\times$ time $\rightarrow$ piles $\cup$ cranes $\cup$ robots and
rloc : robots $\times$ time $\rightarrow$ locations.

Hence rloc(r), for $r \in$ robots, refers to the finite set of state variables that define the locations of the robots in the domain. Note that function symbols of more than one argument are only a syntactic facility of the representation because we are assuming that all of the object variables have finite domains.

■

Formally, let D be the set of all constant symbols in a planning domain partitioned into various *classes* of constants, such as the classes of robots, locations, cranes, and containers. The representation describes a domain with a finite set of n-ary state variables where each $x(v_1, \ldots, v_n)$ denotes an element of a function

$$x : D_1^x \times \ldots \times D_n^x \times \text{time} \rightarrow D_{n+1}^x,$$

where each $D_i^x \subseteq D$ is the union of one or more classes. All state variables are functions of time. Unless needed, the $(n+1)$th argument of an n-ary state variable will be left implicit.

These state variables will be used as *partially specified functions*. We do not need a complete knowledge of the value of each x for each time point. Planning will proceed from a partial specification by further refinement, i.e., by augmenting and constraining that specification with some change and persistence conditions over the state variables in order to achieve the objectives. In particular, there can be "holes" in the predicted evolution, i.e., time periods during which the value of some

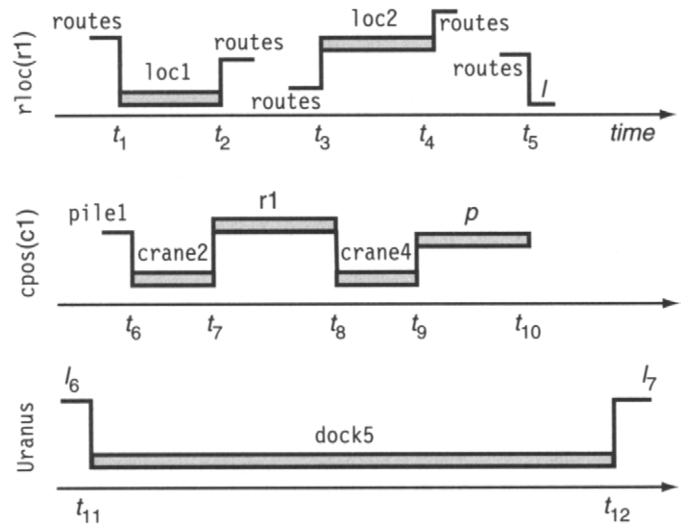

Figure 14.5 Three state variables as partial functions of time. (Persistent values are drawn as dashed rectangles while changes of values are represented as steps.)

x_i is unknown because it is not strictly needed for the plan. A state variable may have an imprecise or incomplete specification, i.e., $x(t) = v$, where v is an object variable whose value is not known but only constrained with respect to other object variables, and where the time point t is constrained with respect to other instants.

Example 14.8 Consider a DWR domain involving, among other objects, a robot r1, a container c1, and a ship Uranus, as well as several locations and cranes. A scenario partially specifying the whereabouts of r1, c1, and Uranus is represented with three state variables (Figure 14.5).

1. The first function says that r1 is in location loc1 at time t_1. It leaves loc1 at time t_2, enters loc2 at t_3, leaves loc2 at time t_4, and at t_5 enters location l.
2. The second function states that container c1 is in pile1 until t_6. It is held by crane2 from t_6 to t_7. Then it sits on r1 until t_8, at which point it is taken by crane4, which puts it on p at t_9, where it stays at least until t_{10}.
3. The last function says that the ship Uranus stays at location dock5 from t_{11} until t_{12}.

The object variable l is of type locations; it can be constrained but may remain unspecified even in a final plan. Similarly, the object variable p can be any pile in some location, e.g., if the goal is to have c1 anywhere in that location.

The temporal variables $t_1, \ldots, t_{12}$ are time points used to specify constraints. For example, the following constraints specify that the crane can start picking up a container when the destination robot enters the location: $t_1 \leq t_6$, $t_7 < t_2$, $t_3 \leq t_8$, and $t_9 < t_4$. Here, all the time points within a function are totally ordered, e.g., $t_1 < t_2 < t_3 < t_4 < t_5$, but the respective position of two time points in two different functions may remain unspecified, e.g., t_6 and t_{12} can be in any order.

There are several respects in which these functions are only partially specified. For example, the whereabouts of r1 are unknown from t_2 to t_3 and from t_4 to t_5. We know about the arrival of r1 to l at t_5, but nothing constrains it to stay there after t_5. The position of c1 is constrained to remain in p from t_9 at least until t_{10}, but it could stay there forever because the figure does not specify any change after this *persistence requirement.*

■

The three state variables in this example are *piecewise constant functions.* They can be specified with a set of temporal assertions using only two primitives, called *events* and *persistence conditions.* Both are primitives for describing the dynamics of a system due to actions or to predicted contingent events (e.g., the predicted arrival of Uranus, over which the planner has no control). These two primitives are defined as follows.

Definition 14.6 A *temporal assertion* on a state variable x is either an *event* or a *persistence condition* on x.

- An *event,* denoted as $x@t{:}(v_1, v_2)$, specifies the instantaneous *change* of the value of x from v_1 to v_2 at time t, with $v_1 \neq v_2$.
- A *persistence condition,* denoted as $x@[t_1, t_2){:}v$, specifies that the value of x *persists* as being equal to v over the interval $[t_1, t_2)$.

In this definition, t, t_1, and t_2 are constants or temporal variables, and v, v_1, and v_2 are constants or object variables in the domain of x.

■

As earlier, we have restricted intervals to be semi-open to avoid any confusion about the value of the state variable at a time t for an event $x@t{:}(v_1, v_2)$, or at the end of an interval for a persistence condition:

$$\begin{aligned}
x@[t_1, t_2){:}v \quad &\equiv \forall t\ (t_1 \leq t < t_2) \quad && \wedge\ (x(t) = v) \\
x@t{:}(v_1, v_2) \quad &\equiv \exists t_0\ \forall t'(t_0 < t' < t) \quad && \wedge\ (x(t') = v_1) \\
& && \wedge\ (x(t) = v_2)
\end{aligned}$$

These two types of assertions are related:

$$\begin{aligned}
x@t{:}(v_1, v_2) \quad &\equiv \exists t_1, t_2\ (t_1 < t < t_2) \quad && \wedge\ x@[t_1, t){:}v_1 \\
& && \wedge\ x@[t, t_2){:}v_2 \\
& && \wedge\ v_1 \neq v_2
\end{aligned}$$

The set of functions that can be described with the events and persistence conditions is limited to piecewise constant functions. This may seem very restricted as compared with the set of usual mathematical functions. Indeed, there are cases where one does need to reason about a continuous change from one value to another for a state variable ranging over a continuous domain. (The use of piecewise linear functions is discussed briefly in Section 14.4.) However, the restriction to piecewise constant functions, which is compatible with state variables ranging over finite domains, is often adequate given the abstraction level taken in planning where one leaves the precise values and the change of values to be taken care of at the execution level.

For example, the localization of a robot in a topological graph that has a finite set of locations and routes is not precise. One needs a more precise metrical localization for path planning and for controlling the robot along planned trajectories (see Chapter 20). However, a topological graph can be sufficient for planning the robot's mission. In that case it is reasonable to consider only discrete values for the state variable rloc(r1) and hence a piecewise constant function to describe its evolution; similarly for the state variable cpos(c1).

Definition 14.7 A *chronicle* for a set of state variables $\{x_i, \ldots, x_j\}$ is a pair $\Phi = (\mathcal{F}, \mathcal{C})$, where $\mathcal{F}$ is a set of temporal assertions, i.e., events and persistence conditions about the state variables $x_i, \ldots, x_j$, and $\mathcal{C}$ is a set of object constraints and temporal constraints. A *timeline* is a chronicle for a single state variable x. ■

Example 14.9 The timeline for the state variable rloc(r1) in Figure 14.5 is described by the following pair:

$$
\begin{array}{ll}
(\{ & \mathsf{rloc(r1)}@t_1 : (l_1, \mathsf{loc1}),\\
& \mathsf{rloc(r1)}@[t_1, t_2) : \mathsf{loc1},\\
& \mathsf{rloc(r1)}@t_2 : (\mathsf{loc1}, l_2),\\
& \mathsf{rloc(r1)}@t_3 : (l_3, \mathsf{loc2}),\\
& \mathsf{rloc(r1)}@[t_3, t_4) : \mathsf{loc2},\\
& \mathsf{rloc(r1)}@t_4 : (\mathsf{loc2}, l_4),\\
& \mathsf{rloc(r1)}@t_5 : (l_5, \mathsf{loc3})\ \},\\
\{ & \mathsf{adjacent}(l_1, \mathsf{loc1}), \mathsf{adjacent}(\mathsf{loc1}, l_2),\\
& \mathsf{adjacent}(l_3, \mathsf{loc2}), \mathsf{adjacent}(\mathsf{loc2}, l_4), \mathsf{adjacent}(l_5, \mathsf{loc3}),\\
& t_1 < t_2 < t_3 < t_4 < t_5\ \})
\end{array}
$$

This timeline for the state variable rloc(r1) refers to constants (loc1, loc2, etc.), object variables (l_1, l_2, etc.), and temporal variables (t_1, t_2, etc.). ■

A chronicle Φ can be seen as composed of several timelines, one for each state variable, and several constraints. It describes through these timelines and the constraints C how the world changes over time. The chronicle is interpreted as a *conjunction* of temporal assertions and temporal and object constraints. The object constraints in a chronicle are restricted to unary or binary constraints over finite domains on object variables. These are constraints of the form $x \in D$, $x = y$, $x \neq y$, conditional expressions such as "if $x \in D_1$ then $y \in D_2$," and pairs of allowed values for two variables, e.g., as defined by a rigid relation. The temporal constraints are here those of the point algebra *PA*. Hence, we will not instantiate temporal variables into numerical values. Other temporal constraint formalisms can be used in the chronicle representation; in particular, the use of TCSPs and the STPs will be discussed in Section 14.3.4.

A timeline Φ for a state variable x is used as a partial specification of a function that gives the value of x over time. Several such functions may correspond to Φ. Intuitively, a timeline Φ is *consistent* when the set of functions specified by Φ is not empty. This will be the case if no pair of assertions in the timeline conflicts, i.e., no specification leaves two possibly distinct values of the same state variable at the same time. We can avoid such a conflict by requiring a timeline to contain, either explicitly or implicitly, *separation constraints* that make each pair of assertions nonconflicting.

Definition 14.8 A timeline $\Phi = (\mathcal{F}, C)$ for the state variable x is *consistent* iff C is consistent and every pair of assertions in Φ is either disjoint or refers to the same value and/or the same time points.

- If two persistence conditions $x@[t_1, t_2){:}v_1$ and $x@[t_3, t_4){:}v_2$ are in $\mathcal{F}$, then one of the following sets of separation constraints is entailed by C: $\{t_2 \leq t_3\}$, $\{t_4 \leq t_1\}$, or $\{v_1 = v_2\}$.
- If an event $x@t{:}(v_1, v_2)$ and persistence condition $x@[t_1, t_2){:}v$ are in $\mathcal{F}$, then one of the following sets of separation constraints is entailed by C: $\{t < t_1\}$, $\{t_2 < t\}$, $\{t_1 = t, v = v_2\}$, or $\{t_2 = t, v = v_1\}$.
- If two events $x@t{:}(v_1, v_2)$ and $x@t'{:}(v'_1, v'_2)$ are in $\mathcal{F}$, then one of the following sets of separation constraints is entailed by C: $\{t \neq t'\}$ or $\{v_1 = v'_1, v_2 = v'_2\}$.

A chronicle Φ is consistent iff the timelines for all the state variables in Φ are consistent.

■

Recall that an n-ary state variable, e.g., rloc(r), is also a function of object variables. Consequently, two assertions on the same n-ary state variable in a chronicle may also be made disjoint by requiring their arguments to be distinct. For example, two events on rloc(r) and rloc(r') are made disjoint with the constraint $r \neq r'$. This additional constraint also has to be taken into account among the possible separation constraints for maintaining the consistency of a chronicle.

Note that the notion of a consistent chronicle is different from the usual notion of a consistent set of constraints. This definition requires a set of separation constraints to be entailed by C, i.e., to be in C either explicitly or implicitly through other constraints. This is stronger than requiring only the possible consistency of a set of separation constraints with $\mathcal{C}$, which would make Φ only *possibly* consistent. Definition 14.8 requires $\mathcal{C}$ to be *necessarily* consistent with the separation constraints.

Example 14.10 Let $\mathcal{F}$ be the three timelines of Figure 14.5 and $\mathcal{C} = \{t_1 \leq t_6, t_7 < t_2, t_3 \leq t_8, t_9 < t_4, \text{attached}(p, \text{loc2})\}$. This last constraint means that pile p belongs to location loc2. The pair $(\mathcal{F}, \mathcal{C})$ is a consistent chronicle because in each timeline all pairs of assertions are either disjoint or they refer to the same value at the same time point.

■

In Example 14.10, the time points in different timelines remain unordered with respect to each other, although within each timeline they are required to be totally ordered. This latter condition obviously simplifies checking the consistency of the chronicle, but even such a total order is not a sufficient consistency condition. For example, the chronicle $(\{x@[t', t){:}v, x@t{:}(v_1, v_2)\}, \emptyset)$ has its time points totally ordered but is not consistent because the constraint $(v = v_1)$ is not entailed from the chronicle.

Similarly to timelines, a chronicle is a partial specification for a collection of functions of time. We can refine such a specification by adding to a chronicle new temporal assertions and constraints, while keeping it consistent.

In order to do that precisely, let us define union and inclusion operations on chronicles. Because a chronicle is a pair of sets $(\mathcal{F}, \mathcal{C})$, the union of $\Phi = (\mathcal{F}, \mathcal{C})$ and $\Phi' = (\mathcal{F}', \mathcal{C}')$ is $\Phi \cup \Phi' = (\mathcal{F} \cup \mathcal{F}', \mathcal{C} \cup \mathcal{C}')$. The inclusion $\Phi \subseteq \Phi'$ holds iff $\mathcal{F} \subseteq \mathcal{F}'$ and $\mathcal{C} \subseteq \mathcal{C}'$.

We now define the notion of a chronicle *supporting* other chronicles, which is a notion quite different from that defined for temporal databases. Φ *supports* an assertion α when there is another assertion β in Φ that can be used as a causal support for α and when α can be added to Φ consistently. More precisely, when α asserts a value v or a change of value from v to some v' at time t for a state variable x, we require that β *establish* this value v for x at a time τ before t and that this value can *persist* consistently until t.

Definition 14.9 A consistent chronicle $\Phi = (\mathcal{F}, \mathcal{C})$ *supports* an assertion α (α being an event $\alpha = x@t{:}(v, v')$ or a persistence condition $\alpha = x@[t, t'){:}v$) iff there is in $\mathcal{F}$ an assertion β that asserts a value w for x (i.e., $\beta = x@\tau{:}(w', w)$ or $\beta = x@[\tau', \tau){:}w$) and there exists a set of separation constraints c such that $\Phi \cup (\{\alpha, x@[\tau, t){:}v\}, \{w = v, \tau < t\} \cup c)$ is a consistent chronicle.

■

This definition requires β to establish the value v of x, needed by α, at some time τ prior to t, and that the persistence of this value from τ to t and the assertion α itself, together with separation constraints, can be added consistently to Φ. Here, β

is called the *support* of α in Φ. The pair $\delta = (\{\alpha,\ x@[\tau, t){:}v\},\ \{w = v, \tau < t\} \cup c)$ is an *enabler* of α in Φ. Several remarks are in order:

- This definition requires Φ to be consistent before enabling α.
- The enabler δ is a chronicle: its assertions are α itself and a persistence condition for the value v; its constraints are the precedence and binding constraints from β to α, as well as the separation constraints c required to make the new assertions nonconflicting with those of Φ.
- For an event $\alpha = x@t{:}(v, v')$, we are requiring a support in Φ only for the value v, not for v', and only *before* t. This is because the support will be used as a *causal explanation* for α, to appear before α because time orders causality.[8] The change from v to v' will be explained by an action.
- There can be several ways to enable an assertion α in Φ: the support β and the separation constraints c are not unique, hence the enabler is not unique.

Example 14.11 The chronicle Φ of Example 14.10 supports in two different ways the event α = rloc(r1)@t:(routes,loc3) with the following two supports:

$$\beta = \text{rloc(r1)@t}_2 : (\text{loc1, routes})$$

$$\beta = \text{rloc(r1)@t}_4 : (\text{loc2, routes})$$

An enabler for this latter case is:

$$\delta = (\{\text{rloc(r1)@}[t_4, t) : \text{routes}, \text{rloc(r1)@}t : (\text{routes, loc3})\}, \{t_4 < t < t_5\}$$ ■

Let $\mathcal{E} = \{\alpha_1, \ldots, \alpha_k\}$ be a set of assertions. Assume that for each α_i in $\mathcal{E}$ there is a support β_i in $\mathcal{F} \cup \mathcal{E}$, $\beta_i \neq \alpha_i$, and an enabler δ_i. Let ϕ be the union of all these enablers: $\phi = \cup_i \delta_i$. Then we can create a new definition.

Definition 14.10 A consistent chronicle $\Phi = (\mathcal{F}, \mathcal{C})$ *supports* a set of assertions $\mathcal{E}$ iff each assertion α_i in $\mathcal{E}$ is supported by $(\mathcal{F} \cup \mathcal{E} - \{\alpha_i\}, \mathcal{C})$ with an enabler δ_i such that $\Phi \cup \phi$ is a consistent chronicle, where $\phi = \cup_i \delta_i$. ■

This definition allows an assertion $\alpha_i \in \mathcal{E}$ to support another assertion $\alpha_j \in \mathcal{E}$, $\alpha_j \neq \alpha_i$, with respect to Φ, as long as the union of the enablers is consistent with Φ. This union ϕ is called the *enabler* of $\mathcal{E}$ in Φ. Note that the assertions of ϕ include the set $\mathcal{E}$. Furthermore, ϕ is not unique because each α_i in $\mathcal{E}$ may have several supports and separation constraints with respect to Φ.

Example 14.12 The chronicle $\Phi = (\mathcal{F}, \mathcal{C})$ of Example 14.10 supports the pair of assertions $\{\alpha_1$ = rloc(r1)@t:(routes, loc3), α_2 = rloc(r1)@$[t', t'')$:loc3$\}$ in four different ways: with the two supports of Example 14.11 for α_1; and in each case α_2 can be

8. This is similar to causal links for preconditions in plan-space planning (Chapter 5).

supported either by α_1 or by the event in Φ: β = rloc(r1)@t_5:(routes,l), with the binding l=loc3. ■

Let $\Phi' = (\mathcal{F}', \mathcal{C}')$ be a chronicle such that Φ supports $\mathcal{F}'$, and let $\theta(\Phi'/\Phi)$ be the set of all possible enablers of $\mathcal{F}'$ in Φ augmented with the constraints $\mathcal{C}'$ of Φ', i.e., $\theta(\Phi'/\Phi) = \{\phi \cup (\emptyset, \mathcal{C}') \mid \phi \text{ is an enabler of } \mathcal{F}'\}$. $\theta(\Phi'/\Phi)$ is empty when Φ does not support $\mathcal{F}'$. Note that every element of $\theta(\Phi'/\Phi)$ is a chronicle that contains Φ'; it is also called an *enabler* for Φ'.

Definition 14.11 A consistent chronicle $\Phi = (\mathcal{F}, \mathcal{C})$ *supports* a chronicle $\Phi' = (\mathcal{F}', \mathcal{C}')$ iff Φ supports $\mathcal{F}'$ and there is an enabler $\phi \in \theta(\Phi'/\Phi)$ such that $\Phi \cup \phi$ is a consistent chronicle. Chronicle Φ *entails* Φ' iff it supports Φ' and there is an enabler $\phi \in \theta(\Phi'/\Phi)$ such that $\phi \subseteq \Phi$. ■

14.3.2 Chronicles as Planning Operators

We are now ready to use chronicles as building blocks for temporal planning.

Let $X = \{x_1, \ldots, x_n\}$ be a set of state variables. A chronicle planning operator on X is a pair $o = (\text{name}(o), (\mathcal{F}(o), \mathcal{C}(o))$, where:

- name(o) is a syntactic expression of the form $o(t_s, t_e, t_1, \ldots, v_1, v_2, \ldots)$, where o is an operator symbol, and $(t_s, t_e, t_1, \ldots, v_1, v_2, \ldots)$ are all the temporal and object variables in o.
- $(\mathcal{F}(o), \mathcal{C}(o))$ is a chronicle on the state variables of X.

Instead of such a pair, we will write operators as illustrated in the following example.

Example 14.13 A move operator for the DWR domain can be defined as follows (see Figure 14.6).

$$\begin{aligned} \mathsf{move}(t_s, t_e, t_1, t_2, r, l, l') = \{\ & \mathsf{rloc}(r)@t_s && : (l, \mathsf{routes}), \\ & \mathsf{rloc}(r)@[t_s, t_e) && : \mathsf{routes}, \\ & \mathsf{rloc}(r)@t_e && : (\mathsf{routes}, l'), \\ & \mathsf{contains}(l)@t_1 && : (r, \mathsf{empty}), \\ & \mathsf{contains}(l')@t_2 && : (\mathsf{empty}, r), \\ & t_s < t_1 < t_2 < t_e, \\ & \mathsf{adjacent}(l, l')\ \} \end{aligned}$$

Here, rloc : robots × time → locations ∪ {routes}, where routes refers to any route between locations; contains : locations × time → robots ∪ {empty}, where empty means a free location. Note that the object variables r and l in rloc and contains, respectively, are both arguments of a state variable and values of another state variable.

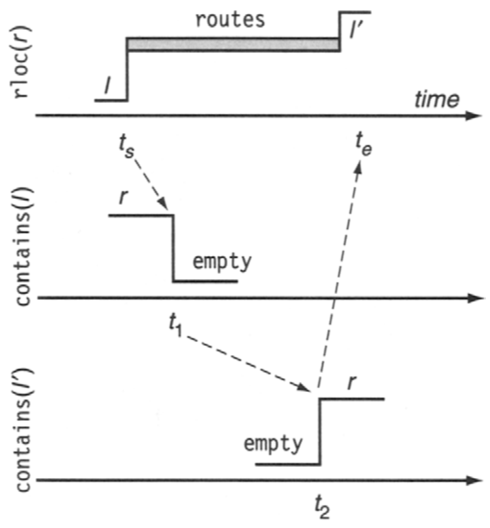

Figure 14.6 A planning operator as a chronicle. The figure depicts three timelines for the three state variables for this operator and temporal constraints on four time points. Note the the first timeline on rloc involves two events and a persistence condition.

These specifications are conservative in the sense that an origin location becomes empty *after* the robot leaves it ($t_s < t_1$), and a destination location ceases to be empty after the origin is free and *before* the robot enters it ($t_1 < t_2 < t_e$). ■

A chronicle planning operator o departs significantly from a classical planning operator.

- o does not specify preconditions and effects explicitly and separately. For example, contains$(l)@t_1 : (r, \text{empty})$ is both a precondition and an effect. It is a precondition for the value contains$(l) = r$, for which a support will be required for the action to be applicable. It is an effect for the value contains(l) = empty, which can be used as a support for other actions.
- o is applied not to a state but to a chronicle. The applicability of o is defined with respect to an entire chronicle.
- The result of applying an instance of o to a chronicle is not unique; it is a set of chronicles.

Note that all object and temporal variables in o are parameters of o existentially quantified with respect to a chronicle. As in Section 14.3.1, temporal variables will not be instantiated into numerical values.

An *action* $a = \sigma(o)$, for some substitution σ, is a partially instantiated planning operator o. Since an action a is a chronicle $a = (\mathcal{F}(a), \mathcal{C}(a))$, Definition 14.10 (see page 333) is used directly to specify when an action is applicable and what is its result.

An action a is *applicable* to Φ iff Φ supports the chronicle $(\mathcal{F}(a), \mathcal{C}(a))$. The result of applying a to Φ is not a unique chronicle but a set:

$$\gamma(\Phi, a) = \{\Phi \cup \phi \mid \phi \in \theta(a/\Phi)\}$$

An action a is a chronicle, $\theta(a/\Phi)$, as defined according to Definition 14.11. Hence, every element of $\gamma(\Phi, a)$ is a consistent chronicle.

A set $\pi = \{a_1, \ldots, a_n\}$ of actions is also a chronicle $\Phi_\pi = \bigcup_i(\mathcal{F}(a_i), \mathcal{C}(a_i))$. Again, a plan π will be defined as a set, not as a sequence. The general constructs of Section 14.3.1 enable us to specify directly the applicability and the result of a set π of overlapping and interacting actions.

Definition 14.12 A set $\pi = \{a_1, \ldots, a_n\}$ of actions is applicable to Φ iff Φ supports $\Phi_\pi = \bigcup_i(\mathcal{F}(a_i), \mathcal{C}(a_i))$. The result of π is the set of chronicles $\gamma(\Phi, \pi) = \{\Phi \cup \phi \mid \phi \in \theta(\Phi_\pi/\Phi)\}$.

■

It is important to underline that the definition of a chronicle supporting another chronicle allows some assertions of Φ_π to be supported by other assertions of Φ_π (see Definition 14.10 (see page 333)). Hence, we can handle concurrent actions with interfering effects.

14.3.3 Chronicle Planning Procedures

It is now easy to pursue an approach similar to that developed for temporal databases in order to define with chronicles temporal planning domains, problems, and solution plans.

A temporal planning *domain* is a pair $\mathcal{D} = (\Lambda_\Phi, \mathcal{O})$ where $\mathcal{O}$ is a set of chronicle planning operators, and Λ_Φ is the set of all chronicles that can be defined with assertions and constraints on the state variables of X using the constants and the temporal and object variables of the representation.

A temporal planning *problem* on $\mathcal{D}$ is a tuple $\mathcal{P} = (\mathcal{D}, \Phi_0, \Phi_g)$, where Φ_0 is a consistent chronicle that represents an initial scenario describing the rigid relations, the initial state of the domain, and the expected evolution that will take place independently of the actions to be planned, and Φ_g is a consistent chronicle that represents the goals of the problem. The *statement* of a problem $\mathcal{P}$ is given by $P = (\mathcal{O}, \Phi_0, \Phi_g)$.

A *solution plan* for a problem P is a set $\pi = \{a_1, \ldots, a_n\}$ of actions, each being an instance of an operator in $\mathcal{O}$, such that there is a chronicle in $\gamma(\Phi, \pi)$ that entails Φ_g. We will require a planner to find not only the set π but also a chronicle in $\gamma(\Phi, \pi)$ that entails Φ_g.

The approach to planning appears here simpler than in Section 14.2 because we have not introduced domain axioms. This is due to the functional representation and to the fact that every change and persistence effect of an action is specified explicitly in its description as a chronicle. We are going to present a planning procedure for chronicles that refines the TPS schema using CSP-based techniques and a search space similar to that of partial plans.

The TPS procedure of Figure 14.4 and the notations of Section 14.2.6 apply here with the following modifications: only two types of flaws have to be considered (open goals and threats) and enablers are here chronicles, not just constraints.

The resolution stage of the planning problem is now the tuple $\Omega = (\Phi, G, \mathcal{K}, \pi)$, where $\Phi = (\mathcal{F}, \mathcal{C})$ is the current chronicle, G is a set of assertions corresponding to the current open goals, $\mathcal{K} = \{C_1, \ldots, C_i\}$ is a set of pending sets of enablers, and π is the current plan.

For the problem $P = (\mathcal{O}, \Phi_0, \Phi_g)$, where $\Phi_0 = (\mathcal{F}_0, \mathcal{C}_0)$ and $\Phi_g = (\mathcal{G}, \mathcal{C}_g)$, we start initially with $\Phi = (\mathcal{F}_0, \mathcal{C}_0 \cup \mathcal{C}_g)$, $G = \mathcal{G}$, $\pi = \emptyset$, and $\mathcal{K} = \emptyset$. The current chronicle Φ is then refined until it has no flaw.

If P is successfully solved, then in the final stage $G = \mathcal{K} = \emptyset$, π is a set of actions that are instances of operators in $\mathcal{O}$, and $\Phi \in \gamma(\Phi_0, \pi)$ such that Φ entails Φ_g. Actions in the final plan π are partially instantiated and their time points are partially constrained.

The two types of flaws are processed as follows.

Open Goals. If $G \neq \emptyset$, then every assertion $\alpha \in G$ is a goal not yet supported by Φ.

- If $\theta(\alpha/\Phi) \neq \emptyset$, then there is in Φ a support for α. A resolver for this flaw is an enabler consistent with Φ. The refinement of Ω consist of the following updates:

$$\mathcal{K} \leftarrow \mathcal{K} \cup \{\theta(\alpha/\Phi)\}$$
$$G \leftarrow G - \{\alpha\}$$

- Otherwise, a resolver for the flaw is an action a that is relevant for this flaw, i.e., there is a support for α in $\mathcal{F}(a)$. The procedure chooses nondeterministically such an action and refines Ω with the following updates:

$$\pi \leftarrow \pi \cup \{a\}$$
$$\Phi \leftarrow \Phi \cup (\mathcal{F}(a), \mathcal{C}(a))$$
$$G \leftarrow G \cup \mathcal{F}(a)$$
$$\mathcal{K} \leftarrow \mathcal{K} \cup \{\theta(a/\Phi)\}$$

Notice that we are not requesting a to be supported by the current stage of Φ but only to be relevant for α. Hence, in the latter update of $\mathcal{K}$, $\theta(a/\Phi)$ is not necessarily completely defined at that stage; it remains as a set of pending consistency conditions.

```
CP($\Phi, G, \mathcal{K}, \pi$)
    if $G = \mathcal{K} = \emptyset$ then return($\pi$)
    perform the two following steps in any order
        if $G \neq \emptyset$ then do
            select any $\alpha \in G$
            if $\theta(\alpha/\Phi) \neq \emptyset$ then return(CP($\Phi, G - \{\alpha\}, \mathcal{K} \cup \theta(\alpha/\Phi), \pi$))
            else do
                relevant $\leftarrow \{a \mid a$ contains a support for $\alpha\}$
                if relevant $= \emptyset$ then return(failure)
                nondeterministically choose $a \in$ relevant
                return(CP($\Phi \cup (\mathcal{F}(a), \mathcal{C}(a)), G \cup \mathcal{F}(a), \mathcal{K} \cup \{\theta(a/\Phi)\}, \pi \cup \{a\}$))
        if $\mathcal{K} \neq \emptyset$ then do
            select any $C \in \mathcal{K}$
            threat-resolvers $\leftarrow \{\phi \in C \mid \phi$ consistent with $\Phi\}$
            if threat-resolvers $= \emptyset$ then return(failure)
            nondeterministically choose $\phi \in$ threat-resolvers
            return(CP($\Phi \cup \phi, G, \mathcal{K} - C, \pi$))
end
```

Figure 14.7 CP, a chronicle planning procedure.

Threats. If $\mathcal{K} \neq \emptyset$, then every $C \in \mathcal{K}$ is a pending set of enablers that are not yet entailed by the chronicle Φ. A resolver for this flaw is an enabler $\phi \in C$ that is consistent with Φ. The procedure chooses nondeterministically such an enabler if it exists and refines Ω with the following updates:

$$\mathcal{K} \leftarrow \mathcal{K} - \{C\}$$

$$\Phi \leftarrow \Phi \cup \phi \quad \text{for } \phi \in C \text{ consistent with } \Phi$$

Backtracking is needed when a flaw has no resolver. The corresponding procedure, called CP, for *Chronicle Planning*, is given in Figure 14.7. At each recursion, CP performs a termination test and addresses one particular flaw. Either it checks for open goals first, then deals with threats if there are no open goals, or it focuses on threats first.

For an implementation of CP, several steps in the above pseudocode can be refined.

- The choice of a relevant action for an open goal has to be heuristically guided but also restricted to actions that are consistent with the current stage of Φ.
- When an action a is added to the current plan, the update of G can be restricted to the sole assertions of $\mathcal{F}(a)$ that are not supported by a itself.

- The link between a support brought by an action and the assertion α can be performed immediately when a is added to the current π, enabling one to remove α from G.

In addition, the implementation of CP into a practical planner requires two important design steps: how to handle the constraints and how to control the search.

14.3.4 Constraint Management in CP

Two types of constraints are involved in this representation: temporal constraints and object constraints. Here, these two types of constraints are *decoupled*: no constraint in the representation introduced so far restricts the value of a time point as a function of object variables or vice versa. In other words, a set of constraints C in a chronicle $\Phi = (\mathcal{F}, C)$ is consistent iff the temporal constraints in C are consistent and the object constraints in C are consistent.[9] Consequently, constraint handling relies on two independent constraint managers for the two types of constraints.

The Time-Map Manager. Here we have to rely on the techniques developed in Chapter 13 in order to handle the temporal constraints. For simplicity we used the symbolic calculus. However, more expressive constraints would be needed in a practical planner. Simple temporal networks (the STPs of Section 13.4.1) allow for all of the *PA* constraints as well as for quantitative constraints. Minimal STP networks can be maintained, enabling the computation of queries in $O(1)$ and incremental updates in $O(n^2)$. The more expressive TCSPs with disjunctive quantitative constraints allow for a richer planning language but also for a disjunctive-refinement process. Although they are more complex to maintain, they may be a viable alternative if the control benefits from a disjunctive-refinement procedure.

However, the use of quantitative constraints and the instantiation of temporal variables into numerical values raises another important issue. According to the semantics of existentially quantified parameters of planning operators, the planner is free to choose any value for its temporal parameters that meets the specified constraints. One way to view this is to proceed with temporal variables as with object variables. For example, in the DWR operator take(k, l, c, d, p) (Example 2.8), variables c and d are existentially quantified, but because of the relation on(c, d), the container d is a function of c. Hence, choosing c fixes d.

If we are going to instantiate numerically the temporal variables according to such an approach, we would obviously need to specify additional knowledge in a temporal operator in order to predict precisely, according to the physics of the world, the duration of an action and the other temporal periods involved in it. In the move operator of Example 14.13 (see page 334), we would need to predict how long

9. Note that a consistent C does not guarantee a consistent Φ, as defined in Definition 14.8.

it takes to leave the initial location, to travel the routes, and to enter the destination location. The additional constraints would make t_e, t_1, and t_2 functions of t_s and other domain features, leaving the value of the starting point t_s as the only temporal choice point available to the planner among the set of possible instances.

Because a precise model of an action cannot be achieved, in practice one relies on conservative estimates that may provide for some robustness in the action models. These estimates are not easy to assess. Furthermore, conservative estimates for each action may make the planner fail to find a plan for a time-constrained goal even though such a goal is most often achievable.

A more reasonable approach would not need precise predictive knowledge but only prediction about possible intervals, leaving a margin for uncertainty. This approach requires the introduction of another category of temporal variables that cannot be determined and instantiated at planning time but can only be *observed* at execution time. These *contingent temporal variables* can be considered as random variables ranging in some intervals specified in the operator constraints. They are not existentially quantified parameters but universally quantified variables, distinct from free variables in the sense that whatever their observed values at execution time, within their allowed intervals of uncertainty the plan should meet all its constraints. This issue of contingent temporal variables and constraints was briefly discussed Section 13.5 and requires further extension to planning procedures and to the STP techniques for handling these contingent constraints and variables.

The Object Constraints Manager. This manager handles unary and binary constraints on object variables that come from binding and separation constraints it also handles rigid relations. Here we have to maintain a general CSP over finite domains, whose consistency checking is an NP-complete problem. There appears to be no useful restriction of the constraint language to a tractable CSP for object constraints. Indeed, the separation constraints of type $x \neq y$ make consistency checking NP-complete.[10] Filtering techniques, such as incremental arc consistency, are not complete. Nonetheless, they are very efficient, and they offer a reasonable trade-off for testing the consistency of object constraint networks. They may be used jointly with complete resolution algorithms, such as Forward-checking, at regular stages of the search with a risk of late backtracking.

In addition to the temporal and object constraints, other constraint managers can be useful if planning proceeds along a disjunctive-refinement scheme. The disjunction of resolvers for a flaw can be handled with a meta-CSP. Here a flaw is a variable whose domain is the set of resolvers of that flaw. The technique for managing this meta-CSP is similar to that illustrated (in Section 13.4.2) for TCSPs and in Section 8.6.1.

A Meta-CSP for Pending Enablers. Let us explain briefly how a meta-CSP can be used for handling the set $\mathcal{K}$ of pending sets of enablers. $\mathcal{K}$ will be associated with a network denoted $X_{\mathcal{K}}$, where each CSP variable x_C corresponds to a set of enablers

10. Graph coloring, an NP-complete problem, gives a good example of a CSP with such a constraint.

C in $\mathcal{K}$. At an update such as $\mathcal{K} \leftarrow \mathcal{K} \cup \{C\}$, if C is not yet known explicitly, it is left pending.[11] Otherwise, the consistency of each enabler $\phi \in C$ is checked with respect to the current Φ; ϕ is removed from C if ϕ is not consistent with Φ. At the end, one of the following occurs.

- If $C = \emptyset$, then this is a backtracking point.
- If $|C| = 1$, then there is a single enabler. Its constraints are directly added to the temporal network and to the object variable network, with the corresponding updates for $\mathcal{K}$ and Φ.
- Otherwise, the meta-CSP $X_{\mathcal{K}}$ is updated with a new variable x_C, whose domain is the set of remaining enablers in C.

Consistent tuples of values of the variables in $X_{\mathcal{K}}$ are conjunctions of enablers consistent with Φ. The management of the meta-CSP $X_{\mathcal{K}}$ may rely on incremental arc consistency, performed when a new variable is added to $X_{\mathcal{K}}$. If filtering leads to inconsistency, then backtracking is needed. If filtering reduces the domain of a variable to a single possible resolver, that variable is removed from the CSP network and the corresponding constraints are added to the temporal network and to the object variable network. Otherwise, inconsistent pairs of enablers are explicitly recorded in order to avoid using them in resolving future flaws.

The principle of managing disjunctions with a meta-CSP may be used at even a finer level for handling the set $\mathcal{K}$. Each enabler ϕ corresponds to a set of possible supports β_i for the assertions supported by ϕ and to separation constraints. Each such set of supports may have several sets of separation constraints. Hence it can be worthwhile to manage separately in two meta-CSPs the disjunctions over a set of supports and the disjunctions over separation constraints. The relations between the two are kept as binary constraints. Here also the consistency of each disjunction is checked separately with respect to the temporal and object networks. Inconsistent disjunctions are filtered out. Arc-consistency checking may further reduce disjunctions.

14.3.5 Search Control in CP

The control of the search proceeds here, as in plan space, along the principle of least commitment. This means that no action and no temporal or object constraint in enablers is added to the current Ω unless strictly needed to solve a flaw. This principle is very appealing because it involves no unnecessary nondeterministic choices that may lead to conflicts and to backtracking. Furthermore, least commitment leads to a final plan that remains as flexible as possible for execution, with partially instantiated and partially constrained actions.

11. This is the case for an action a not yet supported by the current Φ.

However, the least commitment principle alone is too general and insufficient to focus the search. It has to be combined with more informed and/or problem-specific strategies and heuristics for ordering flaws and choosing resolvers.

The principles for designing flaw-ordering heuristics are similar to those used in the resolution of CSPs for variable ordering. The idea is to start with the most constrained flaws that have the smallest number of resolvers in order to backtrack, if needed, as early as possible. If the quality of each resolver can be assessed, more complex flaw-selection heuristic functions that combine the number of resolvers and their quality are possible. These heuristics, however, require finding and assessing all resolvers for all flaws. They involve a significant overhead.

If disjunctive-refinement techniques are used for managing threats, we are left only with open goal flaws. Because a resolver for a threat does not introduce new open goals, a possible strategy is to solve all open goals, then to address threats with CSP resolution and filtering techniques. In this strategy, the search has two distinct phases: the open goal phase, while posting disjunctive constraints, and the threat resolution phase, which solves these constraints. This two-phases strategy is effective as long as there is no backtracking from the second phase back to the first. This can be assisted during the open goal phase with the filtering techniques for the disjunctive refinement in order to find inconsistent constraints as early as possible.

The resolution of open goals requires flaw-ordering heuristics and resolver-selection heuristics. A good compromise for flaw-ordering heuristics is to select the assertion α that correspond to the smallest set of relevant actions that have a support for α. Resolver-selection heuristics are the most important control element for the efficiency of the search. Good distance-based heuristics, as discussed in Chapter 9, rely on a trade-off between relaxation and regression.

The *relaxation principle* here is to ignore the consistency constraints between assertions of $\Phi \cup (\mathcal{F}(a), \mathcal{C}(a))$. Hence an action a is evaluated only with respect to what assertions are possibly supported by a, as compared to what new assertions a adds to the open goals. Furthermore, the binding and temporal constraints are not taken into account in this evaluation, whereas the role variable instantiations due to the partial instantiation of a with the current open goal of interest are.

The *regression principle* may rely on a heuristic evaluation of the cost of solving an open goal α defined as follows:

$$h(\alpha) = \min_{a \in relevant} \{\text{cost}(a) + \sum_{\alpha_i \in \mathcal{F}(a)} h(\alpha_i)\}$$

In this expression, the sum can be restricted to the sole assertions in $\mathcal{F}(a)$ that are not supported by a itself; cost(a) can be either a unit cost for all actions or a user-given value that reflects the cost of planning with a.[12] The regression may proceed along an a priori bounded AND/OR graph whose root is the assertion α. Or-branches correspond to actions relevant for α. From such an action, AND-branches are the assertions of a for which there is no support, neither in Φ nor in

12. This cost should not be confused with the cost of executing a within a plan.

the current AND/OR graph. At some maximal depth, the formula is computed with heuristics estimates for leaf assertions and regressed backward until the root. The action corresponding to the minimum is chosen for resolving the current flaw.

A final element for controlling the search may rely on a partial order on the state variables. Flaws are processed according to this order: no flaw is considered for the state variable x until all flaws for all state variables that precede x have been addressed. This scheme works as long as the resolution of flaws on x does not introduce new flaws for state variables that precede x. It is easy to synthesize automatically a partial order that guarantees this property for a particular chronicle planning problem. The techniques for this are similar to those used in classical planning for generating hierarchies with the ordered monotonic property [325] (see Section 24.6).

14.4 Discussion and Historical Remarks

As discussed in the introduction to Part IV, an explicit representation of time in planning extends the expressiveness of planning models. In particular, concurrent actions that are not independent and have interfering effects can be dealt with. Note, however, that this capability is limited to effects explicit in the action models: we have not addressed the much more general *ramification problem*, where additional effects are deduced from explicit effects and a general world model [463].

Temporal planning can be pursued either along the state-oriented view or along the time-oriented view. The early work on temporal planning, as well as most of the more recent temporal planners, adopt the state-oriented view. This can be explained mainly because this view is much closer to the classical planning paradigm, and thus it is easier to integrate with classical planning techniques and to benefit from any new progress in these techniques. The state-oriented approach was taken by several planners.

- In state-space planning, following on the early work on Deviser [528], several recent heuristically guided planners have been generalized to handle time, e.g., TLplan and following planners [29, 32], TALplanner [160, 342, 343], and the HS planner [261].
- In plan-space planning, the ZENO system [437] introduced a rich representation with variable durations and linear constraints.
- In the planning-graph approaches, planners such as TGP [489], SAPA [159], and TPSYS [208] handle durations and other temporal constructs.
- In HTN planning, several planners such as O-Plan [135], SIPE [549], and SHOP2 [416] integrate time windows and constraints in their representations and search processes.

Among these approaches, several of the recent planners have been shown to be quite efficient and to allow a reasonable scaling up (e.g., see the AIPS'02 planning competition [195]). However, most often these approaches lead to restricted models of concurrency of actions. The assumptions usually made are that an action has a duration but there is no possible reference to intermediate time points during this duration interval. Preconditions and effects are specified at the end points of the interval, with eventually invariant conditions that should prevail during the duration (along the model of Sandewall and Rönnquist [464]). These assumptions are discussed in the specification of the PDDL2.1 domain description language [196]. They lead to a weak notion of concurrency that basically requires independence between actions, as in layered plans. Interfering effects, as discussed in Section 14.2.5, need additional modeling efforts, e.g., adding specific actions and states, or constructs such as the processes and events proposed in PDDL+ [194].

Planning along the time-oriented view followed the pioneering work of Allen [18]. The example of a door with a spring lock[13] and similar interfering actions are discussed and correctly handled through the planner of [13, 17]. This planner is based on Allen's interval algebra (Section 13.3.2) for handling qualitative temporal constraints between propositions that are conditions and effects of actions, anywhere during their duration intervals. The planner does a plan-space search in which causal links are handled through *IA* constraints.

Another significant contribution to planning along the time-oriented view is the Time-Map Manager of Dean and McDermott [145] and several planners based on it [78, 143]. These planners handle what we presented here as temporal databases and implement several variants of the techniques described in Section 14.2.

Planning with chronicles started with the IxTeT system [223, 224], whose kernel is an efficient manager of the *PA* and STP temporal constraints [225]. This system seeks a compromise between efficiency and expressiveness for handling concurrency and interfering actions, external events, and goals situated in time. Distance-based heuristics were originally proposed within IxTeT, which also integrates several extensions to the abstraction techniques of plan-space planning [205]. However, this planner still remains noncompetitive with the most efficient state-based or HTN temporal planners.

Several other planners adopt variants of the chronicle-based approach. These are notably ParcPlan [170, 361], and RAP and its follower EUROPA [198, 285, 409]. We will come back to the latter planners in Chapter 19 because these planners enabled a major planning application. ParcPlan has a control strategy that handles open goals in a simplified manner, then it manages threats with a meta-CSP technique, along the lines of the two-stages control technique discussed earlier.

Several temporal planners are able to handle more complex temporal constraints than those of *IA*, *PA*, STP and TCSP (Chapter 13). For example, in the state-oriented view, ZENO handles linear constraints; the planner of Dimopoulos and Gerevini [155] relies on mixed linear and integer programming techniques. In the time-oriented view, even if most temporal planners rely basically on piecewise constant functions, generalization to piecewise linear functions appears feasible in many cases [512].

13. This example is known as the "University of Rochester CS Building door"; see Exercise 14.7.

The issue of robust estimates for temporal periods and other parameters is well known in scheduling and is being explored in planning (e.g., [139]). The problem of contingent temporal variables and constraints is fairly well addressed [403, 530, 531, 532]. But a good integration of the proposed techniques into planning procedures remain to be further explored, eventually within the wider context of CSPs with contingent variables along approaches such as [185, 542].

The two approaches presented in this chapter are very close and involve similar techniques. The temporal database representation has didactic merits because its operators are closer to the familiar classical planning operators and it allows an easier introduction to the main concepts. The chronicle-based approach benefits from the state-variable representation and the lack of domain axioms in conciseness and probably also efficiency.

14.5 Exercises

14.1 Modify the move operator of Figure 14.2 to take into account within this action the time it takes the robot to get out of its initial location, i.e., between t_s and the moment at which the robot leaves location l. Work out the consistency constraints with domain axioms of Example 14.5 (see page 318) for this modified operator.

14.2 Using the representation of Figure 14.2, define temporal planning operators for the actions load, unload, take, and put of the DWR domain.

14.3 Define domain axioms corresponding to the operators of Exercise 14.2.

14.4 Extend the database of Example 14.1 (see page 311) to include two containers, respectively in loc1 and loc2, and apply the operators and axioms of the previous exercises to achieve the goal of permuting the positions of these two containers. Develop two possible plans, one that assumes a domain with three adjacent locations, and one restricted to only two locations.

14.5 Consider a DWR domain where each robot is equipped with a container fixture that guides the loading of a container on the robot with a crane and that maintains the container in position while the robot is moving. This fixture remains closed; it can be opened by the robot, but it closes as soon as the opening action ceases. With such a system, a robot has to open its container fixture and maintain it open during the load and unload operations. Define an open-fixture operator and revise the load and unload operators to have the desired joint effects of loading and fixing a container, or freeing and unloading it.

14.6 In the TPS procedure (Section 14.2.6), analyze the processing of open goal flaws. Explain how this processing makes TPS consistent with the definition of an applicable set of concurrent actions with interfering effects.

14.7 Consider two operators open and unlock for handling the "University of Rochester CS Building door" domain: open turns the knob and pushes the door but requires

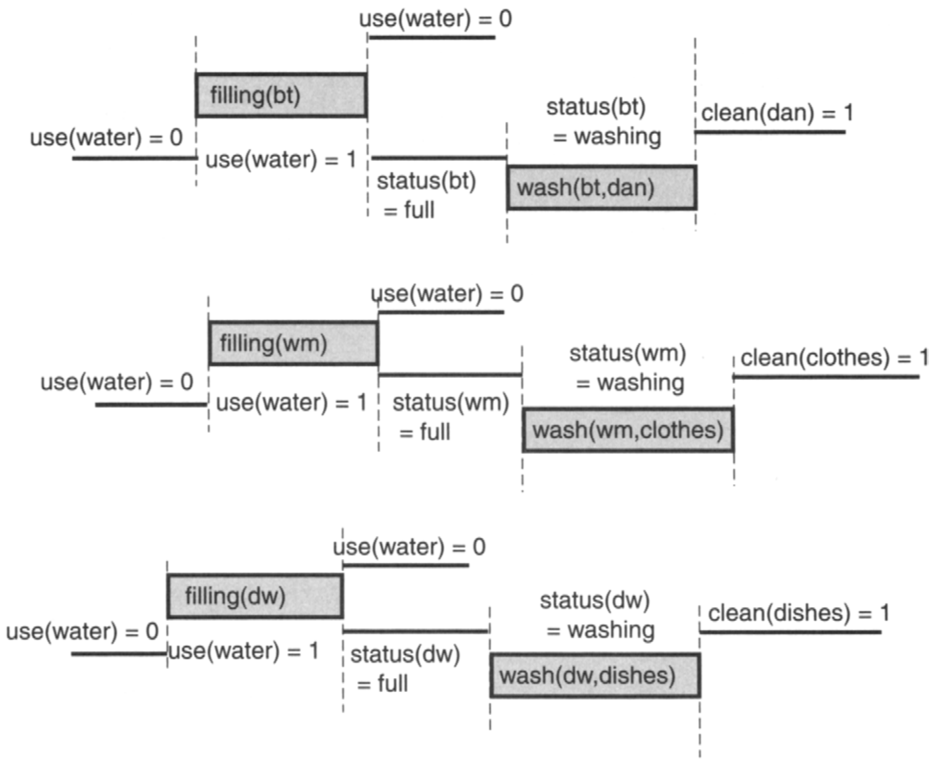

Figure 14.8 A chronicle for a washing problem.

the lock to be unlocked when turning; unlock pushes a spring and maintains it to keep the lock unlocked. Define these two operators in the temporal database approach and in the chronicle approach. Compare the two models.

14.8 Using the representation of Example 14.13 (see page 334), define four chronicle operators for the actions load, unload, take, and put of the DWR domain.

14.9 Extend the domain of Example 14.8 (see page 328) by considering that the ship Uranus contains two containers c2 and c3 that need to be unloaded, and the container c1 has to be loaded on Uranus while the ship is on dock. Assume that crane4 belongs to dock5. Find a plan for this problem.

14.10 Extend the chronicle planning representation to include numerical temporal constraints of the STP type. Discuss the required change in the CP procedure.

14.11 Here is a temporal planning adaptation of the planning problem described in Exercise 5.7. Dan wants to make a plan to wash his clothes with a washing machine wm, wash his dishes in a dishwasher dw, and bathe in a bathtub bt. The water supply doesn't have enough pressure to do more than one of these activities at once. Here are the operators for the planning domain:

fill(t_s, t_e, x)

status(x)@t_s	: (ready,filling)
status(x)@$[t_s, t_e)$	: (filling)
status(x)@t_e	: (filling,full)
use(water)@t_s	: (0,1)
use(water)@$[t_s, t_e)$	: 1
use(water)@t_e	: (1,0)
$t_s < t_e$	

wash(t_s, t_e, x, y)

status(x)@t_s	: (full,washing)
status(x)@$[t_s, t_e)$	: (washing)
status(x)@t_e	: (washing,ready)
clean(y)@t_e	: (0,1)
$t_s < t_e$	

(a) Write the initial chronicle for the planning problem.

(b) Figure 14.8 is a depiction of a chronicle at an intermediate point in the planning process. What are all of the flaws in this chronicle?

(c) Write the chronicle as a set of temporal assertions and a set of constraints.

(d) How many solution plans can be generated from the chronicle? Write one of them.

(e) How (if at all) would your answer to part (c) change if the planning domain also included operators to make everything dirty again?

14.12 Redo Exercise 14.11 using a temporal database representation.