Exploiting Temporal Knowledge to Organize Constraints

Stephen F. Smith

CMU-RI-TR-83-12

Intelligent Systems Laboratory
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

July 1983

This research was supported by the Air Force Office of Scientific Research under contract F49620-82-K0017.

Table of Contents

List of Figures

Abstract

This paper examines the role of temporal knowledge in regard to the representation and utilization of constraints within time dependent problem domains. We first consider the representation of constraints whose interpretations may vary in different temporal regions of the solution space. A representation that organizes constraint variants via the temporal relationships among them is presented and seen to support a simple mechanism for determining the applicable variant at any point in time. We then turn our attention to the management of temporal constraints that are dynamically imposed as various commitments are made by the reasoning system, e.g., as resources are allocated to activities in a plan. Constraint propagation techniques which collectively insure consistency in the hypotheses under development are presented and discussed. These techniques are driven by the temporal relationships present in the domain model. This work is motivated by ongoing research with ISIS, an intelligent scheduling and information system currently being applied to the problem of scheduling job shops, and examples throughout the paper are drawn from this domain.

1. Introduction

The use of constraints to direct problem solving activity is common within AI (e.g. [Goldstein 75, Kornfeld 81, Stefik 81, Sussman 80, Steels 81, Waltz 75]) but the manner in which they are employed can vary. Typically, they are utilized in a generative capacity, with constraint propagation serving to restrict the number of hypotheses generated. This strategy often yields a unique solution in domains where the satisfaction of constraints can be assumed. However, in complex domains, it is often not possible to satisfy all of the constraints present. Constraints must be selectively relaxed, and the strategy becomes one of identifying the hypothesis that best satisfies the constraints. In this case, constraints must additionally serve as a basis for discriminating among the alternative hypotheses generated. The dual role played by constraints in these latter domains raises some interesting issues with respect to the underlying constraint representation and organization.

The research reported here is concerned with these issues in problem domains where matters are further complicated by a need to reason about time. Specifically, we will consider a methodology for organizing and managing constraints based on the temporal relationships that are present between them. The work is motivated by research underway at the CMU Robotics Institute on an intelligent scheduling and information system (ISIS), currently being applied to the problem of scheduling job shops [Fox 82, Fox 83a]. Briefly, the problem is one of generating a schedule to govern the manufacture of products for which orders have been received. The schedule must carry each product order through an appropriate sequence of operations such that a set of organizational goals (e.g. meeting due dates, minimizing work in process time, maintaining production levels) are met. The operations associated with the manufacture of different items share a common set of resources (e.g. machines, tools, etc.), whose allocation is the essence of the scheduling problem. Casting this as a complex constraint satisfaction problem, our approach in ISIS has focused on the expression and utilization of the large variety of constraints that influence job shop schedules.

Two broad issues relating to constraint representation and organization arise in time dependent problem domains. First, many of the constraints maintained by the reasoning system are likely to be *time varying*. That is, distinct variants of the constraints may be applicable when the reasoning system is focusing on distinct intervals of time in the solution space. Such constraints may be imposed at the outset or result from the relaxation of an unsatisfiable variant. A constraint representation is required that promotes efficient resolution of the applicable variant at any point in time. A second issue involves the management of temporal constraints that are dynamically imposed as various commitments are made during the search process. These constraints must be propagated through the domain model to ensure consistency of the hypotheses under development. This paper examines the role of temporal knowledge in both of these contexts.

The remainder of the paper is organized as follows. Section 2 considers previous work in temporal representations. In Section 3, a set of modeling primitives are introduced to provide a basis for representing temporal knowledge. Sections 4 and 5 then consider the processes of constraint resolution and constraint propagation, respectively. In Section 6 the main ideas of the paper are summarized. Because of our experience with job shop scheduling, much of the discussion will center around this domain. The techniques under consideration, however, appear relevant to any time dependent problem domain.

2. Background

Traditionally, temporal considerations have not played a major role in the reasoning processes of problem solving and planning systems. Systems that operate in the blocks world, for example, focus on planning and executing solutions to problems without concern to the intervals of time encompassed by these solutions. Only an implicit notion of time, as embodied in the relationships between states in a given problem space, is present. In attempting to solve problems that place time restrictions on the completion of activities and achievement of goals, however, the inadequacy of this *instantaneous time slices* approach to modeling time becomes apparent. The ability to reason about time requires an explicit representation.

Despite a recognition of this need, the explicit treatment of time in problem solving/planning systems is fairly uncommon. The AUTOPILOT system [Thorndike 81], a special purpose, distributed planning system for guiding multiple aircraft through a common airspace, utilizes a specific notion of time in representing aircraft flight plans although this technique appears to have limited applicability. The NUDGE system [Goldstein 77] also takes a domain specific approach, utilizing a rich set of knowledge about the time requirements of various activities and the time preferences of specific individuals to produce a schedule of an individual's weekly activities and appointments. Vere has described a more general technique for planning within imposed time spans [Vere 81] that associates start time *windows* and *durations* with the various activities under consideration. This temporal information is refined and propagated to other activities in the plan as the plan crystalizes. A similar approach is taken in [Fukumori 80] in developing train schedules.

The general issue of representing and reasoning with temporal knowledge has also been considered [Bruce 72, Kahn 77, Allen 81a, Vilain 82], although primarily in the context of natural language comprehension and generation. These efforts focused on providing schemes for efficiently organizing a body of temporal knowledge and deductive methods that exploit these representations in responding to queries. Some recent proposals [Allen 81b, McDermott 82] have attempted to place these temporal models within a larger framework in which plans and actions can be expressed. Many of the ideas that have emerged from these studies are directly applicable to the problems associated with representing and manipulating constraints in time constrained domains, and we will draw freely on this work below.

3. A representational framework for modeling time

Before proceeding with the issue of reasoning with a time varying set of constraints, it is necessary to define an underlying framework for modeling time. In this section, we present a set of application independent primitives that will provide us with a basis for representing temporal knowledge within particular problem domains. The primitives are expressed in SRL, a schema (or frame) based knowledge representation language [Fox 79]. Lack of space prohibits an in depth discussion of the language itself and the reader is referred to [Wright 83] for a complete description. An overview of the SRL schema construct is contained in Appendix I.

Generally speaking, there are two forms of temporal knowledge to which the underlying primitives must attend:

- absolute temporal knowledge - knowledge that is explicitly linked with a particular period of time along some time coordinate system (e.g. machine *x* has been reserved for the milling operation from June 1, 1982 through June 15, 1982)

- relative temporal knowledge - knowledge that relates temporal objects without referencing any specific periods of time (e.g. the milling operation precedes the drilling operation)

The representation of both forms is addressed, in turn, in the following two subsections.

3.1. Primitive temporal objects

Defining an underlying temporal representation establishes a specific perspective through which the problem solver may view and reason about time. Problem solving systems that have explicitly dealt with time have typically employed a single representation, and, hence, a single perspective of time. However, for the applications we have in mind, such a single perspective is inadequate. In the job-shop scheduling domain. for example, resources are allocated along a calendar, work weeks are defined in terms of days of the week, and work shifts are expressed as occurring over hours in the day. Each of these perspectives requires a distinct underlying representation. Thus, rather than restricting ourselves to a single representation, we have chosen a representational framework that allows the creation of multiple temporal representations.

A particular temporal representation is specified by instantiating the **time-line** schema shown in Figure 3-1. As can be seen, this schema packages the complete specifications of a given time coordinate system. More specifically, this information defines a fully ordered set of time points, the granularity of the time line, and a set of operations for manipulating temporal knowledge bound to this time line. The operations include arithmetic functions, conversion functions, and functions for testing and deriving the relationships between temporal objects.

```
{{time-line
    POINT-FORM:
    START-POINT:
    END-POINT:
    GRANULARITY:
    ADD:
    DIFF:
    BEFOREP:
    AFTERP:
     . . . }}
```

Figure 3-1: time-line schema

The role played by **time-line** schemata will become clear as the remainder of the representational framework is presented. For now, consider the **calendar** schema in Figure 3-2 which defines a temporal representation utilized in the job shop scheduling domain. A time point consists of an ordered triple (week day hour) where weeks are indexed from some base week.

```
{{calendar
    {INSTANCE time-line
        POINT-FORM: (LIST (sexp (lambda (x) (and (fixp(x)) (not (lessp x 0)))))
                          (sexp (lambda (x) (and (fixp(x)) (not (lessp x 0)) (not (greaterp (x 6))))))
                          (sexp (lambda (x) (and (fixp(x)) (not (lessp x 0)) (not (greaterp (x 23)))))))
        START-POINT: (0 0 0)
        END-POINT: (99999 0 0)
        GRANULARITY: (0 0 1)
        ADD:
          . . . } }}
```

Figure 3-2: **calendar** schema

Given that most activities/events are best described as occurring over an interval of time, the basic temporal object provided in the representational framework is the **interval** schema. This is defined in Figure 3-3.

```
{{interval
    START-TIME:
    END-TIME:
    DURATION:
    DATED-BY: }}
```

Figure 3-3: **interval** schema

By associating a specific **time-line** schema we can differentiate between sub-classes of intervals, as, for example, in Figure 3-4.

```
{{calendar-interval
    {IS-A interval
        DATED-BY: calendar } }}
```

Figure 3-4: **calendar-interval** schema

Specific instances of the **interval** schema may be expressed in different ways, depending on the type of temporal information that is available. An interval may be associated with a particular period of time, in which case the START-TIME and END-TIME slots contain time-points delineating the endpoints of the interval. Alternatively, the DURATION slot may be used to indicate the span of an interval without necessarily binding it to a particular period of time. Note that, in either case, an interval is bound to a particular time-line (i.e. it is defined with respect to a particular temporal perspective).

As suggested above, the association of a specific time line to a class of points and/or intervals establishes a specific perspective of time, and is meant to govern the representation and manipulation of temporal objects bound to this perspective. This is accomplished by the **dated-by** relation which allows for the passage of information from the time line to the time points and/or intervals to which it is associated. To invoke a particular temporal function, for example, the value of the slot corresponding to the function name is inherited and applied to the arguments. The **dated-by** schema is depicted in Figure 3-5.[1]

```
{{dated-by
   {IS-A relation
      DOMAIN: (TYPE INSTANCE interval)
      RANGE: (TYPE INSTANCE time-line)
      INCLUSIONS: {{INSTANCE inclusion-spec
                        DOMAIN: all
                        RANGE: all
                        TYPE: value
                        SLOT: all
                        VALUE: all
                        CONDITION: t }}
      INVERSE: dates  } }}
```

Figure 3-5: dated-by schema

3.2. Primitive temporal relations

Much of the temporal knowledge required by a reasoning system is relative in nature, indicating temporal relationships between activities or events without designating specific intervals on an underlying time line. Such knowledge is characterized through the use of a predefined set of temporal relations. We will see in Section 4 that these relations can also be useful in situations where absolute temporal information is available. The following relations,taken from [Allen 81a], have been incorporated as basic primitives:

- e_1 **before** e_2 -- event e_1 occurs before event e_2 with some intervening interval of time
- e_1 **after** e_2 -- e_1 occurs after e_2 with some intervening interval of time (inverse of **before**)
- e_1 **during** e_2 -- e_1 occurs during e_2
- e_1 **contains** e_2 -- e_2 occurs during e_1
- e_1 **meets** e_2 -- e_1 and e_2 occur consecutively in time with e_1 preceding e_2
- e_1 **met-by** e_2 -- e_1 and e_2 occur consecutively in time with e_2 preceding e_1

[1]Within SRL, all slots are viewed as relations. Most relations (slots) do not allow inheritance (as this is the default) but the model builder does have the ability to associate inheritance specifications with a relation. In this case, the INCLUSIONS slot is used to specify that all slots and values of the RANGE schema may be inherited through the **dated-by** relation by the DOMAIN schema. See [Wright 83] for details.

- e_1 **overlaps** e_2 -- e_2 begins prior to the termination of e_1
- e_1 **overlapped-by** e_2 -- e_1 begins prior to the termination of e_2
- e_1 **equal** e_2 -- e_1 and e_2 occupy the same interval of time

The **during** schema shown in Figure 3-6 typifies the form of the above relations. The class name **event** appearing in the DOMAIN slot of the schema is used loosely to represent any entity that occurs within or occupies a specific interval of time (i.e. it has an **interval** schema associated with it). The restriction imposed on the RANGE of the relation specifies that the intervals associated with the two schemata being related must be bound to the same time line.

```
{{during
   {IS-A relation
      DOMAIN: (TYPE INSTANCE event)
      RANGE: (EQ DOMAIN)
      INVERSE: contains } }}
```

Figure 3-6: during schema

Note that the issue of inheritance is not addressed in specifying the primitive temporal relations. Such properties depend on the context in which a given relation is employed. Inheritance may be desirable, for example, in situations where the relation is used as a knowledge structuring mechanism (see Section 4). On the other hand, inheritance is unnecessary (in fact undesirable) if the relation is used to express a partial ordering among activities in a plan (see Section 5). Thus, the inheritance properties associated with a given temporal relation are left to be specified when it is specialized for a particular application.

4. Temporal knowledge as a basis for constraint resolution

As indicated at the outset, an integral step in the constraint-directed reasoning control cycle is constraint resolution, the process of identifying the set of constraints relevant to the current situation. If the overall set of constraints maintained by the system is large and varied, an organizational framework in which to embed the constraints becomes increasingly important to the ease with which this resolution process can be carried out. Given an object oriented knowledge representation such as SRL, *constraint residency* is a useful and simple basis for determining relevancy. Constraints (represented as schemata) are directly attached to the objects they constrain, and the relevant constraints at any point are collected by examining the objects associated with the current state. Constraint resolution is complicated, however, in the case of *time varying* constraints. There might be several variants of a given constraint associated with a particular object, each applicable during a different interval of time. In these situations it is necessary to impose further organizational structure on the constraint representation.

The following subsections focus on the expression of time varying constraints. We will examine a representational technique that exploits the use of temporal relations to provide an organization whereby the relevant variant at any point in time can be easily identified.[2]

4.1. Time varying constraints

Within the job shop scheduling domain, there are many constraints that vary over time with respect to the value that the constrained attribute must satisfy. Consider the manpower levels that are set in various work areas of the shop. At any point in time these levels are fixed, and any schedule that is generated for the shop must adhere to the specified levels. Nonetheless, particular settings for manpower levels are only applicable for a limited period of time. They are adjusted over time as the production requirements in the shop change. Furthermore, the possible adjustments that may be made to manpower levels are bounded by the maximum and minimum number of workers that can be assigned to a given work area, and each individual setting within this range carries its own degree of desirability.

This example illustrates the kinds of information that must be captured in a representation of time varying constraints. However, given that such constraints may be altered periodically, a primary requirement of the representation is a means of structuring the set of alterations currently in force. Temporal relations provide a natural basis for this organization and lead to the definition of the **time-varying-constraint** schema depicted in Figure 4-1.

```
{{time-varying-constraint
    {IS-A constraint
        TEMPORAL-SCOPE:
            range: (TYPE INSTANCE interval)
        VALUE:
        UTILITY:
        ALTERNATIVES:
            range: (TYPE INSTANCE relaxation-spec)
        CURRENT-ALTERATIONS:
            range: (SET (TYPE INSTANCE time-varying-constraint))
            comment: list of alterations currently in force for this constraint.
        ALTERATION-OF:
            range: (TYPE INSTANCE time-varying-constraint)
            comment: the constraint for which this constraint is a alteration } }}
```

Figure 4-1: time-varying-constraint schema

Each instance of a time varying constraint has associated with it a specific TEMPORAL-SCOPE (i.e. an interval of time) which delineates its period of applicability. Assuming no alterations have been made to the constraint, it is applicable at any point in time within this temporal scope. In this case, VALUE

[2] The fundamental issue of categorizing and representing the large variety of constraints that may be found in complex domains has been investigated extensively in [Fox 83a], resulting in a taxonomy of constraint types and associated representations. To simplify matters, we will limit our discussion to constraints that specify a single value to be satisfied, and develop the notion of a time varying constraint type for this class of constraints. Within ISIS, the temporal organization to be detailed below is instead integrated into the specification of each constraint type identified in [Fox 83a].

contains the current value of the constraint and UTILITY its associated utility. The constraint also possesses a set of ALTERNATIVES, specified by means of an attached *relaxation spec*. These alternatives enumerate the possible alterations that can be made to the constraint, and are instantiated (i.e. a temporal scope is associated) to produce specific alterations.[3]

If a decision is made to alter a time varying constraint, the temporal scope associated with the alteration will necessarily encompass a subinterval of the original constraint's temporal scope. As such, there is an inherent temporal relationship linking a constraint to its alterations (*contains*), and, likewise, an inverse relationship (*during*) in the other direction. These relationships are exploited for organizational purposes in defining the CURRENT-ALTERATIONS and ALTERATION-OF relations found in the **time-varying-constraint** schema. To make this explicit, consider the complete definition of the ALTERATION-OF relation in Figure 4-2.

```
{{alteration-of
   {IS-A during
      DOMAIN: (SET (TYPE INSTANCE time-varying-constraint))
      RANGE: (TYPE INSTANCE time-varying-constraint)
      INCLUSIONS: {{INSTANCE inclusion-spec
                        DOMAIN: all
                        RANGE: all
                        TYPE: value
                        SLOT: all
                        VALUE: all
                        CONDITION: t }}
      INVERSE: current-alterations } }}
```

Figure 4-2: alteration-of schema

We are now in a position to examine the dynamics of the representation. Ignoring the issue of when to relax or strengthen a time varying constraint, suppose that at some point a decision is made to alter the constraint along the lines of one of its specified alternatives, and an appropriate temporal scope for the alteration has been determined. Since an alteration is itself a time varying constraint, it is instantiated as such and attached to the original constraint's CURRENT-ALTERATIONS slot. This newly attached constraint now takes precedence over the original constraint at points in time within its temporal scope, while the original constraint is still applicable at points in time within its temporal scope but outside that of its alterations. Note that an alteration may be subsequently altered itself in the same manner. Thus, each instance of a time varying constraint instance sits at the root of a tree structure containing its current alterations. The resolution process defined over a given *alteration*

[3]We can approach the specification of alternatives from different perspectives. If the constraint directed reasoning process proceeds in a one way direction, attempting to satisfy the relevant constraints and relaxing those that cannot be satisfied, then the set of alternatives associated with a given constraint need only include the subset of possible values whose associated utilities are less than the constraint's utility (i.e. the possible relaxations of the constraint). On the other hand, if the reasoning process is also capable of strengthening a constraint (necessitated, perhaps, by the inappropriateness of a previous relaxation that was made) then it is necessary to include all possible alternatives. We have adopted this more general view and take the alternatives of a given time varying constraint to include means for both relaxing and strengthening the constraint. The name *relaxation spec* is retained to maintain consistency with the terminology in [Fox 83a].

tree is straightforward. To locate the currently applicable constraint at time *t*, we find the lowest constraint in the tree whose temporal scope contains *t*.[4]

As is the case with all constraints, time varying constraints can be associated with the objects they constrain through attachment to the appropriate schemata. Time varying constraints specifying *a priori* determined work shifts, for example, might be placed in the schemata representing the specific machines in a job shop. More generally, it is useful to place constraints at various levels within the hierarchical model of the environment, and rely on inheritance relations to import the constraints relevant to objects at lower levels. This allows the association of a *default* constraint with a class of objects which can be overridden by the attachment of more specific constraints to individual members of the class. However, this technique falls short in the case of time varying constraints. Because of their distinct periods of applicability, it may be necessary to inherit a constraint from a higher level even though constraints reside at the current level.

Given the inadequacy of inheritance relations in this context, it is necessary to extend the representation to explicitly identify the potentially applicable constraints residing at higher levels in the hierarchy. This is accomplished through the introduction of the **time-varying-constraint-root** schema depicted in Figure 4-3.

```
{{time-varying-constraint-root
    {IS-A time-varying-constraint
            TEMPORAL-SCOPE: always }
    SPECIALIZATION-OF:
        range: (TYPE INSTANCE time-varying-constraint-root) }}
```

Figure 4-3: time-varying-constraint-root schema

Within the extended representation, an instance of the **time-varying-constraint-root** schema is attached to the object being constrained in lieu of the actual set of associated time varying constraints. The actual constraints, in turn, are placed in the CURRENT-ALTERATIONS slot of the root, combining the constraints and their associated alterations into an alteration tree.[5] The resolution process proceeds by first examining the alteration tree designated by the constraint root in the

[4]A second temporal organization we might have adopted here is that of maintaining a strict *relaxation tree* where in each node is a relaxation of its parent. In this case, a decision to strengthen a given constraint would be accommodated by restructuring the relaxation tree to which it belongs (e.g. pruning one or more constraints from the tree or reducing their temporal scope so that the desired stronger constraint residing at a higher level in the tree becomes applicable, or splitting the constraint into two and inserting the desired constraint between the two resulting temporal scopes at the same level in the tree). This would likely result in a more compact representation, as the height of each tree would be bounded by the number of alternatives at its root. However, the adopted strategy of growing a new leaf in the tree for each alteration (be it a relaxation or a strengthening of the constraint) provides a complete record of the sequences of alterations made to each time varying constraint in the system. This provides the meta-level reasoning system with a basis for incorporating knowledge of its past decisions when contemplating subsequent alterations. Moreover, the tree can be augmented with dependency information to enable explanations of the meta-level actions taken. These issues are beyond the scope of this paper and will not be discussed further.

[5]In this sense the constraint root bears a strong resemblance to Allen's *reference interval* [Allen 81a].

manner described above. If an applicable constraint is not located the process is recursively applied to the alteration tree designated by the SPECIALIZATION-OF relation in the root.

The constraint root may also serve a dual role as a repository for information related to the constraint that is invariant across all possible alterations. Such information is made available to all constraints currently residing in the alteration tree through the ALTERATION-OF inheritance relation defined in Figure 4-2.

4.2. An example: representing shift constraints

To amplify the ideas of the last section, let us examine the representational framework in a specific context drawn from the job shop scheduling domain. One type of time varying constraint that needs to be expressed in this domain is the number of work shifts associated with a machine, work area, or facility. Figure 4-4 defines this class of constraints as a type of time varying constraint. In this case, the value of the constraint consists of a specification of the work shifts involved.

```
{{shift-constraint
    {IS-A time-varying-constraint
        [elaborate VALUE --> SHIFT:
                              range: (LIST (TYPE INSTANCE hours-of-day-interval)) ] }}
```

Figure 4-4: shift-constraint schema

The definition of the **shift-constraint** schema presumes the existence of an additional *hours-of-day* time line for representing work shifts. This leads to the definition of the **shift** schema depicted in Figure 4-5. Within this definition, a shift's WORK-WEEK is itself represented as a temporal interval, in this case with respect to a *days-of-week* time line.

```
{{shift
    {IS-A hours-of-day-interval }
        FOREMAN:
        WORK-WEEK:
            range: (TYPE INSTANCE days-of-week-interval)] } }}
```

Figure 4-5: shift schema

An instance of the **shift** schema representing the first shift associated with machine *mach1* and its corresponding work week specification are contained in Figure 4-6.

```
{{mach1-5day-1st-shift
    {INSTANCE shift
        START-TIME: (7 30)
        END-TIME:  (15 30)
        FOREMAN: Jones
        WORK-WEEK: 5day-workwk  } }}

{{5day-workwk
    {INSTANCE days-of-week-interval
        START-TIME: monday
        END-TIME: friday }  }}
```

Figure 4-6: **mach1-5day-1st-shift** schema

With these basic primitives in hand, let us examine the representation of a specific configuration of shift constraints for the *mach1* machine identified above. Assuming that a **shift-constraint-root** schema has been defined as a subtype of the **time-varying-constraint-root**, first consider the specific constraint root that is attached to the *mach1* machine description. This is displayed in Figure 4-7.

```
{{mach1-shift-constraint-root
    {INSTANCE shift-constraint-root
        SPECIALIZATION-OF: milling-area-shift-constraint-root
        ALTERNATIVES: mach1-shifts
        CURRENT-ALTERATIONS: mach1-shift-constraint1 mach1-shift-constraint2  }}
```

Figure 4-7: **mach1-shift-constraint-root** schema

The root indicates that the encompassed collection of shift constraints is a specialization of those associated with the *milling* work area. It also identifies **mach1-shifts** as the relaxation spec which contains the possible values that the shift constraint can assume. The **mach1-shifts** schema is depicted in Figure 4-8.[6]

[6]Relaxation specs are defined in [Fox 83a] and the reader is referred there for a complete discussion of their structure and interpretation. For our purposes here, it is sufficient to note that the **shift-constraint-relaxation-spec.** schema of which **mach1-shifts** is an instance is defined as a particular type of relaxation spec called a *discrete choice*. As the name implies, a *discrete choice* specifies a discrete set of alternative values.

```
{{mach1-shifts
    {INSTANCE shift-constraint-relaxation-spec
        RELAXATION: (mach1-alt1 mach1-alt2 mach1-alt3
                     mach1-alt4 mach1-alt5 mach1-alt6) } }}
```

Figure 4-8: mach1-shifts schema

The alternatives listed in **mach1-shifts**, which vary the number of shifts and the length of the work week, are enumerated in Figure 4-9.[7]

```
{{mach1-alt1
    {INSTANCE shift-constraint-relaxation
        SHIFT: mach1-5day-1st-shift
        UTILITY: 2.0 } }}

{{mach1-alt2
    {INSTANCE shift-constraint-relaxation
        SHIFT: mach1-7day-1st-shift
        UTILITY: 1.6 } }}

{{mach1-alt3
    {INSTANCE shift-constraint-relaxation
        SHIFT: (mach1-5day-1st-shift mach1-5day-2nd-shift)
        UTILITY: 1.4 } }}

{{mach1-alt4
    {INSTANCE shift-constraint-relaxation
        SHIFT: (mach1-7day-1st-shift mach1-7day-2nd-shift)
        UTILITY: 1.2 } }}

{{mach1-alt5
    {INSTANCE shift-constraint-relaxation
        SHIFT: (mach1-5day-1st-shift mach1-5day-2nd-shift mach1-5day-3rd-shift)
        UTILITY: 0.9 } }}

{{mach1-alt6
    {INSTANCE shift-constraint-relaxation
        SHIFT: (mach1-7day-1st-shift mach1-7day-2nd-shift mach1-7day-3rd-shift)
        UTILITY: 0.7 } }}
```

Figure 4-9: shift constraint alternatives for machine *mach1*

[7] By historical convention, the utilities associated with constraints range from 0.0 to 2.0.

The constraints currently bound to *mach1* are contained in the CURRENT-ALTERATIONS slot of the **mach1-shift-constraint-root** schema (see Figure 4-7). These constraints are defined in Figure 4-10. **mach1-shift-constraint1** is seen to be applicable from the beginning of week 0 to the beginning of week 20 and specifies a single 8 hour shift operating over a five day work week. Its associated utility of 2.0 indicates the desirability of satisfying this constraint. Alternatively, **mach1-shift-constraint2** is relevant from the beginning of week 50 to the beginning of week 80 and specifies two 8 hour shifts operating over a five day work week. It possesses a reduced utility of 1.4 to reflect the cost of adding a second shift. There are no constraints directly associated with *mach1* during the temporal intervals not covered by **mach1-shift-constraint1** and **mach1-shift-constraint2**. At points in time during those intervals the applicable constraint will be found at a higher level in the SPECIALIZATION-OF hierarchy.

```
{{mach1-shift-constraint1
   {INSTANCE shift-constraint
      TEMPORAL-SCOPE: {{INSTANCE calendar-interval
                               START-TIME: (0 0 0)
                               END-TIME: (20 0 0) }}
      SHIFT: mach1-5day-1st-shift
      UTILITY: 2
      ALTERATION-OF: mach1-shift-constraint-root } }}

{{mach1-shift-constraint2
   {INSTANCE shift-constraint
      TEMPORAL-SCOPE: {{INSTANCE calendar-interval
                               START-TIME: (50 0 0)
                               END-TIME: (80 0 0) }}
      SHIFT: (mach1-5day-1st-shift mach1-5day-2nd-shift)
      UTILITY: 1.4
      ALTERATION-OF: mach1-shift-constraint-root } }}
```

Figure 4-10: the actual constraints associated with machine *mach1*

Having now completely specified *mach1*'s configuration of shift constraints, let us conclude the example by providing a flavor of how the representation will evolve as alterations are made. It will be helpful at this point to move to a graphical notation, and the above described representation is recast in these terms in Figure 4-11 (along with the addition of a *default* constraint residing in the *milling* work area description). Let us assume that this configuration of constraints is the result of *a priori* made decisions, motivated perhaps by forecasted load levels in the job shop.

As the search for a constraint satisfying job shop schedule proceeds, we would expect a more accurate assessment of the load levels in the shop to emerge. For our purposes here, suppose that the following sequence of alterations is made by the meta level reasoning system as this knowledge about the search space accumulates:

1. **shift-constraint1** is relaxed to include a second shift during the temporal interval

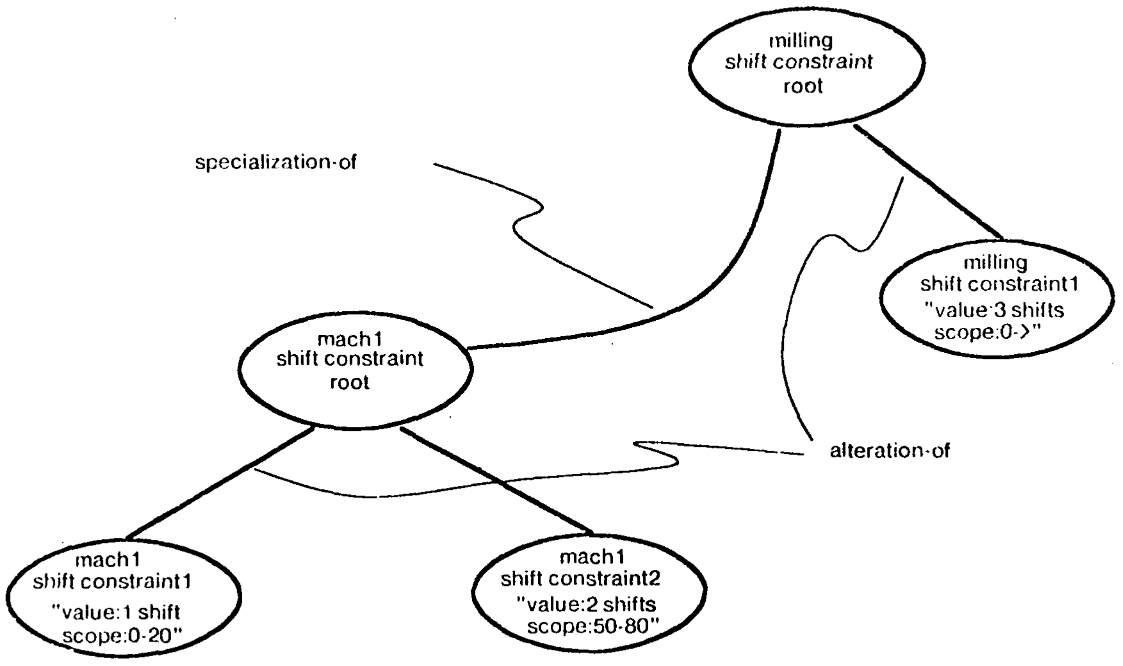


Figure 4-11: Initial configuration of shift constraints for machine *mach1*

ranging from the beginning of week 5 to the beginning of week 12.

2. This relaxation is subsequently relaxed to include a third shift from the beginning of week 10 to the beginning of week 12.

3. The default constraint residing with the *milling* work area is strengthened to specify only two shifts from the beginning of week 25 to the beginning of week 35.

Each decision to alter a shift constraint results in the attachment of a new shift constraint (embodying the desired alteration) to the CURRENT-ALTERATIONS slot of the constraint found to be unsatisfactory. Thus, given the above sequence of alterations, the configuration of shift constraints associated with *mach1* would now appear as in Figure 4-12. According to the resolution mechanism defined in Section 4.1, each new constraint now takes precedence over the original (i.e. the constraint of which it is an alteration) at points within its temporal scope.

5. Temporal knowledge as a basis for constraint propagation

In contrast to constraints which provide a basis for discriminating among alternative decisions proposed by the reasoning system, there are constraints that are applied in a generative fashion. These constraints determine the admissibility of decisions, delineating boundaries within which the exploration of alternative decisions may proceed. In many cases, these constraints are embedded in the model of the domain (e.g. operation *x* can only be performed on machine *y*) and are invariant over time. An important exception, however, is the collection of constraints characterizing the nonavailability of resources. The majority of these constraints evolve dynamically as explorations in

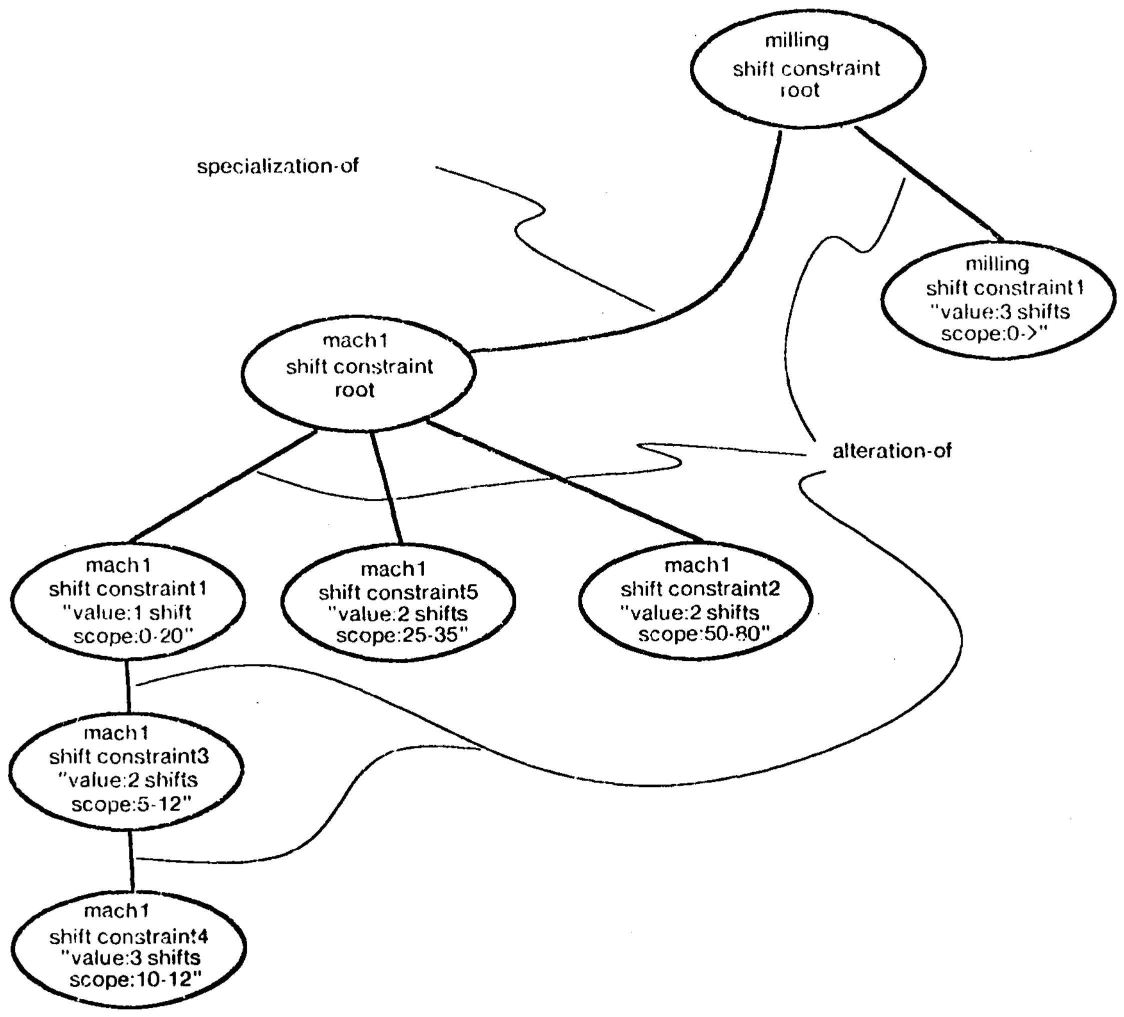


Figure 4-12: Resulting configuration of shift constraints for machine *mach1*

various temporal regions of the space culminate in the establishment of resource reservations.

Within the job shop scheduling domain, resource reservations may be established or removed by either the system or the user. These scheduling decisions are made in an opportunistic fashion [Erman 80] and may be expressed at various levels of abstraction. Given these characteristics, the issue of maintaining the consistency of a dynamically changing collection of reservations must be addressed. Specifically, the temporal constraints imposed by particular scheduling decisions must be propagated through the rest of the partially developed schedule so as to influence the scheduling decisions that remain to be made. This, in turn, requires an appropriate organizational framework in which to embed these constraints.

Once again, temporal relations provide the necessary structure, in this case as provided in the representation of the *plan knowledge* from which the schedules of individual orders are derived. Such a representation and an associated set of constraint propagation techniques are examined in the following subsections.

5.1. Multiple descriptions of activities and resources

Each product producible in the job shop has associated with it an *operations graph* which defines the alternative process routings that may be employed to produce the product. An operations graph is a network of activities, partially ordered with respect to time. Any path through the network constitutes a viable plan and a complete assignment of resources along a given path constitutes a schedule. The operations graph also exhibits a hierarchical structure in that activities are modeled at various levels of abstraction. Thus, an operations graph is not unlike a *procedural net* formulation of plan knowledge [Sacerdoti 77]. Its distinguishing characteristics include

- flexibility in the types of abstractions that are representable,
- an explicit representation of the temporal structure inherent in the plans, and
- the existence of schedulable resources at all levels of abstraction.

There appear to be two forms of abstraction relevant to the expression of process routings: abstraction by aggregation of sequences of activities into composite activities (the form typically found in plan expressions) and abstraction by omission of activities. The latter form emphasizes an approach whereby critical facilities (i.e. bottleneck resources) are scheduled before an attempt is made to develop a complete schedule, and is particularly useful in supporting the interactive development of portions of the schedule by the user. A given operations graph will generally incorporate both forms of abstraction, distinguishing between the two by the specific temporal relationships linking various activities.

Within the the job shop scheduling domain, an activity is refined into an operation. The prototypical operation schema is displayed below in Figure 5-1.[8]

```
{{operation
    {IS-A activity
        RESOURCE:
        OPERATION-TIME:
            range: (TYPE INSTANCE calendar-interval)
        SCHEDULING-CONSTRAINTS:
        AFTER:
        MET-BY:
        BEFORE:
        MEETS:
        DURING:
        CONTAINS: } }}
```

Figure 5-1: operation schema

[8] In this schema and those to be presented subsequently, only the slots relevant to the discussion will be listed.

Operations are organized into operations graphs via the temporal relations contained in the **operation** schema definition. The AFTER and MET-BY relations associate an operation with alternative previous operations in the set of process routings, with MET-BY indicating that the alternative previous operations and the current operation occur consecutively in the production of the product and AFTER signaling the omission of one or more intervening operations at this level of abstraction. MET-BY is always in use if previous operations exist. AFTER is utilized only by the user interface to shield nonbottleneck operations from the user. Similarly, the BEFORE and MEETS relations associate an operation with alternative next operations in the set of process routings. In this case, MEETS indicates the consecutive occurrence of operations and is always in force if following operations exist, while BEFORE indicates the omission of one or more intervening operations and is utilized in the same manner as AFTER. Thus, the MEETS and MET-BY relations of operations collectively define the set of process routings at each level of abstraction.[9] The remaining temporal relations are used to define the operation abstraction hierarchy. The DURING relation links an operation with its superoperation (i.e. the abstract operation of which this operation is a suboperation). The CONTAINS relation associates an operation with alternative sequences of more detailed operations. A portion of a specific operations graph, illustrating the temporal framework, is contained in Figure 5-2.

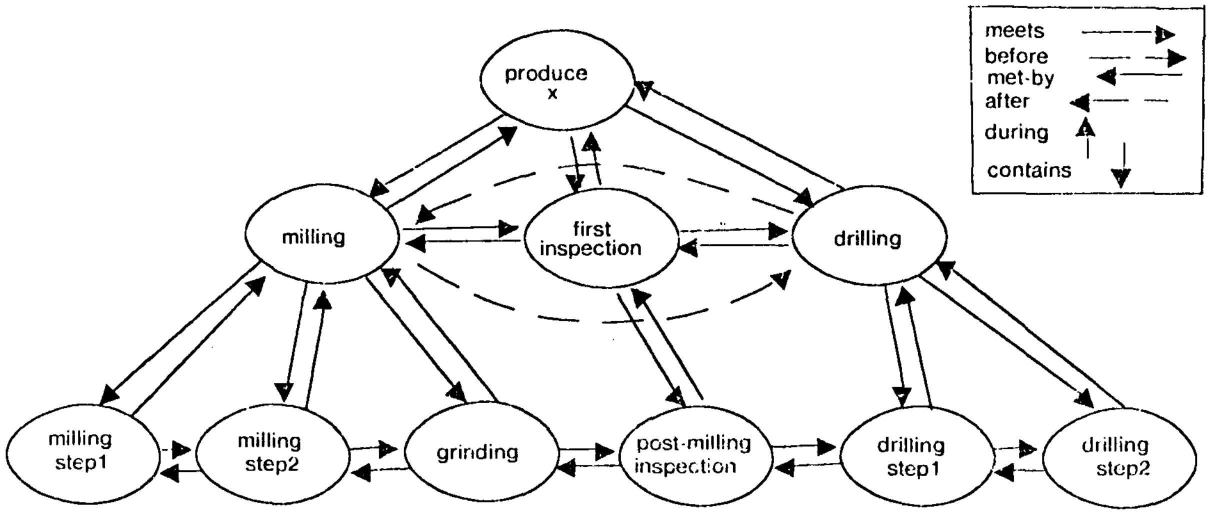


Figure 5-2: a portion of the operations graph for product *x*

The remaining slots listed in the **operation** schema are attributes relevant to the scheduling of operations. OPERATION-TIME contains an estimate of the duration of the operation, SCHEDULING-CONSTRAINTS contains a set of temporal constraints that must be satisfied when scheduling the operation (see Section 5.2), and RESOURCE contains the resource that must be reserved. Resources are associated with operations at all levels in the operation abstraction

[9] It should be noted that OVERLAPS and OVERLAPPED-BY are also viable relations for associating alternative next and previous operations respectively. However, the need to distinguish between these relations and MEETS/MET-BY arises only when reasoning about the specific bounds of resource reservations (e.g. detecting conflicts). We will assume such an ability in the discussion below and omit further consideration of the OVERLAPS and OVERLAPPED-BY relations.

hierarchy, with the resources residing at a given level embodying abstractions of the resources required by lower level suboperations. The taxonomy of resource types depicted in Figure 5-3 forms the basis for describing resources within the job shop scheduling domain.

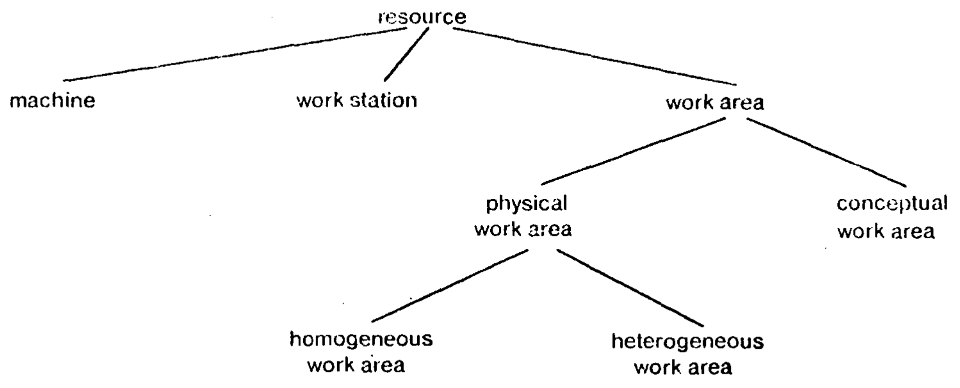


Figure 5-3: taxonomy of resource descriptions

At the lowest level in the operation abstraction hierarchy, the resource required by an operation is assumed to be a specific machine or work station. We can restrict our attention to machines only without loss of generality. There may be one such machine in the shop or several identical machines from which to choose. The first case is represented by the **machine** schema in Figure 5-4 while the **homogeneous-work-area** schema in Figure 5-5 describes the second case.

```
{{machine
    {IS-A resource
        RESERVATIONS:
        PHYSICAL-WORK-AREA:
        CONCEPTUAL-WORK-AREA: } }}
```

Figure 5-4: **machine schema**

```
{{homogeneous-work-area
    {IS-A physical-work-area
        RESERVATIONS:
        PART-OF:
        PHYSICAL-WORK-AREA:
        CONCEPTUAL-WORK-AREA: } }}
```

Figure 5-5: **homogeneous-work-area** schema

Both the **machine** and **homogeneous-work-area** schemata possess a RESERVATION slot which contains the reservations currently established for the resource. The PHYSICAL-WORK-AREA and CONCEPTUAL-WORK-AREA relations are links to the physical work area and various conceptual work areas (see below) to which the resource belongs. The **homogeneous-work-area** schema possesses an additional PART-OF relation that associates the resource with the specific machines that comprise it.

The resource associated with a given abstract operation is characterized by the **conceptual-work-area** schema (Figure 5-6), the PARTS-OF which are the resources required by each of the constituent lower level operations embodied by the abstract operation. The need for distinguishing between physical and conceptual work areas arises from the fact that the two are derived along different organizational dimensions. Physical work areas imply a physical proximity of their constituent resources and are useful in organizing constraints pertaining to work shifts, manpower levels, etc. at higher levels. To map operation abstractions to physical work areas would severely restrict the range of possible abstractions. Conceptual work areas, alternatively, permit a grouping of resources that parallels the operation abstraction hierarchy.

```
{{conceptual-work-area
    {IS-A work-area
        RESERVATIONS:
        PART-OF:
        CONCEPTUAL-WORK-AREA: } }}
```

Figure 5-6: conceptual-work-area schema

Resource reservations at all levels in the abstraction hierarchy are represented by the **reservation** schema shown in Figure 5-7.

```
{{reservation
    {IS-A constraint
        RESERVER:
        OPERATION:
        RESOURCE:
        TEMPORAL-SCOPE:
            range: (TYPE INSTANCE calendar-interval)
        STATUS:
            range: (OR valid invalid incomplete)
        ORIGIN:
            range: (OR imposed generated) } }}
```

Figure 5-7: reservation schema

Instantiations of the **reservation** schema are attached directly to the resources involved as commitments are made to particular reservations. As we will see in the next section, however, a commitment to a reservation does not necessarily imply its validity in the job shop schedule under development. Reservations residing at a given level of abstraction are supported by reservations residing at lower levels and a reservation is valid if and only if all of its supporting reservations are valid. Reservations may also lead to conflicts with subsequent commitments and become invalidated. The STATUS of a reservation reflects its current relationship to other-reservations and may assume one of the following values:

- valid - the reservation is valid (i.e all of its supporting reservations have been established and validated)
- incomplete - at least one of the reservation's supporting reservations remain to be established and/or validated
- invalid - the reservation has been invalidated due to the validation a of conflicting reservation or an inability to validate the reservation's supporting reservations.

A reservation's ORIGIN indicates whether the reservation was externally *imposed* by either the reasoning system or the user, or *generated* during the propagation process. Its significance will become clear below.

5.2. Maintaining the consistency of resource reservations through the posting of constraints

A commitment to a particular scheduling decision (i.e. the establishment of a resource reservation in a specific order's name for a specific operation during a specific time interval) imposes constraints which must be satisfied by the scheduling decisions that remain to be made. Preceding operations for the order, for example, are now constrained to end by a certain point in time. Likewise, following operations are now constrained to start on or after a certain point in time. Ensuring the consistency of the dynamically evolving set of resource reservations requires an ability to bring the constraints imposed by previous scheduling decisions to bear when contemplating the scheduling decisions that remain to be made. The strategy adopted below is one of explicitly representing these constraints and *posting* them with the operation(s) for which they are relevant. The temporal framework of the operations graph provides the basis for identifying the operation(s) with which a particular constraint should be posted.

The constraints to be posted are specializations of a general class of constraints referred to as predicate constraints in [Fox 83a]. The defining schema is depicted in Figure 5-8. It includes a PREDICATE which is applied to its ARGUMENTS and the value being constrained to yield either the SATISFIED-UTILITY (if the predicate evaluates true) or the RELAXED-UTILITY (if the predicate evaluates false). In the current context, where the constraints of interest are nonrelaxable, we will assume a SATISFIED-UTILITY of *t*, and a RELAXED-UTILITY of *nil*.

```
{{predicate-constraint
   {IS-A constraint
      ARGUMENTS:
      PREDICATE:
      SATISFIED-UTILITY:
      RELAXED-UTILITY: } }}
```

Figure 5-8: predicate-constraint schema

5.2.1. Potential inconsistencies and associated constraint types

We can partition the inconsistencies that can potentially arise as a result of the imposition of a new resource reservation into two types:

- those involving conflicts between reservations belonging to the same order, and
- those involving conflicts between reservations belonging to different orders.

Conflicts of the former type constitute violations of the temporal relationships embodied in the operations graph associated with the product ordered, and it is these conflicts that will be avoided through the posting of constraints. Conflicts of the latter type, alternatively, involve contention for the same resources. Since the priority of an order can often dictate the preemption of another order, these conflicts will not be prevented but will be resolved upon detection. For the present, let us confine our attention only to the conflicts that can arise within a given order's schedule and identify the types of constraints needed to ensure their avoidance.

Conflicts between the reservations of temporally related operations residing at different levels in the operations graph (i.e. those related via DURING and CONTAINS) constitute one form of inconsistency that might arise as the set of reservations evolves. Avoidance of these conflicts requires propagation of the consequences of each new scheduling decision in both directions. Reservations residing at higher levels can be immediately adjusted (and created if necessary) to reflect the more detailed estimate provided by the new imposition. In the other direction, the subsequent scheduling of supporting lower level operations must be constrained to occur within the temporal scope of the newly imposed decision. This is accomplished by posting operation time bound constraints with each of these operations.

The operation time bound constraint schema is defined in Figure 5-9. It is a predicate constraint in which the ARGUMENTS are elaborated into a set of TIME-BOUNDS. The predicate *op-time-boundp*, when applied to a perspective scheduling decision for ORDER, returns *t* if the temporal scope of the decision falls within the specified TIME-BOUNDS and *nil* otherwise. The POSTED-BY relation designates the originator of the constraint and its STATUS indicates whether the constraint is currently active or has been rendered inactive by the posting of a more accurate instance of the constraint. The generation and maintenance of these constraints (and those to be described below) will be addressed in the following subsections.

```
{{operation-time-bound-constraint
   {IS-A predicate-constraint
      [elaborate ARGUMENTS --> TIME-BOUNDS:
                                   range: (TYPE INSTANCE calendar-interval)]
      PREDICATE: op-time-boundp
      ORDER:
      POSTED-BY:
         range: (TYPE INSTANCE operation)
      STATUS:
         range: (OR active inactive) } }}
```

Figure 5-9: operation-time-bound-constraint schema

A second form of inconsistency that might arise involves conflicts among the reservations of operations that are related temporally at the same level in the operations graph (i.e. via MEETS and MET-BY). These conflicts are avoided by propagating the start and end times associated with a newly imposed scheduling decision laterally in the graph. Specifically, either start time constraints or end time constraints (whichever is appropriate) are posted with the other operations at this level. The schemata defining these constraints are contained in Figures 5-10 and 5-11 respectively, and have their obvious interpretation.

```
{{start-time-constraint
   {IS-A predicate-constraint
      [elaborate ARGUMENTS --> START-TIME
                                   range: (TYPE INSTANCE calendar-time-point)]
      PREDICATE: start-timep
      ORDER:
      POSTED-BY:
         range: (TYPE INSTANCE operation)
      STATUS:
         range: (OR active inactive) } }}
```

Figure 5-10: start-time-constraint schema

```
{{end-time-constraint
   {IS-A predicate-constraint
      [elaborate ARGUMENTS --> END-TIME
                                  range: (TYPE INSTANCE calendar-time-point)]
      PREDICATE: end-timep
      ORDER:
      POSTED-BY:
         range: (TYPE INSTANCE operation)
      STATUS:
         range: (OR active inactive) } }}
```

Figure 5-11: end-time-constraint schema

A final form of inconsistency will arise if mutually exclusive operations in the operations graph are scheduled. These conflicts are prevented by posting operation restriction constraints with any operation that should be eliminated from consideration as a result of the imposition of a particular scheduling decision.

```
{{operation-restriction-constraint
   {IS-A predicate-constraint
      [elaborate ARGUMENTS --> OPERATION]
      PREDICATE: op-restrictionp
      ORDER:
      POSTED-BY:
         range: (TYPE INSTANCE operation) } }}
```

Figure 5-12: operation-restriction-constraint schema

Given the above constraint posting approach to consistency maintenance, let us consider the techniques responsible for generating and propagating constraints as the set of resource reservations evolves. First, we will examine the propagation of a new scheduling commitment through the existing set of resource reservations. We will then examine the inverse operation of propagating a decision to remove a particular reservation. These considerations will lead to a subsequent discussion of techniques for propagating decisions to invalidate and restore particular reservations.

5.2.2. Propagating a commitment to a new resource reservation

For purposes of discussion, let us assume that a scheduling decision with the parameters *order op resource stime* and *etime* has been imposed, and it has been verified that the imposition satisfies all of the constraints that have been posted with operation *op* for *order*. This results in the creation of a reservation schema instance that represents the decision and the ORIGIN is designated as *imposed*. The status of this newly imposed reservation depends on the level of abstraction embodied by *op*. If *op* resides at the lowest level in the operations graph (i.e. it CONTAINS no suboperations) then the newly imposed reservation is considered valid. If *op* resides at an abstract level in the operations graph, the reservation is necessarily *incomplete* as its validity is contingent on the validity of supporting lower level reservations that have yet to be imposed.

In the former case, the reservation has been imposed at a level where physical resources (e.g. specific machines) are involved. As such, it is necessary to detect and resolve any *resource contention conflicts* that have been introduced. In the latter case, it is necessary to propagate the imposition to the supporting suboperations at lower levels in the operations graph. As indicated above, this is accomplished by posting appropriate operation time bound constraints with these operations.

The resolution of resource contention conflicts requires the invalidation of one of the offending reservations, and there are several candidate strategies for deciding which to invalidate (e.g. a comparison of order priority, a comparison of the authority of each reservation's creator, an appeal to the user). For simplicity, we will assume in what follows that all conflicts are resolved in favor of the newly imposed reservation. The use of more complex strategies is a straightforward substitution. The choice to invalidate rather than remove the selected reservation allows its restoration if subsequent scheduling actions eliminate the conflict (see Section 5.2.4).

Downward propagation

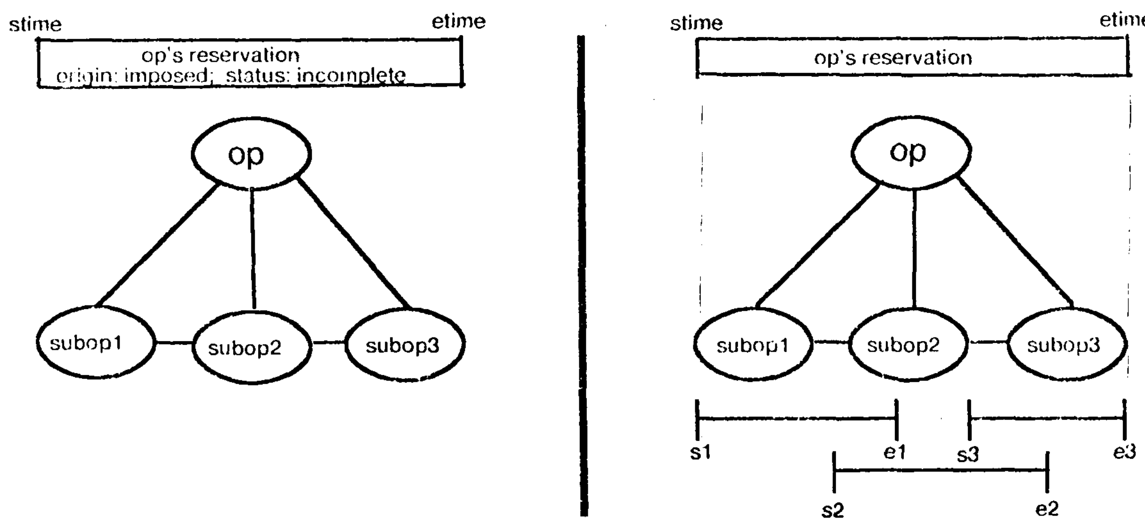
The operation time bound constraints are derived by applying a critical path method (CPM) analysis to the operations related to *op* by the *contains* relation. Estimates of the durations of these operations drive this analysis and existing reservations are taken into account. The CPM analysis is first applied in a forward direction through the sequence(s) of lower level operations abstracted by *op* to determine the earliest start time (the first bound) of each operation. It is then applied in a backward direction to determine the latest end time (the second bound) for each operation. The analysis is recursively applied to any lower level operation which is itself an abstraction of more detailed operations. Upon completion of the CPM analysis, the posted constraints dictate acceptable intervals within which the associated operations may be scheduled (see Figure 5-13). Any operation time bound constraints previously posted with these operations (i.e. constraints that originated from reservations residing higher that *op* in the operations graph) are rendered inactive.[10]

Lateral propagation

The imposition is also propagated to the operations preceding and following *op* at the same level in the operations graph. An end time constraint of *stime* is created and propagated to each operation encountered while moving through the MET-BY relations of *op* and its predecessors. Upon each posting of the constraint, its value is reduced by the duration of the associated operation. Similarly, a start time constraint with an initial value of *etime* is created and propagated to each operation encountered while moving through the MEETS relation of *op* and its successors.

As various scheduling decisions are imposed, distinct start time/end time constraints may accumulate with the same operation. At any point in time, however, only the start time/end time

[10]It should be noted that the CPM analysis can also be used to verify that the temporal scope of the imposition (delineated by *stime* and *etime*) allows sufficient time for performing *op*.



a. before downward propagation b. after downward propagation

Figure 5-13: Downward propagation of operation time bound constraints

constraint originating from the "closest" operation in the operation graph is active. Thus, the imposition of *op*'s reservation may result in the posting of inactive start time/end time constraints with the operations some distance away from *op* in the operations graph. This is necessary to insure the existence of an appropriate start time/end time constraint if the reservations associated with intervening operations are subsequently removed (see Section 5.2.3).

Upward propagation

Finally, the imposition of *op*'s reservation for *order* is propagated to the super operation DURING which *op* occurs. If the super operation does not possess a reservation belonging to *order*, a new reservation is created, its ORIGIN is marked *generated*, and the temporal scope of *op*'s reservation is assigned. If such a reservation does exist, its temporal scope is adjusted to reflect the inclusion of *op*'s newly imposed reservation. Propagation of the appropriate status change is accomplished by examining the sequence of operations aggregated by *op*'s super operation that includes *op*.[11] If all operations in the sequence now possess valid reservations, the super operation's reservation is marked valid. Otherwise, it is marked incomplete.

The specific nature of the adjustment to the temporal scope of the super operation's reservation depends on the ORIGIN of the reservation. The scope can only decrease if this reservation was previously *imposed* by the reasoning system or the user. This will occur only if *op* is the initial (final) operation in the sequence of suboperations including *op* and *stime* (*etime*) falls within the operation time bound previously posted with *op*. Figure 5-14 illustrates this case. Alternatively, the temporal scope of the super operation's reservation can only increase if it was initially *generated* during the

[11] There will be more than one sequence if the super operation CONTAINS mutually exclusive alternatives.

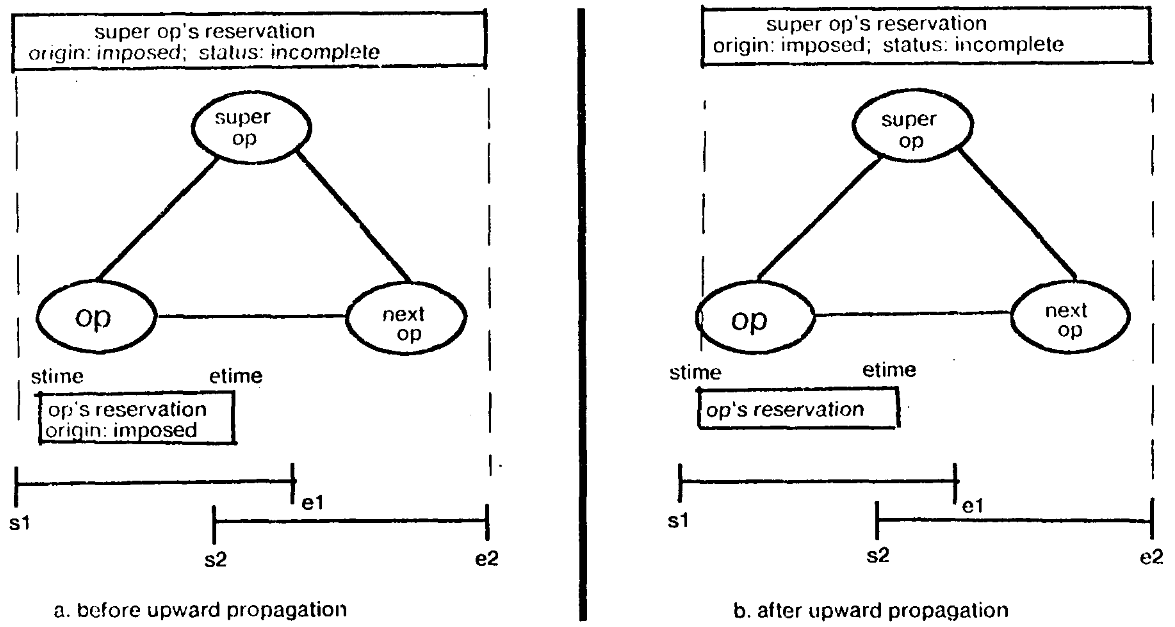


Figure 5-14: Adjusting the scope of a previously imposed parent reservation to reflect the imposition of *op*'s reservation

upward propagation of a previous imposition. A change will occur in this case only if there are no operations preceding (following) *op* in the sequence of suboperations that possess a valid or incomplete reservation for *order* (see Figure 5-15).

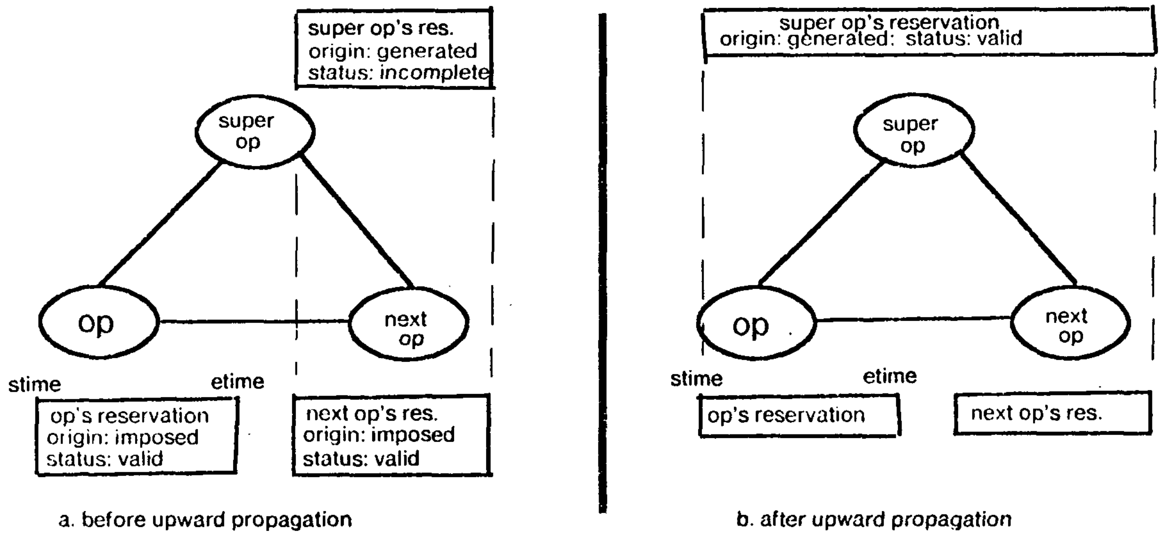


Figure 5-15: Adjusting the scope of a previously generated parent reservation to reflect the imposition of *op*'s reservation

If a change is made to the temporal scope of the super operation's reservation, it is necessary to propagate new start time and/or end time constraints to the operations residing at the same level in the operations graph. This is handled in the same manner as above, with the new constraints replacing those that previously originated from this operation. The new temporal scope may also introduce inconsistencies with respect to the start time and end time constraints posted with the super operation. Such a situation is depicted in Figure 5-16. This does not indicate an actual conflict, however, as the originator of the constraint is necessarily an operation possessing an incomplete reservation for *order*. If the operation did possess a valid reservation, then the start time/end time constraint originating from its supporting lower level reservations would have prevented the imposition of *op*'s reservation for *order* at the outset. The start time/end time constraints posted by *op* will insure the eventual consistency of the reservations.[12]

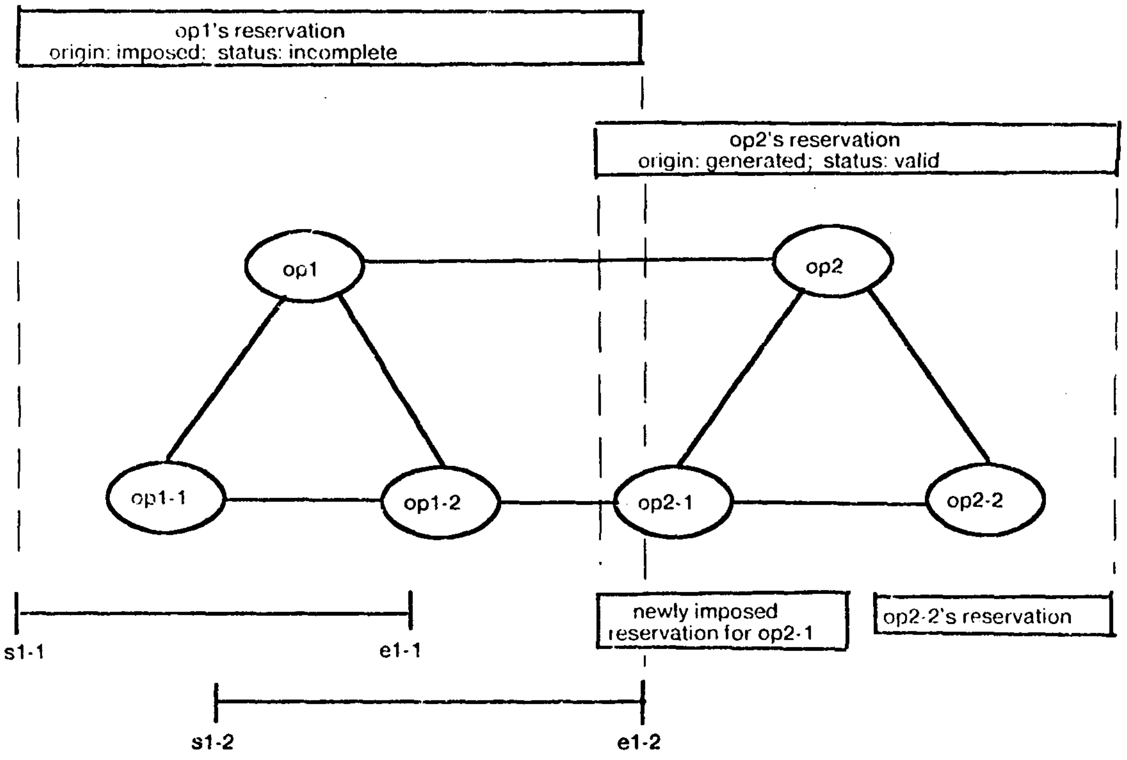

op1's reservation
origin: imposed; status: incomplete
op2's reservation
origin: generated; status: valid
op1
op2
op1-1
op1-2
op2-1
op2-2
newly imposed reservation for op2-1
op2-2's reservation
s1-1
e1-1
s1-2
e1-2

Figure 5-16: Inconsistency among reservations residing at an abstract level

By reflecting the imposition of *op*'s reservation for *order*, the super operation's reservation now constitutes a commitment to the sequence of suboperations that includes *op*. As such, further consideration of any alternative sequences of operations CONTAINED by the super operation should be prohibited. This is enforced by generating and posting operation restriction constraints with each

[12] We could adjust the temporal scope of the reservation belonging to the operation from which the start time/end time constraint originated to immediately remove the inconsistency. If this approach is taken, it is also necessary to readjust the temporal scope if the super operation's reservation is subsequently removed. For simplicity, we will tolerate these temporary inconsistencies.

operation in the alternative sequences if the imposition of *op*'s reservation for *order* represents the initial commitment to the sequence of operations including *op*.

If any of the above actions result in a change to the super operation's reservation, the process is recursively applied to the operation appearing in the range of its DURING relation.

5.2.3. Propagating the removal a resource reservation

The removal of a resource reservation with the parameters *order op* and *resource* has an inverse effect on the existing collection of resource reservations. *op*'s reservation for *order* is removed, and, if the reservation's status was not *invalid*, any posted start time, end time, operation time bound, and operation restriction constraints that originated from *op* are retracted (if *op*'s reservation was previously invalidated, these constraints have already been eliminated - see Section 5.2.4). If *op* resides at a level in the operations graph where physical resources are involved, *resource*'s other reservations are examined to determine whether any reservations that were previously invalidated due to resource contention conflicts may now be restored. Otherwise, the above actions are recursively applied to any reservation for *order* associated with a suboperation CONTAINED by *op*.

The removal of *op*'s reservation for *order* is also propagated upward in the operations graph to the super operation appearing in the range of *op*'s DURING relation. Adjustments may be necessary to the status and temporal scope of this super operation's reservation for *order* as well as the posted constraints that originally resulted from this reservation. In considering these adjustments below, it is once again necessary to distinguish whether the reservation was originally *imposed* or *generated* during the propagation of a previous imposition.

If the ORIGIN the super operation's reservation is marked *generated*, then its existence is a consequence of the impositions of its supporting reservations, and its resulting status must reflect the current status of the reservations associated with the sequence of suboperations CONTAINED by the super operation that includes *op*. Specifically, the super operation's reservation is removed if no reservations in *order*'s name remain for any of the suboperations in the sequence, invalidated if no valid or incomplete reservations remain, or marked as incomplete if neither of the above conditions are met. In either of the first two cases, any previously posted start time, end time, or operation restriction constraints originating from the super operation are retracted as well.

If, alternatively, the super operation's reservation was originally *imposed* then its existence does not depend on the existence of supporting lower level reservations. Its resulting status is necessarily *incomplete* since there is now at least one supporting reservation (i.e. *op*'s) that is no longer valid. Nonetheless, it is necessary to retract any previously propagated operation restriction constraints if no valid or incomplete reservations for *order* remain in the sequence of suboperations that includes *op*.

Adjustments to the temporal scope of the super operation's reservation need be considered only if the status of *op*'s reservation was valid or incomplete, due to the similar nature of the invalidation process. Assuming this to be the case, the action taken again depends on the ORIGIN of the super operation's reservation. If the reservation was *imposed*, an adjustment is necessary only if *op* is the initial and/or final operation in the sequence of suboperations CONTAINED by the super operation. In this case the initial imposition of *op*'s reservation might have reduced the temporal scope of super operation's reservation (see Section 5.2.2), and the endpoint(s) delineating the scope of the original imposition must be restored. The appropriate endpoint(s) can be obtained from the operation time bound constraint previously posted with *op* (see Figure 5-17).

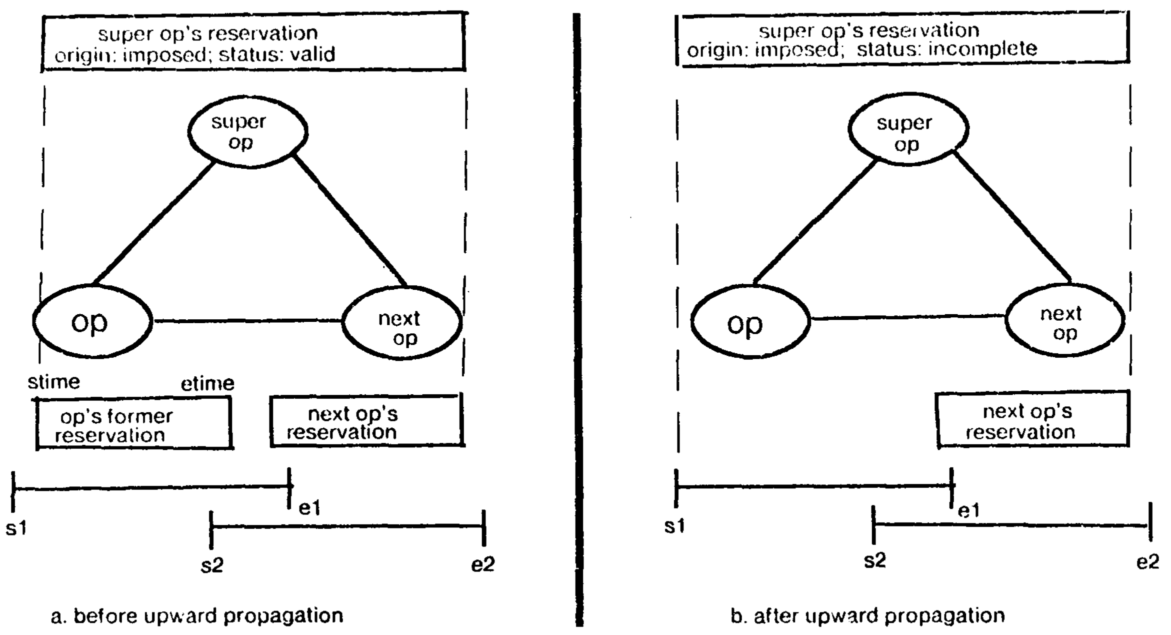


Figure 5-17: Adjusting the scope of a previously imposed parent reservation to reflect the removal of *op*'s reservation

If the super operation's reservation was *generated*, then its temporal scope is adjusted to reflect the removal of *op*'s reservation. This will result in either a reduction in the scope (if *op* was the first/last operation in the sequence of suboperations CONTAINED by the super operation possessing a valid or incomplete reservation for *order*) or no change (see Figure 5-18). Any adjustments made to the scope of the super operation's reservation necessitate the propagation of new start time and/or end time constraints to the other operations residing at this level in the operations graph.

If any of the above actions result in a change to the super operation's reservation, the upward propagation process is recursively applied to the operation appearing in the range of its DURING relation.

5.2.4. Invalidating and restoring resource reservations

The invalidation of a resource reservation can only occur at the lowest level in the operations graph since this is the level at which resource contention conflicts are detected. Consequently, after setting the reservation's status to invalid and retracting the start time and end time constraints that were propagated as a result of the original imposition of the reservation, upward propagation of the invalidation is all that remains. This is handled in precisely the same manner as that of propagating the removal of a reservation upward.

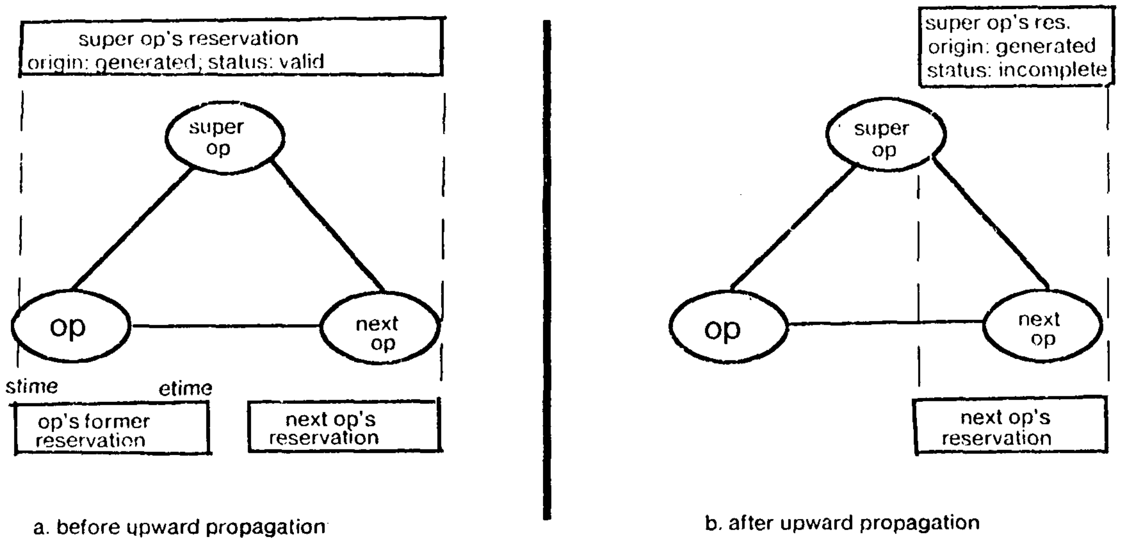


Figure 5-18: Adjusting the scope of a previously generated parent reservation to reflect the removal of *op*'s reservation

The restoration of a previously invalidated reservation can also only occur at the lowest level in the operations graph. In this case, the reservation's status reverts back to valid, and appropriate start time and end time constraints are posted with the other operations residing at this level. The restoration is propagated upward in precisely the same manner as a newly imposed reservation.

6. Summary

In the preceding sections we have addressed some issues surrounding the organization and utilization of constraints in time dependent problem domains. In doing so, we have attempted to demonstrate the utility of an explicit representation of temporal knowledge. Let us reiterate the main ideas of the paper.

We first considered the issue of constraint resolution in the context of constraints that may vary over time with respect to the value(s) to be satisfied. In this case a constraint organization is required that promotes efficient extraction of the relevant variant at any point in time from the collection of constraints associated with the object being constrained. This was accomplished by associating a temporal scope with each such constraint and relating the collection of alterations that have been made to the constraint temporally. A constraint root was introduced to collapse the organization into a single tree structure, and provide a common access point. The resulting organization afforded simple mechanisms for constraint resolution (including the ability to import constraints, if necessary, from higher levels in the model) and constraint alteration (both relaxation and strengthening).

We then turned our attention to the allocation of resources to activities, and considered the problem of utilizing the temporal constraints imposed as this process proceeds so as to maintain the consistency of the hypotheses under development. A constraint posting approach was proposed as a means of ensuring consistency, and techniques for propagating the constraints imposed by the establishment, removal, invalidation and restoration of resource reservations to temporally related activities were defined. These techniques were driven by the temporal relationships embedded in the system's plan knowledge. This approach to resource allocation has been used to provide an interactive scheduling capability within the ISIS-II job shop scheduling system [Fox 83b].

Acknowledgements

This work has benefited greatly from many discussions with Mark Fox and the other members of the ISIS research team: Brad Allen, and Gary Strohm. Thanks should also go to Mark Fox for his helpful comments on earlier drafts of this paper. This research was supported by the Air Force Office of Scientific Research under contract F49620-82-K0017.

References

[Allen 81a] Allen, James F.
Maintaining Knowledge about Temporal Intervals.
Technical Report 86, University of Rochester, January, 1981.

[Allen 81b] Allen, James F.
A General Model of Action and Time.
Technical Report 97, University of Rochester, November, 1981.

[Bobrow 77] Bobrow, D., and T. Winograd.
KRL: Knowledge Representation Language.
Cognitive Science 1(1), 1977.

[Bruce 72] Bruce, Bertram C.
A Model for Temporal References and Its Application in a Question Answering Program.
Artificial Intelligence 3:1-25, 1972.

[Erman 80] Erman, Lee D., Frederick Hayes-Roth, Victor Lesser, and D. Raj Reddy.
The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty.
Computing Surveys 12(2):213-253, June, 1980.

[Fox 79] Fox, Mark S.
On Inheritence in Knowledge Representation.
In *Proceedings 6th IJCAI*, pages 282-284. August, 1979.

[Fox 82] Fox, Mark, Bradley Allen and Gary Strohm.
Job-Shop Scheduling: An Investigation in Constraint-Directed Reasoning.
In *Proceedings of the 2nd National Conference on Artificial Intelligence*, pages 155-158. American Association of Artificial Intelligence, August, 1982.

[Fox 83a] Fox, Mark S.
Constraint-Directed Reasoning: An Investigation of Job Shop Scheduling.
PhD thesis, Carnegie-Mellon University, 1983.

[Fox 83b] Fox, M.S., B.P. Allen, S.F. Smith, and G.A. Strohm.
ISIS: A Constraint Directed Reasoning Approach to Job Shop Scheduling, System Summary.
Technical Report, The Robotics Institute, Carnegie-Mellon University, 1983.

[Fukumori 80] Fukumori, Koji.
Fundamental Scheme for Train Sceduling (Application of Range-Constriction Search.
A.I. Memo 596, Massachusetts Institute of Technology, AI Lab, September, 1980.

[Goldstein 75] Goldstein, Ira P.
Bargaining Between Goals.
In *Proceedings 4th IJCAI*, pages 175-180. 1975.

[Goldstein 77] Goldstein, Ira P.
NUDGE: A Knowledge-Based Scheduling Program.
In *Proceedings 5th IJCAI*, pages 257-263. 1977.

[Kahn 77] Kahn, Kenneth and Gorry, G. Anthony.
Mechanizing Temporal Knowledge.
Artificial Intelligence 9:87-108, 1977.

[Kornfeld 81] Kornfeld, William A.
The use of Parallelism to Implement a Heuristic Search.
In *Proceedings 6th IJCAI*, pages 575-580. August, 1981.

[Lenet 76] Lenet, D.
AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search.
PhD thesis, Computer Science Dept., Stanford University, 1976.

[McDermott 82] McDermott, Drew.
A Temporal Logic for Reasoning About Processes and Plans.
Cognitive Science 6:101-155, 1982.

[Minsky 75] Minsky, M.
A Framework for Representing Knowledge.
In P. Winston (editor), *The Psychology of Computer Vision*, . McGraw Hill, New York, 1975.

[Sacerdoti 77] Sacerdoti, Earl D.
A Structure for Plans and Behavior.
Elsevier, New York, 1977.

[Steels 81] Steels, Luc.
Constraints as Consultants.
AI Memo 14, Schlumberger-Doll, December, 1981.

[Stefik 81] Stefik, Mark.
Planning with Constraints (MOLGEN: Part 1).
Artificial Intelligence 16:111-139, 1981.

[Sussman 80] Sussman, G.J. and G.L. Steele.
CONSTRAINTS: A Language for Expressing Almost-Hierarchical Descriptions.
Artificial Intelligence 14(1):1-39, August, 1980.

[Thorndike 81] Thorndike, Perry, McArthur, David, and Cammarata. Stephanie.
AUTOPILOT: A Distributed Planner for Air Fleet Control.
Technical Note N-1731-ARPA, The Rand Corporation, July, 1981.

[Vere 81] Vere, Steven A.
Planning in Time: Windows and Durations for Activities and Goals.
Technical Report, Jet Propulsion Laboratory, November, 1981.

[Vilain 82] Vilain, Marc B.
A System for Reasoning About Time.
In *Proceedings of the 2nd National Conference on Artificial Intelligence*, pages 197-201. American Association of Artificial Intelligence, August, 1982.

[Waltz 75] Waltz, David.
Understanding Line Drawings of Scenes with Shadows.
In P.H. Winston (editor), *The Psychology of Computer Vision*, . McGraw Hill, New York, 1975.

[Wright 83] Wright, Mark and Mark S. Fox.
SRL/1.5 User Manual.
Technical Report, The Robotics Institute, Carnegie-Mellon University, December, 1983.

I. The SRL schema construct

The basic representational unit within SRL is the *schema*. The schema provides a means for constructing symbolic descriptions of concepts and is similar in spirit to the constructs found in other declarative knowledge representations (e.g. frames [Minsky 75], concepts [Lenet 76], units [Bobrow 77]). This appendix provides a brief overview of its structure.

Syntactically, a schema is composed of a schema name (printed in bold font) and a set of slots (printed in small caps). The slots collectively define the attributive, structural and relational properties of a concept. and each may assume an arbitrary lisp expression as its value. A schema is always enclosed in double braces with the schema name appearing at the top. Figure I-1 illustrates these basic conventions in defining an **operation** schema.

```
{{operation
   OPERATION-TIME:
   RESOURCE: }}
```

Figure I-1: A schema definition

Additional descriptive capabilities are made possible through the association of *meta-information* with a schema. Meta-information is itself represented as a schema (referred to as a *meta-schema*), and may be *attached* to any component of a schema (i.e. the entire schema, a particular slot in the schema, or a particular value in a slot). The slots of an attached meta-schema (printed in italics) provide information about the schema, slot or value. These slots are referred to as *facets* if the meta-schema is attached to a slot. Figure I-2 illustrates the attachment of a meta-schema to the RESOURCE slot in the **operation** schema. The *range* facet specifies restrictions on the values RESOURCE may assume, and the *default* facet specifies its default value.

```
{{operation
    OPERATION-TIME:
    RESOURCE:
        range:
        default: }}
```

Figure I-2: The attachment of meta-information

Schemata can be organized into relational networks through which information (i.e. slots and values) may be inherited. Relations are represented as slots, as illustrated in Figure I-3.

```
{{milling-operation
    IS-A: operation }}
```

Figure I-3: Slots as relations

The relation associated with a slot is enclosed in single braces, with the relation type and target schema (e.g. IS-A **operation**) appearing at its head. *Opening* a relation within a schema establishes a particular view of that schema, and allows slots and values to be inherited from the target schema. This information, which is unique to the view defined by the relation, is stored with the relation itself and not directly in the inheriting schema.[13] This is illustrated in the **milling-operation** schema contained in Figure I-4. In this case, the slots OPERATION-TIME and RESOURCE are inherited through the "milling-operation is an operation" relation.

```
{{milling-operation
    {IS-A operation
        OPERATION-TIME:
        RESOURCE: milling-machine } }}
```

Figure I-4: Opening a relation

SRL provides two standard inheritance relations for purposes of constructing taxonomic descriptions of concepts: IS-A (for relating prototypes hierarchically) and INSTANCE (for relating

[13] This solves the *residency problem* in situations where equivalently named attributes can be inherited along different relations.

instances to their prototypes). Mechanisms are also provided for defining specialized inheritance relations that reflect the idiosyncrasies of the domain being modeled. The reader is referred to [Wright 83] for an in depth discussion of these and other facilities provided by SRL.