Reasoning About Recurrence

by

Johannes A. G. M. Koomen

Submitted in Partial Fulfillment

of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

Supervised by James F. Allen

Department of Computer Science

University of Rochester

Rochester, New York

July 1989

Curriculum Vitae

Johannes A. G. M. Koomen¹ was born on February 25, 1951 in The Netherlands. After a brief stint at the Psychology Department of the University of Utrecht, Mr. Koomen moved to Vancouver, Canada in the Spring of 1974. Starting over again that Fall, he enrolled as a Psychology student at the University of British Columbia. In the Fall of 1976, during a seminar on Psychology and Artificial Intelligence, he encountered computers for the first time, was thrilled with their potential, and enthusiastically switched majors. In 1978 he graduated with a B.Sc. (Hons.) in Computer Science, got married, and continued on to graduate studies at UBC. He obtained his M.Sc. degree in 1980 under the supervision of Alan Mackworth. His thesis research dealt with the design and implementation of virtual machines for Lisp systems. To test his ideas against the demands of “the real world,” he joined the staff at the Information Sciences Institute of the University of Southern California. There he designed and implemented the kernel of the Interlisp-Vax system, eventually in use at several hundred sites.

Returning to his old dream of academic teaching, Mr. Koomen decided to resume graduate school to pursue a doctoral degree. In the Fall of 1981 he joined the Department of Computer Science at the University of Rochester. Four months later the first of his four children was born, and another four months later, in order to support his growing family, he joined the Xerox Corporation as a part-time consultant on symbolic computing environments and user interface design tools. He subsequently spent over three years at Xerox, teaching in-house courses, supervising major software releases, and designing, coordinating and implementing system enhancements.

At the University of Rochester Mr. Koomen served as teaching assistant in courses on Artificial Intelligence and Symbolic Computing between 1982 and 1984, and as principal instructor of a course on A. I. Programming in 1985. Soon after joining the department Mr. Koomen became interested and involved in the work of his advisor, James Allen, to whom he served as research assistant for the remainder of his tenure as graduate student. His primary interests lie in the areas of planning, temporal reasoning and knowledge representation.

¹Also known as Hans — I doubt you care about all those initials!

Acknowledgments

It is appropriate to express first my gratitude to my advisor, James Allen. Without his staunch support I would never have made it through these years. His scholarship and keen mind have been an inspiration, and his gentle guidance and encouragement always helped me to keep on plodding. I also thank the one-time and current members of my thesis committee, Lucia French, Gary Dell, Pat Hayes, Dana Ballard and Henry Kyburg. James' and their willingness to go out on a limb for me have made it possible to actually complete my somewhat unorthodox graduate career! My advisors of old, Alan Mackworth and Barry Pollack, also deserve honorable mention here — they believed in my graduate potential long before I did.

I am especially indebted to Josh Tenenberg, whose humor and genuine concern and friendship have on many occasions helped me change focus from the tunnel walls to the light at the end. I also thank my fellow students, in particular Jay Weber, Henry Kautz and Richard Pelavin, for many interesting and challenging discussions and “Gang of N” sessions, and for the many helpful comments and suggestions during presentations of my work. I gratefully acknowledge all the prompt and helpful assistance I received from Brad Miller whenever I harassed him with my incessant questions regarding the *Rhetorical* system. I only hope that the use of the fruits of my labors, TEMPOS, will repay him adequately.

Graduate life here would have been considerably rougher if it hadn't been for the helpfulness and friendliness of the systems and secretarial staff. I wish to thank especially Jill Forster and Rose Peet for their interest, help and caring over the years.

Thanks to my many Corpus friends, especially Jim, Jim & Julie and Dan & Kate, as well as my friends at Xerox, especially David, Jim, Bill and Jim & Donna, for their moral and financial support, and for making living in Rochester an experience I will treasure always.

Alas, it will be quite impossible to adequately thank my wife, Trisha. Nevertheless, Trish, I thank you most sincerely and abundantly for your love and support, for standing by me all these years, for your faith and encouragement, and for putting up with the doubts and the uncertainties.

Finally, it is equally appropriate to express last, but not least, my gratitude to my parents Willie and Jan, for instilling a love of learning, for encouraging me always to reach and to persevere, and for their patient faith. I dedicate this dissertation to Willie, whose qualities of compassion, love of life, strength of mind and desire to learn will always inspire me, qualities all the more treasured for their loss to the ravages of Alzheimer's Disease.

This research was supported by NSF Coordinated Experimental Research grant no. DCR-8320136 and NSF research grant no. 8351665.

Abstract

This dissertation examines the issue of representing knowledge about recurring states and events, and develops mechanisms for reasoning about recurrence. A first-order axiomatization is developed based on the interval calculus that allows *entities* (events, properties, processes, *etc.*) to be associated, incidentally or repeatedly, with temporal intervals. We describe the implementation of a self-organizing interval database manager and inference engine, as well as a system which integrates the interval reasoner with a general purpose theorem prover and provides a restricted version of the recurrence logic.

The recurrence formalism is applied to the domain of planning in a world with traffic control lights. An extensive example is developed, showing how the use of recurrence can enable a planner to determine that a plan to travel from a source to a destination is feasible, even if an intervening traffic light is red at the time it is encountered. In order to develop this example, a novel application of the interval logic to the spatial domain is presented, which allows us to reason about trajectories and their traversal over time. The combination of paths and temporal intervals, together with the recurrence axioms and actions such as **traverse**, allow us to prove several theorems about the use of traffic lights, and to show how an agent can validate plans to get past them.

Table of Contents

1	Introduction	1
1.1	Perspectives on time	1
1.2	Representation of Time	3
1.2.1	Situations	3
1.2.2	Problems	4
1.2.3	Reification	4
1.2.4	Points of Time	5
1.2.5	Intervals of Time	6
1.2.6	Representing recurrence	7
1.3	Planning in Time	9
1.4	Planning on Recurrence	11
1.5	Thesis Organization	11
2	Logic of Recurrence	13
2.1	Syntax and Semantics	13
2.2	Interval Logic	14
2.2.1	Axiomatization	14
2.2.2	Extended definitions	15
2.2.3	Inferring interval relations	17
2.3	States, properties and events over time	19
2.3.1	Entity classes	20
2.3.2	Logical connections	22
2.3.3	Subsumption of Allen's HOLDS and OCCURS	23
2.3.4	Maximal Truth	24
2.4	Sequential composition	26

2.4.1	Reasoning about sequences	28
2.5	Recurrence	30
2.5.1	Occurrence from recurrence	31
2.6	Discussion	33
2.7	Summary	35
3	A Temporal Reasoning System	36
3.1	Introduction	37
3.2	The TimeLogic system	37
3.3	Interval Constraints	38
3.3.1	Specification of relational constraints	38
3.3.2	Inferring relational constraints	39
3.3.3	Specification of durational constraints	41
3.3.4	Inferring durational constraints	41
3.3.5	Inferring durational from relational constraints	42
3.3.6	Inferring relational from durational constraints	43
3.4	Interval Hierarchies	44
3.4.1	Constraint propagation: how bad does it get?	44
3.4.2	Limiting propagation: reference intervals	45
3.4.3	Automatic referencing	47
3.4.4	Empirical results	50
3.5	Contexts and Backtracking	51
3.5.1	Contexts	52
3.5.2	Backtracking	52
3.5.3	Auto-referencing and backtracking	52
3.6	The Tempos system	54
3.6.1	Intervals in Tempos	55
3.6.2	Constraints on Tempos intervals	55
3.7	Truth over time in Tempos	57
3.7.1	Truth Throughout	58
3.7.2	Verification of temporal constraints	59
3.7.3	Maximal Truth from Sequential Truth	60
3.7.4	Maximal Truth from Repeated Truth	61
3.8	Summary	62

4	Reasoning About Recurrence	64
4.1	Path Logic	66
4.1.1	Being places	66
4.1.2	Going places	68
4.2	The logic of traffic lights	71
4.3	Taking a trip	72
4.3.1	Driving through a traffic light	74
4.3.2	Driving past a traffic light	76
4.3.3	Driving from A to B past a traffic light	80
4.4	Summary	81
5	Conclusions & Future Work	83
5.1	Summary	83
5.2	Extending the temporal logic	84
5.2.1	Relative durations	84
5.2.2	Composite durations	86
5.2.3	Conventional time	88
5.2.4	Instantaneous events	88
5.3	Extending the recurrence logic	90
5.3.1	Relaxing restrictions on sequences	90
5.3.2	Relaxing restrictions on the Recurrence Axiom	90
5.3.3	Recurrence and full planning	91
5.4	Extending the spatial logic	91
5.4.1	What goes around...	91
5.4.2	Beyond one dimensional space	92
	Bibliography	93
A	TIMELOGIC Interface	98
A.1	Terminology and conventions	99
A.1.1	Properties	99
A.1.2	Contexts	99
A.1.3	Types	99
A.1.4	Errors	100
A.2	Initialization	100

A.3	Manipulating contexts	100
A.4	Manipulating intervals	101
A.5	Manipulating constraints	102
A.6	Tracing constraint propagation	103
A.7	Statistics	104
A.8	Customization	105
B	TEMPOS Interface	108
B.1	Integration of TimeLogic and Rhet	109
B.2	Initialization and Customization	109
B.3	Interval definition	110
B.4	Interval Predicates	110
B.4.1	Interval relation predicates	110
B.4.2	Interval duration predicates	112
B.4.3	Interval Skolemization	113
B.4.4	Contextual Reasoning	113
B.5	Truth over Time	113
B.5.1	Inferred Truth over Time	114
C	Automatic Referencing Algorithm	117
C.1	Reducing the branching factor	118
C.2	Constructing the reference hierarchy	119
C.3	Asserting relations	121
C.4	Propagating equality	121

List of Tables

3.1	Simple interval relations in TIMELOGIC	40
3.2	Compound interval relations in TIMELOGIC	40
3.3	Durational constraint examples	42
3.4	Relational to durational constraint map	43
3.5	Durational to relational constraint map	43
3.6	Propagation statistics on 101 randomly related intervals	51
B.1	Specializations of the TIME-RELN predicate	111
B.2	Specializations of the TIME-DURN predicate	112

List of Figures

1.1	Temporal layout of the cup drinking rule	7
3.1	The 13 possible relational constraints between two intervals	39
3.2	Example of interval reference structure	46
3.3	Algorithm for constructing the reference hierarchy	49
3.4	Algorithm for reducing the reference hierarchy branching factor .	50
4.1	Temporal layout of the trip to the store	70
4.2	One possible temporal layout, second half of the proof of Theo- rem 4.22	75
4.3	One possible temporal layout of the robot delivery problem	80

Chapter 1

Introduction

The issue of representing and reasoning about time is receiving increased attention as researchers in Artificial Intelligence are recognizing the relevance, if not crucial importance of time in domains such as natural language processing, planning, robotics, problem solving, and expert systems. The models developed so far are usually based on either points or intervals, and primarily deal with linear sequences of events, sometimes allowing for some parallelism. Model complexity has ranged from simply using integers or reals as temporal indices onto a time line, to fairly elaborate formal logics where modal operators allow reasoning about future and past possibilities and necessities. The research described in this thesis concerns itself with an aspect of time barely touched upon in Artificial Intelligence, namely recurrence — the phenomenon of an event or state [or a sequence thereof] repeated periodically. Although prominent in daily life it has not been studied at all for its own sake in A.I. Yet from the specific vantage point taken in this work — the reasoning required of an intelligent agent to arrive at workable plans of action — situations and examples abound where the ability to represent and make use of explicit knowledge about cyclic events is crucial. For instance a space robot might reason about orbit-related events; a factory robot might need to know the cyclic patterns of components coming by on the conveyor belt; a domestic robot might have to deal with the demand to do the laundry every Monday; and a mobile robot might have to reason about traffic lights in order to navigate intersections.

1.1 Perspectives on time

A useful distinction is often made in the philosophical and psychological literature between three categories of time, namely natural, conventional and logical time. Cognition of natural time is based on phenomena occurring in the natural world

and includes the notions of night and day, the (lunar) month and the seasons. Conventional time refers to the temporal structure based on social conventions, some of which are derived from natural time, and includes seconds, minutes, hours and weeks (arbitrary conventions), and days, months and years (natural conventions). Logical time refers to the logico-mathematical structure of time and operations thereon, such as ordering and transitivity. In terms of this classification, most of our daily experience is guided by our knowledge of natural or conventional time, which is defined almost exclusively in terms of recurrence.

Recurrence is a crucial element of our “common sense” notion of time. From our earliest days we learn to perceive time as a result of two important cognitive abilities: the awareness of change in the world around us, and the ability to detect regularities in that change. Without change there is no awareness of anything, and without regularities in change there is awareness of chaos only. Perceived regularities form the underpinnings of the cognition of recurrence: the realization that the same state or event, or one sufficiently similar, is repeated regularly.

It is sobering to realize how basic the notion of recurrence is to our human perception and understanding of time, and how it has been largely ignored so far in both computational theories and practical A.I. systems. To date no attempt has been made to construct a computational theory or an A.I. system based on natural time. Presumably this is so because no A.I. system has been sophisticated enough to actually operate as an independent agent in “the real world” for any length of time for this to be an issue.

Some work has been done on the theory and on systems that capture the notion of conventional, or calendar time. These we will discuss briefly in Section 1.2.6. Most work in A.I., however, has concentrated on logical time. Many formalisms have been proposed, some of which we will discuss here.

There are two major foci in A.I. with respect to time:

- Defining a formalism so that knowledge of time gleaned from the *input* to an intelligent system can be represented. This is primarily the focus of computational linguistics, which attempts to find a representation of linguistic constructs such as the utterance “I hope to be home before it starts raining,” although some work in computational vision has also had this focus, attempting to extract temporal information from a series of images.
- Defining a formalism that is suitable for reasoning about time for the purpose of forming new knowledge (inference) or determining the *output* (actions) of an intelligent agent, such as deciding to go to the bank at a certain time

in order to get there before closing time. This is the focus of research in problem solving and planning.

Obviously, the ideal formalism, if it exists, fulfills both purposes. The research reported here falls solidly in the last category.

1.2 Representation of Time

1.2.1 Situations

The prototypical formalism that first incorporated the notion of temporal progression is the Situation Calculus [McCarthy and Hayes, 1969]. In this calculus, each description of an aspect of the world, called a *property*, is associated with a *situation*, with the intended meaning that the property is true or *holds* in that situation. For example, the fact that a cup *c* is full in situation *s*₁ is stated as

$$\text{FULL}(c, s_1)$$

In the Situation Calculus *actions* are functions that take a situation, among other arguments, and produce a new situation. Actions affect properties by means of *rules*, i.e., logical implications. For instance, the rule “If you drink any full cup in any situation, then that gets you to a situation in which the cup is empty” is stated as

$$\forall c, s. \text{FULL}(c, s) \supset \text{EMPTY}(c, \text{drink}(c, s))$$

The fact that situations are produced only by actions has a number of ramifications, namely

- There is a single agent acting in an otherwise completely static world — no other agents or external events can be accounted for;
- Situations chop the world up into discrete states, and nothing can be said about what happens during state transitions;
- Although series of actions provide a temporal “progression,” reasoning about time itself is not possible, as situations do not have any inherent structure.
- Continuous processes, or simultaneous or overlapping events cannot be represented.

Nevertheless, the Situation Calculus has been extremely influential, as it provided a starting formalism, and because it first gave rise to two major problems: the *Qualification Problem* and the *Frame Problem*.

1.2.2 Problems

The Qualification Problem [Ginsberg and Smith, 1988] stems from the impossibility, in general, of stating all conditions that could prevent an action from occurring. For instance, the above drinking rule failed to ensure that the cup did not have a lid on it, that the water in the cup wasn't frozen, that there was no poison in the cup that would have killed the drinker instantly, *etc.* As the Qualification Problem is a general problem, and dealing with it is beyond the scope of this thesis, we shall ignore it.

The Frame Problem stems from the difficulty in deciding what does *not* change because of an action, and can be illustrated as follows. Suppose in the previous example there were two cups, c_1 and c_2 , both full. If drinking c_1 in situation s_1 leads to situation s_2 , the drinking rule therefore implies that c_1 is empty in s_2 ; but what is the state of c_2 in s_2 ? The Frame Problem is a general problem, not specific to the Situation calculus, and has been the subject of extensive research [Haas, 1987; Pylyshyn, 1987; Hanks and McDermott, 1986; Hayes, 1973].

The Frame Problem is sometimes rephrased as the *Extended Prediction Problem*, also known as the *Persistence Problem* [Cohen and Levesque, 1987; Lifschitz, 1986]. Given that we know x is true in some situation or at some time, how long (in how many subsequent situations) will it stay true? The general belief is that, albeit elusive so far, a generic solution to this problem is possible. One particular attempt is noteworthy, namely *Chronological Ignorance* [Shoham, 1986]. This approach, incorporated in Dean's time map database manager [Dean, 1984], is characterized by the principle of least commitment, *i.e.*, everything stays the same until (just before) it is known otherwise. However, Kautz [1987] illustrated with a simple example¹ how this approach is inadequate and falls short of providing a general solution, and Weber [1989] and others have argued that any solution will of necessity be domain-dependent. We underwrite this view, and shall provide domain-specific persistence axioms where necessary.

1.2.3 Reification

It has become virtually standard practice to *reify* properties. Rather than representing properties using predicates (such as `EMPTY`) they are represented by domain objects and related to situations by a general predicate (such as `HOLDS` [Lifschitz, 1987]). This allows properties to be arguments to or constructed from

¹If I park my car in the parking lot at time 0 and look out the window at time 1000 to find it gone, is it reasonable to assume it was there until time 999?

functions, as well as be in relation to other objects, including other properties. whereas this is not possible with property predicates (in first order predicate calculus). For instance, the above drinking example would be stated in reified form as

$$\begin{aligned} & \text{HOLDS}(\text{full}(c), s_1) \\ & \forall c, s. \text{HOLDS}(\text{full}(c), s) \supset \text{HOLDS}(\text{empty}(c), \text{drink}(c, s)) \end{aligned}$$

Although at first appearance this increases the expressiveness of the formalism, Bacchus *et al.* [1989] present a convincing argument that reified logics are no more expressive (although perhaps more convenient), and in some case less so [Shoham, 1987], than using property predicates with a situational or temporal argument.

1.2.4 Points of Time

In order to reason about time itself, rather than time being implicit in the succession of situations, time must be represented explicitly. This is done by adding temporal objects to the domain and imposing a structure on them. A debate has been raging for some time what the basic elements of this structure should be: time *points* or *intervals*. Many formalisms have been based on a single time line made up of a totally ordered set of time points. A *discrete* model maps these time points onto the integers, which has the advantage that a successor function is available, but the disadvantage that there are “gaps” between the points. A *continuous* model maps the points to the reals, which is a more appropriate foundation for reasoning about continuous processes. However, there is no such thing as “the next point in time.”

A more sophisticated model allows multiple time lines. For instance, the model developed by McDermott [1982] uses the notion of *chronicles*, which are sets of time points belonging to a possible world. Hence his logic allows for hypothetical reasoning. Something may be true in one world (chronicle) at some point in time but not in other worlds.

Sentences describing states or properties in the world look very similar to those in the Situation Calculus, except that situational objects are replaced by time points. The fact that a cup is full at time τ_1 is written as $\text{HOLDS}(\text{full}(c), \tau_1)$. Reified actions, however, have a different interpretation: instead of being functions producing the next situation they merely produce a domain object that represents the action. A general predicate OCCURS relates such action objects to a time point, or rather, since actions generally take time, a pair of points. The above drinking rule is now stated as

$$\begin{aligned} \forall c, t, t'. \text{ HOLDS}(\text{full}(c), t) \wedge t < t' \wedge \text{OCCURS}(\text{drink}(c), t, t') \\ \supset \text{HOLDS}(\text{empty}(c), t') \end{aligned}$$

where t' is often obtained from the successor function in discrete models, or represented as some appropriate durational offset from t in continuous models.

Shoham [1987], in an attempt to provide a solid semantics to representing and reasoning about time, developed a point-based temporal logic where the structure of time was not axiomatically defined, as in many other systems, but model-theoretically. All predicates, variables and terms in his logic are *interpreted* with respect to two time points, bracketing the time over which the predicate evaluates to truth. For instance, the sentence

$$\text{COLOR}(\text{HOUSE}, \text{RED})[[t_1, t_2]]$$

is true if the tuple formed from the interpretation of `HOUSE` with respect to the interpretations of t_1 and t_2 , and the interpretation of `RED` with respect to the interpretations of t_1 and t_2 is in the relation denoted by the interpretation of `COLOR` with respect to the interpretations of t_1 and t_2 . Since the time dependency is specified semantically the syntax is rigid with respect to the temporal terms. Aside from the ordering and equality relations between time points built into the semantics, temporal terms cannot occur in arbitrary predicates, no atemporal (“eternal”) relations are possible (such as “individual `B12` is a `BLOCK`”), and statements involving more than two time points are awkward to phrase. Moreover, Bacchus *et al.* [1989] have shown that Shoham’s logic is completely subsumed by a normal first-order logic, retaining a standard semantics and proof theory.

A major difficulty with using pairs of points to denote the time over which a property is true or an event takes place, is deciding whether to include the endpoints, and if so, which ones. The danger is that one can end up with truth value “holes,” *i.e.*, points in time when something is neither true nor false, or with truth value contradictions, *i.e.*, points where something is both true and false.

1.2.5 Intervals of Time

Allen [1983] developed a temporal logic using intervals of time, foregoing the notion of points altogether, claiming that “instantaneous” events still require time and are instantaneous only due to our perspective: one can always “turn up the magnification.” For instance, in normal parlance “the light turns on” is an instantaneous event. However, to an electrical engineer studying filament longevity

Figure 1.1: Temporal layout of the cup drinking rule

various phases in temperature and electrical resistance are distinguishable within this event.

The interval logic has provided the foundation for a general theory of action and time [Allen, 1984; Allen and Hayes, 1985]. Properties, events, actions, and processes are reified and related to intervals with the predicates `HOLDS` and `OCCURS`, and hence possess an inherent continuity. Allen's interval calculus is based on the thirteen relations that can hold between any two intervals (equality, plus the relations before, meets, overlaps, starts, finishes, and during, as well as their inverses). Our water drinking example would be phrased in Allen's logic as

$$\begin{aligned} & \text{HOLDS}(\text{full}(c), i_1) \\ & \forall c, i, j. \text{HOLDS}(\text{full}(c), i) \wedge \text{MEETS}(i, j) \wedge \text{OCCURS}(\text{drink}(c), j) \\ & \supset \exists k. \text{MEETS}(j, k) \wedge \text{HOLDS}(\text{empty}(c), k) \end{aligned}$$

Again, sentences describing states or properties in the world look very similar to those in the Situation Calculus, but now the temporal terms refer to intervals. The drinking rule involves three intervals, and the association between them and the properties and event can be visualized as in Figure 1.1: Although the association of properties and events with intervals provide them with inherent continuity, the calculus admits a discrete model as long as one does not insist that all intervals have an infinite number of subintervals. However, because an interval can overlap or contain another, we only have a partial order defined over intervals.

A relation between two intervals can also be described as a disjunctive set of these basic relations, and his calculus provides a method for inferring the transitive relation set between two intervals i and k given the relations between i and j and between j and k .

1.2.6 Representing recurrence

The two primary temporal reasoning frameworks we have looked at above, McDermott's point-based logic and Allen's interval logic, both permit recurrence. Neither, however, have dealt with the issue. McDermott did provide a `WHILE`

predicate, but does not elaborate on how this might actually be used in reasoning.

The interval algebra has been extended to include *non-convex* intervals [Ladkin, 1986a; Ladkin, 1986b], which are arbitrary unions of maximally convex (*i.e.*, normal) subintervals. Ladkin defines operators to combine convex intervals into non-convex ones, to dissect non-convex intervals into sets of convex ones. Ladkin extends Allen's temporal relations, which he labels convex relations, by a number of relations on non-convex intervals in terms of convex relations. For instance,

$$\text{MOSTLY-}\mathcal{R}(i, j)$$

is a relation between two non-convex intervals which is true if, for every component of j , there is a component of i that is $\mathcal{R}$ to it, where $\mathcal{R}$ is one of the basic convex relations such as MEETS.

One major difficulty in using this extended algebra in a practical reasoning system is the combinatorial explosion of non-convex relations. As Ladkin showed, the number of relations between unions of convex intervals is at least exponential in the cardinality of the sets.

Regarding the representation of conventional, or calendar time, a promising proposal was made by Leban *et al.* [1986]. As part of a scheduling system they developed the notion of *collections* of intervals, which can be recursively built up from the most primitive collections called *calendars*. A calendar is a collection consisting of an infinite sequence of intervals that span the time line, each meeting the next in the sequence. Each interval is defined as a pair of points, Leban *et al.* explicitly ignore the issues of boundary inclusion and whether the underlying model is discrete or continuous. Two operations on collections are defined, namely *dicing* and *slicing*. Dicing is an extension of the relations on intervals. For each interval relation there are two dicing operators: strict ($: \mathcal{R} :$) and relaxed ($(\mathcal{R})$). The effect of a strict dicing operation on an interval i with respect to a collection c is to break up i according to c . For instance, given an interval *Year-1989* and a collection *Weeks*, the expression *Weeks* :overlaps: *Year-1989* will result in a collection of intervals, each a whole or partial week contained in the year 1989. The result of the relaxed operator would be a similar collection but containing only whole weeks, each having some or all time in common with 1989.

The slicing operation is used to pick certain components out of a collection in a regular fashion. For instance, the expression *5/Days* :during: *Weeks* refers to a collection consisting of the fifth day of every week. Calendars themselves are

specified using a simple notation: a (by definition) circular list started with a unit specification, followed by 1 or more repetition factors. For example,

$$Weeks \equiv \langle Days; 7 \rangle$$

indicates that the first interval in the *Weeks* calendar is the union of the first 7 intervals of the *Days* calendar; the second interval in the *Weeks* calendar is the union of the second 7 intervals of the *Days* calendar; *etc.*

This representation and construction of regular sets appears quite useful in construction of time schedules, *i.e.*, recurrent references to conventional time segments: every Thursday; the first Monday of every month; and so on. However, it is not clear how well suited the notion of interval collections is for reasoning about properties and events over time that do not have an explicit reference to some external reference point. In many cases a reasoner is more concerned with relative order and duration than with particular dates.

1.3 Planning in Time

Traditional planning systems have often operated in a timeless problem space. The prototypical system illustrating some of the features of these planning theories is the STRIPS system [Fikes and Nilsson, 1971]. STRIPS represents actions as three sets of atomic formulae: a set of preconditions which must be true before the action is executed, a set referred to as the add-list containing formulae to be added to the global database describing the world upon execution of the action, and a set called the delete-list whose members should be deleted from the global database. The world consists of a single agent who can apply operators (execute actions) to static objects in a serial fashion only. No external influences, forces, events, states or processes can be considered. As temporally impoverished this domain may be, STRIPS tackled difficult problems, and continues to inspire planning researchers. The three basic action conditions and effects can be found in most modern planning systems. Another principle that has found widespread usage is Fikes and Nilsson's solution to the Frame Problem, namely the STRIPS Assumption: assume that everything not mentioned explicitly in add- or delete-lists remains unchanged.

Hendrix [1973] described a system that cast all actions as processes with continuous effects on the environment. He allowed for events and processes external to the agent, both those that occur naturally and in spite of the agent, and those that the agent can initiate but which proceed independently, such as the filling of

a bucket once the agent places it under a running tap. Time is represented simply as a set of points on the real line, and numerical comparisons underly process scheduling in a simulator. Time elapses by the simulator moving on to the next “interesting” time. Hendrix did not describe any planning or reasoning system, but did provide a fairly rich domain description in which such a system might be developed.

The NOAH system [Sacerdoti, 1977] was one of the first planning systems that performed some primitive temporal reasoning. The planner described plans as sets of actions partially ordered in time. These sets were computed based on the principle of least commitment, which states that temporal orderings between actions are added only when such an ordering is required to avoid conflicts in achieving preconditions or maintaining action effects. If temporal ordering is added, it is a simple before/after relationship. The planner is hierarchical, so the principle of least commitment is important to avoid imposing temporal order prematurely. Temporal reasoning in NOAH is limited to resolving goal or resource conflicts, and is otherwise not a critical part of either the plan or the world description.

The planning system DEVISER [Vere, 1983] is able to reason about events occurring at certain (absolute) points on the time line, and about events occurring independent of the agent. In developing plans the system is able to take into account time constraints in the form of windows within which certain activities could or should be scheduled. Its major shortcomings include being restricted to the real time line which presents difficulties in hypothetical reasoning; having to deal with individual points in time; and not being able to reason about time explicitly, only as it relates to events. Although Vere’s system allows the posting of future external events, no model is defined for these future occurrences, and hence it is not possible to describe recurring events.

The power of a planning system based on the interval logic (and hence any system that fully incorporates time) has been demonstrated by Allen and Koomen [1983]. Their planning system successfully disambiguated the ordering of overlapping tasks on the basis of propagation of temporal constraints derived from transitivity. In the blocks world example described, a two-armed agent is required to switch the top two blocks of a stack with the interesting environmental constraint that there is not enough room to put a block on the table temporarily. Hence the actions of picking up the blocks, holding them and putting them down in the right order must be interleaved. Temporal constraints in the blocks world domain, such as “any interval during which $ON(block_1, block_2)$ is true must be disjoint from any interval during which $CLEAR(block_2)$ is true,” are sufficient to derive the correct plan for this task.

1.4 Planning on Recurrence

Recurrence in planning systems arises in two ways, namely the repetition of a sequence of states or events for a certain number of times (*iterative* recurrence), and repetition of states or events while or until some condition holds (*conditional* recurrence). A number of formalisms have made use of *regular expressions* to denote recurrence [Cohen and Levesque, 1987; Lansky, 1987; Lansky, 1988]. In particular, In Cohen and Levesque's framework the two operators $*$ and $+$, when applied to action terms, indicate a repetition of 0 or more and 1 or more occurrences, respectively. These operators apply only to events, however, and not to states or properties. Moreover, in their framework time is embedded in the semantics and is not explicitly represented.

Lansky's GEM system is primarily event-based, where time is not accorded full citizen status. Events do, however, have temporal constraints associated with them through the use of modal operators. In addition to the standard monadic operators $\Box$ (*Henceforth*) and $\Diamond$ (*Eventually*) the diadic operator $P \cup Q$ is defined as P *until* Q . Although at least conditional repetition is clearly representable, it cannot be done with explicit access to the time of repetition. Lansky does not demonstrate the use of this operator in reasoning.

Miller [1985] is one of the first and few to deal explicitly with recurrence in a planning system. In his system, the planner can describe the repetition of some subtask for a known number of times, *e.g.*, painting and putting away 10 widgets. A plan incorporating iteration is represented as a list of actions to be executed by the agent which contains a special marker, or "meta-action," indicating the return to a previous action in the list. Unfortunately, the temporal logic underlying his system for planning through simulation is somewhat impoverished. Additionally, Miller's focus is primarily on scheduling and resource allocation, he does not deal with the possibility of external events and processes, and he makes no claim to developing a theory of recurrence.

1.5 Thesis Organization

Chapter 2 presents a formal system for representing and reasoning about recurrence. It is based on Allen's interval calculus, which we include in our axiomatization. Two predicates, TT (*True Throughout*) and MT (*Maximally True*) are developed in order to associate *entities* (events, properties, processes, *etc.*) with temporal intervals. We define a taxonomy of such entities (a subset of a taxonomy

presented by Shoham [1987]) to differentiate the temporal behavior of entities over time. The predicates together with the taxonomy are shown to subsume Allen's `HOLDS` and `OCCURS` predicates. We then deal with the issue of sequential composition of entities and their truth over time. And finally, we axiomatize the predicate `RT` (*Repeatedly True*) which repeatedly maps entities, including sequences of entities, over time.

In Chapter 3 we describe the implementation of two systems, namely `TIMELOGIC` and `TEMPOS`. The `TIMELOGIC` system is a generic database manager and forward chaining inference engine of interval relations and durations. One particular noteworthy feature of this system is a mechanism for automatically creating and maintaining a hierarchy of intervals, thereby localizing constraints and constraint propagation, resulting in a more efficient system.

The `TEMPOS` system is an implementation of a restricted version of the formalism developed in Chapter 2. The restriction lies in the fact that `TEMPOS` is layered on top of the *Rhetorical* system, which is a Horn-clause based knowledge representation and reasoning system developed locally. We will discuss the limitations imposed on the recurrence axioms by fitting them into a Horn-clause formulation instead of general first order predicate calculus.

The formalism developed in Chapter 2 is applied in Chapter 4 to the domain of traffic control lights. An extensive example is developed, showing how the use of recurrence can enable a planner to determine that a plan to travel from A to B is feasible, even if an intervening traffic light is red at the time it is encountered. In order to develop this example, a novel application of the interval logic to the spatial domain is presented, which allows us to reason about trajectories and their traversal over time. The combination of paths and temporal intervals, together with the recurrence axioms of Chapter 2 and actions such as `traverse`, allow us to prove several theorems about the use of traffic lights, and how an agent can plan to get passed them.

Finally, Chapter 5 provides a summary of this work, and sketches interesting avenues of further research. In particular, we discuss a number of extensions to the repertoire of temporal objects and predicates, such as incorporating time points and *moments* (minimal intervals). We will also look at various restrictions imposed on the recurrence axioms and how they might be lifted or eased. A major extension of the current work to be discussed is how durational reasoning might fit in, and why it would be important to do so.

Chapter 2

Logic of Recurrence

2.1 Syntax and Semantics

The logic of recurrence we will develop in this chapter and use in subsequent chapters is based on standard sorted first-order predicate calculus with equality. Well-formed formulas are constructed, in the usual way, by combining atomic sentences with the logical operators $\forall$ and $\exists$ (*universal* and *existential quantification*), $\neg$ (*negation*), $\wedge$ (*conjunction*), $\vee$ (*disjunction*), $\otimes$ (*exclusive disjunction*), $\supset$ (*implication*), and $\iff$ (*biconditional*). An atomic sentence is formed from an n -ary relation constant ρ and n terms $\tau_1, \dots, \tau_n$ by writing them as $\rho(\tau_1, \dots, \tau_n)$. Although not strictly necessary, typographic conventions facilitate distinguishing predicates (P) from individuals (P), variables (p) and functions (p). Hence, an atomic formula may look like

$$\forall x \exists y. \text{PREDICATE}(x, \text{function}(y, \text{INDIVIDUAL93}))$$

Equality will be written in infix form. Unless obvious from context or by variable naming convention, variables will be typed using a $*$ notation, where

$$\begin{aligned} \forall \nu * \rho. \phi &\stackrel{\text{def}}{=} \forall \nu. \rho(\nu) \supset \phi \\ \exists \nu * \rho. \phi &\stackrel{\text{def}}{=} \exists \nu. \rho(\nu) \wedge \phi \end{aligned}$$

The interpretation of formulas is provided by a model, which maps constant terms in the language to individuals, functional terms to functions from tuples of individuals to individuals, and predicates to sets of tuples of individuals. The mapping of free variables in sentences to individuals, and the interpretation of quantification and propositional connectives are likewise standard. $(\phi \otimes \theta)$ is interpreted as $(\phi \wedge \neg \theta) \vee (\theta \wedge \neg \phi)$. Proofs in this thesis will use natural deduction, using standard rules and transformations only.

2.2 Interval Logic

In order to represent time we shall make use of Allen's interval logic [Allen, 1983], which is a logic of linear time. Intervals of time are individuals in a sort called *times*, distinguished by the predicate **TIME**. For ease of reading we shall use the convention that the (possibly subscripted) variables i, j, k and l are implicitly typed with the **TIME** predicate.

2.2.1 Axiomatization

The complete interval logic can be defined by five axioms based on the primitive relation **MEETS** on **TIME** individuals [Allen and Hayes, 1985; Allen and Hayes, 1987]. These five axioms are included here for completeness and notational consistency:

Axiom 2.1 (Meets Unique) *Intervals meet in unique places, i.e., if two intervals meet a third, then any interval met by one must also be met by the other (Allen & Hayes' axiom M1):¹*

$$\forall i, j, k, l. \text{MEETS}(i, j) \wedge \text{MEETS}(i, k) \wedge \text{MEETS}(l, j) \\ \supset \text{MEETS}(l, k)$$

Axiom 2.2 (Total Order) *Meeting places are totally ordered, i.e., given two places where intervals meet then these places are either equal or one precedes the other (Allen & Hayes' axiom M2):*

$$\forall i, j, k, l. \text{MEETS}(i, j) \wedge \text{MEETS}(k, l) \supset \\ \text{MEETS}(i, l) \\ \otimes [\exists m. \text{MEETS}(i, m) \wedge \text{MEETS}(m, l)] \\ \otimes [\exists m. \text{MEETS}(k, m) \wedge \text{MEETS}(m, j)]$$

Axiom 2.3 (Continuity) *The universe of intervals is continuous, i.e., every interval has two intervals on either side (Allen & Hayes' axiom M3):*

$$\forall i \exists j, k. \text{MEETS}(j, i) \wedge \text{MEETS}(i, k)$$

Axiom 2.4 (Composition) *Given two adjacent intervals there exists a third interval which is the union of the first two (Allen & Hayes' axiom M4):*

$$\forall i, j. \text{MEETS}(i, j) \supset \\ \exists k, l, m. \text{MEETS}(k, i) \wedge \text{MEETS}(j, m) \\ \wedge \text{MEETS}(k, l) \wedge \text{MEETS}(l, m)$$

¹M1-M5 numbering according to [Allen and Hayes, 1987]

Axiom 2.5 (Intervals Unique) *There is only one interval between any two meeting places (Allen & Hayes' axiom M5):*

$$\begin{aligned} \forall i, j, k, l. \text{ MEETS}(i, j) \wedge \text{MEETS}(j, l) \\ \wedge \text{MEETS}(i, k) \wedge \text{MEETS}(k, l) \\ \supset j = k \end{aligned}$$

Note that we have a Composition axiom but no Decomposition axiom. Not insisting that intervals can be arbitrarily decomposed allows us to adopt the notion of *moments* [Koomen, 1986; Allen and Hayes, 1987]. Although not used in this thesis, the use of moments can be a powerful addition to any reasoning system due to the added temporal constraints imposed by the semantics of moments.

2.2.2 Extended definitions

Although the MEETS relation is sufficient, it is convenient to extend the vocabulary by defining other basic relations on intervals, as well as by introducing a convenient shorthand for disjunctions of basic relations.

2.2.2.1 Basic interval relations

In addition to the MEETS relation as defined above, we introduce the basic interval predicates BEFORE, EQUALS, STARTS, OVERLAPS, DURING and FINISHES. As Allen and Hayes showed, these can all be defined in terms of the MEETS relation (see also Section 3.3):

$$\forall i, j. \text{ BEFORE}(i, j) \iff \exists k. \text{ MEETS}(i, k) \wedge \text{MEETS}(k, j)$$

$$\begin{aligned} \forall i, j. \text{ EQUALS}(i, j) \iff \exists k_1, k_2. \text{ MEETS}(k_1, i) \wedge \text{MEETS}(i, k_2) \\ \wedge \text{MEETS}(k_1, j) \wedge \text{MEETS}(j, k_2) \end{aligned}$$

$$\begin{aligned} \forall i, j. \text{ STARTS}(i, j) \iff \exists k_1, k_2, k_3. \text{ MEETS}(k_1, i) \wedge \text{MEETS}(i, k_2) \\ \wedge \text{MEETS}(k_1, j) \wedge \text{MEETS}(j, k_3) \\ \wedge \text{MEETS}(k_2, k_3) \end{aligned}$$

$$\begin{aligned} \forall i, j. \text{ FINISHES}(i, j) \iff \exists k_1, k_2, k_3. \text{ MEETS}(k_2, i) \wedge \text{MEETS}(i, k_3) \\ \wedge \text{MEETS}(k_1, j) \wedge \text{MEETS}(j, k_3) \\ \wedge \text{MEETS}(k_1, k_2) \end{aligned}$$

$$\begin{aligned} \forall i, j. \text{DURING}(i, j) &\iff \\ &\exists k_1, k_2, k_3, k_4. \text{MEETS}(k_1, k_2) \wedge \text{MEETS}(k_3, k_4) \\ &\quad \wedge \text{MEETS}(k_1, j) \wedge \text{MEETS}(j, k_4) \\ &\quad \wedge \text{MEETS}(k_2, i) \wedge \text{MEETS}(i, k_3) \end{aligned}$$

$$\begin{aligned} \forall i, j. \text{OVERLAPS}(i, j) &\iff \\ &\exists k_1, k_2, k_3, k_4, k_5. \text{MEETS}(k_1, k_2) \wedge \text{MEETS}(k_2, k_3) \\ &\quad \wedge \text{MEETS}(k_3, k_4) \wedge \text{MEETS}(k_4, k_5) \\ &\quad \wedge \text{MEETS}(k_1, i) \wedge \text{MEETS}(i, k_4) \\ &\quad \wedge \text{MEETS}(k_2, j) \wedge \text{MEETS}(j, k_5) \end{aligned}$$

Given the above definition of EQUALS and Axiom 2.5, it is obvious that

$$\forall i, j. \text{EQUALS}(i, j) \iff i = j$$

The definitions of the inverse relations, such as MET-BY(j, i), are omitted, since they are obvious and also not used in this text.

2.2.2.2 Compound interval relations

Often the relation between two intervals is only known as a disjunction of basic interval relations. For the sake of brevity and to improve readability we will define the infix notation $i \mathcal{R} j$ as the disjunction² of all the basic interval relations in the set $\mathcal{R}$ between i and j . Possible elements of $\mathcal{R}$ and their corresponding relations are as follows (overstriked elements indicate the inverse relation):

a	AFTER	(same as $\overline{b}$)	f	FINISHES
b	BEFORE	(same as $\overline{a}$)	m	MEETS
c	CONTAINS	(same as $\overline{d}$)	o	OVERLAPS
d	DURING	(same as $\overline{c}$)	s	STARTS
e	EQUALS			

For example, $i \{a, b, e\} j$ is equivalent to

$$\text{AFTER}(i, j) \vee \text{BEFORE}(i, j) \vee \text{EQUALS}(i, j)$$

A number of predicates, useful in subsequent proofs, are defined using this set notation. The *Containment* predicates define how one interval is contained, inclusively or properly, in another:³

²Actually, since the interval logic axioms entail that intervals can be related to one another by one and only one of the basic relations, the definition could be stated more strongly using exclusive or's. However, the weaker definition suffices.

³In general, any predicate " ρ !" is a stronger version of the corresponding predicate " ρ ".

$$\forall i, j. \text{ WITHIN}(i, j) \iff i \{e, d, f, s\} j$$

$$\forall i, j. \text{ WITHIN!}(i, j) \iff i \{d, f, s\} j$$

The *Disjoint* predicate defines how two intervals are temporally disjoint, *i.e.*, do not share any common time:

$$\forall i, j. \text{ DISJOINT}(i, j) \iff i \{a, b, m, \overline{m}\} j$$

When an interval i either starts later than or starts at the same time and ends later than an interval j , we shall call it the *LATER* one:

$$\forall i, j. \text{ LATER}(i, j) \iff i \{a, d, e, f, \overline{m}, \overline{o}, \overline{s}\} j$$

An important notion is that of interval *cover*. Interval k covers intervals i and j if and only if i starts k , and j finishes k (and i meets j , in the strict case). Note that, in the strict case, the Composition axiom (Axiom 2.4) ensures that such a cover k exists if i and j exist; the existence of a cover using the weaker formulation (in which $i \{b, m, o\} j$) is easily proved by case analysis.

$$\forall i, j, k. \text{ COVER}(i, j, k) \iff \text{ STARTS}(i, k) \wedge \text{ FINISHES}(j, k)$$

$$\forall i, j, k. \text{ COVER!}(i, j, k) \iff \text{ COVER}(i, j, k) \wedge \text{ MEETS}(i, j)$$

2.2.3 Inferring interval relations

For any three intervals, given the relations r_1 and r_2 between any two pairs, we can infer a maximal relation set s_3 between the third pair, *i.e.*, the actual relation r_3 between the intervals of the third pair is an element of s_3 , all relations in s_3 are possible *given only the relations between the first two pairs*, and any relation not in s_3 is impossible. Allen [1983] showed how this can be extended to accommodate sets of relations s_1 and s_2 between the first two pairs, by making s_3 the union of all the sets inferred from considering each r_i in s_1 and r_j in s_2 separately (see also Section 3.3.2).

All such transitive relationships between intervals can be stated as theorems in our logic, for instance

$$\forall i, j, k. \text{ BEFORE}(i, j) \wedge \text{ BEFORE}(j, k) \supset \text{ BEFORE}(i, k)$$

and can be derived from the definitions of the basic interval relations and the axioms of Section 2.2.1. In particular, subsequent proofs will make use of the following theorems:

Theorem 2.6 *The WITHIN predicate is transitive:*

$$\forall i, j, k. \text{WITHIN}(i, j) \wedge \text{WITHIN}(j, k) \supset \text{WITHIN}(i, k)$$

Theorem 2.7 *The LATER predicate is transitive:*

$$\forall i, j, k. \text{LATER}(i, j) \wedge \text{LATER}(j, k) \supset \text{LATER}(i, k)$$

Theorem 2.8 *For any intervals i , j and k , i is later than k if i is contained in j and j is later than k :*

$$\forall i, j, k. \text{WITHIN}(i, j) \wedge \text{LATER}(j, k) \supset \text{LATER}(i, k)$$

Theorem 2.9 *For any intervals i , j and k , i is later than k if i is later than j and j is contained in k :*

$$\forall i, j, k. \text{LATER}(i, j) \wedge \text{WITHIN}(j, k) \supset \text{LATER}(i, k)$$

Theorem 2.10 *For any two overlapping intervals i and j there exists an interval k such that k is a subinterval of both i and j :*

$$\forall i, j. \text{OVERLAPS}(i, j) \supset \exists k. \text{WITHIN}(k, i) \wedge \text{WITHIN}(k, j)$$

Theorem 2.11 *For any subinterval i of j there exists an interval k such that k is later than i and finishes j :*

$$\forall i, j. \text{WITHIN!}(i, j) \supset \exists k. \text{LATER}(k, i) \wedge \text{FINISHES}(k, j)$$

Theorem 2.12 *For any two intervals i and j finishing interval k there exists an interval l such that l is a subinterval of both i and j :*

$$\begin{aligned} \forall i, j, k. \text{FINISHES}(i, k) \wedge \text{FINISHES}(j, k) \supset \\ \exists l. \text{WITHIN}(l, i) \wedge \text{WITHIN}(l, j) \end{aligned}$$

Theorem 2.13 *If an interval k is covered by intervals i and j , then any subinterval l of k is either a subinterval of i or of j or it straddles i and j :*

$$\begin{aligned} \forall i, j, k, l. \text{COVER!}(i, j, k) \wedge \text{WITHIN}(l, k) \supset \\ \text{WITHIN}(l, i) \vee \text{WITHIN}(l, j) \vee \text{OVERLAPS}(i, l) \end{aligned}$$

The proofs of these theorems are omitted as they are straightforward and add little to the understanding of the topic at hand.

2.3 States, properties and events over time

We now posit a sort of individuals in the ontology which we shall refer to as *entities* and which will be distinguished by the predicate `ENTITY`. Entities are intended to correspond to the usual notions of states, events, processes, *etc.* We shall use the convention that variables v , w , x , y and z are implicitly typed with the `ENTITY` predicate. We do not make the usual distinctions between events and states or properties, as we are only interested in their temporal aspects. A more traditional classification, such as in [Allen, 1984], is not precluded by our approach (as is shown in Section 2.3.3), but it is unnecessary in this thesis.

We shall relate these atemporal entities to time, *i.e.*, they are said to be true (“hold” or “occur”) over intervals of time, using the predicate `TT` (“True Throughout”). We shall take

$$\text{TT}(i, x)$$

to mean “ x is true over interval i .” Note that this does not preclude x from being true on either or both sides of i , nor does this say anything about the truth of x over subintervals of i . So, for example, a fact like “It is winter in January 1989” is represented as

$$\text{TT}(\text{JAN-89}, \text{WINTER}) \tag{2.1}$$

It should be clear that entities represent *types* in the sense of sets of event or state occurrences, *i.e.*, `WINTER` may appear in `TT` statements with different intervals. Others [Kautz, 1987; Pelavin and Allen, 1986] have argued that it is better to reason about event *instances* in the form of tokens because (a) their existence can be asserted without completely specifying all their parameters (as opposed to using a functional construction of parameterized types), and (b) they allow multiple distinct instances of the same abstract event type to occur simultaneously [Kautz, 1987, page 29].

The first reason is not really an issue — the use of functional construction is cleaner when only few parameters are needed, as is the case in this thesis. But more complicated cases could be accommodated by adopting a “token-like” formulation. For instance, instead of

$$\text{TT}(\text{T44}, \text{readbook}(\text{FRED})) \tag{2.2}$$

we could say

$$\text{READBOOK}(\text{ENTITY17}) \wedge \text{agent}(\text{ENTITY17}) = \text{FRED} \wedge \text{TT}(\text{T44}, \text{ENTITY17}) \quad (2.3)$$

This kind of formulation is indeed more flexible, but it is no more expressive than functional construction, and more verbose when only few parameters are actually specified.

The second reason for using tokens rather than timeless event types, namely multiple, simultaneous occurrences of the same abstract type, is harder to counter. We agree that dealing with simultaneity of this kind indeeds mandates the use of tokens. However, and most importantly, it is exactly the use of timeless entities that will later enable us to reason explicitly about recurrence *without* reference to a (possibly infinite) set of tokens. A token-based system would need to define another universe of individuals corresponding to sets of tokens, and provide ways of mapping from one to the other. Kautz' method of event composition would not suffice.

2.3.1 Entity classes

Certain inferences regarding the truths of entities over several intervals are possible on the basis of entity *classes*.⁴ Shoham [1987] introduced a comprehensive as well as intuitively appealing taxonomy of objects, distinguishing the various ways in which truth distributes over time intervals. We will axiomatize here the subset of his taxonomy that makes sense in an interval-based logic, namely the classes *downward hereditary*, *upward hereditary*, *liquid*, *concatenable*, *gestalt* and *solid*.

Axiom 2.14 (Downward Hereditary) *An entity x is in the class downward hereditary if and only if x being true over interval i implies that it is true over each subinterval of i :*

$$\forall x. \text{DOWNWARD}(x) \iff [\forall i, j. \text{WITHIN}(i, j) \wedge \text{TT}(j, x) \supset \text{TT}(i, x)]$$

Axiom 2.15 (Upward Hereditary) *An entity x is in the class upward hereditary if and only if x being true over each subinterval i of j implies that it is true over interval j :*

$$\forall x. \text{UPWARD}(x) \iff [\forall i, j. [\text{WITHIN}(i, j) \supset \text{TT}(i, x)] \supset \text{TT}(j, x)]$$

⁴We use the label “class” rather than “type” so as not to confuse them with the types of the previous section, or the sorts in the ontology.

Axiom 2.16 (Liquid) *An entity x is in the class liquid if and only if it is both upward and downward hereditary:*

$$\forall x. \text{LIQUID}(x) \iff \text{UPWARD}(x) \wedge \text{DOWNWARD}(x)$$

Axiom 2.17 (Concatenable) *An entity x is in the class concatenable if and only if x being true over two meeting intervals i and j implies that it is true over the minimal covering interval k :*

$$\forall x. \text{CONCATENABLE}(x) \iff [\forall i, j, k. \text{TT}(i, x) \wedge \text{TT}(j, x) \wedge \text{COVER!}(i, j, k) \supset \text{TT}(k, x)]$$

Axiom 2.18 (Gestalt) *An entity x is in the class gestalt if and only if x being true over interval i implies that it is not true over any subinterval j of i :*

$$\forall x. \text{GESTALT}(x) \iff [\forall i, j. \text{TT}(i, x) \wedge \text{WITHIN!}(j, i) \supset \neg \text{TT}(j, x)]$$

Axiom 2.19 (Solid) *An entity x is in the class solid if and only if x being true over intervals i and j implies that i and j are equal or disjoint:*

$$\forall x. \text{SOLID}(x) \iff [\forall i, j. \text{TT}(i, x) \wedge \text{TT}(j, x) \supset \text{EQUALS}(i, j) \vee \text{DISJOINT}(i, j)]$$

To Shoham's taxonomy we add one useful diadic predicate to capture the notion of *mutual exclusion*. Knowing that two things can never be true simultaneously is a powerful constraint that can be used to great advantage by constraint-driven systems such as planners and schedulers.

Axiom 2.20 (Mutual Exclusion) *Two entities x and y are mutually exclusive if and only if x being true over interval i and y being true over interval j implies that i and j are disjoint:*

$$\forall x, y. \text{MUTEX}(x, y) \iff [\forall i, j. \text{TT}(i, x) \wedge \text{TT}(j, y) \supset \text{DISJOINT}(i, j)]$$

2.3.1.1 An example of class-based temporal inference

The class axioms 2.14 through 2.19 enable us to make inferences regarding the truth of entities over various intervals based on the temporal relations between these intervals. For instance, we can now prove that it is winter on 18 January 1989, assuming that it is winter in January 1989, winter is in the class Liquid, and 18 January 1989 is a subinterval of January 1989:

$$\begin{aligned} & \text{TT}(\text{JAN-89}, \text{WINTER}) \wedge \text{LIQUID}(\text{WINTER}) \wedge \text{DURING}(\text{18-JAN-89}, \text{JAN-89}) \\ & \supset \text{TT}(\text{18-JAN-89}, \text{WINTER}) \end{aligned} \tag{2.4}$$

Proof

- 1: TT(JAN-89, WINTER) ; hypothesis
- 2: LIQUID(WINTER) ; hypothesis
- 3: DURING(18-JAN-89, JAN-89) ; hypothesis
- 4: DOWNWARD(WINTER) ; MP(2)(A2.16)
- 5: WITHIN(18-JAN-89, JAN-89) ; (3)+def'n(WITHIN)
- 6: DOWNWARD(WINTER) $\wedge$ WITHIN(18-JAN-89, JAN-89) ; conj.intro.(4)(5)(1)
 $\wedge$ TT(JAN-89, WINTER)
- 7: TT(18-JAN-89, WINTER) ; MP(6)(A2.14)

■

2.3.2 Logical connections

As we have indicated earlier, we will use functions to construct entities based on a number of parameters. There is an obvious temptation to define functions that mirror logical connectives in order to simulate logical inference. For instance, Allen [1984] defines the property functions **and** and **not**, and derives **or**. The function **and** is defined easily enough:⁵

$$\forall i, x, y. \text{TT}(i, \text{and}(x, y)) \iff \text{TT}(i, x) \wedge \text{TT}(i, y)$$

However, negation presents problems. First of all, negation does not move freely through the TT predicate, as conjunction does: $\text{TT}(i, \text{not}(x))$ is *not* the same as $\neg \text{TT}(i, x)$ since the latter still admits $\text{TT}(j, x)$ for some subinterval j of i !

Secondly, ensuring $\text{TT}(i, \text{not}(\text{not}(x))) \iff \text{TT}(i, x)$ requires a clumsy, doubly nested definition of TT with respect to subintervals. Such a definition is not compatible with the taxonomy described above, as it would be correct only for liquid entities.

In order to reason about negated entities, we instead introduce the predicate FT (“False Throughout”), and axiomatize negation of entities in terms of it and the TT predicate:

Axiom 2.21 *Nothing is ever true and false simultaneously:*

$$\forall i, j, x. \text{TT}(i, x) \wedge \text{FT}(j, x) \supset \text{DISJOINT}(i, j)$$

⁵Perhaps a more appropriate name for **and** is **sim**, indicating the fact that the argument entities are true simultaneously. Note also that this definition is a specialization of Allen’s event function **composite**. The function **seq**, introduced in Section 2.4, is another one.

Axiom 2.22 *If x is not true over interval i then it must be false over at least one subinterval, and vice versa:*

$$\forall i, x. \neg \text{TT}(i, x) \supset \exists j. \text{WITHIN}(j, i) \wedge \text{FT}(j, x)$$

$$\forall i, x. \neg \text{FT}(i, x) \supset \exists j. \text{WITHIN}(j, i) \wedge \text{TT}(j, x)$$

Axiom 2.21 has an obvious corollary:

Theorem 2.23 *Being true implies not being false, and vice versa:*

$$\forall i, x. \text{TT}(i, x) \supset \neg \text{FT}(i, x)$$

Proof

By contradiction:

$\neg [\text{TT}(i, x) \supset \neg \text{FT}(i, x)]$ is equivalent to $[\text{TT}(i, x) \wedge \text{FT}(i, x)]$, which by Axiom 2.21 implies $\text{DISJOINT}(i, i)$, which by the basic interval axioms is a contradiction. ■

Note that this is an implication, and not a biconditional. If x is not true throughout i it does not follow that it is false throughout i — it might be true throughout some subinterval of i . We can make a stronger claim than in the theorem above if we know that two entities are mutually exclusive.

Axiom 2.24 (Mutual Exclusion II) *Two entities x and y are mutually exclusive if and only if x being true over interval i implies y being false over i :*

$$\forall x, y. \text{MUTEX}(x, y) \iff [\forall i. \text{TT}(i, x) \supset \text{FT}(i, y)]$$

2.3.3 Subsumption of Allen's HOLDS and OCCURS

Allen's primitive axioms over properties and events [Allen, 1984] can be cast in our current terminology quite easily:⁶

$$\forall i, x. \text{HOLDS}(i, x) \iff \text{TT}(i, x) \wedge \text{PROPERTY}(x)$$

$$\forall i, x. \text{OCCURS}(i, x) \iff \text{TT}(i, x) \wedge \text{EVENT}(x)$$

⁶We reverse the order of the time and property/event entities in Allen's presentation for the sake of consistency in our presentation.

where the predicates PROPERTY and EVENT are defined as:

$$\forall x. \text{PROPERTY}(x) \supset \text{LIQUID}(x) \wedge \text{CONCATENABLE}(x)$$

$$\forall x. \text{EVENT}(x) \supset \text{SOLID}(x) \wedge \text{GESTALT}(x)$$

Incorporating Allen's notion of processes requires more care. Allen denotes a process occurring over time simply by associating the corresponding event with that time using the predicate OCCURRING. That would not work in our logic since we classify event entities as solid, which implies they cannot be true over subintervals. Allen claims that processes occurring over interval i fall between properties and events with respect to how they distribute over time, as they are true over some subintervals of i but not necessarily over *all* subintervals of i [Allen, 1984, page 132]. This we consider vacuous, as it just depends on the definition of the event or process. If the walk event can mean either moving or pausing to smell the flowers, but the walking process must refer strictly to moving, then we are comparing apples and oranges. We claim that any process that reflects an event occurring must have the same meaning as the event and hence is true over all subintervals of the interval over which the event occurs. Therefore processes behave like properties with respect to time. Since properties differ from events in the way they distribute over time, we cannot simply use the event entity. Instead we introduce the function **prog** [Galton, 1987] which, when applied to an event entity, produces the progressive form of the event. The axiom

$$\forall x. \text{EVENT}(x) \supset \text{PROPERTY}(\text{prog}(x))$$

ensures that the progressive has the appropriate temporal behavior. We can then define Allen's OCCURRING as

$$\begin{aligned} \forall i, x. \text{OCCURRING}(i, x) &\iff \\ &\exists j, y. \text{TT}(i, x) \wedge \text{WITHIN}(i, j) \wedge x = \text{prog}(y) \wedge \text{OCCURS}(j, y) \end{aligned}$$

2.3.4 Maximal Truth

Just as the axiomatization of the interval logic prevents infinite intervals, we will constrain the entity logic by insisting that there are no time-dependent but infinite truths, *i.e.*, for every truth assertion there is a corresponding maximal truth assertion. This is a useful property of our logic, primarily due to Axiom 2.26 below: any interval over which an entity x is true is either contained in or

disjoint from any interval over which x is maximally true, *i.e.*, no overlaps. Allen-type events, which are **SOLID**, are by definition true over their maximal intervals only, but extending this to arbitrary entities gives us added power: the additional constraint often enables temporal inferences which otherwise would have been impossible. This effect will be illustrated below.

We shall take

$$MT(i, x)$$

to mean “ x is true over interval i , and it is not true over any interval j which contains or intersects with i .” For example,

$$MT(\tau_{23}, \text{eat}(\text{JOHN}, \text{DINNER})) \quad (2.5)$$

represents the fact that John ate dinner at time τ_{23} and that he did not eat dinner at any time overlapping τ_{23} .

Formally, the following three axioms define the **MT** predicate:

Axiom 2.25 *Entity x is true over interval i if it is maximally true over i :*

$$\forall i, x. MT(i, x) \supset TT(i, x)$$

Axiom 2.26 *Any interval over which an entity x is true is either contained in or disjoint from any interval over which x is maximally true:*

$$\forall i, j, x. TT(i, x) \wedge MT(j, x) \supset \text{WITHIN}(i, j) \vee \text{DISJOINT}(i, j)$$

A special case of this is two maximally true intervals. If they are not disjoint then each has to be within the other, *i.e.*, they must be equal.

Theorem 2.27 *All intervals i and j over which an entity x is maximally true are either equal or disjoint:*

$$\forall i, j, x. MT(i, x) \wedge MT(j, x) \supset \text{EQUALS}(i, j) \vee \text{DISJOINT}(i, j)$$

Axiom 2.28 *For every interval i over which an entity x is true there exists a containing interval j over which x is maximally true:*

$$\forall i, x. TT(i, x) \supset [\exists j. \text{WITHIN}(i, j) \wedge MT(j, x)]$$

Obviously, a similar set of axioms can be used to defined the predicate **MF** (“Maximally False”) by substituting **FT** for all occurrences of **TT**.

Note that we explicitly allow for an entity to be maximally true over two intervals that meet (see the definition of the DISJOINT predicate). As a consequence it is not true in general that every MT interval must be followed by a FT one. This is especially easy to see in the case of events: it is not uncommon that, for instance, the event “eat a potato chip” (which by definition occurs over a maximal interval) is immediately followed by another such event.⁷ However, it is true for concatenable entities (due to the class axioms), and hence for properties.

Theorem 2.29 *Concatenable entities alternate maximal truth and falsehood:*

$$\forall i, x. \text{MT}(i, x) \wedge \text{CONCATENABLE}(x) \supset [\exists j, k. \text{MEETS}(j, i) \wedge \text{MEETS}(i, k) \wedge \text{FT}(j, x) \wedge \text{FT}(k, x)]$$

or alternatively,

$$\forall i, j, x. \text{MT}(i, x) \wedge \text{MT}(j, x) \wedge \text{CONCATENABLE}(x) \supset i \{a, b, e\} j$$

Proof

If x is true over i and j , and i meets j , then by Axiom 2.17 x is true over the cover of i and j , k . Both i and j are properly contained in k and therefore cannot be intervals over which x is maximally true. ■

2.4 Sequential composition

We now introduce the notion of *sequences* of entities. It is often necessary or convenient to represent and to reason about chains of events or successive states. One common approach is to define a sequencing operator which can be applied to two actions. For instance, Cohen and Levesque [1987] define the ; operation. However, they define the interpretation of this operation model-theoretically, and explicit mapping of the component actions onto time is not possible.

Another approach is to introduce the notion of sets of intervals. For instance, Ladkin [1986a] defines *non-convex* intervals as sets of (maximal) regular intervals with temporal gaps between them. Although Ladkin does not discuss this, presumably we could then use the TT predicate to associate compound states and events with these sets. It is not clear, however, how the component actions would be mapped onto the interval set members, and how 1:1 correspondence would be

⁷Actually, experience has shown that it is very hard to make them not immediately follow!

enforced. Moreover, the presence of temporal gaps would prevent any sequence of temporally contiguous events or states.

Poesio [1986] has suggested mapping sequences of events onto sequences of intervals, the latter being Ladkin-style non-convex intervals. The elements of one sequence, s_1 , are connected to those of the other sequence, s_2 , by means of the second order construct $\text{TIED}(s_1, s_2, \rho)$, where ρ is some arbitrary relation. Aside from the problems introduced by departing from standard first order predicate calculus, Poesio also posits the existence of *Dense Time Sequences*, which are sequences of time intervals which meet each other. This particular construct is explicitly ruled out by Ladkin's algebra, making the foundation rather tenuous.

We will define sequences as a special kind of entity, and precisely specify how the composition of sequences is established and how the components are mapped onto time. Sequences are defined axiomatically as follows:

Axiom 2.30 *Sequences, distinguished by the predicate `SEQUENCE`, are a subset of the solid entities:*

$$\forall z. \text{SEQUENCE}(z) \supset \text{ENTITY}(z) \wedge \text{SOLID}(z)$$

Axiom 2.31 *The set of entities is closed under the sequence operator `seq`:*

$$\forall x, y. \text{SEQUENCE}(\text{seq}(x, y))$$

$$\forall z. \text{SEQUENCE}(z) \supset \exists x, y. \text{seq}(x, y) = z$$

Axiom 2.32 *The sequence operator is associative:*

$$\forall x, y, z. \text{seq}(x, \text{seq}(y, z)) = \text{seq}(\text{seq}(x, y), z)$$

Axiom 2.33 (Sequential Truth) *A sequence is true over an interval if and only if the two entities composing the sequence are sequentially true over that interval:*

$$\forall k, x, y. \text{MT}(k, \text{seq}(x, y)) \iff [\exists i, j. \text{COVER!}(i, j, k) \wedge \text{MT}(i, x) \wedge \text{MT}(j, y)]$$

Using this formalism, we are now able to represent, for instance, the fact that it was initially cloudy but later sunny on some day as

$$\text{MT}(\text{DAY93}, \text{seq}(\text{CLOUDY}, \text{SUNNY})) \tag{2.6}$$

and the fact that thunder followed shortly after a lightning strike as

$$\text{MT}(\text{TIME13}, \text{seq}(\text{LIGHTNING}, \text{seq}(\text{DELAY}, \text{THUNDER}))) \quad (2.7)$$

Note that we insist on strict sequentiality: the second entity in a sequence starts being true when the first ceases to be true. In Section 5.3.1 we shall discuss what relaxing this restriction entails; for now, this insistence simplifies subsequent proofs considerably.

2.4.1 Reasoning about sequences

Given that we have a representation for sequences and their occurrences, what can we infer from this information? Obviously, the inference that the components will be true over certain intervals given the truth of a sequence falls directly out of our axiomatization. An interesting variation is inferring that, given any subinterval of a sequence, the second component will be true over some interval later than the given one⁸ provided the second component is downward hereditary. For instance, given

$$\text{MT}(\text{TIME97}, \text{seq}(\text{NIGHT}, \text{DAY})) \wedge \text{LIQUID}(\text{NIGHT}) \wedge \text{LIQUID}(\text{DAY}) \quad (2.8)$$

we'd like to be able to infer that for any interval i properly within TIME97 there exists an interval j within TIME97 such that j is later than i and DAY is true over j . Intuitively, this is clear, as either

- i falls within the NIGHT interval, in which case the day interval itself will do; or
- i overlaps the DAY interval, in which case there exists, by definition of the predicate OVERLAPS , a trailing interval over which DAY is true by virtue of DAY being downward hereditary; or
- i falls within the DAY interval, in which case there is, by definition of the predicate WITHIN! , a trailing interval over which DAY is true by virtue of DAY being downward hereditary.

Formally, we state this inference as Theorem 2.34, the proof of which follows:

⁸Or, alternatively, the first component will be true over some interval *earlier* than the given one.

Theorem 2.34 *If a sequence of x and y occurs over interval k , then there will always be a subinterval j of k after any proper subinterval i of k such that y is true over j , provided y is downward hereditary:*

$$\begin{aligned} & \forall k, x, y, i. \text{MT}(k, \text{seq}(x, y)) \wedge \text{WITHIN!}(i, k) \wedge \text{DOWNWARD}(y) \\ & \supset \exists j. \text{WITHIN}(j, k) \wedge \text{LATER}(j, i) \wedge \text{TT}(j, y) \end{aligned}$$

Proof

1: $\text{MT}(k, \text{seq}(x, y))$	; hypothesis
2: $\text{WITHIN!}(i, k)$	; hypothesis
3: $\text{DOWNWARD}(y)$	; hypothesis
4: $\exists i, j. \text{COVER!}(i, j, k) \wedge \text{MT}(i, x) \wedge \text{MT}(j, y)$	; MP(1)(A2.33)
5: $\text{COVER!}(\text{sk}_1, \text{sk}_2, k) \wedge \text{MT}(\text{sk}_1, x) \wedge \text{MT}(\text{sk}_2, y)$	; exist.elim.(4)
6: $\text{COVER!}(\text{sk}_1, \text{sk}_2, k)$	; conj.elim.(5)
7: $\text{MT}(\text{sk}_2, y)$	; conj.elim.(5)
8: $\text{FINISHES}(\text{sk}_2, k)$	; (6)+defn(COVER!)
9: $\exists j. \text{LATER}(j, i) \wedge \text{FINISHES}(j, k)$	; MP(2)(T2.11)
10: $\text{LATER}(\text{sk}_3, i) \wedge \text{FINISHES}(\text{sk}_3, k)$	; exist.elim.(9)
11: $\text{LATER}(\text{sk}_3, i)$	; conj.elim.(10)
12: $\text{FINISHES}(\text{sk}_3, k)$	; conj.elim.(10)
13: $\text{FINISHES}(\text{sk}_2, k) \wedge \text{FINISHES}(\text{sk}_3, k)$	; conj.intro.(8)(12)
14: $\exists l. \text{WITHIN}(l, \text{sk}_2) \wedge \text{WITHIN}(l, \text{sk}_3)$	; MP(13)(T2.12)
15: $\text{WITHIN}(\text{sk}_4, \text{sk}_2) \wedge \text{WITHIN}(\text{sk}_4, \text{sk}_3)$	; exist.elim.(14)
16: $\text{WITHIN}(\text{sk}_4, \text{sk}_2)$	; conj.elim.(15)
17: $\text{WITHIN}(\text{sk}_4, \text{sk}_3)$	; conj.elim.(15)
18: $\text{TT}(\text{sk}_2, y)$	; MP(7)(A2.25)
19: $\text{DOWNWARD}(y) \wedge \text{WITHIN}(\text{sk}_4, \text{sk}_2) \wedge \text{TT}(\text{sk}_2, y)$	; conj.intro.(3)(16)(18)
20: $\text{TT}(\text{sk}_4, y)$	; MP(19)(A2.14)
21: $\text{WITHIN}(\text{sk}_4, \text{sk}_3) \wedge \text{LATER}(\text{sk}_3, i)$	; conj.intro.(17)(11)
22: $\text{LATER}(\text{sk}_4, i)$	; MP(21)(T2.8)
23: $\text{WITHIN}(\text{sk}_4, \text{sk}_2) \wedge \text{FINISHES}(\text{sk}_2, k)$	; conj.intro.(16)(8)
24: $\text{WITHIN}(\text{sk}_4, k)$	; MP(23)(T2.6)
25: $\text{WITHIN}(\text{sk}_4, k) \wedge \text{LATER}(\text{sk}_4, i) \wedge \text{TT}(\text{sk}_4, y)$	; conj.intro.(20)(22)(24)
26: $\exists j. \text{WITHIN}(j, k) \wedge \text{LATER}(j, i) \wedge \text{TT}(j, y)$	; exist.intro.(25)

■

2.5 Recurrence

Finally, we are ready to define recurrence! As with sequential occurrence, a common approach to representing and to reasoning about recurring events or states is to define an iteration operator. Cohen and Levesque [1987], for instance, define the operators $*$ (zero or more repetitions of an action) and $+$ (one or more repetitions). Again, however, the meaning of these operations is defined model-theoretically, and explicit mapping of the repeated actions onto time is not possible.

Using sequences of the same (repeated) event and mapping them onto sequences of time intervals [Poesio, 1986] makes recurrence implicit in the structure of the individuals. Poesio also uses a special annotation on some predicates, namely *Number Restriction*, to infer a sequence of events when what otherwise looks like a single event is associated with a sequence of intervals. This sequence is then also tied to the temporal sequence.

To facilitate reasoning about the occurrence of components, we prefer to use an explicit representation of recurrence. Hence we will define recurrence with the predicate RT . We shall take

$$RT(i, x)$$

to mean “ x is repeatedly true over interval i .” For instance, to express the fact that Tommy hit the wall repeatedly we say

$$RT(\text{TIME65}, \text{hit}(\text{TOMMY}, \text{WALL37})) \quad (2.9)$$

The fact that night and day alternate for the duration of the earth can be expressed as

$$RT(\text{EARTHTIME}, \text{seq}(\text{NIGHT}, \text{DAY})) \quad (2.10)$$

The following axiom formally defines recurrence:

Axiom 2.35 *If an entity x is repeatedly true over interval k , then either it is maximally true over k or there exists a subdivision of k into i and j such that x is maximally true over i and repeatedly true over j :*

$$\forall k, x. RT(k, x) \supset \\ MT(k, x) \vee [\exists i, j. \text{COVER!}(i, j, k) \wedge MT(i, x) \wedge RT(j, x)]$$

2.5.1 Occurrence from recurrence

As we did with the occurrence of sequences, we shall demonstrate the use of RT to represent recurrence. Given a recurring entity, the fact that the entity is true at least once within the time of recurrence obviously falls out of the definition. A more interesting question is: given a recurring entity, will the entity be true any time later than some arbitrary time during the interval of recurrence? In other words, armed with the above fact (2.10), can E.T. confidently predict that, no matter when he visits Earth, he'll see the light of day?

This inference can be expressed in our logic as a theorem, which is an extension of Theorem 2.34, as follows:

Theorem 2.36 *If a sequence of x and y repeatedly occurs over interval k , then there will always be a subinterval j of k after any proper subinterval i of k such that y is true over j , provided y is downward hereditary:*

$$\begin{aligned} \forall i, k, x, y. & \text{RT}(k, \text{seq}(x, y)) \wedge \text{WITHIN!}(i, k) \wedge \text{DOWNWARD}(y) \\ & \supset \exists j. \text{WITHIN}(j, k) \wedge \text{LATER}(j, i) \wedge \text{TT}(j, y) \end{aligned}$$

Proof

- 1: $\text{RT}(k, \text{seq}(x, y))$; hypothesis
- 2: $\text{WITHIN}(i, k)$; hypothesis
- 3: $\text{DOWNWARD}(y)$; hypothesis
- 4: $\text{MT}(k, \text{seq}(x, y)) \vee$; MP(1)(A2.35)
- $[\exists i, j. \text{COVER!}(i, j, k) \wedge$
- $\text{MT}(i, \text{seq}(x, y)) \wedge \text{RT}(j, \text{seq}(x, y))]$

Prove by case:

a) *The left term of disjunction (4) ...*

- 5: $\text{MT}(k, \text{seq}(x, y))$; disj(4), left term
- 6: $\text{MT}(k, \text{seq}(x, y)) \wedge \text{WITHIN}(i, k) \wedge \text{DOWNWARD}(y)$; conj.intro.(5)(2)(3)
- 7: $\exists j. \text{WITHIN}(j, k) \wedge \text{LATER}(j, i) \wedge \text{TT}(j, y)$; MP(6)(T2.34)

□

b) *The right term of the disjunction (4) ...*

- 8: $\exists i, j. \text{COVER!}(i, j, k)$; disj(4), right term
- $\wedge \text{MT}(i, \text{seq}(x, y)) \wedge \text{RT}(j, \text{seq}(x, y))$
- 9: $\text{COVER!}(\text{sk}_1, \text{sk}_2, k)$; exist.elim.(8)
- $\wedge \text{MT}(\text{sk}_1, \text{seq}(x, y)) \wedge \text{RT}(\text{sk}_2, \text{seq}(x, y))$

10: COVER!(SK ₁ , SK ₂ , k)	; conj.elim.(9)
11: MT(SK ₁ , seq(x, y))	; conj.elim.(9)
12: RT(SK ₂ , seq(x, y))	; conj.elim.(9)
13: WITHIN(SK ₁ , k)	; (10)+def'n(COVER!)
14: WITHIN(SK ₂ , k)	; (10)+def'n(COVER!)
15: COVER!(SK ₁ , SK ₂ , k) ∧ WITHIN(i, k)	; conj.intro.(10)(2)
16: WITHIN(i, SK ₁) ∨ WITHIN(i, SK ₂) ∨ OVERLAPS(SK ₁ , i)	; MP(15)(T2.13)

Again, prove by case:

b1) The left term of disjunction (16) ...

17: WITHIN(i, SK ₁)	; (16) left term
18: MT(SK ₁ , seq(x, y)) ∧ WITHIN(i, SK ₁) ∧ DOWNWARD(y)	; conj.intro.(11)(17)(3)
19: ∃ j . WITHIN(j, SK ₁) ∧ LATER(j, i) ∧ TT(j, y)	; MP(18)(T2.34)
20: WITHIN(SK ₃ , SK ₁) ∧ LATER(SK ₃ , i) ∧ TT(SK ₃ , y)	; exist.elim.(19)
21: WITHIN(SK ₃ , SK ₁)	; conj.elim.(20)
22: LATER(SK ₃ , i)	; conj.elim.(20)
23: ∧ TT(SK ₃ , y)	; conj.elim.(20)
24: WITHIN(SK ₃ , SK ₁) ∧ WITHIN(SK ₁ , k)	; conj.intro.(21)(13)
25: WITHIN(SK ₃ , k)	; MP(24)(T2.6)
26: WITHIN(SK ₃ , k) ∧ LATER(SK ₃ , i) ∧ TT(SK ₃ , y)	; conj.intro.(25)(22)(23)
27: ∃ j . WITHIN(j, k) ∧ LATER(j, i) ∧ TT(j, y)	; exist.intro.(26)

□

b2) The middle term of disjunction (16) ...

28: WITHIN(i, SK ₂)	; (16) middle term
29: RT(SK ₂ , seq(x, y)) ∧ WITHIN(i, SK ₂) ∧ DOWNWARD(y)	; conj.intro.(12)(28)(3)
30: ∃ j . WITHIN(j, SK ₂) ∧ LATER(j, i) ∧ TT(j, y)	; (29), RT induction
31: WITHIN(SK ₄ , SK ₂) ∧ LATER(SK ₄ , i) ∧ TT(SK ₄ , y)	; exist.elim.(30)
32: WITHIN(SK ₄ , SK ₂)	; conj.elim.(31)
33: LATER(SK ₄ , i)	; conj.elim.(31)
34: TT(SK ₄ , y)	; conj.elim.(31)
35: WITHIN(SK ₄ , SK ₂) ∧ WITHIN(SK ₂ , k)	; conj.intro.(32)(14)

36: $\text{WITHIN}(\text{SK}_4, k)$; MP(35)(T2.6)
 37: $\text{WITHIN}(\text{SK}_4, k) \wedge \text{LATER}(\text{SK}_4, i) \wedge \text{TT}(\text{SK}_4, y)$; conj.intro.(36)(33)(34)
 38: $\exists j. \text{WITHIN}(j, k) \wedge \text{LATER}(j, i) \wedge \text{TT}(j, y)$; exist.intro.(37)
 □
b3) The right term of disjunction (16) ...
 39: $\text{OVERLAPS}(\text{SK}_1, i)$; (16) right term
 40: $\exists j. \text{WITHIN}(j, \text{SK}_1) \wedge \text{WITHIN}(j, i)$; MP(39)(T2.10)
 41: $\text{WITHIN}(\text{SK}_5, \text{SK}_1) \wedge \text{WITHIN}(\text{SK}_5, i)$; exist.elim.(40)
 42: $\text{WITHIN}(\text{SK}_5, \text{SK}_1)$; conj.elim.(41)
 43: $\text{WITHIN}(\text{SK}_5, i)$; conj.elim.(41)
 44: $\text{MT}(\text{SK}_1, \text{seq}(x, y)) \wedge \wedge \text{DOWNWARD}(y)$; conj.intro.(11)(42)(3)
 45: $\exists j. \text{WITHIN}(j, \text{SK}_1) \wedge \text{LATER}(j, \text{SK}_5) \wedge \text{TT}(j, y)$; MP(44)(T2.34)
 46: $\text{WITHIN}(\text{SK}_6, \text{SK}_1) \wedge \text{LATER}(\text{SK}_6, \text{SK}_5) \wedge \text{TT}(\text{SK}_6, y)$; exist.elim.(45)
 47: $\text{WITHIN}(\text{SK}_6, \text{SK}_1)$; conj.elim.(46)
 48: $\text{LATER}(\text{SK}_6, \text{SK}_5)$; conj.elim.(46)
 49: $\text{TT}(\text{SK}_6, y)$; conj.elim.(46)
 50: $\text{LATER}(\text{SK}_6, \text{SK}_5) \wedge \text{WITHIN}(\text{SK}_5, i)$; conj.intro.(48)(43)
 51: $\text{LATER}(\text{SK}_6, i)$; MP(50)(T2.9)
 52: $\text{WITHIN}(\text{SK}_6, \text{SK}_1) \wedge \text{WITHIN}(\text{SK}_1, k)$; conj.intro.(47)(13)
 53: $\text{WITHIN}(\text{SK}_6, k)$; MP(52)(T2.6)
 54: $\text{WITHIN}(\text{SK}_6, k) \wedge \text{LATER}(\text{SK}_6, i) \wedge \text{TT}(\text{SK}_6, y)$; conj.intro.(53)(51)(49)
 55: $\exists j. \text{WITHIN}(j, k) \wedge \text{LATER}(j, i) \wedge \text{TT}(j, y)$; exist.intro.(54)
 ■

2.6 Discussion

The definition of recurrence as stated above (Axiom 2.35) is such that an entity is repeatedly true if it is at least true once. This may seem somewhat odd, as intuitively we think of repetition as something being true at least two or more times. Hence, an alternative to Axiom 2.35 is the following:

Axiom 2.35.a *If an entity x is repeatedly true over interval k , then there exists a subdivision of k into i and j such that x is maximally true over i and it is either maximally or repeatedly true over j :*

$$\forall k, x. \text{RT}(k, x) \iff \exists i, j. \text{COVER!}(i, j, k) \wedge \text{MT}(i, x) \wedge [\text{MT}(j, x) \vee \text{RT}(j, x)]$$

This formulation also has the advantage of the biconditional, which means we can infer recurrence given repeated occurrence as well as infer repeated occurrence given recurrence.

Given this alternate axiom, we can still prove Theorem 2.36, although the proof is significantly longer due to the nested disjunction. The proof outline is identical, however.

Another possible point of contention with Axiom 2.35 is that recurring sequences are forced to repeat in their entirety. If we wish to consider an elevator repeatedly going up and down in its shaft over its lifetime, it might be considered reasonable to force it to always start out going up and finish by going down. But, given the alternation of night and day for the duration of the earth (fact 2.10), the earth may well come to an end when it is nighttime!⁹ In order to allow this, we would need to restrict Axiom 2.35 (or 2.35.a) to the recurrence of non-sequences only, and add the following axiom:

Axiom 2.35.b *If a sequence of x and y is repeatedly true over interval k , then there exists a subdivision of k into i and j such that x is maximally true over i and either y is maximally true over j or the sequence of y and x is repeatedly true over j :*

$$\forall k, x, y. \text{RT}(k, \text{seq}(x, y)) \iff \exists i, j. \text{COVER!}(i, j, k) \wedge \text{MT}(i, x) \wedge [\text{MT}(j, y) \vee \text{RT}(j, \text{seq}(y, x))]$$

This formulation, of course, plays havoc with the prediction theorem (Theorem 2.36): the arbitrary subinterval may coincide with the last interval over which x is true, which also happens to be the interval that terminates the recurrence interval. Hence, we can no longer conclude that y will be true later than the arbitrary subinterval.

We can complicate things further by insisting that, instead of possibly just a single occurrence of the sequence, a repeated sequence of x and y should imply that at least $x, y, x, \dots$ occurs. This would expand Axiom 2.35.b with another nested disjunction.

⁹From the subjective viewpoint of the reasoner, of course!

2.7 Summary

In this chapter we have developed a formal system for reasoning about temporal intervals, and about the truth of entities (actions, events, and states) over these intervals. A series of axioms have been presented that defined the concepts of truth over time, maximal truth over time, and repeated truth over time. We have shown how entities can be composed into sequences and defined how these sequences behave over time, both incidentally and repeatedly.

The primary advantage of the approach we have taken is that events and states are explicitly associated, by means of predicates, with the times over which they are true (occur or hold), which facilitates analysis and prediction for a reasoning system. Making predictions, naturally, is of great value to a planning system. After describing in the next chapter the implementation of a temporal reasoner based on the logic developed in this chapter, we will explore an extensive example in Chapter 4.

Chapter 3

A Temporal Reasoning System

3.1 Introduction

The previous chapter described a formal system for representing and reasoning about intervals of time, and the truth of entities (states holding, events occurring) over time. This chapter describes the implementation of a temporal reasoning system based on this formalism. The implementation consists of two parts, namely,

- The TIMELOGIC system: an interval-based forward chaining inference engine and database manager of temporal constraints; and
- The TEMPOS system: an extension to the Horne-based logic programming and knowledge representation system *Rhetorical*, providing a predicate interface to TIMELOGIC as well as the predicates TT, MT, FT, MF and RT.

The two systems, and their syntax, structure and capabilities, will be described in detail here. Appendices A and B provide details of the functional interface to the TIMELOGIC and TEMPOS systems, respectively, including initialization, interval definition, relational and durational constraint posting and retrieval, and system customization.

3.2 The TimeLogic system¹

Interval-based temporal logic, introduced by Allen [Allen, 1983], has proven to be a powerful paradigm. In many A.I. systems, information about the temporal relation between two events or propositions is as important, if not more so, as

¹The bulk of the TIMELOGIC description first appeared as [Koomen, 1987].

absolute length or position on a time line of any one such event or proposition. Recognizing that intervals can be related to other intervals in at most seven different ways (before, meets, overlaps, starts, during, finishes and equals) plus their inverses, Allen proposed a way of storing partial temporal knowledge by maintaining sets of possible temporal relations between any two intervals, and provided a temporal constraint propagation algorithm. Knowledge can be added to a network of interval constraints by reducing the set of possible temporal relations between two intervals, and propagating the effects of this reduction to all other intervals related to these two.

Given the relational constraint sets between intervals i and j and between j and k , the relational constraint set between i and k can be obtained through a transitivity function which successively indexes a transitivity table by the elements of the $i-j$ and $j-k$ sets and computes the union of the table entries. If there already existed a set of constraints between i and k , this set is intersected with the computed union to arrive at a new constraint set. If the cardinality of the resulting set is 1, then the constraint between i and k has been uniquely determined; if the intersection resulted in the empty set, then a conflict exists in the network.

TIMELOGIC is a stand-alone system, implemented in Common Lisp [Steele Jr., 1984], that provides a facility for storing, retrieving and inferring temporal constraints between intervals. Intervals can be named by any Lisp object with the proviso that two objects which are EQL refer to the same interval. Two types of constraints can be manipulated, namely interval relations (as described above), and interval durations. Section 3.3 describes how these relational and durational constraints are specified and manipulated. The TIMELOGIC system provides facilities for the management of a hierarchical interval structure. Section 3.4 describes how intervals can be explicitly or automatically clustered by associating them with reference intervals, and how reference intervals are used to limit constraint propagation and to deduce inter-cluster interval constraints. TIMELOGIC also supports contextual reasoning and backtracking, which is documented in Section 3.5.

3.3 Interval Constraints

3.3.1 Specification of relational constraints

Relational constraints specify ordinal relationships between intervals, such as before, equals, during, *etc.* A total of 13 relational constraints are possible between any two intervals, as illustrated in Figure 3.1. These constraints are binary and pairwise disjoint, *i.e.*, two intervals are related by exactly one of the constraints.

x Relation y	
<i>After</i>
<i>Before</i>
<i>Contains</i>
<i>During</i>
<i>Equals</i>
<i>Finishes</i>
<i>Finished by</i>
<i>Meets</i>
<i>Met by</i>
<i>Overlaps</i>
<i>Overlapped by</i>
<i>Starts</i>
<i>Started by</i>

Figure 3.1: The 13 possible relational constraints between two intervals

A set of relational constraints denotes the “exclusive or” of the members of the set.

In the TIMELOGIC system, a relational constraint set is specified either by the keyword of one of the simple relations listed in Table 3.1, or by the keyword of one of the compound relations listed in Table 3.2, or by a list of one or more simple or compound relations.

Hence, a relational constraint set such as “either before or after or met by” is denoted in formulae as $\{a, b, \overline{m}\}$, in running text as $\{before, after, met\ by\}$, and in TIMELOGIC as $(:A :B :Mi)$.

3.3.2 Inferring relational constraints

If relational constraints exist between intervals i and j and intervals j and k , it is possible to infer what the relational constraint is between intervals i and k (see Section 2.2.3). For instance, if $X \{meets\} Y$ and $Y \{meets\} Z$ then it can be concluded that $X \{before\} Z$. This inference is accomplished by the transitivity function *Constraints* which indexes a 13x13 transitivity table [Allen, 1983]. Note

relation	key	inverse	key
<i>before</i>	:B	<i>after</i>	:A
<i>during</i>	:D	<i>contains</i>	:C
<i>equals</i>	:E	<i>equal</i>	:E
<i>finishes</i>	:F	<i>finished by</i>	:Fi
<i>meets</i>	:M	<i>met by</i>	:Mi
<i>overlaps</i>	:O	<i>overlapped by</i>	:Oi
<i>starts</i>	:S	<i>started by</i>	:Si

Table 3.1: Simple interval relations in TIMELOGIC

relation	key	equivalence
<i>somehow during</i>	:DUR	(:D :F :S)
<i>somehow contains</i>	:CON	(:C :Fi :Si)
<i>disjoint from</i>	:DIS	(:A :B :M :Mi)
<i>intersects with</i>	:INT	(:CON :DUR :O :Oi :E)
<i>not equal to</i>	:NEQ	(:CON :DUR :O :Oi :DIS)
<i>all</i>	:ALL	(:CON :DUR :O :Oi :DIS :E)

Table 3.2: Compound interval relations in TIMELOGIC

that this inference is limited in that the actual constraint between i and k might be more precisely defined than could be concluded from the transitivity table. For instance, if $X \{contains\} Y$ and $Y \{during\} Z$, then nothing can be concluded from this about the constraint between X and Z .

If the constraints between i and j and between j and k are sets, the constraint between i and k can be computed as the set union of the result of mapping the transitivity function over the cross product of the two sets. For instance, if $X \{starts, met by\} Y$ and $Y \{overlaps, equals\} Z$, then it can be concluded that $X \{before, during, finishes, meets, met by, overlaps, overlapped by, starts\} Z$, as follows:

If $X \{starts\} Y$ and $Y \{overlaps\} Z$ then $X \{before, meets, overlaps\} Z$;

If $X \{starts\} Y$ and $Y \{equals\} Z$ then $X \{starts\} Z$;

If $X \{met by\} Y$ and $Y \{overlaps\} Z$ then $X \{during, finishes, overlapped by\} Z$;

If $X \{met by\} Y$ and $Y \{equals\} Z$ then $X \{met by\} Z$;

Therefore conclude:

If $X \{starts, met\ by\} Y$ and $Y \{overlaps, equals\} Z$ then
 $X \{before, during, finishes, meets, met\ by, overlaps, overlapped\ by, starts\} Z$.

The following code fragment casts this last example in TIMELOGIC terminology:

```

◇ (ADD-INTERVAL-CONSTRAINT 'X' '(:Mi :S) 'Y)
◇ (ADD-INTERVAL-CONSTRAINT 'Y' '(:E :O) 'Z)
◇ (GET-INTERVAL-CONSTRAINT 'X' 'Z)
→ (:B :D :F :M :Mi :O :Oi :S)

```

3.3.3 Specification of durational constraints

Durational constraints specify relative magnitudes between intervals, such as half as long, equal or three times longer than, *etc.* Formally, a durational constraint is either a range of open or closed bounds, or a disjunctive set of ranges. A single bound may be used to specify a range whose lower bound is equal to its upper bound.² A bound is either a non-negative number or the keyword `:INF` meaning infinity. Open bounds, whose values are not included in the range they bound, are indicated as a one-element list containing a closed bound. The bounds `0` and `:INF` are by definition open bounds, and do not need to be embedded in a list. Disjunctive sets of durational constraints are indicated by a list whose first element is the symbol `OR` and whose remaining elements are closed bounds or ranges. The keywords `:=`, `<`, `<=`, `>`, `>=` and `<>` are recognized as abbreviations for the ranges `1`, `(0 (1))`, `(0 1)`, `((1) :INF)`, `(1 :INF)` and `(OR < >)`, respectively. Examples of durational constraints between intervals X and Y , and their meanings, are given in Table 3.3.

All arithmetic on bounds of durational constraints results in a floating point number if either of the bounds is a floating point number and the `:FLOATS` property is enabled (see Section A.8 on customization), otherwise rational arithmetic is used.

3.3.4 Inferring durational constraints

If durational constraints exist between intervals i and j and intervals j and k , it is likewise possible to infer what the durational constraint is between intervals i and k . For instance, if X takes 3 times as long as Y and Y takes between 4 and 5 times as long as Z then it can be concluded that X must take between 12 and 15

²which obviously must be closed!

X constraint Y	X takes ... times as long as Y
6	exactly 6
(0 6)	at most 6
(0 (6))	less than 6
((3) 6)	more than 3 but at most 6
(0 :INF)	undetermined
(3/2 25/4)	between $1\frac{1}{2}$ and $6\frac{1}{4}$, inclusive
(1.5 6.25)	between 1.5 and 6.25, inclusive
(OR :< (8 10))	either less than 1 or between 8 and 10
(OR (0 (5)) ((5) :INF))	either less than 5 or more than 5

Table 3.3: Durational constraint examples

times as long as Z. This inference, which is accomplished using a multiplicative transitivity rule, is limited in that the actual constraint between i and k might be more precisely defined than could be concluded from the transitivity rule. The product of two ranges r_1 and r_2 is defined as the range whose lower bound is the product of the lower bounds of r_1 and r_2 and whose upper bound is the product of the upper bounds of r_1 and r_2 . The product of two bounds b_1 and b_2 is open unless both b_1 and b_2 are closed.

If the constraints between i and j and between j and k are sets, the constraint between i and k can be computed as the set union of the result of mapping the transitivity rule over the cross product of the two sets. For instance, if X takes either the same time or between 4 and 6 times as long as Y and Y takes either between 2 and 5 times or between 9 and 10 times as long as Z then conclude that X must take either between 2 and 5 times or between 8 and 30 times or between 36 and 60 times as long as Z. Note that set union over ranges collapses overlapping ranges. The following code fragment casts this last example in TIMELOGIC terminology:

```

◇ (ADD-INTERVAL-CONSTRAINT 'X '(OR := (4 6)) 'Y :TYPE :DUR)
◇ (ADD-INTERVAL-CONSTRAINT 'Y '(OR (2 5) (9 10)) 'Z :TYPE :DUR)
◇ (GET-INTERVAL-CONSTRAINT 'X 'Z :TYPE :DUR)
→ (OR (2 5) (8 30) (36 60))

```

3.3.5 Inferring durational from relational constraints

Durational constraints can be inferred from certain relational constraint sets. If a relational constraint between intervals i and j is a subset of ρ , then the durational

constraint between i and j must be a subset of δ , where the interpretation of ρ and δ are given in Table 3.4.

ρ	δ
(:E)	:=
(:D :F :S)	:<
(:D :E :F :S)	:<=
(:C :Fi :Si)	:>
(:C :E :Fi :Si)	:>=
(:C :D :F :Fi :S :Si)	:<>

Table 3.4: Relational to durational constraint map

3.3.6 Inferring relational from durational constraints

Similarly, relational constraints can be deduced from those durational constraints that leave no ambiguity with respect to one interval being longer than, shorter than or equal in length to another. If a durational constraint between intervals i and j is a subset of δ , then the relational constraint between i and j must be a subset of ρ , where δ and ρ are defined according to Table 3.5. Note that the order

δ	ρ
:=	(:DIS :E :O :Oi)
:<	(:DIS :DUR :O :Oi)
:<=	(:DIS :DUR :E :O :Oi)
:>	(:DIS :CON :O :Oi)
:>=	(:DIS :CON :E :O :Oi)
:<>	(:DIS :CON :DUR :O :Oi)

Table 3.5: Durational to relational constraint map

in which the subset tests are performed is significant: if a durational constraint is a subset of $:<$ then it is also a subset of $:<=$ and $:<>$.

The TIMELOGIC system will perform these cross inferences automatically when both relational and durational subsystems are enabled.

3.4 Interval Hierarchies³

3.4.1 Constraint propagation: how bad does it get?

Since the cardinality of any relational constraint set is at most 13, each relational constraint can be asserted and updated at most 12 times.⁴ However, propagating the effects of such an update is proportional to the branching factor, *i.e.*, the number of other intervals related to the two intervals whose constraint changed. Without imposing structure, a network of temporal relations between n intervals tends toward full connectivity. Hence, for each relational constraint added, Allen's basic algorithm attempts to propagate the effect to all other "Comparable" (*i.e.*, connected) intervals, invoking the (expensive!) transitivity function *Constraints* $2n-2$ times. Each time propagation reduces another constraint, this reduction too will be propagated across the network. Vilain and Kautz [1986] showed that the time complexity of this algorithm in terms of *Constraints* invocations is $O(n^3)$. As the number of intervals in the network grows, this becomes entirely insufferable. Moreover, in a system using the interval logic, activity tends to focus on the most recently added intervals; new information does not affect old relations much, and propagating effects back to older intervals tends to be useful only for consistency checking.

Clearly, the cubed time and quadratic space complexity for a temporal database of this nature is prohibitive for all but the smallest problem sets. Allen proposed the use of reference intervals to limit constraint propagation, thereby reducing the average time and space requirements. This section first demonstrates that, in general, arbitrary reference clusters fail to propagate crucial constraints, resulting in an underconstrained network. Then a new algorithm is presented which allows a temporal database to dynamically create, without loss of information, a reference hierarchy based on temporal containment. Directly stored relations between disjoint intervals can be eliminated if the same relation can be found along a path through the containment hierarchy. As such a hierarchy tends to be tree-like, this search is acceptable (*i.e.*, logarithmic). Empirical results of experiments based on random networks of intervals and on a more realistic planning problem bear out the superior performance of this algorithm.

³A version of this section also appeared as [Koomen, 1989].

⁴Asserting the set of all 13 possible relations is equivalent to having no information at all.

3.4.2 Limiting propagation: reference intervals

Allen proposed adopting a notion of reference intervals similar to that of Kahn and Gorry [1977] to reduce the space requirements of an interval network. Clusters are formed within the network by associating intervals with one or more reference intervals. Constraint propagation takes place within each cluster only, while inter-cluster relations are obtained indirectly by applying the transitivity function along paths through the network, where paths consist of reference intervals only. For instance, consider the two activities *gardening* and *dinner* with associated intervals **Gardening** and **Dinner**. The intervals associated with all activities relating to this instance of gardening, such as **Mowing** and **Raking**, would be related directly to **Gardening**, and likewise, intervals such as **BoilWater**, **CookSpaghetti**, *etc.*, would be directly related to this instance of preparing for and having dinner. The intervals **BoilWater**, **CookSpaghetti**, *etc.*, would not be directly related to **Gardening** or any of its subintervals, however; those relationships would be determined by applying the constraint propagation algorithm along the path through **Dinner**. The effect of further constraining two dinner-related intervals would not be propagated to any gardening-related intervals.

This referencing mechanism obviously can reduce the branching factor at each node considerably, resulting in a significant space savings (fewer relations to store) as well as a speed-up in adding new constraints due to propagation to fewer related intervals. The time to fetch inter-cluster relations is increased, but this can be kept manageable (*i.e.*, logarithmic) if the reference intervals are organized in a tree-like hierarchy. In other words, we are trading somewhat increased query time ($O(\log n)$ versus $O(1)$ in the non-hierarchical case) for significantly decreased assertion time ($O(\rho^2 n)$ versus $O(n^3)$, where ρ is the average/maximum number of intervals per reference cluster). Although n grows as more information is added to the temporal database, ρ is expected to stay small and relatively constant.

Allen claims that imposing a reference hierarchy “can be done with little loss of information,” as long as “care is taken.” He does not make clear, however, of what this care consists. The fact is that it is rather easy to lose information due to the structure of the reference hierarchy, particularly between intervals in different, temporally overlapping clusters. To flesh out the example above, assume that the following constraints have been asserted:

- c_1 : **Mowing** {*starts*} **Gardening**
- c_2 : **Mowing** {*before*} **Raking**
- c_3 : **Raking** {*finishes*} **Gardening**
- c_4 : **BoilWater** {*starts*} **Dinner**

Figure 3.2: Example of interval reference structure

- c_5 : BoilWater {meets} CookSpaghetti
- c_6 : CookSpaghetti {before,meets} EatSpaghetti
- c_7 : EatSpaghetti {finishes} Dinner
- c_8 : Dinner {meets} Dishes

Now suppose I decided to boil the water while I'm raking the lawn:

- c_9 : Raking {equals} BoilWater

In a "flat" network, the following additional constraints, among others, would be derived automatically:

- c_{10} : Gardening {overlaps} Dinner
- c_{11} : Mowing {before} Dinner
- c_{12} : Mowing {before} Dishes
- c_{13} : CookSpaghetti {during} Dinner

However, consider a structured system where **Gardening** is a reference interval for **Mowing** and **Raking**, and **Dinner** is a reference interval for **BoilWater**, **CookSpaghetti** and **EatSpaghetti**. The reference structure is depicted in Figure 3.2.

In this case constraint c_{10} would be derived, but the interval **Mowing** would not be comparable with **Dinner** and **Dishes** (they do not share a reference interval, nor have explicit relationships been asserted between them), and hence constraints c_{11} and c_{12} would not be derived. Unfortunately, computing the constraint between

Mowing and Dinner along the path **Mowing–Gardening–Dinner** results in the constraint $\{before, meets, overlaps\}$ which is clearly weaker than constraint c_{11} obtained in the flat system. Note that overlapping reference intervals and multiple reference intervals per simple interval are by no means contrived situations; for instance, they can easily arise in planning situations [Allen and Koomen, 1983]. Moreover, this example shows that merely requiring that intervals are wholly contained in their reference intervals, as Allen suggests, is insufficient to ensure proper inferencing. Allen already noted that merging two reference subtrees whenever such an overlap is detected tends to quickly flatten the entire reference hierarchy, defeating the purpose for reference intervals.

3.4.3 Automatic referencing

The TIMELOGIC system allows intervals to be explicitly designated as references for other intervals. However, devising a reference structure for a temporal constraint network such that no information is lost turns out to be a very nasty problem, and certainly one that a temporal reasoning system ought not to leave up to higher level reasoning systems, as Allen suggests. It is hard, if not impossible, to predict what effect predefining a reference structure has on temporal inferencing, and it certainly requires taking into account the order in which constraints are posted as well as detailed knowledge of the temporal constraint propagation algorithm, details of which would best be kept in the specialized temporal reasoning system. It would be very desirable if the temporal reasoning system itself could structure the network dynamically on the basis of posted and derived constraints, and do it without losing information. This and the following sections describe a method for doing this.

As Allen [1983] and Kahn and Gorry [1977] suggested, a reference hierarchy is naturally based on temporal containment: higher level intervals have a greater span in time, and lower level intervals are contained in higher ones. Propositional inheritance is an obvious reason for basing the structure on containment: any proposition that holds over an interval i necessarily holds over any subinterval of i , and any relation between i and another, disjoint interval j , is necessarily true between any subinterval of i and j . Another reason a reference structure based on temporal containment is desirable is that it reflects temporal locality of reference: most queries to a temporal database concern relations between intervals associated with propositions currently under consideration, such as planning a task, where propagation is likely to be useful, whereas very few queries would be concerned with the constraints between two (semantically) unrelated and tempo-

rally distant propositions. However, this reasoning breaks down if a lot of global domain constraints apply to the problem at hand [Allen and Koomen, 1983]. For instance, in the blocks world a rule such as

$$\forall x, y. \text{BLOCK}(x) \wedge \text{BLOCK}(y) \supset \text{MUTEX}(\text{on}(x, y), \text{clear}(y))$$

would cause task-unrelated intervals to be temporally related, violating the notion of temporal locality of reference.

The automatic referencing method described in this section revolves around two notions, namely, dynamically constructing a reference hierarchy based on containment, and reducing the interval branching factor by removing direct constraints that can be obtained indirectly through this reference hierarchy.

In the following discussion, the constraints between lower and higher level intervals in the reference hierarchy will be referred to as *uplinks* (with *downlinks* going the other way), and constraints between disjoint intervals as *sidelinks*, with the proviso that sidelinks are singular, *i.e.*, relational constraint sets with cardinality 1. Let $\text{Refs}^*(i)$ be the closure of Allen's Refs function, *i.e.*, the set of all reference intervals directly or indirectly above i . Let $C(i, j)$ be the direct constraint between intervals i and j and $N(i, j)$ be the constraint between i and j derivable from the hierarchy. (See appendix C for formal definitions of these links and more detailed descriptions of the automatic referencing algorithms.)

3.4.3.1 Constructing the hierarchy

The reference hierarchy is constructed on the basis of asserted uplinks and downlinks. If an uplink between intervals i and j is asserted (either explicitly or through propagation), *i.e.*, the new constraint set between i and j is a subset of $\{\text{starts}, \text{during}, \text{finishes}\}$, then j is a candidate reference interval for i . Adding a reference link from i to j , if necessary, is accomplished by the algorithm listed in Figure 3.3. In the gardening/dinner example above, for instance, the same hierarchy as depicted in Figure 3.2 is constructed dynamically as the constraints c_1 , c_3 , c_4 and c_7 are posted. Assuming that all gardening and dinner intervals are initially constrained to be during **Today**, constraint c_1 causes **Gardening** to become a reference interval for **Mowing** while the reference link between **Mowing** and **Today** is removed. Likewise, the reference links between **EatSpaghetti** and **Dinner** (by assertion of c_7) and between **CookSpaghetti** and **Dinner** (by implication) are created, and the corresponding reference links to **Today** are removed.

```

to AddRef(i,j)
  if  $j \in \text{Refs}(i)$  then
    Exit
  else if  $j \in \text{Refs}^*(i)$  then
    :: i is already below j
     $\text{Refs}(i) \leftarrow \text{Refs}(i) - \{j\}$ 
  else begin
     $\text{Refs}(i) \leftarrow \text{Refs}(i) \cup \{j\}$ 
    for each node  $k \in \text{Refs}^*(j)$  do
      :: remove now indirect links, if any,
      :: from i to ancestors of j
       $\text{Refs}(i) \leftarrow \text{Refs}(i) - \{k\}$ 
    for each node  $h$  such that  $i \in \text{Refs}^*(h)$  do
      :: remove now indirect links, if any,
      :: from descendants of i to j
       $\text{Refs}(h) \leftarrow \text{Refs}(h) - \{j\}$ 
  end
end AddRef

```

Figure 3.3: Algorithm for constructing the reference hierarchy

3.4.3.2 Reducing the branching factor

The branching factor of an interval is reduced by removing singular⁵ sidelinks and indirect uplinks with other intervals, provided those links can be derived from a path through the reference hierarchy. If a singular sidelink between intervals i and j is asserted (either explicitly or through propagation), *i.e.*, some interval is disjoint from another and the constraint set is singular, the procedure `RemoveLink` (see Figure 3.4) attempts to remove the sidelink. This is successful if and only if the same relation can be obtained by applying the constraint propagation algorithm along all paths through the reference hierarchy starting from i and ending up at j , and i and j do not have a reference interval in common. If a singular uplink between intervals i and j is asserted (either explicitly or through propagation), *i.e.*, some interval is contained in another and the constraint set is singular, the uplink can be removed if it is an indirect uplink. Additionally, all singular sidelinks or indirect uplinks between i or any of its descendants and j or any of its

⁵Non-singular uplinks or sidelinks should not be removed, as they are still subject to revision and should remain in the active network.

```

to RemoveLink(i,j)
  if SingularP(C(i,j))
    and Refs(i)  $\cup$  Refs(j) =  $\emptyset$ 
    and C(i,j) = N(i,j)
  then Remove C(i,j) from the network
end RemoveLink

```

Figure 3.4: Algorithm for reducing the reference hierarchy branching factor

ancestors may now be removable, since this uplink has added information to the hierarchy. In the example above, the constraints c_{10} , c_{11} and c_{12} are all derived and recorded. Both c_{11} and c_{12} are sidelinks, so both are candidates for removal. There exists a path along which c_{12} can be obtained exactly, hence the relation c_{12} can be removed. However, there exists no reference path that obtains the same relation as c_{11} and therefore the relation c_{11} cannot be removed without loss of information. Note also that c_2 cannot be removed; in general, this is true for all relations between disjoint intervals that have a common reference interval.

3.4.4 Empirical results

The automatic referencing mechanism implemented in TIMELOGIC suffers no loss of information, since each relation is initially computed, and it is not deleted until it has been uniquely determined and it can be obtained indirectly through the reference hierarchy. This initial computation of a relation followed by its deletion may seem to be a lot of extra and unnecessary work—how have we gained anything? It turns out that the greatly reduced average branching factor at each node (and hence reduced propagation) more than offsets any extra work entailed in the automatic management of the reference hierarchy. Unfortunately, it is difficult to get a handle on the kind of improvement this mechanism affords, since it is very dynamic in nature, and therefore very much dependent on the order and kind of relations asserted.

Nevertheless, two experiments have borne out that this approach is generally superior in performance to the flat system. The first experiment consisted of generating a network of 101 intervals and randomly assigning singular relations between any two intervals until all pairs of intervals have singular relations, either posted or derived, that are locally consistent [Vilain and Kautz, 1986]. In all, 450 relations were posted to obtain the total of 5050 singular relations. The results

Statistic	NoRefs	AutoRefs
Branching factor, average	100.0	18.3
# UpLinks, average	19.4	1.4
# SideLinks, average	47.3	1.6
Reference depth, average	1.0	13.8
Propagations, total	775,015.0	252,411.0
<i>Constraints</i> invocations	1,105,346.0	475,585.0

Table 3.6: Propagation statistics on 101 randomly related intervals

are given in Table 3.6, which compares average values per node and total values of several statistics for the flat system (NoRefs) and the automatic referencing system (AutoRefs). Note in particular the sharply reduced average number of uplinks and sidelinks per node.

The second experiment involved planning the task of exchanging the top two blocks of a stack using a two-armed robot, with the additional constraint that there is no room to place either block on the surface supporting the stack in an intermediate state (see [Allen and Koomen, 1983] for a more detailed description of this problem). The planner generates 33 intervals, asserting 65 temporal relations between them. In a flat system, another 1063 constraints are asserted due to constraint propagation, causing some 43000 invocations of the transitivity function *Constraints* and resulting in an average branching factor of 31, *i.e.*, the network ends up almost completely connected. Using the automatic referencing mechanism as described above, the average branching factor and the number of invocations of the transitivity function are roughly cut in half.

3.5 Contexts and Backtracking

A computational reasoning system often needs to add temporary assertions to the knowledge base, such as during the development of alternative branches in a plan, or “what if” reasoning. The standard ways of achieving this is by backtracking over these temporary assertions, or by asserting them in contexts which may later be discarded. The TIMELOGIC system fully supports contextual reasoning as well as backtracking.

3.5.1 Contexts

The TIMELOGIC system starts out with a predefined context with the name τ . This context forms the root of a tree of user-defined contexts. Exactly one context is designated the “current” context at all times. All TIMELOGIC database queries and assertions take place in the current context. Unless a constraint between two intervals is explicitly added to a context, the constraint is inherited from the nearest ancestral context, if there is one. Facilities are provided for defining and deleting contexts, pushing and popping contexts, and explicit context switching. In addition, most database query and assertion functions have an optional context parameter, which when given becomes the current context within the scope of the function. Adding assertions to a context with children is not allowed, as this may cause inconsistencies in the children.

3.5.2 Backtracking

The TIMELOGIC system keeps a full history of intervals defined and constraints asserted. Each successful assertion⁶ results in a new backtracking “point.” If such a backtracking point is given to the function `TIMELOGIC-BACKTRACK`, the TIMELOGIC system reverts to the state it was in *before* the assertion was made that resulted in the backtracking point.

This backtracking mechanism is also used to automatically undo all propagated effects from an assertion that resulted in a database inconsistency, provided the TIMELOGIC property `:AUTO-BACKTRACK` is `:ON`.

3.5.3 Auto-referencing and backtracking

The automatic referencing algorithm described in Section 3.4.3, as well as the algorithms in [Allen, 1983], were presented with the assumption of monotonic updates to the temporal database. In order to support efficient backtracking a forward chaining constraint propagation system must store both the derived constraints themselves and adequate dependency information. Hence, removing relations from the network as suggested in Section 3.4.3. is out of the question. Limited to $O(n^2)$ space, we can still benefit from the auto-referencing algorithm as presented by simply marking relations “broken” where they would have been removed. Propagation does not need to take place across links marked “broken,”

⁶Note that a successful assertion may not result in any changes to the database, but serve only as a consistency check.

and in addition, these marked links can still be used to obtain relational information, circumventing the need for traversing the reference hierarchy.

3.6 The Tempos system

TEMPOS is an extension of *Rhetorical*⁷ (RHET), which is a Horn-clause based knowledge representation and reasoning system implemented in Common Lisp [Allen and Miller, 1988]. In addition to the usual Prolog-like pattern matching and backward chaining capabilities, RHET provides forward chaining, equality reasoning, constraint posting, hierarchical types, and facilities for representing knowledge in the form of frames. The syntactic form of Horn clauses in RHET is a single *consequent* followed by zero or more *antecedents*. Specifically, the format of backward and forward chaining rules is

[*consequent* < ... *antecedents* ...]

[*consequent* < ... *antecedents* ... :FORWARD *trigger*]

Variables are distinguished from literals by preceding them with a question mark.

For instance, the formula

$$\forall x. \text{HUMAN}(x) \supset \text{MORTAL}(x)$$

would be stated in RHET as

[[MORTAL ?x] < [HUMAN ?x]]

or, provided T-HUMAN is a declared RHET type, equivalently as

[[MORTAL ?x*T-HUMAN] <]

The TEMPOS system provides a convenient interface to the TIMELOGIC system in the form of RHET builtins. Facilities are provided for associating TIMELOGIC intervals with RHET terms and asserting and querying temporal constraints between these interval terms, which is described in Sections 3.6.1 and 3.6.2, respectively. In addition, TEMPOS optionally provides axioms for reasoning about the occurrence and recurrence of states and events over time, formally introduced in Chapter 2. These are described in Section 3.7.

⁷Rochester Horn clause Extended Tool Of Research In Characterizing Applied natural Language

3.6.1 Intervals in Tempos

Once the TEMPOS system has been initialized (using the function `RESET-TEMPOS`), any ground RHET term can be (uniquely) associated with a TIMELOGIC interval by passing it to the function `DEFINE-TIME`. This automatically declares the term to be of type `T-TIME`, which is supplied by TEMPOS. Only these predefined terms may be used in the TEMPOS predicates. The association between RHET terms and TIMELOGIC intervals remains in effect until either RHET or TEMPOS is reset.

3.6.2 Constraints on Tempos intervals

TEMPOS provides the full interval logic defined in Section 2.2 as a set of built-in predicates. The primary predicates are

```
[TIME-RELN ?i*T-TIME ?r*T-LISP ?j*T-TIME]
```

```
[TIME-DURN ?i*T-TIME ?d*T-LISP ?j*T-TIME]
```

where `?i` and `?j` are interval terms and `?r` (`?d`) is a lisp term denoting a relational (durational) constraint set (see Sections 3.3.1 and 3.3.3). For convenience, TEMPOS supplies a large number of abbreviated predicates, such as `TIME-BEFORE`, `TIME-WITHIN!`, `TIME-<=`, *etc.* See Section B.4 for a complete list of them.

Asserting a `TIME-RELN` or `TIME-DURN` clause⁸, or any of their abbreviations, causes the indicated constraint between the intervals associated with the interval terms to be added to the TIMELOGIC database. The assertion will fail if adding the constraint would lead to a TIMELOGIC database inconsistency. For instance, the fact that I am leaving for vacation on January 18, 1989, which also happens to be the day my library books are due, is stated as

```
(ASSERT-AXIOMS [TIME-OVERLAPS 18-JAN-89 VACATION-11]
               [TIME-RELN LIBRARY-VISIT-47 (:B :D) 18-JAN-89])
```

In query mode, a ground `TIME-RELN` clause succeeds if the constraint set known to TIMELOGIC is a *subset* of the indicated constraint. A non-grounded clause succeeds if the variables can be bound such that the TIMELOGIC constraints are satisfied, and will backtrack as long as there are more appropriate bindings. For instance, given the above facts the query

```
(PROVE [TIME-RELN LIBRARY-VISIT-47 ?r*T-LISP VACATION-11])
```

⁸which must be fully grounded

succeeds, binding ?*r* to (:B :D :M :O :S).

The predicates **TIME-RELN-P** and **TIME-DURN-P** succeed in query mode if the indicated constraint has a non-empty *intersection* with the constraint set known to **TIMELOGIC**. This predicate may not be asserted. So, given the facts above,

```
(PROVE [TIME-RELN ?i*T-TIME (:D :F :S) 18-JAN-89])
```

will fail since neither :O nor (:B :D) is a subset of (:D :F :S), but

```
(PROVE [TIME-RELN-P ?i*T-TIME (:D :F :S) 18-JAN-89])
```

succeeds, binding ?*i* to **LIBRARY-VISIT-47**. In other words, **TIME-RELN** succeeds if the indicated constraint is *implied* by the actual constraint, and **TIME-RELN-P** succeeds if the indicated constraint is *consistent* with the actual constraint.

Note that the -P predicates are non-monotonic! It may happen that an initially provable -P clause fails once more information has been added. For instance, adding

```
(ASSERT-AXIOMS [TIME-BEFORE LIBRARY-VISIT-47 18-JAN-89])
```

which is consistent with the previously asserted constraint (:B :D), will cause the last query to fail.

3.6.2.1 Skolemizing and constraint posting

It is often necessary or useful to reason about derived intervals without explicitly naming them. **TEMPOS** provides these “Skolem” intervals through the predicate **TIME-SKOLEM**. This predicate fails if its single argument is bound; otherwise it succeeds binding the argument variable to a newly generated **TIMELOGIC** interval.

A powerful feature of **RHET** is the ability to post constraints. Posting a (non-grounded) clause succeeds immediately after annotating the embedded variables appropriately. Subsequent attempts to bind these variables will succeed only if the posted clauses can be proven given the bindings. The **TIME-SKOLEM** predicate makes use of this feature by adding to the **TIMELOGIC** database all temporal constraints posted on its argument variable. Obviously, if the posted constraints turn out to be inconsistent with the **TIMELOGIC** database, these constraints are removed from **TIMELOGIC** and the predicate fails. The constraints are also removed if **RHET** ever backtracks through the **TIME-SKOLEM** predicate. Hence,

```
[AND [POST [TIME-BEFORE ?i*T-TIME 18-JAN-89]]
      [TIME-SKOLEM ?i*T-TIME]]
```

is the same as

```
[AND [TIME-SKOLEM ?i*T-TIME]
      [ASSERT-FACT [TIME-BEFORE ?i*T-TIME 18-JAN-89]]]
```

This capability is exploited heavily by TEMPOS itself in the definition of the occurrence and recurrence axioms.

3.6.2.2 Tempos constraints in context

The TEMPOS system takes care to (automatically) map the tree structure of RHET's contexts onto an isomorphic TIMELOGIC context structure. Hence, contextual reasoning can be applied to temporal facts or assumptions. For example,

```
(ASSERT-AXIOMS [TIME-DISJOINT DINNER-TIME BIKING-TIME]
                [WITH "Scenario-1" [TIME-AFTER BIKING-TIME DINNER-TIME]]
                [WITH "Scenario-2" [TIME-BEFORE BIKING-TIME DINNER-TIME]])
```

causes **DINNER-TIME** to be disjoint from **BIKING-TIME** in the root context, to be before it in the "Scenario-1" context, and to be after it in the "Scenario-2" context. Note that the association of RHET terms with TIMELOGIC intervals is global — only the temporal constraints between intervals are context-sensitive.

3.7 Truth over time in Tempos

The TEMPOS system is also an implementation of the axioms for reasoning about states and events over time developed in Chapter 2. It provides the predicates **TT** (true throughout), **MT** (maximally true), **FT** (false throughout) and **RT** (repeatedly true), and handles sequential truth. Obviously, not the full power of the formal axioms is available — those were defined using first order predicate calculus, whereas the implementation is restricted to Horn clauses. This restriction forces a number of design decisions:

- Axioms using biconditionals ($\iff$) in their formal definitions are implemented using simple, one-way implication. For instance, occurrence can be proven given recurrence, but not *vice versa*.

- Although the formal system contains the universe of all intervals, the implementation is restricted to only those intervals explicitly defined or derived during a proof; hence, nothing is provable in the implementation that depends on all subintervals of a given interval. For instance, the Upward Hereditary axiom (Axiom 2.15) can not be implemented.

3.7.1 Truth Throughout

TEMPOS assumes that the normal way of handling terms over time is that maximal truth is asserted, and truth throughout is queried (by the user or a higher level system). Obviously, one way TEMPOS can prove truth throughout is by showing maximal truth⁹

`[[TT ?i ?x] < [MT ?i ?x]]`

On the other hand, if truth throughout interval i is ever asserted, a forward chaining rule asserts maximal truth over a newly generated interval k containing i (note that the WITHIN predicate allows equality):

`[[AND [MT ?k ?x] [TIME-WITHIN ?i ?k]]
< [TT ?i ?x]
[TIME-SKOLEM ?k] :FORWARD [TT ?i ?x]]`

Since upward inheritance presents difficulties, as explained above, the elements of Shoham's taxonomy used in TEMPOS are downward inheritance, liquidity and concatenation. Downward inheritance can be asserted by the user or inferred from liquidity:

`[[TIME-DOWNWARD ?x] < [TIME-LIQUID ?x]]`

Then another way to prove the truth of x throughout i is by showing that x is downward hereditary and maximally true over j containing i :

`[[TT ?i ?x] < [TIME-DOWNWARD ?x]
[TIME-WITHIN ?i ?j]
[MT ?j ?x]]`

⁹This and subsequent TEMPOS axioms have been simplified for presentation purposes. To avoid textual clutter we assume that the variables ?i, ?j and ?k are of type T-TIME. For the same reason, portions of axioms which prevent circularities or improve performance (*e.g.*, CUT) have been omitted.

Finally, the last way to prove the truth of x throughout i is by showing that x is concatenable and true throughout j and k which together are covered by i :

```
[[TT ?i ?x] < [TIME-CONCATENABLE ?x]
               [TIME-COVER! ?j ?k ?i]
               [TT ?j ?x]
               [TT ?k ?x]]
```

Armed with these axioms, TEMPOS succeeds in the following example:

```
◇ (DEFINE-TIME [JAN-89] [18-JAN-89])
→ ([JAN-89] [18-JAN-89])
◇ (ASSERT-AXIOMS [TT JAN-89 WINTER]
                  [TIME-DOWNWARD WINTER]
                  [TIME-DURING 18-JAN-89 JAN-89])
→ (T T T)
◇ (PROVE [TT 18-JAN-89 WINTER])
→ [TT 18-JAN-89 WINTER]
```

3.7.2 Verification of temporal constraints

The remaining TEMPOS axioms to be discussed, namely occurrence of sequence components and recurrence, assert the existence of subintervals (see Axioms 2.33 and 2.35) over which the components occur. Moreover, the occurrence of the components of these subintervals often imposes other temporal constraints, both due to the axiomatization and to domain knowledge. The **TIME-SKOLEM** predicate is particularly effective here, by both generating intervals and installing the posted temporal constraints on them.

When constructing an interval i over which x is maximally true, TEMPOS makes use of Axiom 2.26 to ensure that i is either equal to or disjoint from any other interval j over which x is maximally true. This could be done using a forward chaining rule triggered by the assertion that x is maximally true over i but this leads to backtracking problems in the case where the temporal constraints turn out to be impossible. Hence the following TEMPOS axiom is used to post the constraints prior to the creation of i .

```
[[MT-Verify-Unique ?i ?x]
 < [FORALL ?j [MT ?j ?x]
   [POST [TIME-RELN ?i (:E :DIS) ?j]]]]
```

In a similar way, TEMPOS aids in maintaining domain consistency by providing the predicate

[TIME-MUTEX ?x ?y]

This can be asserted for any two entities to indicate that any occurrences of ?x and ?y must be mutually exclusive, *i.e.*, the intervals over which they occur must be disjoint. For instance,

(ASSERT-AXIOMS [TIME-MUTEX NIGHT DAY])

ensures that it is never night and day at the same time.

TEMPOS ensures that all such domain constraints are applied when constructing intervals with the following axiom:

```
[[MT-Verify-Mutex ?i ?x]
 < [FORALL (?j ?y) [AND [MT ?j ?y] [TIME-MUTEX ?x ?y]]
   [POST [TIME-DISJOINT ?i ?j]]]]
```

These two axioms are combined into a single verification axiom which posts the various verification constraints and then ensures that the temporal argument either was bound (in which case all the postings would have been proven) or gets bound to a newly generated interval (which causes all posted temporal constraints to be added to the TIMELOGIC database).

```
[[MT-Verify ?i] < [MT-Verify-Unique ?i]
                  [MT-Verify-Mutex ?i]
                  [OR [BOUND ?i]
                      [TIME-SKOLEM ?i]]]
```

3.7.3 Maximal Truth from Sequential Truth

TEMPOS is able to prove that some entity x is maximally true over some interval i if the fact has been asserted that some sequence containing x occurs over interval k . It does this by generating ("Skolemizing") an interval and applying the appropriate temporal constraints, as explained in the previous section. A sequence is any three-element compound RHET term starting with the literal SEQ. There are two TEMPOS axioms, one for each component of a sequence, that accomplish the inference:

```
[[MT ?i ?x] < [MT ?k [SEQ ?x ?y]]
               [POST [TIME-COVER! ?i ?j ?k]]
               [MT-Verify ?i ?x]
               [MT-Verify ?j ?y]]
```

```

[[MT ?j ?y] < [MT ?k [SEQ ?x ?y]]
  [POST [TIME-COVER! ?i ?j ?k]]
  [MT-Verify ?i ?x]
  [MT-Verify ?j ?y]]

```

Note that these axioms prove occurrence of sequence components but do not assert them. Repeated proofs will generate different Skolem intervals for each component occurrence, but the **MT-Verify** clauses will ensure that each of those Skolem intervals will coincide.

The following is a trace of a sequential occurrence example in TEMPOS, proving that a store visit occurs over some interval finishing the interval **TESTTIME**, given that a sequence of a bank visit followed by a store visit occurs over **TESTTIME**:

```

◇ (DEFINE-TIME [TESTTIME])
→ ([TESTTIME])
◇ (ASSERT-AXIOMS [MT TESTTIME [SEQ BANK-VISIT STORE-VISIT]])
→ (T)
◇ (PROVE [AND [MT ?when STORE-VISIT]
  [TIME-RELN ?when ?rel TESTTIME]])
→ [AND [MT ?when~Time-Skolem-0043 STORE-VISIT]
  [TIME-RELN ?when~Time-Skolem-0043 ?rel~(:F) TESTTIME]]

```

3.7.4 Maximal Truth from Repeated Truth

Two TEMPOS axioms, based on Axiom 2.35, provide the capability to deduce maximal truth of an entity given the recurrence of the entity. The first proves that, given recurrence of x over interval k , x occurs over some interval i which either starts or is equal to k . As above, i is generated automatically by the verification axiom, if necessary.

```

[[MT ?i ?x] < [RT ?k ?x]
  [POST [Time-Reln ?i (:S :E) ?k]]
  [MT-Verify ?i ?x]]

```

The second recurrence axiom handles the recursion in Axiom 2.35, by proving that x recurs over some interval j given that x recurs over some interval k finished by j and that x occurs over some interval i starting or during k and meeting j :

```

[[RT ?j ?x] < [MT ?i ?x]
    [Time-Reln ?i (:S :D) ?k]
    [RT ?k ?x]
    [POST [Time-Meets ?i ?j]]
    [POST [Time-Finishes ?j ?k]]
    [RT-Verify ?j ?x]]

```

Actually, TEMPOS has two more axioms to prove nested recurrence which are specializations of the last axiom in the case of repeated sequences. They are practically identical to the above and therefore omitted.

The **RT-Verify** axiom is a variation on the theme of **MT-Verify**. It verifies that some (possibly generated) interval over which an entity recurs is either equal to or disjoint from or a tail of any other interval over which the entity recurs.

As a final example of the capabilities of TEMPOS, here is a trace of the proof that E.T. will indeed see the light of day even if he lands on earth at night (see Theorem 2.36):

```

◇ (DEFINE-TIME [EARTHTIME] [E-T-LANDS])
→ ([EARTHTIME] [E-T-LANDS])
◇ (ASSERT-AXIOMS [TT E-T-LANDS NIGHT]
    [TIME-DURING E-T-LANDS EARTHTIME]
    [RT EARTHTIME [SEQ NIGHT DAY]]
    [TIME-MUTEX NIGHT DAY])
→ (T T T T)
◇ (PROVE [AND [POST [TIME-RELN ?when (:A :Mi) E-T-LANDS]]
    [POST [TIME-WITHIN ?when EARTHTIME]]
    [MT ?when DAY]])
→ [AND [TIME-RELN ?when~Time-Skolem-0061 (:A :Mi) E-T-LANDS]
    [TIME-WITHIN ?when~Time-Skolem-0061 EARTHTIME]
    [MT ?when~Time-Skolem-0061 DAY]]

```

3.8 Summary

This chapter has presented the implementation of the two systems TIMELOGIC and TEMPOS. The former is a general utility for managing relational and durational constraints in a network of temporal intervals; the latter is an extension of the logic programming and knowledge representation system Rhetorical. The salient features of the two systems have been discussed and illustrated with various examples, all verified in the implementation. One particularly noteworthy

concept discussed and used in various TEMPOS axioms is the ability to generate intervals during proofs and incorporating these intervals in the TIMELOGIC interval network using posted temporal constraints. Details of the actual user interface can be found in appendices A and B.

Chapter 4

Reasoning About Recurrence

In this chapter we take a look at how the logic of recurrence developed in Chapter 2 might be employed in problem solving in a particular domain. The problem we will investigate is that of driving to work with a traffic light along the way. A traffic light is an obvious and illustrative example of events (or states) recurring over time. A planning system for, say, a robot delivery van, can reasonably be expected to deal with traffic lights, which means it must be able to reason about alternating lights and derive the fact that encountering a red light will not prevent it from reaching its destination.

In order to develop this example, we shall need a representation and a reasoning mechanism for the dual of time: space. The next section will present a logic suitable for a restricted form of spatial reasoning, namely reasoning about trajectories, or *paths*, which is adequate for our purposes. Then we define axioms for reasoning about traversing paths and prove a number of theorems, eventually showing how an agent can prove that a destination can be reached no matter what the status of an intervening traffic light is at the time it is reached.

Since we are primarily interested in problem solving involving repeated states and events, we will ignore the more traditional issues in planning, such as the representation of actions, be they primitive or hierarchical, intentionality, and feedback from the environment regarding success or failure of action attempts (see, for instance, Pelavin's modal operators INEV and IFTRIED [Pelavin, 1988]). As these issues are orthogonal to the notion of reasoning about recurrence we believe that our treatment of recurrence can easily be incorporated into such planning systems.

4.1 Path Logic

We shall use spatial intervals to represent trajectories, or paths. Intervals of one-dimensional space are individuals in a sort called *paths*, distinguished by the predicate `PATH`. Since they are the only spatial objects we will consider, paths will also occasionally be referred to as *locations*.

The complete path logic is obtained simply by taking the set of temporal predicates defined in Section 2.2 and prefixing them with `P-`, with the understanding that the resulting predicates apply only to individuals of type `PATH`. So, for instance, the fact that Main Street connects to South Street is represented as

`P-MEETS(MAIN, SOUTH)`

For ease of reading we shall use the convention that variables *p*, *q*, *r* and *s* are implicitly typed with the `PATH` predicate.

Obviously, restricting ourselves to one-dimensional space is a rather gross simplification,¹ but we claim that in many cases a reasoner benefits from (initial) reasoning in simplified worlds, and to postpone reasoning about finer details until later when a good chunk of an otherwise very large search space will have been eliminated. After all, one dimension is adequate for reasoning about getting from *P* to *Q*, as long as we don't need to account for loops or multiple, distinct paths to *Q*. A more realistic treatment of our traffic problem would need to represent space in at least two dimensions. The ramifications of extending the spatial reasoning to two dimensions will be explored later in Section 5.4.2.

4.1.1 Being places

In order to reason about the location of entities, we introduce the function `at`, which maps pairs of entities and locations $\langle x, p \rangle$ to entities representing the presence of entity *x* at location *p*. We can use this function, for instance, to express the fact that my dog Fido spends the day in the house as

`TT(18-JAN-89, at(FIDO, HOUSE34))`

¹Directing one-dimensional traffic, with or without a light, surely is an academic exercise! Perhaps it comes as no surprise that the implementation of the path logic, a straightforward copy of the TEMPOS system applying TIMELOGIC to a separate network of spatial intervals, is called PATHOS...

The following axioms formally define the notion of presence. First of all, being somewhere is a property of the entity involved: if Fido spends the day in the house then he also spends the morning and the afternoon in the house.

Axiom 4.1 *Being present anywhere is concatenable and both downward and upward hereditary:*

$$\forall x, p. \text{PROPERTY}(\text{at}(x, p))$$

The next axiom reflects our intuition that being in, on or at a location implies that we are in, on or at all containing locations: if Fido is in his basket, he is also in the room, in the house, in the city, *etc.*

Axiom 4.2 *Presence is transitive with respect to location:*

$$\forall i, x, p, q. \text{TT}(i, \text{at}(x, p)) \wedge \text{P-WITHIN}(p, q) \supset \text{TT}(i, \text{at}(x, q))$$

And finally, Fido had better not be in the yard and the house at the same time!

Axiom 4.3 *Entities can only be in one location at the time:*

$$\forall x, p, q. \text{P-DISJOINT}(p, q) \supset \text{MUTEX}(\text{at}(x, p), \text{at}(x, q))$$

To illustrate the various inferences possible with these three axioms together with those developed in Chapter 2, let us assume that the following facts have been asserted:²

DURING(18-JAN-89, JAN-89)
P-DURING(ROCHESTER, NYSTATE)
P-BEFORE(PARIS, NYSTATE)
TT(JAN-89, at(ME, ROCHESTER))
TT(VACATION79, at(ME, PARIS))

Then the following facts are derivable, by virtue of the axioms:

P-BEFORE(PARIS, ROCHESTER)	– Path Logic
DISJOINT(JAN-89, VACATION79)	– Axioms 4.3 and 2.20
DISJOINT!(18-JAN-89, VACATION79)	– Axioms 4.3 and 2.20, Interval Logic
TT(JAN-89, at(ME, NYSTATE))	– Axiom 4.2
TT(18-JAN-89, at(ME, NYSTATE))	– Axioms 4.1 and 2.14
FT(VACATION79, at(ME, ROCHESTER))	– Axioms 4.3 and 2.24
FT(VACATION79, at(ME, NYSTATE))	– Axioms 4.3 and 2.24
FT(JAN-89, at(ME, PARIS))	– Axioms 4.3 and 2.24
FT(18-JAN-89, at(ME, PARIS))	– Axioms 4.3 and 2.24

²For the sake of brevity, we leave it to the reader to infer the appropriate type assertions.

Actually, quite a few more facts are known — these are just the interesting ones. The time and path logic axioms assert the existence of “bridging” intervals, such as the path between PARIS and NYSTATE and the two times on either side of 18-JAN-89 that make up the rest of JAN-89. These, of course, would also appear in derived facts. It is also very obvious from this example that the knowledge of mutual exclusion contributes significantly to the derivation of facts.

Closely related to transitivity of presence with respect to location is the notion that the time an entity is present at a certain location must be within the time it is at an enclosing location: the time Fido is in his basket must be within the time he is at home (unless it is a different occasion, in which case the times had better be disjoint).

Theorem 4.4 *Presence and spatial containment imply either temporal containment or temporal separation:*

$$\forall i, j, x, p, q. MT(i, at(x, p)) \wedge MT(j, at(x, q)) \wedge P\text{-}WITHIN(p, q) \\ \supset WITHIN(i, j) \vee DISJOINT(i, j)$$

Proof

Follows trivially from Axioms 4.2 and 2.26.

■

4.1.2 Going places

Now that we can represent and reason about space as well as time we are ready to combine the two into reasoning about motion or *path traversal*. Obviously, one-dimensional space allows bi-directional motion. Because of the symmetry, however, we shall concern ourselves only with motion in one direction. Functions and axioms analogous to the ones defined in this section could easily be defined to account for motion in the reverse direction.

Let *traverse* be a function mapping pairs of entities and paths $\langle x, p \rangle$ into entities representing the event of entity x traversing path p . The following four axioms define the behavior of traversal entities with respect to time and space. First of all, traversing a path can occur over nonoverlapping intervals only:

Axiom 4.5 *Any traversal entity is solid:*

$$\forall x, p. SOLID(traverse(x, p))$$

Secondly, we can't hope to traverse a path over an interval of time unless we are actually on the path during that time:

Axiom 4.6 *Traversing path p implies being on p :*

$$\text{MT}(i, \text{traverse}(x, p)) \supset \text{TT}(i, \text{at}(x, p))$$

Note that we could have used TT instead of MT to indicate $\text{traverse}(x, p)$. Axioms 4.5 and 2.25 ensure that they are effectively equivalent for traversal. Note also that the time x is at p may be longer than the time x is traversing p .

Thirdly, we can traverse a path over an interval of time by first traversing an initial path segment and then traversing the remaining path in the remaining time.

Axiom 4.7 *Path traversal is constructive:*

$$\begin{aligned} \forall k, x, r. [\exists i, j, p, q. \text{COVER}(i, j, k) \wedge \text{P-COVER!}(p, q, r) \\ \wedge \text{MT}(i, \text{traverse}(x, p)) \wedge \text{MT}(j, \text{traverse}(x, q))] \\ \supset \text{MT}(k, \text{traverse}(x, r)) \end{aligned}$$

The inverse, stated as two axioms, forces traversal of paths to be smooth and continuous, *i.e.*, no subpath can be skipped and all subpaths are traversed in the appropriate order. This is a bit more obscure than the previous axiom because the axiomatization of the interval logic (both time and space) forces us to be careful about asserting the existence of subintervals or subpaths.

Axiom 4.8 *Path traversal is smooth and continuous:*

$$\begin{aligned} \forall i, j, k, x, r. \text{COVER}(i, j, k) \wedge \text{MT}(k, \text{traverse}(x, r)) \\ \supset [\exists p, q. \text{P-COVER!}(p, q, r) \wedge \\ \text{MT}(i, \text{traverse}(x, p)) \wedge \text{MT}(j, \text{traverse}(x, q))] \\ \\ \forall k, x, p, q, r. \text{P-COVER!}(p, q, r) \wedge \text{MT}(k, \text{traverse}(x, r)) \\ \supset [\exists i, j. \text{COVER}(i, j, k) \wedge \\ \text{MT}(i, \text{traverse}(x, p)) \wedge \text{MT}(j, \text{traverse}(x, q))] \end{aligned}$$

Note that motion is implied by the combination of Axioms 4.6 and 4.8. Any traversal of a path r for which a beginning segment p and a concluding segment q can be distinguished implies presence on p initially and presence on q finally. Axiom 4.3 ensures that the time of presence on p is strictly before the time of presence on q .

Finally, the purpose of path traversal is to end up at a location next to the one being traversed.

Figure 4.1: Temporal layout of the trip to the store

Axiom 4.9 *Traversing a path p over time i causes entity x to be at all paths q adjacent to p over some time j met by i :*

$$\forall i, x, p, q. MT(i, \text{traverse}(x, p)) \wedge \text{P-MEETS}(p, q) \\ \supset \exists j. \text{MEETS}(i, j) \wedge MT(j, \text{at}(x, q))$$

Let's take a little trip to the store along Main Street and South Street, and see where we end up! Here are the facts:

P-COVER!(MAIN, SOUTH, TRIP49)
P-MEETS(TRIP49, STORE711)
MT(TIME66, traverse(WE, TRIP49))

Figure 4.1 shows the layout of the relevant intervals, either given or introduced in the following derivation. First of all, from Axiom 4.6 we know that we are at the trip location while traversing it:

$$TT(\text{TIME66}, \text{at}(\text{WE}, \text{TRIP49}))$$

Secondly, from Axiom 4.8 we know that there are two times, say TSK_1 and TSK_2 , during which we traverse Main and South, respectively:

$$MT(\text{TSK}_1, \text{traverse}(\text{WE}, \text{MAIN})) \wedge MT(\text{TSK}_2, \text{traverse}(\text{WE}, \text{SOUTH}))$$

which, by an other application of Axiom 4.6, implies

$$TT(\text{TSK}_1, \text{at}(\text{WE}, \text{MAIN})) \wedge TT(\text{TSK}_2, \text{at}(\text{WE}, \text{SOUTH}))$$

Finally, from Axiom 4.9 we know that there is a time, say TSK_3 , at which we are indeed at the store:

$$\text{MEETS}(\text{TIME66}, \text{TSK}_3) \wedge MT(\text{TSK}_3, \text{at}(\text{WE}, \text{STORE711}))$$

Again, there are numerous other facts as well, all derivable from the initial three, including the fact that there is some location PSK_1 which covers TRIP49 and STORE711 that we were on over some time TSK_4 which includes TIME66 and TSK_3 .

4.2 The logic of traffic lights

A traffic light is defined as a light at some location switching between red and green.³ Let **light** be a function mapping a location to the property of a traffic light placed at that location; and let **red** and **green** be functions that map locations to the property of the light at that location being red and green, respectively. The following axioms define the spatio-temporal behavior of traffic lights.

Axiom 4.10 *The presence of a light at some location, and the redness and greenness of such a light are properties:*

$$\forall p. \text{PROPERTY}(\text{light}(p)) \wedge \\ \text{PROPERTY}(\text{red}(p)) \wedge \text{PROPERTY}(\text{green}(p))$$

Axiom 4.11 *Having a traffic light at location p over time i implies a light at p repeatedly switching between red and green over i :*

$$\forall i, p. \text{MT}(i, \text{light}(p)) \supset \text{RT}(i, \text{seq}(\text{red}(p), \text{green}(p)))$$

Axiom 4.12 *Redness and greenness of a light are time-wise mutually exclusive:*

$$\forall p. \text{MUTEX}(\text{red}(p), \text{green}(p))$$

We now introduce the functions **guarded**, which maps a path into the condition indicating that there is a traffic light somewhere along the path, and **blocked**, which maps a path into the condition that prevents it from being traversed due to the light on it being red.

Axiom 4.13 *Being guarded or blocked is a property of all paths:*

$$\forall p. \text{PROPERTY}(\text{blocked}(p))$$

$$\forall p. \text{PROPERTY}(\text{guarded}(p))$$

Note that the fact that guardedness is a property refers to how it distributes over time, *not* over space. If a path p is guarded over time i then it is guarded over all subintervals of i , but it is not necessarily true that any or all (proper) subpaths of p are guarded! However, we will define guardedness such that the inverse is true: if any subpath is guarded then the whole path is considered guarded.

³Without loss of generality we shall restrict ourselves to a simple red/green alternation. Including an amber light expands the proofs but does not alter the approach.

Axiom 4.14 *A path is guarded if and only if it has a light on it or it has a guarded subpath:*

$$\forall i, p \exists q. \text{P-WITHIN}(q, p) \wedge \text{TT}(i, \text{light}(q)) \iff \text{TT}(i, \text{guarded}(p))$$

There is (in our world) exactly one way for a path to be blocked for some time, namely by having a traffic light on it that is red at the time.

Axiom 4.15 *A path with a red light is a blocked path:*

$$\forall i, p. \text{TT}(i, \text{red}(p)) \supset \text{TT}(i, \text{blocked}(p))$$

Axiom 4.16 *A path with a green light is not a blocked path:*

$$\forall i, p. \text{TT}(i, \text{green}(p)) \supset \text{FT}(i, \text{blocked}(p))$$

Axiom 4.17 *An unguarded path is never blocked:*

$$\forall i, p. \text{FT}(i, \text{guarded}(p)) \supset \text{FT}(i, \text{blocked}(p))$$

These axioms circumscribe the **blocked** condition to exactly the case of a red light [McCarthy, 1980]. We could have accomplished this more simply by making the implication in Axiom 4.15 a biconditional and omitting Axiom 4.16. Then Axiom 4.17 would have been a theorem. However, we used the above approach because it is more general. A more fully developed world description might add conditions such as **construction** or **congested** as alternative reasons for a path to be blocked. Note that it is important to use FT instead of $\neg\text{TT}$:

$$\forall i, p. \neg \text{TT}(i, \text{guarded}(p)) \wedge \neg \text{TT}(i, \text{congested}(p))$$

does not imply that p is unguarded and uncongested at the same time, hence it is still possible for p to be blocked throughout i .

4.3 Taking a trip

Since pedestrians and bicyclists tend to ignore traffic lights, let us define what it means to be driving in our one-dimensional, uni-directional world. Let **driving** be a function mapping the pairs $\langle x, p \rangle$ to an entity representing the state of entity x driving along path p . Since driving is a property of an individual at a certain location, we first assert

Axiom 4.18 *Driving anywhere is concatenable and both downward and upward hereditary:*

$$\forall x, p. \text{PROPERTY}(\text{driving}(x, p))$$

Axiom 4.19 *Driving anywhere implies being there:*

$$\forall i, x, p. \text{TT}(i, \text{driving}(x, p)) \supset \text{TT}(i, \text{at}(x, p))$$

The purpose of driving on a path is to traverse the path. However, as anyone familiar with urban driving knows, driving does not mean continuously making progress towards a destination — sometimes driving means waiting due to various causes. Hence driving on a path results in traversing it only if it is not blocked:

Axiom 4.20 *Driving along an unblocked path implies traversing it:*

$$\begin{aligned} \forall i, x, p. \text{TT}(i, \text{driving}(x, p)) \wedge \text{FT}(i, \text{blocked}(p)) \\ \supset \exists j. \text{WITHIN}(j, i) \wedge \text{MT}(j, \text{traverse}(x, p)) \end{aligned}$$

An interesting implication of this definition is that, if traversal occurs at all, it occurs at the smallest subinterval j that finishes i . This is so because the definition of traversal (Axiom 4.9) implies that x ends up at some location q adjacent to p over some interval k met by j . Axiom 4.3 forces x to be in one place at the time only, and since p and q are disjoint i and k must also be. So we have

$$j \{d, e, f, s\} i \wedge i \{a, b, m, \overline{m}\} k \wedge j \{m\} k$$

The interval logic axioms force this to be equivalent to

$$j \{e, f\} i \wedge i \{m\} k \wedge j \{m\} k$$

The only times traversal is going to occur are those times that both **driving** is TT and **blocked** is not TT. However, if there are such times at all, then the condition also holds for all subintervals of those times since both **driving** and **blocked** are properties. This forces the existential time for traversal to be the smallest subinterval. The obvious way to avoid this is to rephrase the axiom so as to restrict the universal quantification of i to only those maximal intervals over which both **driving** is TT and **blocked** is not TT. This would make our subsequent proofs significantly more complex, however, and we would gain little by it in terms of recurrence reasoning.

Finally, we need a persistence axiom that states that, if you ever find yourself driving on a blocked path, then you keep on driving (*i.e.*, waiting) until the path is no longer blocked. Without this we would not be able to infer any progress at all as soon as a blocked path is encountered.

Axiom 4.21 *If x is maximally driving on path p over time i and path p is maximally blocked over time j then either i and j are disjoint or i ends later than j :*

$$\begin{aligned} & \forall i, j, x, p. \text{MT}(i, \text{driving}(x, p)) \wedge \text{MT}(j, \text{blocked}(p)) \\ & \supset \text{DISJOINT}(i, j) \otimes [\exists k. \text{MEETS}(j, k) \wedge \text{FINISHES}(k, i)] \end{aligned}$$

4.3.1 Driving through a traffic light

The question now before us is: given the knowledge we have built up about spatial and temporal intervals, traffic lights, path traversal and recurrence, is it possible to drive through a traffic light? We will state this as a theorem, and prove it as follows.

Theorem 4.22 *If x is driving at a location p over time i during k while a light is active, then there exists a time j within k at which x traverses p :*

$$\begin{aligned} & \forall i, k, x, p. \text{TT}(i, \text{driving}(x, p)) \wedge \text{DURING}(i, k) \wedge \text{MT}(k, \text{light}(p)) \\ & \supset \exists j. \text{WITHIN}(j, k) \wedge \text{MT}(j, \text{traverse}(x, p)) \end{aligned}$$

Note that this theorem implies that the traversal occurs while the light is operating. Since the light operates over the maximal interval k , picking an interval j after k to traverse p would trivialize the problem!

Proof

1: $\text{TT}(i, \text{driving}(x, p))$	; hypothesis
2: $\text{DURING}(i, k)$	; hypothesis
3: $\text{MT}(k, \text{light}(p))$	; hypothesis
4: $\text{DOWNWARD}(\text{driving}(x, p))$	; (A4.18)
5: $\text{WITHIN}(i, k)$	; (2)+def'n(WITHIN)
6: $\text{TT}(i, \text{blocked}(p)) \vee \neg \text{TT}(i, \text{blocked}(p))$	; tautology

Prove by case:

a) *Suppose p is not blocked at time i ...*

7: $\neg \text{TT}(i, \text{blocked}(p))$	; assumption
8: $\exists j. \text{WITHIN}(j, i) \wedge \text{FT}(j, \text{blocked}(p))$	; MP(7)(A2.22)
9: $\text{WITHIN}(\text{sk}_1, i)$	; exist.elim.(8)
10: $\text{FT}(\text{sk}_1, \text{blocked}(p))$	; exist.elim.(8)
11: $\text{WITHIN}(\text{sk}_1, k)$	; MP(9)(5)(T2.6)

Figure 4.2: One possible temporal layout, second half of the proof of Theorem 4.22

-
12: $\text{TT}(\text{sk}_1, \text{driving}(x, p))$	; MP(4)(9)(1)(A2.14)
13: $\exists j. \text{WITHIN}(j, \text{sk}_1) \wedge \text{MT}(j, \text{traverse}(x, p))$	; MP(12)(10)(A4.20)
14: $\text{WITHIN}(\text{sk}_2, \text{sk}_1)$	; exist.elim.(13)
15: $\text{MT}(\text{sk}_2, \text{traverse}(x, p))$	; exist.elim.(13)
16: $\text{WITHIN}(\text{sk}_2, k)$	; MP(14)(11)(T2.6)
17: $\exists j. \text{WITHIN}(j, k) \wedge \text{MT}(j, \text{traverse}(x, p))$	; exist.intro.(16)(15)

□

b) Suppose p is blocked at time i ...

Figure 4.2 shows one possible interval layout for this branch of the proof.

18: $\text{TT}(i, \text{blocked}(p))$	; assumption
19: $\exists j. \text{WITHIN}(i, j) \wedge \text{MT}(j, \text{blocked}(p))$	; MP(18)(A2.28)
20: $\text{WITHIN}(i, \text{sk}_3)$	; exist.elim.(19)
21: $\text{MT}(\text{sk}_3, \text{blocked}(p))$	; exist.elim.(19)
22: $\text{RT}(k, \text{seq}(\text{red}(p), \text{green}(p)))$	; MP(3)(A4.11)
23: $\text{DOWNWARD}(\text{green}(p))$	; MP(A4.10)(A2.16)
24: $\exists j. \text{WITHIN}(j, k) \wedge \text{LATER}(j, i) \wedge \text{TT}(j, \text{green}(p))$	; MP(22)(2)(23)(T2.36)
25: $\text{WITHIN}(\text{sk}_4, k)$	; exist.elim.(24)
26: $\text{LATER}(\text{sk}_4, i)$	; exist.elim.(24)

27: $TT(sk_4, \text{green}(p))$	; exist.elim.(24)
28: $FT(sk_4, \text{blocked}(p))$	; MP(27)(A4.16)
29: $\exists j. \text{WITHIN}(i, j) \wedge MT(j, \text{driving}(x, p))$	; MP(1)(A2.28)
30: $\text{WITHIN}(i, sk_5)$	; exist.elim.(29)
31: $MT(sk_5, \text{driving}(x, p))$	; exist.elim.(29)
32: $\neg \text{DISJOINT}(sk_3, sk_5)$	; MP(20)(30)(Int.logic)
33: $\exists k. \text{MEETS}(sk_3, k) \wedge \text{FINISHES}(k, sk_5)$	; MP(21)(31)(32)(A4.21)
34: $\text{MEETS}(sk_3, sk_6)$	; exist.elim.(33)
35: $\text{FINISHES}(sk_6, sk_5)$	; exist.elim.(33)
36: $TT(sk_6, \text{driving}(x, p))$	; MP(35)(31)(A2.14)
37: $\exists k. \text{MEETS}(sk_3, k) \wedge FT(k, \text{blocked}(x, p))$	; MP(21)(T2.29)
38: $\text{MEETS}(sk_3, sk_7)$	; exist.elim.(37)
39: $FT(sk_7, \text{blocked}(x, p))$	; exist.elim.(37)
40: $\exists j. \text{WITHIN}(j, sk_6) \wedge \text{WITHIN}(j, sk_7)$	; MP(34)(38)(Int.logic)
41: $\text{WITHIN}(sk_8, sk_6)$	; exist.elim.(40)
42: $\text{WITHIN}(sk_8, sk_7)$	; exist.elim.(40)
43: $TT(sk_8, \text{driving}(x, p))$	; MP(36)(41)(A2.14)
44: $FT(sk_8, \text{blocked}(x, p))$	; MP(39)(42)(A2.14)
45: $\exists j. \text{WITHIN}(j, sk_8) \wedge MT(j, \text{traverse}(x, p))$	; MP(43)(44)(A4.20)
46: $\text{WITHIN}(sk_9, sk_8)$	; exist.elim.(45)
47: $MT(sk_9, \text{traverse}(x, p))$	; exist.elim.(45)
48: $\text{WITHIN}(sk_9, k)$	; MP(46)(44)(28)(25)
49: $\exists j. \text{WITHIN}(j, k) \wedge MT(j, \text{traverse}(x, p))$	; exist.intro.(48)(47)

■

4.3.2 Driving past a traffic light

Now that we have shown that it is possible to drive through a traffic light, can we extend this to show that it is possible to traverse a path by driving, even if the path contains a subpath with a traffic light on it?

Before we proceed to state this as a theorem and sketch the proof we will need another persistence axiom. The first persistence axiom ensured that we would continue to drive until the blocked path on which we are driving stops being blocked. The next persistence axiom ensures that we keep driving as long as we haven't reached the end of the road being driven.

Axiom 4.23 *If x is driving on path p then x continues driving while not at the end of the path:*

$$\begin{aligned} \forall i, j, x, p, q. & \text{TT}(i, \text{driving}(x, p)) \wedge \text{TT}(j, \text{at}(x, q)) \\ & \wedge \text{FINISHES}(j, i) \wedge \text{P-WITHIN!}(q, p) \wedge \neg \text{P-FINISHES}(q, p) \\ & \supset \exists k. \text{MEETS}(i, k) \wedge \text{TT}(k, \text{driving}(x, p)) \end{aligned}$$

The problem of traversing a path by driving, even if it means getting past a traffic light, is stated as a theorem and proven as follows:

Theorem 4.24 *If x is located at path p at time i and starts driving on path q next to p at time j met by i , then there exists a time k over which x traverses q :*

$$\begin{aligned} \forall i, j, x, p, q. & \text{MEETS}(i, j) \wedge \text{P-MEETS}(p, q) \\ & \wedge \text{TT}(i, \text{at}(x, p)) \wedge \text{TT}(j, \text{driving}(x, q)) \\ & \supset \exists k. \text{LATER}(k, j) \wedge \text{MT}(k, \text{traverse}(x, q)) \end{aligned}$$

Note that the precondition that x initially be on p preceding q forces q to be traversed in its entirety, if at all.

Proof

1: $\text{MEETS}(i, j)$	; hypothesis
2: $\text{P-MEETS}(p, q)$	; hypothesis
3: $\text{TT}(i, \text{at}(x, p))$	; hypothesis
4: $\text{TT}(j, \text{driving}(x, q))$	; hypothesis
5: $\text{TT}(j, \text{guarded}(q)) \vee \neg \text{TT}(j, \text{guarded}(q))$	; tautology

Prove by case:

a) *Suppose q is not guarded at time j ...*

6: $\neg \text{TT}(j, \text{guarded}(q))$	; assumption
7: $\exists l. \text{WITHIN}(l, j) \wedge \text{FT}(l, \text{guarded}(q))$	; MP(6)(A2.22)
8: $\text{WITHIN}(\text{TSK}_1, j)$	; exist.elim.(7)
9: $\text{FT}(\text{TSK}_1, \text{guarded}(q))$	; exist.elim.(7)
10: $\text{FT}(\text{TSK}_1, \text{blocked}(q))$	; MP(9)(A4.17)
11: $\text{TT}(\text{TSK}_1, \text{driving}(x, q))$	; MP(8)(4)(A2.14)
12: $\exists l. \text{WITHIN}(l, \text{TSK}_1) \wedge \text{MT}(l, \text{traverse}(x, q))$	; MP(11)(10)(A4.20)

□

b) Suppose q is guarded at time j ...

- 13: $TT(j, \text{guarded}(q))$; assumption
- 14: $\exists r. P\text{-WITHIN}(r, q) \wedge TT(j, \text{light}(r))$; MP(13)(A4.14)
- 15: $P\text{-WITHIN}(\text{PSK}_1, q)$; exist.elim.(14)
- 16: $TT(j, \text{light}(\text{PSK}_1))$; exist.elim.(14)
- 17: $P\text{-EQUALS}(\text{PSK}_1, q) \vee P\text{-WITHIN}!(\text{PSK}_1, q)$; def'n(P-WITHIN)

b.1) Suppose there is a light on q ...

- 18: $P\text{-EQUALS}(\text{PSK}_1, q)$; assumption
- 19: $TT(j, \text{light}(q))$; MP(16)(18)
- 20: $\exists l. \text{WITHIN}(j, l) \wedge MT(l, \text{light}(q))$; MP(19)(A2.28)
- 21: $\text{WITHIN}(j, \text{TSK}_2)$; exist.elim.(20)
- 22: $MT(\text{TSK}_2, \text{light}(q))$; exist.elim.(20)
- 23: $\exists l. \text{WITHIN}(l, \text{TSK}_2) \wedge MT(l, \text{traverse}(x, q))$; MP(4)(21)(22)(T4.22)

□

b.2) Suppose the light is on a proper subpath of q ...

This can be handled by induction, traversing the relevant subpaths and using the construction axiom to prove that the entire path is traversed.

- 24: $P\text{-WITHIN}!(\text{PSK}_1, q)$; assumption
- 25: $P\text{-STARTS}(\text{PSK}_1, q) \vee P\text{-DURING}(\text{PSK}_1, q)$; (24)+def'n(P-WITHIN!)
 $\vee P\text{-FINISHES}(\text{PSK}_1, q)$

b.2.1) Suppose the light is on a path starting q ...

- 26: $P\text{-STARTS}(\text{PSK}_1, q)$; assumption
- 27: $\exists r. P\text{-MEETS}(\text{PSK}_1, r) \wedge P\text{-FINISHES}(r, q)$; (26)+def'n(P-STARTS)
- 28: $P\text{-MEETS}(\text{PSK}_1, \text{PSK}_2)$; exist.elim.(27)
- 29: $P\text{-FINISHES}(\text{PSK}_2, q)$; exist.elim.(27)
- 30: $\exists l. \text{STARTS}(l, j) \wedge TT(l, \text{driving}(x, \text{PSK}_1))$; MP(1)(2)(26)
- 31: $\text{STARTS}(\text{TSK}_3, j)$; exist.elim.(30)
- 32: $TT(\text{TSK}_3, \text{driving}(x, \text{PSK}_1))$; exist.elim.(30)
- 33: $\text{MEETS}(i, \text{TSK}_3)$; MP(1)(31)(Int.logic)
- 34: $P\text{-MEETS}(p, \text{PSK}_1)$; MP(1)(26)(Path logic)

Inductive step on beginning of path...

- 35: $\text{MEETS}(i, \text{TSK}_3) \wedge P\text{-MEETS}(p, \text{PSK}_1)$; conj.intro.(33)(34)
- 36: $TT(i, \text{at}(x, p)) \wedge TT(\text{TSK}_3, \text{driving}(x, \text{PSK}_1))$; conj.intro.(3)(32)

- 37: $\exists k. \text{LATER}(k, \text{TSK}_3) \wedge \text{MT}(k, \text{traverse}(x, \text{PSK}_1))$; MP(35)(36)(T4.24)
38: $\text{LATER}(\text{TSK}_4, \text{TSK}_3)$; exist.elim.(37)
39: $\text{MT}(\text{TSK}_4, \text{traverse}(x, \text{PSK}_1))$; exist.elim.(37)
40: $\exists l. \text{MEETS}(\text{TSK}_4, l) \wedge \text{MT}(l, \text{at}(x, \text{PSK}_2))$; MP(39)(28)(A4.9)
41: $\text{MEETS}(\text{TSK}_4, \text{TSK}_5)$; exist.elim.(40)
42: $\text{MT}(\text{TSK}_5, \text{at}(x, \text{PSK}_2))$; exist.elim.(40)
43: $\text{TT}(\text{TSK}_4, \text{at}(x, \text{PSK}_1))$; MP(39)(A4.6)
44: $\exists l. \text{MEETS}(\text{TSK}_4, l) \wedge \text{TT}(l, \text{driving}(x, p))$; MP(43)(4)(26)(A4.23)
45: $\text{MEETS}(\text{TSK}_4, \text{TSK}_6)$; exist.elim.(44)
46: $\text{TT}(\text{TSK}_6, \text{driving}(x, p))$; exist.elim.(44)
47: $\text{TT}(\text{TSK}_6, \text{driving}(x, \text{at}(x, \text{PSK}_2)))$; MP(41)(45)(42)(46)

Inductive step on remainder of path...

- 48: $\text{MEETS}(\text{TSK}_4, \text{TSK}_6) \wedge \text{P-MEETS}(\text{PSK}_1, \text{PSK}_2)$; conj.intro.(45)(28)
49: $\text{TT}(\text{TSK}_4, \text{at}(x, \text{PSK}_1))$; conj.intro.(43)(47)
 $\quad \wedge \text{TT}(\text{TSK}_6, \text{driving}(x, \text{at}(x, \text{PSK}_2)))$
50: $\exists k. \text{LATER}(k, \text{TSK}_6) \wedge \text{MT}(k, \text{traverse}(x, \text{PSK}_2))$; MP(48)(49)(T4.24)
51: $\text{LATER}(\text{TSK}_7, \text{TSK}_6)$; exist.elim.(50)
52: $\text{MT}(\text{TSK}_7, \text{traverse}(x, \text{PSK}_2))$; exist.elim.(50)
53: $\exists k. \text{COVER}(\text{TSK}_4, \text{TSK}_7, k)$; MP(45)(51)(Int.logic)
54: $\text{COVER}(\text{TSK}_4, \text{TSK}_7, \text{TSK}_8)$; exist.elim.(53)
55: $\text{P-COVER}!(\text{PSK}_1, \text{PSK}_2, q)$; MP(26)(28)(29)
56: $\text{COVER}(\text{TSK}_4, \text{TSK}_7, \text{TSK}_8) \wedge \text{P-COVER}!(\text{PSK}_1, \text{PSK}_2, q)$; conj.intro.(54)(55)
57: $\text{MT}(\text{TSK}_4, \text{traverse}(x, \text{PSK}_1))$; conj.intro.(39)(52)
 $\quad \wedge \text{MT}(\text{TSK}_7, \text{traverse}(x, \text{PSK}_2))$
58: $\text{MT}(\text{TSK}_8, \text{traverse}(x, q))$; MP(56)(57)(A4.7)
59: $\text{COVER}(\text{TSK}_4, \text{TSK}_7, \text{TSK}_8) \wedge \text{LATER}(\text{TSK}_4, \text{TSK}_3)$; conj.intro.(54)(38)
60: $\text{LATER}(\text{TSK}_8, j)$; MP(59)(31)(Int.logic)
61: $\exists k. \text{LATER}(k, j) \wedge \text{MT}(k, \text{traverse}(x, q))$; exist.intro.(60)(58)

□

b.2.2) Suppose the light is on a path in the middle of q...

b.2.3) Suppose the light is on a path finishing q...

These cases are analogous to case b.2.1) and need not be elaborated.

■

Figure 4.3: One possible temporal layout of the robot delivery problem

4.3.3 Driving from A to B past a traffic light

Finally, since we have shown that it is possible to traverse any path by driving, even past a traffic light, we now show that it is possible to drive from anywhere along a path, which may contain a traffic light, to anywhere else. To cast this problem in more concrete terms, consider an automated delivery truck at the farm, faced with the task of getting apples to a store along a path that contains a traffic light. Can it conclude that if it starts driving it will get there eventually? If it does, we would expect the relevant intervals to line up as in Figure 4.3, at least in the scenario where the light was encountered at the time it was red.

The problem of getting from A to B via a traffic light is stated as a theorem and proven as follows:

Theorem 4.25 *If x is located at p at time i and starts driving over a path q started by p which contains a proper subpath r that has a light, then there exists a time j later than i at which x is at a location s following q :*

$$\begin{aligned}
 & \forall i, k, x, p, q, r, s. \text{ P-STARTS}(p, q) \wedge \text{BEFORE}(p, r) \\
 & \quad \wedge \text{P-DURING}(r, q) \wedge \text{P-MEETS}(q, s) \\
 & \quad \wedge \text{TT}(i, \text{at}(x, p)) \wedge \text{TT}(i, \text{driving}(x, q)) \\
 & \quad \wedge \text{DURING}(i, k) \wedge \text{MT}(k, \text{light}(r)) \\
 & \supset \exists j. \text{LATER}(j, i) \wedge \text{TT}(j, \text{at}(x, s))
 \end{aligned}$$

Although Theorem 4.24 shows that any path driven eventually will be traversed, and hence the conclusion follows simply from that theorem and Axiom 4.9, we will sketch the reasoning for completeness. The proof proceeds in three steps:

- Getting from the starting point (p) to the light (r);
- Getting through the light; and
- Getting from just after the light to the destination (s).

Proof

1. *Getting to the light:* Since we know that p is before r the path logic gives us a path between them, say, p_1 . We know that x is driving on p at time i and that p is not guarded, so Axioms 4.20 and 4.9 give us a time i_1 at which x is on p_1 . Since p_1 does not finish q (the whole trip), Axiom 4.23 ensures that x is still driving at time i_1 . Then the same reasoning gets us from p_1 to r where we arrive at time i_2 .
2. *Getting through the light:* Since r is properly within q we know from the path logic that there is a path r_1 immediately following r and finishing q . From the interval logic we know that, since i_2 is after i and i is during k , i_2 must be either within, disjoint from or overlapped by k .
 - (a) If i_2 is within k then we know that x is driving at a time and location where the light is operating. But then Theorem 4.22 shows that there is a time i_3 later than i_2 at which time x will be at r_1 .
 - (b) If i_2 is disjoint from k then we know that r is not guarded, and hence can be traversed. So by Axiom 4.9 we know that there is a time i_3 met by (i.e., later than) i_2 at which time x will be at r_1 .
 - (c) If k overlaps i_2 then from the interval logic we know there is a time i'_2 met by (i.e., disjoint from) k and finishing i_2 . Since being at a location and driving are properties holding over i_2 they also hold over i'_2 . So by the same argument as in (b) there is a time i_3 later than i_2 at which time x will be at r_1 .
3. *Getting to the destination:* Since r_1 does not contain a subpath with a light, we know that r_1 is not guarded, and hence can be traversed. So by Axiom 4.9 we know that there is a time j met by i_3 (and by the interval logic axioms therefore later than i), at which time x will be at s .

■

4.4 Summary

In this chapter we have considered how the logic of recurrence developed in Chapter 2 might be employed in problem solving in a particular domain, namely that of driving in a world with traffic lights. In order to represent knowledge about this world we developed a logic suitable for a restricted form of spatial reasoning, namely one dimensional paths. Then we defined axioms for reasoning about

traversing paths and what it means to drive in this world. Finally we proved two theorems, showing how an agent can be sure that a destination can be reached no matter what the status of an intervening traffic light is at the time it is encountered. This was accomplished by using the recurrence axioms and theorems developed in Chapter 2 and the knowledge that traffic lights are alternating between red and green to show that appropriate intervals for traversing a path containing a light indeed exist.

Chapter 5

Conclusions & Future Work

5.1 Summary

Knowledge of the regular repetition of states and events plays an important role in our understanding of and our ability to act in the world, and we would expect that any artificially intelligent agent would have adequate mechanisms for representing and using such knowledge. The goal of this thesis is to develop a logical formalism based on the interval calculus that provides a problem solver or planner with the capability to represent and reason about knowledge of recurrence.

In Chapter 2 we have presented a formal system for reasoning about temporal intervals, and about the truth of *entities* over these intervals. A series of axioms define the concepts of truth and falsehood over time, maximal truth over time, and repeated truth over time. The logic treats events, states, properties, processes, *etc.*, uniformly as entities (with respect to their “holding” or “occurring” over time), thereby maximizing the generality of occurrence and recurrence reasoning. Entities are differentiated only on the basis of how they distribute over time, which is defined by their *class*, such as downward, liquid, or solid. We have shown how entities can be composed into sequences and defined how these sequences behave over time, both incidentally and repeatedly.

The primary advantage of the approach we have taken is that entities are explicitly associated, by means of predicates, with the times over which they are true (incidentally or repeatedly), which facilitates analysis and prediction for a reasoning system. Making predictions, naturally, is of great value to a planning system.

We presented the implementation of the TIMELOGIC and TEMPOS systems in chapter 3. The TIMELOGIC system is a general utility for managing relational and durational constraints in a network of temporal intervals. The salient and

novel feature of this system is the inclusion of an algorithm for automatically and dynamically creating and maintaining a hierarchy of reference intervals based on temporal containment. Localizing constraint propagation in this way was shown to preserve all information in the original network, and reduce the time complexity of maintaining the network constraints from $O(n^3)$ to $O(\rho^2 n)$. Assuming that in most cases ρ , the size of each reference group, remains small and relatively constant, this is a significant gain.

The TEMPOS system is an extension of the logic programming and knowledge representation system Rhetorical. Although the use of a Horn-clause based system is more restrictive than full First Order Predicate Calculus, the TEMPOS system incorporates most of the axioms developed in Chapter 2, or at least the implicative versions of biconditional axioms. TEMPOS therefore stresses the use of the recurrence axioms in reasoning about the components given the recurrence, rather than in recognizing the recurrence given the components. One particularly noteworthy concept discussed and used in various TEMPOS axioms is the ability to generate intervals during proofs and incorporating these intervals in the TIMELOGIC interval network using posted temporal constraints.

In Chapter 4 we have considered how the logic of recurrence developed in Chapter 2 might be employed in problem solving in a particular domain, namely that of driving in a world with traffic lights. In order to represent knowledge about this world we developed a logic suitable for a restricted form of spatial reasoning, namely one dimensional paths. Then we defined axioms for reasoning about traversing paths and what it means to drive in this world. Finally we proved two theorems, showing how an agent can be sure that a destination can be reached no matter what the status of an intervening traffic light is at the time it is encountered. This was accomplished by using the recurrence axioms and theorems developed in Chapter 2 and the knowledge that traffic lights are alternating between red and green to show that appropriate intervals for traversing a path containing a light indeed exist.

5.2 Extending the temporal logic

5.2.1 Relative durations

Conspicuous by its absence from the entire description of the formalism we have developed for recurrence reasoning is the notion of *duration*. All the reasoning we have demonstrated thus far has been qualitative. How do we express the fact that

Tommy hit the wall repeatedly *for about a minute*? For that matter, how do we prevent any recurrence from being infinite?

An interesting scheme for representing durational information was informally suggested by Allen and Kautz [1985]. They recognized that attaching some number to an interval was meaningless without reference to some scale or some other interval. Hence they proposed to maintain durational information in terms of *ranges* of values that an interval's duration could take on with respect to another interval or unit of time. A range consists of a lower and an upper bound, either of which may be open or closed (indicated with parentheses and square brackets, respectively).¹ A single number n is equivalent to the range $[n\ n]$. For instance, the fact that "The reaction takes two to five minutes to complete" is represented as

REACTION $[2\ 5]$ MINUTES

Allen and Kautz showed how *fuzzy multiplication* could be used to maintain a network of durational constraints. For any three intervals i , j and k , if the durational constraints between i and j and between j and k are known then the durational constraint between i and k is the fuzzy product of the first two durational constraints. The fuzzy product of two ranges r_1 and r_2 is a range whose lower and upper bounds are the numeric products of the lower and upper bounds of r_1 and r_2 , respectively.

Finally, Allen and Kautz showed how the durational logic can be integrated with the (relational) interval logic by exploiting constraints asserted or derived in one to derive constraints in the other. For example, if an interval i is within and interval j , then surely the duration of i with respect to j is less than 1.

The TIMELOGIC system described in Chapter 3 incorporates this duration logic (see Sections 3.3.3 through 3.3.5), and extends it by allowing disjunctions of ranges (*range sets*). Fuzzy multiplication of two range sets results in a set whose members are the fuzzy products of the pairwise combinations of the ranges in each set, where overlapping ranges can be combined. The set notation allows us to distinguish, for instance, the following two propositions regarding John's running around the track:

TOTAL-TRACK-TIME $\{3, 4\}$ SINGLE-TRACK-TIME

TOTAL-TRACK-TIME $[3\ 4]$ SINGLE-TRACK-TIME

¹For simplicity's sake we shall restrict ourselves to closed bounds in the following discussion — incorporating open bounds is straightforward and left as the proverbial exercise for the reader.

The latter formulation would allow partial completion of a track. If the relative variation in SINGLE-TRACK-TIME is large the TOTAL-TRACK-TIME will be the same range in either formulation because of overlap, but if it is small enough, say 95 ± 5 seconds, the range set formulation results in a TOTAL-TRACK-TIME of $\{[270\ 300], [360\ 400]\}$ seconds, and the simple range formulation in $[270\ 400]$ seconds.

5.2.2 Composite durations

Unfortunately, incorporating this durational logic into our formalism is necessary but not sufficient for proper durational recurrence reasoning. This is due to our formulation of recurrence. Recall the Recurrence Axiom (Axiom 2.35), which implied that, if x recurs over interval k , then either x is maximally true over k (which terminates the recurrence), or k covers initial interval i and remainder interval j such that x is maximally true over i and recurs over j . Even if we have durational information about the initial k and we know how long i typically is, we still can't say anything about the duration of the remainder of k , j , and therefore cannot determine when the recurrence ends. In other words, we need *fuzzy addition/subtraction* to determine the duration of j . This would allow us to conclude that, given John ran around the track for 5 minutes and he completes each track in 100 seconds, the interval remaining after the first track has a duration of 200 seconds. Applying the Recurrence Axiom recursively twice more ends the recurrence because the last remainder is equal to the duration of a single track completion and therefore a track running event could not take place over an interval that started the last remainder.

Composite duration can be added fairly easily to a forward chaining system such as TIMELOGIC. Let **duration**(i, j) be a function that returns the duration of i with respect to j . Since the predicate COVER! is not atomic, the system must maintain the equality

$$\forall i, j, k. \text{COVER!}(i, j, k) \supset \\ \forall u. \text{duration}(i, u) + \text{duration}(j, u) = \text{duration}(k, u)$$

by means of three rules:

$$\begin{aligned}
& \forall i, j. \text{MEETS}(i, j) \supset \\
& \quad \forall k. \text{STARTS}(i, k) \wedge \text{FINISHES}(j, k) \supset \\
& \quad \quad \text{duration}(k, j) = 1 + \text{duration}(i, j) \\
& \forall i, k. \text{STARTS}(i, k) \supset \\
& \quad \forall j. \text{MEETS}(i, j) \wedge \text{FINISHES}(j, k) \supset \\
& \quad \quad \text{duration}(j, k) = 1 - \text{duration}(i, k) \\
& \forall j, k. \text{FINISHES}(j, k) \supset \\
& \quad \forall i. \text{MEETS}(i, j) \wedge \text{STARTS}(i, k) \supset \\
& \quad \quad \text{duration}(i, k) = 1 - \text{duration}(j, k)
\end{aligned}$$

where the durations may actually have to be computed up to three different ways because the arithmetic on ranges is not symmetric. Note also that this assumes that a cover exists for every pair of intervals that meets. This is true axiomatically in the formal system, but must be enforced in a system such as TIMELOGIC, where intervals are normally created only as needed.

What makes composite duration an interesting issue is the fact that subtraction of ranges may result in indeterminacy. Supposing, for instance, that Constable John walks his beat typically in anywhere from 40 minutes to an hour and he repeatedly walks his beat over interval k with duration of 3 hours, then the first application of the recurrence axiom gives us the first “walk the beat” event over interval i , with a duration of 40-60 minutes and a remainder j of 120-140 minutes. The second application gives the second walk event, i' , with the same duration, and a second remainder j' with a duration of 60-100 minutes. Once more gives i'' for the walk event and a remainder j'' with a duration of 0-60 minutes. Hence there may or may not be enough time left for another instance of the “walk the beat” event. Without more information about the duration of each preceding event there is no way to decide.

One way a planner may get around this problem, and others induced by the disjunctive nature of the temporal logic, is by *constraint gathering*. Normally, the planner (theorem prover) infers the consequent of a rule if it can prove the antecedent. A modified strategy for a temporal planner goes as follows: if an antecedent can not be proven then assume it is true and collect it, provided it is a temporal constraint (such as MEETS) and it is consistent with the constraints gathered so far. If the rest of the proof succeeds then the plan is *conditionally* viable, i.e., it is viable provided the gathered temporal constraints hold. These constraints can be used by a scheduler, or verified during execution of the plan. Obviously this is a dangerous strategy; if not used with great care, anything becomes provable. More research is needed to explore the ramifications of such a strategy.

5.2.3 Conventional time

As described in Chapter 1, Leban *et al.* [1986] proposed interval *collections*, which provide powerful and flexible ways of partitioning the dateline. Adapting the recurrence logic to apply to interval collections would obviously increase its utility. It would also get around an inconvenience of the current formulation, namely the fact that recurrence is defined as a repeated but continuous occurrence. This is quite adequate for describing recurring states, as was the case in the traffic light example. However, an event recurrence is often more easily thought of as a series of short, possibly even instantaneous, events interspersed with pauses where “nothing” happens. Consider, for instance, the chiming of a clock, or the representation of the sentence frequently found on Automatic Teller machines: “This machine is in service daily from 2:45 to 3 pm.” In our logic we would have to express this as

$$\begin{aligned} \text{DAILY-ATM-SERVICE} &= \text{seq}(\text{SERVICE-ATM}, \text{nothing}(\text{SERVICE-ATM})) \\ &\wedge \text{RT}(\text{ATM-INSTALLED-TIME}, \text{DAILY-ATM-SERVICE}) \\ &\wedge \text{typical-duration}(\text{SERVICE-ATM}, \text{MINUTES}) = 15 \\ &\wedge \text{typical-duration}(\text{DAILY-ATM-SERVICE}, \text{HOURS}) = 24 \end{aligned}$$

where the function **nothing** returns an entity signifying that nothing of interest happens (with respect to the *SERVICE-ATM* event). It seems more intuitive to introduce an axiom *RTC* which defines recurrence of x over a collection c . This definition could, of course, be in terms of *RT*. Given the mechanism of describing interval collections and the *RTC* axiom, the above recurrence could simply be stated at

$$\text{RTC}(64/\text{Quarters} : \text{during: Days}, \text{SERVICE-ATM})$$

which makes the service event repeatedly true over the 64'th quarter of each day.

5.2.4 Instantaneous events

Just as adherents of point-based temporal logics have introduced interval objects as pairs of points, so have followers of the interval-based paradigm felt the need to introduce points. Vilain [1982], in a system similar to *TIMELOGIC*, extended Allen's interval calculus by introducing point objects, doubling the set of temporal relations to include point-to-point, point-to-interval and interval-to-point relations, and extending the transitivity rules to include these point relations. Vilain, however, does not discuss the ramifications of anything being true at points

of time, or whether anything being true over an interval implies it being true at all contained points or at the endpoints.

Allen and Hayes [1985; 1987] provided the theoretical foundation for point-like structures in the interval calculus by defining them as the meeting places of pairs of intervals. Although they motivate the introduction of points by the need to represent *accomplishment* events such as “closing the door,” which occur at a point in time, they do not discuss in detail how the presence of these points affects Allen’s model of events, actions and properties over time.

Galton [1987] also introduced points (*instants* in his terminology) but with the claim that they are necessary to describe continuous change, “for the essence of such change is that the changing object passes through a continuous range of states, each of which obtains for an isolated instant.” [Galton, 1987, page 9] We believe that this perceived difficulty stems primarily from mixing apples and oranges, *i.e.*, from attempting to describe motion as being at a *point* in space over an *interval* in time. Galton’s introduction of points of time so that motion can be described as being at a point in space at a point in time is one solution; Chapter 4 illustrates that introducing intervals of space is equally effective, as motion can be described as being on an interval of space over an interval of time. The latter actually more closely mirrors our notion of *speed* as (the limit of) distance travelled over time.

However, Galton does introduce useful extensions to Allen’s predicates **HOLDS** and **OCCURS**, namely **HOLDS-AT**(t, x) which indicates a property x holding at a point t , and **HOLDS-IN**(i, x) which indicates a property x holding at some instant or over some interval contained in interval i . Likewise for **OCCURS-AT** and **OCCURS-IN**. These could easily be adopted in our logic, collapsing them into **TA** (*True At*) and **TI** (*True In*), as the distinction between properties and events is captured by Shoham’s classification.

Note that the occurrence of accomplishment events at the meeting place of two intervals concurs well with the observation that such events separate two usually opposing conditions, one which holds prior to the event (such as “the door is open”) and one which holds immediately following the event (such as “the door is closed”). These conditions (preconditions and effects, in planner parlance) are *states of position* according to Galton, which hold over intervals. Hence, such punctual events cannot occur infinitely often — they are separated by intervals of time during which their effects hold. We can associate a duration even with such punctual events by ascribing a *recovery* time to them, *i.e.*, the minimal time for the effects to wear off and the preconditions to reestablish themselves.

5.3 Extending the recurrence logic

5.3.1 Relaxing restrictions on sequences

In chapter two we defined the Sequential Truth axiom (Axiom 2.33) such that a sequence is true over interval k if and only if the two sequence components are true over intervals i and j , and i and j are strictly covered by k , i.e., i and j meet. What would happen if we relax the last constraint? Assuming that, in order to speak of sequentiality we will continue to insist that i starts before j starts and finishes before j finishes. Aside from the meets that leaves the relations before and overlaps.

There is really no reason to banish the possibility that i and j overlap. The prediction theorems (Theorems 2.34 and 2.36) still hold, although their proofs are slightly more complicated due to the disjunction. Moreover, in many cases, such as $\text{seq}(\text{NIGHT}, \text{DAY})$, the sequence components are mutually exclusive, eliminating the $\text{OVERLAPS}(i, j)$ from the disjunction.

However, allowing $\text{BEFORE}(i, j)$ makes the whole sequencing notion tenuous unless we apply other constraints. The problem is that, as soon as we allow a gap in between i and j , any number of things might happen during that time, *including other occurrences of the components!* This, of course, is quite at odds with our intent. It would be fairly useless and intuitively incorrect to claim that last night and the day after tomorrow form a sequence. We could amend the sequencing axiom to allow BEFORE in the disjunction provided that we add the constraint that both sequence components must be false throughout the gap. This makes for a fairly complex axiom, and for a very hairy proof!

Axiom 5.1 (Relaxed Sequential Truth) *A sequence is true over an interval if and only if the two entities composing the sequence are sequentially true over that interval:*

$$\begin{aligned} \forall k, x, y. \text{MT}(k, \text{seq}(x, y)) \iff \\ & [\exists i, j. \text{STARTS}(i, k) \wedge \text{FINISHES}(j, k) \wedge \text{MT}(i, x) \wedge \text{MT}(j, y) \\ & \wedge [\text{MEETS}(i, j) \vee \text{OVERLAPS}(i, j) \\ & \vee [\exists l. \text{MEETS}(i, l) \wedge \text{MEETS}(l, j) \\ & \wedge \text{FT}(l, x) \wedge \text{FT}(l, y)]]] \end{aligned}$$

5.3.2 Relaxing restrictions on the Recurrence Axiom

In Section 2.6 we discussed alternative ways of defining recurrence, specifically to relax the constraint that sequences occur in their entirety. The alternative Axiom

2.35.b allowed sequences to recur such that the recurrence ended after some occurrence of the first sequence component, omitting the second component. Although this is intuitively more appealing, as well as more expressive, we lose the power of the prediction theorem (Theorem 2.36). Further work is necessary to ascertain if a planner can make effective use of the alternative axiom in terms of predicting future states of the world.

5.3.3 Recurrence and full planning

In this thesis we have developed the basic reasoning mechanisms that allow an intelligent agent to represent and use knowledge of recurring states and events. An obvious and hopefully fruitful continuation of this research would be to incorporate these mechanisms in a full-fledged planning system. Pelavin's work [1988] would be particularly relevant and suitable for this purpose, as it is also based on the interval logic and deals with issues such as plan representation and the possibility or inevitability of plans of actions accomplishing goals. An interesting difference between the two approaches that would need to be reconciled is the fact that we use an *intentional* representation of actions and events, which allows us to represent infinite (or at least arbitrarily frequent) recurrence without reference to any instance, whereas Pelavin uses an *extensional* representation in the form of event tokens in order to handle multiple simultaneous instances of the same event type.

5.4 Extending the spatial logic

Although tangential to the topic of recurrence, the results obtained in this thesis suggest that the combination of spatial and temporal intervals to express motion is worthy of further study. In particular, it is interesting to speculate on the feasibility of removing two limitations we have placed on our path logic.

5.4.1 What goes around...

Just as our current system patiently waits until the light goes green before proceeding, so should it also be able to wait for an object on a revolving platform to come around before picking it up. What prevents us currently from having circular paths is the Total Order axiom (the equivalent of Axiom 2.2 for the path logic). Allen and Hayes [1987] showed that one consequence of this axiom is that no interval can meet itself:

$$\forall i. \neg \text{MEETS}(i, i)$$

and therefore no path can meet itself. Unfortunately, it is not simply a matter of removing this axiom, as this would allow all sorts of weird models. For instance, an infinite number of paths each meeting itself and no other path; or an infinite number of paths, each describing the same loop — once around, twice around, thrice around, *etc.* Clearly, we do need to impose at least partial order, and ensure that all paths are connected.

5.4.2 Beyond one dimensional space

And finally, is it possible to extend the path logic to capture two or perhaps even three dimensional space? In the two dimensional case paths would be more like patches, and in three dimensions they would be blobs. In either case they would be convex, and so would not be general enough to represent arbitrary objects. However, much approximate spatial reasoning can be accomplished simply with object envelopes, especially regarding motion description and planning.

Bibliography

- [Allen and Hayes, 1985] James F. Allen and Patrick J. Hayes. A commonsense theory of time. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 528–531, Los Angeles, Ca., 1985.
- [Allen and Hayes, 1987] James F. Allen and Patrick J. Hayes. Moments and points in an interval-based temporal logic. Technical Report 180, University of Rochester, Department of Computer Science, 1987. (longer version of [Allen and Hayes, 1985]).
- [Allen and Kautz, 1985] James F. Allen and Henry A. Kautz. A model of naive temporal reasoning. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World*, pages 251–268. Ablex, Norwood, N.J., 1985.
- [Allen and Koomen, 1983] James F. Allen and Johannes A. Koomen. Planning using a temporal world model. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 741–747, Karlsruhe, W. Germany, 1983.
- [Allen and Miller, 1988] James F. Allen and Bradford W. Miller. The rhetorical knowledge representation system: A user’s manual. Technical Report 238, University of Rochester, Department of Computer Science, February 1988. (rerevised March 1989).
- [Allen, 1983] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [Allen, 1984] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.
- [Bacchus *et al.*, 1989] Fahiem Bacchus, Josh Tenenber, and Johannes A. Koomen. A non-reified temporal logic. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 2–10, Toronto, Canada, May 1989.

- [Cohen and Levesque, 1987] Philip R. Cohen and Hector J. Levesque. Persistence, intention, and commitment. In *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, pages 297–340, Timberline, Oregon, 1987.
- [Dean, 1984] Thomas L. Dean. Managing time maps. In *Proceedings of the 5th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, London, Ontario, 1984.
- [Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
- [Galton, 1987] Antony Galton. A critical examination of j. f. allen’s theory of action and time. Technical note, Department of Computer Science, University of Exeter, England, July 1987.
- [Ginsberg and Smith, 1988] Matthew L. Ginsberg and David E. Smith. Reasoning about action II: The qualification problem. *Artificial Intelligence*, 35(3):311–342, 1988.
- [Haas, 1987] Andrew Haas. The case for domain-specific frame axioms. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*. Morgan Kaufman, April 1987.
- [Hanks and McDermott, 1986] Steve Hanks and Drew McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the 5th National Conference on Artificial Intelligence*, August 1986.
- [Hayes, 1973] Patrick J. Hayes. The frame problem and related problems in artificial intelligence. In A. Elithorn and D. Jones, editors, *Artificial and Human Thinking*, pages 45–49. Jossey-Bass, San Francisco, CA., 1973. Also in [Webber and Nilsson, 1980].
- [Hendrix, 1973] Gary G. Hendrix. Modeling simultaneous actions and continuous processes. *Artificial Intelligence*, 4:145–180, 1973.
- [Kahn and Gorry, 1977] K. Kahn and G. A. Gorry. Mechanizing temporal knowledge. *Artificial Intelligence*, 9(1):87–108, 1977.
- [Kautz, 1987] Henry A. Kautz. *A Formal Theory of Plan Recognition*. PhD thesis, University of Rochester, Department of Computer Science, Rochester, N.Y., May 1987.
- [Koomen, 1986] Johannes A. G. M. Koomen. Periodic temporal reasoning. In *Proceedings of 1986 Open House (Technical Report 184)*, pages 27–32. University of Rochester, Department of Computer Science, March 1986.

- [Koomen, 1987] Johannes A. G. M. Koomen. The TIMELOGIC temporal reasoning system. Technical Report 231, University of Rochester, Department of Computer Science, November 1987. (revised March 1989).
- [Koomen, 1989] Johannes A. G. M. Koomen. Localizing temporal constraint propagation. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 198–202, Toronto, Canada, May 1989.
- [Ladkin, 1986a] Peter B. Ladkin. Primitives and units for time specification. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 354–359, Philadelphia, Pa., 1986.
- [Ladkin, 1986b] Peter B. Ladkin. Time representation: A taxonomy of interval relations. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 360–366, Philadelphia, Pa., 1986.
- [Lansky, 1987] Amy L. Lansky. A representation of parallel activity based on events, structure, and causality. In *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, pages 123–160, Timberline, Oregon, 1987.
- [Lansky, 1988] Amy L. Lansky. Localized event-based reasoning for multi-agent domains. Technical Note 423, SRI International, AI Center, Menlo Park, Ca., January 1988.
- [Leban *et al.*, 1986] Bruce Leban, D. McDonald, and D. Forster. A representation for collections of temporal intervals. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 367–371, Philadelphia, Pa., 1986.
- [Lifschitz, 1986] Vladimir Lifschitz. Pointwise circumscription: Preliminary report. In *Proceedings of the 5th National Conference on Artificial Intelligence*, Philadelphia, Pa., August 1986.
- [Lifschitz, 1987] Vladimir Lifschitz. Formal theories of action: Preliminary report. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, pages 35–57. Morgan Kaufman, April 1987.
- [McCarthy and Hayes, 1969] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*. Edinburgh University Press, 1969. Also in [Webber and Nilsson, 1980, pages 431–450].
- [McCarthy, 1980] John McCarthy. Circumscription: A form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980. also in [Webber and Nilsson, 1980].

- [McDermott, 1982] Drew V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.
- [Miller, 1985] David P. Miller. *Planning by Search Through Simulations*. PhD thesis, Yale University, Computer Science Department, 1985.
- [Pelavin and Allen, 1986] Richard N. Pelavin and James F. Allen. A formal logic of plans in temporally rich domains. *Proceedings of the IEEE*, 74(10):1364–1382, 1986.
- [Pelavin, 1988] Richard N. Pelavin. *A Formal Approach to Planning with Concurrent Actions and External Events*. PhD thesis, University of Rochester, Department of Computer Science, Rochester, N.Y., May 1988.
- [Poesio, 1986] Massimo Poesio. Toward a hybrid representation of time. In *European Conference of Artificial Intelligence*, pages 247–252, Munich, FRG, August 1986.
- [Pylyshyn, 1987] Zenon W. Pylyshyn, editor. *The robot's Dilemma: The Frame Problem in Artificial Intelligence*. Ablex, Norwood, N.J., 1987.
- [Sacerdoti, 1977] E. D. Sacerdoti. *A Structure for Plans and Behavior*. Elsevier North-Holland, New York, N.Y., 1977.
- [Shoham, 1986] Yoav Shoham. Chronological ignorance: Time, nonmonotonicity, necessity and causal theories. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 389–393, Philadelphia, Pa., 1986.
- [Shoham, 1987] Yoav Shoham. Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33(1):89–104, 1987.
- [Steele Jr., 1984] Guy L. Steele Jr. *Common Lisp: the Language*. Digital Press, 1984.
- [Vere, 1983] Steven A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):246–267, 1983.
- [Vilain and Kautz, 1986] Mark Vilain and Henry Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 377–382, Philadelphia, Pa., 1986.
- [Vilain, 1982] Mark Vilain. A system for reasoning about time. In *Proceedings of the 2nd National Conference on Artificial Intelligence*, Pittsburgh, Pa., 1982.
- [Webber and Nilsson, 1980] Bonnie Lynn Webber and Nils J. Nilsson, editors. *Readings in Artificial Intelligence*. Morgan Kaufmann, 1980.

[Weber, 1989] Jay Weber. *Principles and Algorithms for Causal Reasoning with Uncertainty*. PhD thesis, University of Rochester, Department of Computer Science, Rochester, N.Y., May 1989.

Appendix A

TIMELOGIC Interface

A.1 Terminology and conventions

A.1.1 Properties

Many TIMELOGIC system features can be enabled, disabled, or otherwise manipulated by the user through TIMELOGIC *properties*. See Section A.8 for a description of these properties, their default values, and how to change them.

A.1.2 Contexts

If any function documented below takes the keyword argument *context*, the function may be invoked with `NIL`, `T` or the name of a previously defined context as the value for this argument, where `NIL` refers to the current context and `T` refers to the root context. The supplied context, if any, becomes the current context within the function's scope.

Note that asserting constraints in a non-leaf context may lead to inconsistencies. Certain constraints may have been inferred in a leaf context given some assertions and inherited constraints. These inferences may no longer be valid if the inherited constraints change. As TIMELOGIC is not able to detect and account for such inconsistencies, asserting constraints in a non-leaf context signals an error if the TIMELOGIC property `:LEAVES-ONLY` is `:ON`, or causes a warning to be printed if this property is `:WARN`.

A.1.3 Types

If any function documented below takes the keyword argument *type*, the function may be invoked with `:REL`, `:DUR` or `NIL` as the value for this argument. If `:DUR` is given, queries and assertions are assumed to be of the durational type; otherwise they are assumed to be of the relational type.

A.1.4 Errors

TIMELOGIC can run into trouble in many places. In general, if an error is detected, the user is given an error message and is then asked whether the Common Lisp `ERROR` function should be invoked. (Presumably facilities are provided for inspecting the current program state, data structures, *etc.*) If the user answered no, or control somehow returns to the TIMELOGIC error function, the user is asked if the most recent changes to the TIMELOGIC database, if any, should be undone. Either way, TIMELOGIC then proceeds by returning `NIL` to wherever the TIMELOGIC error function was invoked. *This may or may not work!* The only exception is the case where a temporal inconsistency is detected during constraint posting, and the TIMELOGIC property `:AUTO-BACKTRACK` is `:ON`. In this case, all database changes, both direct and inferred, that occurred because of the call to the function `ADD-INTERVAL-CONSTRAINT` are undone without user interaction and the function returns `NIL`. See the function description for more detail.

A.2 Initialization

The TIMELOGIC system is loaded by either using the Lisp implementation's system wrapper facilities, or, if this is not available, by loading the file "timelogic.lisp" directly. For instance, one can type "`:Load System TimeLogic`" on the Symbolics machines. Once TIMELOGIC is loaded, all functions documented below are available as external symbols in the TIMELOGIC package (which has "TL" as a convenient nickname). The following function must be called once before attempting any other TIMELOGIC operation, and may be called again as often as desired.

(TIMELOGIC-INIT) [function]
Initializes the TIMELOGIC system, resets the database, creates the root context and makes it the current context. Returns a list of TIMELOGIC properties and their values currently in effect (see the function `TIMELOGIC-RESET-PROPS`).

A.3 Manipulating contexts

(DEFINE-CONTEXT &OPTIONAL *name parent*) [function]
Defines *name* as a TIMELOGIC context with *parent* as its parent context. If *name* is not given, generates a new context name. If *parent* is not given, the current context becomes the parent context. Signals an error if *parent* is not a previously defined context or if *name* is an existing context with a different parent. Returns the new context's name.

(DEFINED-CONTEXTS) [function]
Returns a list of all defined TIMELOGIC contexts.

(CONTEXT-DEFINED-P *name*) [function]
Returns the context's name if *name* is defined context, otherwise NIL.

(DELETE-CONTEXT *name*) [function]
Erases all recorded constraints in context *name* from the database, deletes context *name* and returns it's name if *name* is a leaf context (*i.e.*, if it doesn't have any children), otherwise signals an error.

(SWITCH-CONTEXT *name*) [function]
Makes *name* the current context and returns it's name if *name* is a previously defined context, otherwise signals an error.

(CURRENT-CONTEXT) [function]
Returns the context currently in effect, *i.e.*, either T (root context) or the value of the argument to the last invocation of SWITCH-CONTEXT.

(PUSH-CONTEXT) [function]
Same as (SWITCH-CONTEXT (DEFINE-CONTEXT)).

(POP-CONTEXT &OPTIONAL *dont-delete-p*) [function]
Switches the current context *c* to the parent of *c*, then deletes *c* provided *dont-delete-p* is NIL. Signals an error if *c* does not have a parent (*i.e.*, *c* is the root context).

(CONTEXT-TREE &OPTIONAL (*name* T)) [function]
Returns a tree node for context *name* (or the root context if omitted), where a tree node is either the name of the context if it is a leaf context (*i.e.*, has no children), or a list whose CAR is the name of the context and whose CDR is a list of tree nodes, one for each of the context's children.

A.4 Manipulating intervals

(DEFINE-INTERVAL &OPTIONAL *int* *ref* &KEY *context*) [function]
Unless previously defined, defines *int* as the name of an interval. If *int* is not given, generates a new interval name. If *ref* is given, it is implicitly defined as an interval; if the TIMELOGIC property :AUTO-REFERENCE is :OFF, *ref* also becomes a reference interval for *int* in *context*. Returns the interval's name.

(INTERVAL-DEFINED-P *int*) [function]
Returns T if *int* was previously defined as an interval, otherwise NIL.

(DEFINED-INTERVALS) [function]
Returns a list of all currently defined intervals.

(RELATED-INTERVALS *int* **&KEY** *context type*) [function]
Returns a list of all currently defined intervals for which some explicit constraint with *int* of type *type* has been recorded in *context*. If *type* is **NIL**, the constraint can be of either type.

(REFERENCE-INTERVALS *int* **&OPTIONAL** *inverse-p* **&KEY** *context*) [function]
Returns a list of reference intervals of *int* in *context*. If *inverse-p* is non-**NIL**, returns a list of intervals for which *int* is itself a reference interval.

A.5 Manipulating constraints

(ADD-INTERVAL-CONSTRAINT *x constraint y* **&KEY** *context (type :REL)*) [function]
If the **:AUTO-DEFINE** property is **:ON**, implicitly defines *x* and *y* as intervals with initial reference interval **:ROOT**; otherwise signals an error if *x* or *y* have not been defined previously. Asserts *constraint* as a new constraint between intervals *x* and *y*. Signals an error when *constraint* cannot be parsed according to *type*. Let *oldcon* be the constraint currently known in *context*. Let *newcon* be the intersection of *constraint* and *oldcon*. Signals an error if *newcon* is the null set, unless the **:AUTO-BACKTRACK** property is **:ON**, in which case any intermediate database updates are removed and **NIL** is returned. If *newcon* is not the null set, propagates *newcon* to all constraints between comparable¹ intervals *x* and *z* such that there exists an explicit constraint between *y* and *z*, and to all constraints between comparable intervals *w* and *y* such that there exists an explicit constraint between *w* and *x*, and returns a **TIMELOGIC** backtracking point. If this backtracking point is given to the function **TIMELOGIC-BACKTRACK** (see below), the **TIMELOGIC** database reverts to the state it was in just before this particular invocation of **ADD-INTERVAL-CONSTRAINT**. Note that a backtracking point is returned even if adding *constraint* did not actually result in any database changes (*i.e.*, *newcon* is identical to *oldcon*).

TLR-ASSERT-HOOK [variable]
Either **NIL** or bound to a function. If a call to **ADD-INTERVAL-CONSTRAINT** succeeds and the hook variable is bound, then for each pair of intervals *i* and *j* whose relational constraint changed to *r* due to constraint propagation the hook function is applied to the triple $\langle i, r, j \rangle$. If any such application of the hook function returns **NIL** then **ADD-INTERVAL-CONSTRAINT** signals an error, unless the **:AUTO-BACKTRACK** property

¹Two intervals are comparable (in the current context) if an explicit constraint exists between them or they have one or more reference intervals in common.

is :ON, in which case all database changes are undone and NIL is returned.

Note that, for reasons of efficiency, the hook function is invoked on arguments in internal format. Use `(TL::INTERVAL-NAME i)` to get the object by which the interval *i* is known externally, and `(TL::TLR-DECODE-RLINK r)` to obtain the external form of the relational constraint.

TLD-ASSERT-HOOK [variable]

Same as ***TLR-ASSERT-HOOK*** but invoked on triples $\langle i, d, j \rangle$ for intervals i and j whose durational constraint changed to d . The function **TL::TLD-DECODE-DLINK** will return the external form of the durational constraint.

```
(GET-INTERVAL-CONSTRAINT x y &KEY context (type :REL)) [function]
```

If either x or y are not defined intervals, returns **NIL**; otherwise, returns the currently known relational constraint between x and y , unless *type* is given as **:DUR** in which case the durational constraint is returned.

(TEST-INTERVAL-CONSTRAINT <i>x constraint y</i>	[function]
---	------------

```
&KEY context (type :REL) (test :INTERSECT))
```

Let *curcon* be the currently known constraint between x and y . Tests whether *constraint* is compatible with *curcon*. Three kinds of tests are possible, indicated by the argument *test*:

<i>test</i>	<i>true (returns T) iff</i>
:EQUAL	$curcon = constraint$
:SUBSET	$curcon \subset constraint$
:INTERSECT	$curcon \cap constraint \neq \emptyset$

(TIMELOGIC-BACKTRACK &OPTIONAL *btpoint*) [function]

Backs up the TIMELOGIC constraint database to *btpoint* (see the functions `ADD-INTERVAL-CONSTRAINT`, above, or `TIMELOGIC-CHECKPOINT`, below). If *btpoint* is omitted, backs up to the last backtracking point generated. Returns `T` if succesful, `NIL` otherwise.

```
(TIMELOGIC-CHECKPOINT) [function]
```

Returns an object which, when given to the function `TIMELOGIC-BACKTRACK`, returns the `TIMELOGIC` database to the current state.

(TIMELOGIC-CHECKPOINT-P &OPTIONAL *btpoint*) [function]

Returns **T** if *btpoint* is a valid backtracking point, i.e., if **TIMELOGIC-BACKTRACK** would succeed if given *btpoint*.

A.6 Tracing constraint propagation

(SHOW-INTERVAL-CONSTRAINTS *int* &KEY *with-ints context* (*type* :REL)) [function]
Prints to the standard output stream a listing of directly recorded constraints between *int* and all intervals to which it is related, or between *int* and *with-ints* (which is either an interval or a list of intervals) if given.

(DISPLAY-INTERVALS &KEY *ints context* (*clear* T)) [function]
Puts up a graphic display of all *ints*, lined up properly to reflect the relational constraints between them (relative to *context*). If *clear* is T (the default), the display area is cleared first. If *ints* is T, all intervals explicitly traced with TRACE-INTERVAL will be displayed. If *ints* is NIL, all defined intervals will be displayed.

(GRAPH-INTERVALS &OPTIONAL *ints*) [function]
Puts up a graph of the reference structure (see Section 3.4) over *ints*, which defaults to all defined intervals. *Note: this is currently implemented only in the Xerox and Symbolics Lisp environments.*

(TRACE-INTERVAL *int* &OPTIONAL *with-ints*) [function]
Any subsequent activity between *int* and any of the intervals directly related to it (or any in the list *with-ints* if given) will cause TIMELOGIC to print to the standard output stream a line indicating the activity. Note that this tracing occurs only if the :TRACE property is enabled.

(UNTRACE-INTERVAL &OPTIONAL *int*) [function]
Any subsequent activity between *int* and any of the intervals directly related to it (or any in the list *with-ints* if given previously) will no longer be printed to the standard output stream. If *int* is not given, disables tracing of all intervals explicitly traced with TRACE-INTERVAL. If *int* is a list, all intervals on this list will be untraced.

A.7 Statistics

(TIMELOGIC-STATS) [function]
Prints to the standard output stream a summary of various statistics kept when the TIMELOGIC property :STATS is :ON. For both the relational and durational types, prints the number of constraints attempted, “multiplied” (referring to the number of invocations of the Constraints function) and propagated. Also given is the number of singular constraints (*i.e.*, constraint sets of cardinality 1), the number of broken constraints (*i.e.*, those implied by a path through the reference hierarchy), and the average branching factor (*i.e.*, the number of explicit constraints between intervals). The statistics counters can be reset by setting the TIMELOGIC property :STATS to :RESET.

A.8 Customization

(TIMELOGIC-RESET-PROPS &KEY *keywords*...) [function]

Resets all TIMELOGIC properties to their default values, unless overridden by entries on *keywords*, and returns a list of all TIMELOGIC properties and their current values. For instance, (TIMELOGIC-RESET-PROPS :TRACE :ON :DISPLAY :ON :SORT :ON) will give you a standard TIMELOGIC setup, except that whenever a change is made to the constraint between two traced intervals, a trace message is printed, and the relations between all traced (and defined) intervals are displayed graphically.

(TIMELOGIC-PROP *propkey* &OPTIONAL *newpropvalue*) [function]

If *newpropvalue* is given, sets the property value accordingly. Always returns the current (old) value of the property. Some properties (designated *init* properties) may only be set immediately following a call to TIMELOGIC-INIT. They generally affect the operation of TIMELOGIC such that changing them in mid-session would have unpredictable results. The following properties control the TIMELOGIC system:

:ALL-PATHS [TIMELOGIC init property]

When :ON, TIMELOGIC computes the constraint between two indirectly related intervals as the intersection of the constraints along all paths through the hierarchy. When :OFF, causes TIMELOGIC to compute such a constraint by traversing the first path found through the reference hierarchy. Note that traversing a single path is much faster but may result in an under-constrained relation! Initially set to :ON.

:AUTO-BACKTRACK [TIMELOGIC property]

When :OFF, TIMELOGIC leaves the current state of the database alone when an error or inconsistency occurs. When :ON, the current assertion and all consequences computes sofar will be undone automatically. Initially set to :ON.

:AUTO-DEFINE [TIMELOGIC property]

When :OFF, TIMELOGIC signals an error when a constraint is posted between undefined intervals. When :ON, any undefined interval is implicitly defined with initial reference interval :ROOT. Initially set to :ON.

:AUTO-REFERENCE [TIMELOGIC init property]

When :OFF, TIMELOGIC relies on the user to provide a reference hierarchy of intervals. When :ON, a reference hierarchy is automatically and incrementally defined (see Section 3.4). Initially set to :ON.

:DEPTH-FIRST [TIMELOGIC property]

When :OFF, TIMELOGIC processes constraint inferences in a breadth-first fashion. When :ON, a depth-first strategy is employed. Initially set to :OFF.

:DISPLAY [TIMELOGIC property]

When **:ON**, and the TIMELOGIC property **:TRACE** is not **:OFF**, TIMELOGIC provides a dynamically changing display of relational interval constraints, updated whenever a relevant constraint changes.² If **:TRACE** is **:ON**, only those intervals are displayed which have been explicitly traced. If **:TRACE** is **:ALL** or **:VERBOSE**, all intervals are displayed. Initially set to **:OFF**.

:DURATIONS [TIMELOGIC init property]

When **:ON**, TIMELOGIC allows durational constraints to be posted, and infers them from given durational and relational constraints. Initially set to **:OFF**.

:FLOATS [TIMELOGIC init property]

When **:ON**, TIMELOGIC allows durational constraints to be real-valued; otherwise, all reals will be rationalized. Initially set to **:OFF**. Note that round-off errors in floating-point arithmetic may cause TIMELOGIC to fall into an infinite loop. Setting the **:TOLERANCE** property (see below) to an appropriate value can alleviate (but not eliminate) this problem.

:LEAVES-ONLY [TIMELOGIC init property]

When **:ON**, TIMELOGIC will signal an error if an attempt is made to change a constraint between two intervals in a non-leaf context (*i.e.*, a context with children). When **:WARN**, this attempt would succeed (caveat emptor!) but you will be warned. When **:OFF**, such attempts succeed without fuss (but not necessarily without causing inconsistencies). Initially set to **:WARN**.

:RELATIONS [TIMELOGIC init property]

When **:ON**, TIMELOGIC allows relational constraints to be posted, and infers them from given durational and relational constraints. Initially set to **:ON**.

:SORT [TIMELOGIC property]

When **:ON**, TIMELOGIC attempts to sort intervals along the time axis before displaying them. Initially set to **:OFF**. Note: only operative when the TIMELOGIC property **:DISPLAY** is **:ON**.

:STATS [TIMELOGIC property]

When **:ON**, TIMELOGIC starts (or continues) to keep track of various statistics during its inferencing. Initially set to **:OFF**. Execute (**TIMELOGIC-PROP :STATS :RESET**) to reset the statistics counters.

:TOLERANCE [TIMELOGIC init property]

When the **:FLOATS** property is **:ON**, and the **:TOLERANCE** property is a floating point number f , TIMELOGIC will not add a new constraint if the resulting constraint

²Obviously, since intervals are generally only partially ordered, this display is only partially correct! It is true that two intervals whose endpoints are known not to coincide do not line up in the display, but the inverse is not true.

newcon would be within a factor f of the existing constraint *oldcon*. Initially set to 0.001.

:TRACE [TIMELOGIC property]
 Unless **:OFF**, TIMELOGIC prints various tracing information regarding the state of the temporal database. More detail is obtained by setting this property to **:ON**, **:ALL**, or **:VERBOSE**, successively. When **:ON**, a trace line is printed whenever a constraint changes between some interval and an interval which has been explicitly traced using the function **TRACE-INTERVAL**, or when such an interval is defined, or gets or becomes a reference interval. When **:ALL**, a line is printed tracing each top level activity taking place (*i.e.*, TIMELOGIC changes initiated by the user). When **:VERBOSE**, a trace line is printed whenever anything happens to any interval, direct or inferred. Initially set to **:OFF**. Each trace line starts with one of the following indicators:

TL+ interval: <i>int</i>	Defining interval <i>int</i>
TL+ context: <i>name</i>	Defining context <i>name</i>
TL- context: <i>name</i>	Removing context <i>name</i>
TL> context: <i>name</i>	Making context <i>name</i> the current context
TL+ reference: <i>int ref</i>	Adding <i>ref</i> as reference interval for <i>int</i>
TL- reference: <i>int ref</i>	Removing <i>ref</i> as reference interval for <i>int</i>
TL+ relation [<i>i</i>]: <i>xint yint rel</i>	Adding <i>rel</i> as new constraint between intervals <i>xint</i> and <i>yint</i> on level <i>i</i> (if $i = 1$ then user-defined, otherwise inferred)
TL^ relation [<i>i</i>]: <i>xint yint rel</i>	Restoring <i>rel</i> as backtracked constraint between intervals <i>xint</i> and <i>yint</i> on level <i>i</i>
TL~ relation [<i>i</i>]: <i>xint yint</i>	Breaking direct constraint between intervals <i>xint</i> and <i>yint</i> on level <i>i</i> (see Section 3.4)
TL& relation [<i>i</i>]: <i>xint yint</i>	Reconnecting broken constraint between intervals <i>xint</i> and <i>yint</i> on level <i>i</i>
TL+ duration [<i>i</i>]: <i>xint yint dur</i>	Adding <i>dur</i> as new constraint between intervals <i>xint</i> and <i>yint</i> on level <i>i</i>
TL^ duration [<i>i</i>]: <i>xint yint dur</i>	Restoring <i>dur</i> as backtracked constraint between intervals <i>xint</i> and <i>yint</i> on level <i>i</i>

:WAIT [TIMELOGIC property]
 Either **:ON**, **:OFF** or a positive number. Unless **:OFF**, TIMELOGIC waits (indefinitely if **:ON** or for this many seconds if a number) between displaying intervals and continuing its next inference. Note: only operative when the TIMELOGIC property **:DISPLAY** is **:ON**.

Appendix B

TEMPOS Interface

B.1 Integration of TimeLogic and Rhet

The *Rhetorical* system (“Rhet” for short) is a logic programming and knowledge representation system developed at the University of Rochester [Allen and Miller, 1988]. In addition to the usual Prolog-like pattern-matching and backward-chaining capabilities, it also provides forward-chaining, equality reasoning, constraint posting, hierarchical types, and facilities for representing knowledge in the form of *frames*.

The TEMPOS system provides a convenient interface to the TIMELOGIC system [Koomen, 1987] in the form of Rhet builtins. Facilities are provided for declaring Rhet terms to be of the TEMPOS-supplied type **T-TIME**, associating TIMELOGIC intervals with Rhet terms, and asserting and querying temporal constraints between Rhet terms. In addition, TEMPOS optionally provides axioms for reasoning about the occurrence and recurrence of states and events over time.

B.2 Initialization and Customization

(RESET-TEMPOS &KEY *tt-axioms-p* &REST *timelogic-keywords*) [function]
Initializes the TEMPOS system, defines the builtin interval predicates documented below, initializes the TIMELOGIC system (passing *timelogic-keywords* to the function **TIMELOGIC-RESET-PROPS**; see Section A.8), and, provided *tt-axioms-p* is non-**NIL**, defines the **TT**, **MT** and **RT** predicates documented below.

(TRACE-TEMPOS &OPTIONAL *verbose-p*) [function]
Turns on tracing of TEMPOS queries and assertions. If *verbose-p*, turns on more TEMPOS tracing as well as TIMELOGIC tracing.

(UNTRACE-TEMPOS) [function]
Stops tracing TEMPOS queries and assertions.

B.3 Interval definition

Interval objects must be defined before they can be used. Any Rhet term may be associated with a TIMELOGIC interval by using the following function:

`(DEFINE-TIME &REST terms)` [function]
Defines each of the n terms as a Rhet object of type T-TIME and defines the corresponding TIMELOGIC interval. If the $n-1$ 'th term is the Lisp keyword :REFERENCE then the n 'th term is defined as a reference interval for the terms 1 through $n-2$. For instance,

```
(DEFINE-TIME [JAN-89] [FEB-89] [MAR-89])
```

defines these terms as objects of type T-TIME and as TIMELOGIC intervals with default reference interval, and

```
(DEFINE-TIME [JAN-89] [FEB-89] [MAR-89] :REFERENCE [Q1-89])
```

defines JAN-89, FEB-89, MAR-89 and Q1-89 as Rhet terms of type T-TIME and as TIMELOGIC intervals, and defines Q1-89 to be the reference interval for JAN-89, FEB-89 and MAR-89.

B.4 Interval Predicates

B.4.1 Interval relation predicates

`[TIME-RELN ?i*T-TIME ?reln*T-LISP ?j*T-TIME]` [Rhet predicate]

In query mode, succeeds if the known TIMELOGIC relational constraint between the previously defined intervals ?i and ?j is a subset of ?reln, where ?reln is a TIMELOGIC relational constraint specification (see Section 3.3.1). Any two of the three arguments maybe unbound, in which case they will be bound to known TIMELOGIC entities for which the predicate holds, backtracking as necessary.

In assertion mode, succeeds if ?reln can be added as relational constraint between intervals ?i and ?j. Note that only ground forms can be asserted.

The specialized predicates given in Table B.1 provide convenient alternatives to frequently used forms.

`[TIME-RELN-P ?i*T-TIME ?reln*T-LISP ?j*T-TIME]` [Rhet predicate]

In query mode, succeeds if the known TIMELOGIC relational constraint between the previously defined intervals ?i and ?j intersects with ?reln, where ?reln is a TIMELOGIC relational constraint specification (see Section 3.3.1). This is the way

<i>[time-pred ?i ?j]</i>	<i>[TIME-RELN ?i reln ?j]</i>
TIME-AFTER	:A
TIME-BEFORE	:B
TIME-CONTAINS	:C
TIME-DURING	:D
TIME-EQUALS	:E
TIME-FINISHES	:F
TIME-FINISHED-BY	:Fi
TIME-MEETS	:M
TIME-MET-BY	:Mi
TIME-OVERLAPS	:O
TIME-OVERLAPPED-BY	:Oi
TIME-STARTS	:S
TIME-STARTED-BY	:Si
TIME-WITHIN	(:D :E :F :S)
TIME-WITHIN!	(:D :F :S)
TIME-DISJOINT	(:A :B :M :Mi)
TIME-DISJOINT!	(:A :B)
TIME-INTERSECTS	(:C :D :E :F :Fi :O :Oi :S :Si)
TIME-STARTS-LATER	(:A :D :F :Mi :Oi)
TIME-STARTS-EARLIER	(:B :C :Fi :M :O)
TIME-FINISHES-LATER	(:A :C :Mi :Oi :Si)
TIME-FINISHES-EARLIER	(:B :D :M :O :S)

Table B.1: Specializations of the TIME-RELN predicate

to find out if a certain relation *could* hold rather than holds. Note, however, that success does *not* imply that a subsequent assertion of *?reln* will succeed! Any two of the three arguments maybe unbound, in which case they will be bound to known TIMELOGIC entities for which the predicate holds, backtracking as necessary. This predicate may not be asserted.

The specialized predicates given in Table B.1 may be suffixed with *-P* to obtain convenient alternatives to TIME-RELN-P forms.

[TIME-COVER ?i*T-TIME ?j*T-TIME ?k*T-TIME] [Rhet predicate]
 States that interval *?k* covers intervals *?i* and *?j*, but the latter two are under-constrained: they may meet, overlap or be disjoint.
 Same as **[AND [TIME-STARTS ?i ?k] [TIME-FINISHES ?j ?k]]**

[TIME-COVER! ?i*T-TIME ?j*T-TIME ?k*T-TIME] [Rhet predicate]

<i>[time-pred ?i ?j]</i>	<i>[TIME-DURN ?i durn ?j]</i>
TIME-<i>=</i>	1
TIME-<i><</i>	(0 (1))
TIME-<i>></i>	((1) :INF)
TIME-<i><=</i>	(0 1)
TIME-<i>>=</i>	(1 :INF)
TIME-<i><></i>	(OR (0 (1)) ((1) :INF))

Table B.2: Specializations of the **TIME-DURN** predicate

States that interval *?k* covers intervals *?i* and *?j* which meet.
Same as **[AND [TIME-MEETS ?i ?j] [TIME-COVER ?i ?j ?k]]**

B.4.2 Interval duration predicates

[TIME-DURN ?i*T-TIME ?durn*T-LISP ?j*T-TIME] [Rhet predicate]
In query mode, succeeds if the known **TIMELOGIC** durational constraint between the previously defined intervals *?i* and *?j* is a subset of *?durn*, where *?durn* is a **TIMELOGIC** durational constraint specification (see Section 3.3.3). Any two of the three arguments maybe unbound, in which case they will be bound to known **TIMELOGIC** entities for which the predicate holds, backtracking as necessary.
In assertion mode, succeeds if *?durn* can be added as durational constraint between intervals *?i* and *?j*. Note that only ground forms can be asserted.

The specializations of **TIME-DURN** given in Table B.2 provide convenient alternatives to frequently used forms.

[TIME-DURN-P ?i*T-TIME ?durn*T-LISP ?j*T-TIME] [Rhet predicate]
In query mode, succeeds if the known **TIMELOGIC** durational constraint between the previously defined intervals *?i* and *?j* intersects with *?durn*, where *?durn* is a **TIMELOGIC** durational constraint specification (see Section 3.3.3). This is the way to find out if a certain duration *could* hold rather than holds. Note, however, that success does *not* imply that a subsequent assertion of *?durn* will succeed! Any two of the three arguments maybe unbound, in which case they will be bound to known **TIMELOGIC** entities for which the predicate holds, backtracking as necessary.
This predicate may not be asserted.

The specializations of **TIME-DURN** given in Table B.2 may be suffixed with **-P** to obtain convenient alternatives to **TIME-DURN-P** forms.

B.4.3 Interval Skolemization

[TIME-SKOLEM ?i*T-TIME &OPTIONAL ?ref*T-TIME] [Rhet predicate]
Fails if ?i is bound, otherwise succeeds binding ?i to a freshly created Rhet term of type T-TIME defined as a TIMELOGIC interval (optionally with the Rhet term ?ref as its reference interval). In addition, all posted temporal constraints on ?i are instantiated in the TIMELOGIC database. Obviously, if the posted constraints turn out to be inconsistent, this predicate fails too. If backtracked, fails after removing the posted constraints. This predicate can not be asserted.

B.4.4 Contextual Reasoning

TEMPOS maintains a TIMELOGIC context tree isomorphic to the Rhet context tree. Hence, contextual reasoning can be applied to temporal facts or assumptions. For example,

```
(ASSERT-AXIOMS [TIME-DISJOINT DINNER-TIME BIKING-TIME]
  [WITH "Scenario-1" [TIME-AFTER BIKING-TIME DINNER-TIME]]
  [WITH "Scenario-2" [TIME-BEFORE BIKING-TIME DINNER-TIME]])
```

causes DINNER-TIME to be disjoint from BIKING-TIME in the root context, to be before it in the "Scenario-1" context, and to be after it in the "Scenario-2" context.

B.5 Truth over Time

TEMPOS optionally supports a number of predicates which implement a reified logic of terms held true over time. The basic predicate is TT (*True Throughout*):

[TT ?i*T-TIME ?x*T-U] [Rhet predicate]
is interpreted to mean that ?x is true over interval ?i. In general, it is preferable to query truth in terms of TT, but to assert it in terms of MT (*Maximally True*):

[MT ?i*T-TIME ?x*T-U] [Rhet predicate]
is interpreted to mean that ?x is maximally true over interval ?i, that is, ?x is true over ?i and *not* true over any interval that overlaps or is overlapped by ?i. Another way of saying this is that any interval ?j over which ?x is true must be either equal to, contained in or disjoint from ?i. When an [MT ?i ?x] form is asserted, TEMPOS ensures that the interval ?i is equal to or disjoint from any other interval ?j for which [MT ?j ?x] has been asserted. (Note that this is not complete — logically speaking, “for which [MT ?j ?x] is true” would be more appropriate, but this is intractable.)

[FT ?i*T-TIME ?x*T-U]

[Rhet predicate]

The predicate **FT** (*False Throughout*) is used to assert or query falsehood, and is interpreted to mean that ?x is false over interval ?i. Note that this is not the same as [NOT [TT ?i ?x]]!

[MF ?i*T-TIME ?x*T-U]

[Rhet predicate]

The predicate **MF** (*Maximally False*) is used to assert or query falsehood, and is interpreted to mean that ?x is false over interval ?i such that any other interval *j* over which ?x is false is either within or disjoint from ?i. Note that this is not the same as [NOT [MT ?i ?x]]!

[RT ?i*T-TIME ?x*T-U]

[Rhet predicate]

The predicate **RT** (*Repeatedly True*) is used to query or assert recurrence, and is interpreted to mean that ?x is repeatedly true over interval ?i, that is, there exist a series of intervals ?j₁ ... ?j_n, each meeting the one following, the first one starting ?i and the last one finishing ?i such that ?x is maximally true over each ?j_k. For example,

[RT TIME0046 DISHES]

denotes the fact that the interval **TIME0046** consists of an unspecified number of smaller, consecutive intervals such that **DISHES** is maximally true over each of these subintervals.

B.5.1 Inferred Truth over Time

As mentioned, [TT ?i ?x] is inferred from [MT ?i ?x]. In addition, [TT ?i ?x] is true if [TT ?j ?x] is true, ?x is declared to be of type *liquid*, *downward hereditary* or *concatenable* (see Section 2.3.1), and interval ?i is appropriately related to interval ?j.

B.5.1.1 Liquids, Solids, and other object types

[TIME-UPWARD ?x*T-U]

[Rhet predicate]

?x is true over interval ?i if ?x is true over all intervals ?j contained in ?i. *Note: not currently used in order to avoid circularities. Besides, all subintervals of ?i can not be known in practice.*

[TIME-DOWNWARD ?x*T-U]

[Rhet predicate]

If ?x is true over interval ?i then ?x is true over any interval ?j contained in ?i.

[TIME-LIQUID ?x*T-U]

[Rhet predicate]

Same as [AND [TIME-UPWARD ?x] [TIME-DOWNWARD ?x]].

[TIME-CONCATENABLE ?x*T-U] [Rhet predicate]
 If ?x is true over intervals ?i and ?j, and ?i and ?j meet, then ?x is true over interval ?k, where ?k is the cover of ?i and ?j.

For instance,

```
(DEFINE-TIME [18-JAN-89] [JAN-89])
(ASSERT-AXIOMS [[TT JAN-89 WINTER] <]
               [[TIME-LIQUID WINTER] <]
               [[TIME-DURING 18-JAN-89 JAN-89] <])
(PROVE [TT 18-JAN-89 WINTER])
⇒ [TT 18-JAN-89 WINTER]
```

[TIME-MUTEX ?x*T-U ?y*T-U] [Rhet predicate]
 If true, then any occurrences of ?x and ?y are said to be mutually exclusive, *i.e.*, forced to be temporally disjoint. This is ensured using forward chaining on MT assertions. For instance,

```
(DEFINE-TIME [REDTIME] [GREENTIME])
(ASSERT-AXIOMS [[TIME-MUTEX RED GREEN] <]
               [[MT REDTIME RED] <]
               [[MT GREENTIME GREEN] <])
```

automatically causes the form [TIME-DISJOINT REDTIME GREENTIME] to be asserted.

B.5.1.2 Inferred Truth from Sequences

[MT ?i ?x] is also provable if ?x occurs as part of a known sequence, *i.e.*, if

```
[MT ?j [SEQ ?x ?y]] (or [MT ?j [SEQ ?y ?x]])
```

has been asserted, and interval ?i starts (or finishes) interval ?j.

B.5.1.3 Inferred Truth from Recurrence

Finally, [MT ?i ?x] is provable if ?x occurs as part of a known recurrence, *i.e.*, if [RT ?j ?x] has been asserted, and ?i is an appropriate subinterval of ?j.

Naturally, MT may be inferred from combined sequences and recurrence:

```
(DEFINE-TIME [TESTTIME])
(ASSERT-AXIOMS [[RT TESTTIME [SEQ DAY NIGHT]] <])
(PROVE [AND [POST [TIME-WITHIN ?i*T-TIME TESTTIME]]
         [MT ?i*T-TIME DAY]])
⇒ [AND [POST [TIME-WITHIN ?i~Time-Skolem-0001 TESTTIME]]
     [MT ?i~Time-Skolem-0001 DAY]]
```

Note that the inference mechanism instantiates intervals where necessary. Note also that **PROVE-ALL** *should not be attempted* when using recurrence, as durational reasoning has not yet been incorporated into the TEMPOS recurrence axioms, and hence all recurrence is infinite!

Appendix C

Automatic Referencing Algorithm

In presenting the algorithms, the same nomenclature as in [Allen, 1983] will be adopted to facilitate comparison. For any two intervals i and j , let $N(i, j)$ be the constraint along the arc between i and j in the network, and let $R(i, j)$ be a new constraint between i and j to be added to the network. $N(i, j)$ may be the result of applying the transitivity function along a path through the network involving intervals other than i and j . Let $C(i, j)$ be the explicitly stored constraint between i and j , and $C * (i, j)$ be $C(i, j)$ if it exists, otherwise the result of applying the transitivity function along a path through the network from i to j if such a path exists, otherwise $\emptyset$. If no path exists between i and j , $N(i, j)$ is defined to be the complete set of possible temporal constraints; if no explicit constraint between i and j is stored in the network, $C(i, j)$ is undefined. Hence, $N(i, j)$ can be defined in terms of $C * (i, j)$. The function $N_3(i, k, j)$ will be used to refer to the network relation between intervals i and j forced along the path through the interval k ; in other words,

$$N_3(i, k, j) \equiv Constraints(N(i, k), N(k, j))$$

Modifications to two of Allen's original algorithms are required, namely **Add R(i, j)** which adds a new constraint between intervals i and j , and **Comparable(j, k)** which determines whether a changed constraint between the intervals i and j should be propagated to the relation between intervals j and k . Here is the original algorithm for posting a new constraint, slightly edited to isolate the action of altering the network:

```
to Add R(i, j)
  Assert R(i, j)
  while the queue ToDo is not empty do begin
    Get next <p, q> from queue ToDo
    for each node k such that Comparable(k, q) do
```

```

    Assert  $R(k,q) \leftarrow N(k,q) \cup N3(k,p,q)$ 
    for each node  $k$  such that  $\text{Comparable}(p,k)$  do
        Assert  $R(p,k) \leftarrow N(p,k) \cup N3(p,q,k)$ 
    end
end

```

where Allen would have defined the procedure **Assert** as follows:

```

to Assert  $R(i,j)$ 
    if  $R(i,j) \subset N(i,j)$  then begin
         $C(i,j) \leftarrow R(i,j)$ 
        Add  $\langle i,j \rangle$  to the queue ToDo
    end
end

```

The automatic referencing mechanism can now be embedded in the procedure **Assert**, which will be amended later when the rest of the required mechanisms are in place. Note that this specification is functionally not quite the same as in [Allen, 1983], because changes in constraints take effect immediately rather than during the processing of the corresponding queue entry. The distinction is not crucial except for performance—Allen’s formulation eventually computes the same relations, but tends to waste effort during propagation processing intermediate constraints that are already subsumed by other entries on the queue.

The comparability condition is modified as follows:

$$\text{Comparable}(i,j) \equiv C(i,j) \text{ exists or not } \text{SpecialLinkp}(C^*(i,j))$$

Finally, the various link predicates are defined as follows:

$$\begin{aligned} \text{UpLinkp}(\text{rel}) &\equiv \text{rel} \subseteq \{\text{starts}, \text{during}, \text{finishes}\} \\ \text{DownLinkp}(\text{rel}) &\equiv \text{rel} \subseteq \{\text{started by}, \text{contains}, \text{finished by}\} \\ \text{SideLinkp}(\text{rel}) &\equiv \text{rel} = \{\text{before}\} \text{ or } \text{rel} = \{\text{after}\} \text{ or } \text{rel} = \{\text{meets}\} \text{ or } \text{rel} = \{\text{met by}\} \\ \text{SpecialLinkp}(\text{rel}) &\equiv \text{UpLinkp}(\text{rel}) \text{ or } \text{DownLinkp}(\text{rel}) \text{ or } \text{SideLinkp}(\text{rel}) \end{aligned}$$

C.1 Reducing the branching factor

When a sidelink is being asserted between two intervals i and j , there may be a path from i to j through the reference hierarchy going through a reference interval

$k \in \text{Refs}(i)$. If the cumulative constraint along the path from i through k to j is the same sidelink, then the relation between i and j may be deleted, since, by nature of the way we define and construct the reference hierarchy, this relation is obtainable indirectly without loss of information. The only exception is the case when i and j share a common reference interval k . In this case the relation from i through k to j is under-constrained and cannot be deleted without loss of information. For instance, if $i \{ \text{starts} \} k$ and $j \{ \text{finishes} \} k$ and $i \{ \text{before} \} j$, then the relation $i-k-j$ is $\{ \text{before}, \text{meets}, \text{overlaps} \}$. This check and possible deletion is accomplished by the procedure `CLAssertSideLink`:

```

to AssertSideLink(i,j)
  if Refs(i)  $\cup$  Refs(j) =  $\emptyset$  then
    for each interval  $k \in \text{Refs}(i)$  do
      if  $C(i,j) = N3(i,k,j)$  then begin
        Remove  $C(i,j)$  from the network
        Exit
      end
    end
  end

```

In the context of the gardening/dinner example of Section 3.4.2, after asserting constraint c_9 (`Raking` $\{ \text{equals} \}$ `BoilWater`), a sidelink would be asserted between `Mowing` and `Dinner` (c_{11}) and between `Mowing` and `Dishes`. The latter assertion would be broken because `Gardening` $\in \text{Refs}(\text{Mowing})$ and $N3(\text{Mowing}, \text{Gardening}, \text{Dishes}) = \{ \text{before} \} = C(\text{Mowing}, \text{Dishes})$, but the former assertion would not be broken because $N3(\text{Mowing}, \text{Gardening}, \text{Dinner}) = \{ \text{before}, \text{meets}, \text{overlaps} \}$ which is $\neq C(\text{Mowing}, \text{Dinner})$.

C.2 Constructing the reference hierarchy

When an uplink is asserted between intervals i and j , j is made a reference interval of i ; or when a downlink is asserted between intervals i and j , i can be made a reference interval of j . This is accomplished by the procedure `AddRef(i,j)`, which has to consider a number of special cases:

- j may already be a reference interval of i , since uplinks are not necessarily singular so they can be refined and re-asserted.
- j may already be higher up the reference tree above i ; this can happen, for instance, if this uplink is asserted through propagation. In this case, the current relation between i and j can be deleted.

- i may have a reference interval k which is above j , i.e., $i \{during\} k$ was known as well as $j \{during\} k$, so asserting $i \{during\} j$ implies $i \{during\} j \{during\} k$, which implies the relation $i \{during\} k$ can be obtained indirectly, hence can be deleted. The same holds for any other interval below i .
- i may have a sidelink to an interval k , which could be obtained indirectly through the new reference interval j , hence the current relation between i and k can be deleted. The same holds for any other interval below i .

Let $\mathbf{Refs}^*(i)$ be the closure of the $\mathbf{Refs}$ function, i.e., the set of all reference intervals directly or indirectly above i .

```

to AddRef(i,j)
  if  $j \in \mathbf{Refs}(i)$  then
    Exit
  else if  $j \in \mathbf{Refs}^*(i)$  then
    Remove  $C(i,j)$  from network
  else begin
    DeleteIndirectRefs(i,j)
     $\mathbf{Refs}(i) \leftarrow \mathbf{Refs}(i) \cap \{j\}$ 
    DeleteDiagonals(i,j)
  end
end

```

The recursive procedure $\mathbf{DeleteIndirectRefs}(i,j)$ removes all reference intervals k from the reference sets of i and the intervals below i , if k is j or k is above j . Let $\mathbf{RefdBy}(i)$ be the set of all intervals k such that $i \in \mathbf{Refs}(k)$.

```

to DeleteIndirectRefs(i,j)
  for each node  $k \in \mathbf{Refs}(i)$  do
    if  $k = j$  or  $k \in \mathbf{Refs}^*(j)$  then begin
       $\mathbf{Refs}(i) \leftarrow \mathbf{Refs}(i) - \{k\}$ 
      if  $C(i,k)$  is singular then
        Remove  $C(i,k)$  from the network
    end
  end
  for each node  $k \in \mathbf{RefdBy}(i)$  do
    DeleteIndirectRefs(k,j)
  end
end

```

In general, making an interval j a reference interval of an interval i implies the grafting of two trees, one headed by i , and one which has j as a leaf. Until

this point, i may have had a sidelink to some intervals k to which there existed no reference path. When j is added to the set of reference intervals of i it is possible that there now exists a reference path from i to k going through j , in which case we can delete the sidelink from i to k . The same holds true for any node below i . Removing now superfluous sidelinks is accomplished by the recursive procedure **DeleteDiagonals**:

```

to DeleteDiagonals(i,j)
  for each node k such that SideLinkp (C(i,k)) do
    if Refs(i)  $\cup$  Refs(k) =  $\emptyset$  then
      if C(i,k) = N3(i,j,k) then
        Remove C(i,k) from the network
  for each node k  $\in$  RefdBy(i) do
    DeleteDiagonals(k,j)
end

```

C.3 Asserting relations

All ingredients are now available to modify the procedure **Assert(i,j)**, which adds a newly posted or derived relation to the network. If an uplink or a downlink is asserted then update the reference hierarchy; otherwise, if a sidelink is asserted, then see if it can be removed as described in section C.1:

```

to Assert R(i,j)
  if R(i,j)  $\subset$  N(i,j) then begin
    Add <i,j> to the queue ToDo
    C(i,j)  $\leftarrow$  R(i,j)
    if UpLinkp(C(i,j)) then
      AddRef(i,j)
    else if DownLinkp(C(i,j)) then
      AddRef(j,i)
    else if SideLinkp(C(i,j)) then
      AssertSideLink(i,j)
  end
end

```

C.4 Propagating equality

Equality between intervals can occur frequently in a logic-based reasoning system. Intervals are associated with non-grounded terms which are said to hold over the

intervals. In such a system, unification of terms requires unification of intervals, in other words, asserting their equality. Temporal equality can be viewed as a form of temporal containment, and so should be usable for automatic referencing. The only obstacle is the fact that equality is symmetric, which begs the question: is it an uplink or a downlink? This can be resolved by assigning sequence numbers to intervals as they are introduced, indicating their age. Now the above mechanism for automatic referencing can be extended to include equality by redefining the predicate `UpLinkp` as follows:

```

UpLinkp(rel,from,to)  $\equiv$ 
  if Age(from) > Age(to)
    then Return rel  $\subseteq$  {starts,during,finishes,equals}
    else Return rel  $\subseteq$  {starts,during,finishes}

```

This definition will make the older of two equal intervals a reference interval for the younger one. The predicate `DownLinkp` is modified analogously. Using this extended definition increases the number of potential reference intervals, which in turn tends to reduce the overall network connectivity. Unfortunately, incorporating equality in the referencing mechanism like this will also have the effect of linearizing equivalence sets. Additional gain can be had by using a UNION-FIND algorithm to make the oldest interval in an equivalence set the reference interval for the set, and the youngest the reference interval for all subintervals.