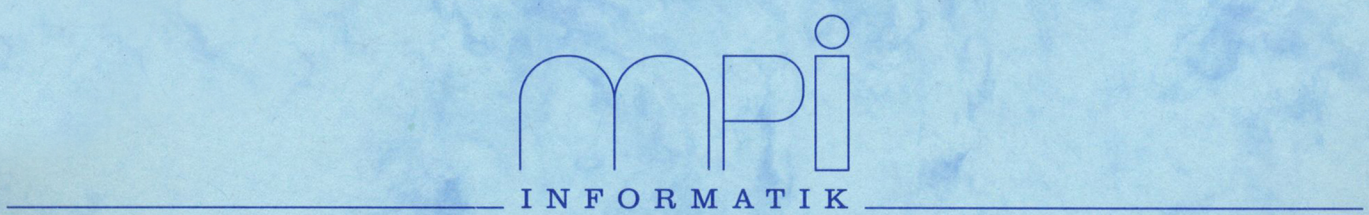
MAX-PLANCK-INSTITUT FÜR INFORMATIK

Temporal Logic
Proceedings of the ICTL Workshop

Hans Jürgen Ohlbach (Editor)

MPI–I–94–230 June 94



Im Stadtwald
66123 Saarbrücken
Germany

Temporal Logic
Proceedings of the ICTL Workshop

Hans Jürgen Ohlbach (Editor)

MPI–I–94–230 June 94

Author's Address

Hans Jürgen Ohlbach (Editor)
Max-Planck-Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken
F. R. Germany
email: ohlbach@mpi-sb.mpg.de

Preface

Time is a very exciting subject. It is one of the few subjects on which everyone is an expert. We move through time continuously and in order to survive and manage ourselves sensibly we constantly have to make temporal decisions. Philosophy, since the days of Aristotle, has been trying to analyse the way we make these decisions. With the rise of computer science, where ideally one wants the machine to do the job for or of the human, there is a new urgency in the precise logical analysis of human temporal activity.

Human (and hence computer) time related activity can be divided into several main areas, all very familiar to us. One of the simplest, and the most important area, is our handling of time dependent data. In computing this is the area of databases. To us ordinary people, it is just time dependent information, involving questions like when to go to the dentist, when to pick up the child from school, until when can one delay in not filing one's tax return and so on. There is another temporal dimension involved in the area of time dependent data besides direct dependency on time. This is the dimension of when a data item is presented to us. For example, if we get a bill to pay our tax on January 1st 1990, it is important when the bill was sent or received, e.g. received September 1989 for January 1990. In database terms there are two times involved: the time dependency of the data and the time when it was introduced into the database. Surprisingly, computing is only now beginning to cope with such things.

Another important area in both human activity and computing is planning. If I have to do the shopping and take my child to visit a friend and cook supper, I have to organise the sequence properly. Going shopping is a simple planning problem but to organise an airport is a more complex planning problem. To be able to let the computer solve it for us we need to develop a logical theory and correctly analyse the steps involved.

Everybody knows the term "time sharing"; what it means in practice is that if neither of us can afford something (e.g. a car or a flat in Spain) then we buy it together and time share. (Computers are more humble, they share things like memory or a printer in order to be more efficient.) We can formulate some intuitive principles on how to share (in computing this is called the *specification*) but there is always the question of exactly how we are going to manage it (what dates am I going to be in the flat and what dates are you, who is going to do the garden and collect the garbage etc.). This is the *implementation* of the principles. Given such an implementation, we have the problem of how to show that it is fair and square and satisfies the specification. One way of doing it is to formulate the procedures in "temporal" logic and then formally prove that it satisfies the specification. In computing the official name is program *specification and verification.*

The main present day research areas of temporal logic are:

1. Philosophical applications. Temporal logic is used in philosophy to clarify various concepts which have been studied since the time of Aristotle. Some examples are causality, historical necessity, identity through time, the notions of events and actions, etc.

2. Applications in computer science as described above.

3. Natural language. Logical analysis of the use of tense and aspect in natural languages. Logical time models for natural language.

4. Pure logical study of temporal logic within the framework of logic itself. Special topics here include:

   (a) Axiom systems, theorem proving and proof theory. Decidability. Model theory.

   (b) Expressive power of temporal languages.

   (c) Applications of temporal logic to the pure theory of other logics (e.g. the notion of provability as a modal logic etc.)

   (d) Deductive reasoning involving time.

To computer science all the above four aspects of the pure logical theory are of great importance.

Temporal logics can be presented in several different ways:

1. Use predicate logic with an additional parameter for time.
2. Use special temporal logics to express temporal phenomena. There are two methods of presentation here.
   (a) Semantic presentation.
   (b) Presentation using axiomatic or other deductive systems for the connectives.
3. The final method is via direct reference to events.

Temporal logic has changed and developed incredibly since its conception as a discipline by Arthur Prior thirty years ago. It is studied by many researchers of numerous and different backgrounds. Different research groups have different conceptions of what temporal logic is and of what it is exactly they themselves do. On many occasions we have heard comments like "that is not logic" referring to a system presented by a colleague. The subject is certainly in a state of accelerated dynamic growth and a new orientation and point of view is currently needed as well as a good coverage of its mathematical and computational aspects. A good understanding, communication and cooperation will enable the subject and the community of researchers to face the challenges of the future.
The ICTL conference is the first *international conference* particularly dedicated to temporal logic. It started with four tutorials: *Programming with Temporal Logics* by Michael Fisher, Manchester Metropolitan University, England, *Incorporating Time in Databases* by Vram Kouramajian, Huston, Texas, USA, *Verification of Finite-State Systems* by Orna Grumberg, The Technion, Haifa, Israel and *Reasoning about Action and Change - Temporal Reasoning in AI* by Erik Sandewall, Linköping University, Sweden. The four invited lectures were given by some of the leading researchers in temporal logic, Johan van Benthem, Hans Kamp, James F. Allen and Amir Pnueli. The main part of the conference was accompanied by a workshop with more informal and spontaneous contributions. This report contains the papers presented at the workshop.

July 1994 Dov Gabbay and Hans Jürgen Ohlbach

Contents

Instantaneous Events

Antony Galton
Department of Computer Science
University of Exeter
Exeter EX4 4PT, UK
Email: antony@uk.ac.exeter.dcs

Abstract

We distinguish strictly *instantaneous* events, which have zero duration, from *momentary* events which have a positive duration that is in some sense minimal. We further classify events, insofar as their occurrence conditions can be given in terms of the holding or not holding of states, into *transitions*, which are characterised in terms of the states holding immediately before and after the event, and *tenures*, which are characterised in terms of a state holding when the event actually happens, but neither immediately before nor after it. These categories are considered in relation to both continuous and discrete models of time. Precise occurrence conditions are given for all events arising from these considerations, using Allen's well-known interval calculus, extended to allow reference to instants. In addition, the concept of *dominance* is introduced to furnish a criterion for whether or not a given qualitative event type admits instantaneous occurrences. We further consider the possibility of temporal models that are neither continuous nor discrete, instancing as examples Allen and Hayes' *moments* and Russell's account of the relation between the psychological subjective present and an underlying physical time continuum.

1 Introduction

Thinkers of an empiricist frame of mind have always tended to view with suspicion the notion of an instantaneous event (that is, an event strictly lacking in duration), claiming that those events which one might be inclined to regard as instantaneous do in fact last for some possibly very short positive time. Russell, for example, states that 'events of which we are conscious do not last for a mathematical instant but always for some finite time, however short' ([Russell, 1914], p.121). Newton-Smith endorses this view with the words 'we have direct experience of events that last for some interval of time and we have no direct experience of anything obtaining for only an instant' ([Newton-Smith, 1980], p.135).

Arguments against this view have generally pointed to beginnings and endings as paradigm cases of truly instantaneous events, with at least a strong suggestion that these are the only such events that there are. Anscombe, rebutting Russell, notes that 'though we cannot think of an instantaneous event falling within our experience that is not a terminus of something that takes time, we can think of plenty of events that are such termini; and we may perhaps reasonably take such events as what we mean by "instantaneous" ones' [Anscombe, 1964]. Similarly, Allen affirms that 'events that appear to refer to a point in time (e.g., finishing a race) are considered to be implicitly referring to another event's beginning or ending' [Allen, 1984].

In this paper I am not concerned to take a stand on the question of whether there really are instantaneous events, either in the physical world or in our experience. Rather I take as my starting point the inescapable fact that most systems that have been proposed for the logical analysis of temporal phenomena have the property that, whether or not their proposers acknowledged this, they admit—indeed require—the description of instantaneous events.

Consider, for example, a simple change of state, say a body's starting to move. In a scientific spirit, one might note that an extended body consists of trillions of atoms that are in a ceaseless state of agitation; the body's being at rest amounts to the fact that the vector sum of all these motions, averaged over time, is zero. The averaging over time makes it impossible to pinpoint, even in principle, an instant at which this condition ceases to obtain. Thus it might be claimed that no real meaning can be attached to the assertion that the body's beginning to move is an instantaneous event.

In the context of an exact qualitative logic of time, however, this kind of objection is beside the point. So long as one admits the notions of

(a) one time interval's meeting, or abutting, another, expressed by, say, the predicate $Meets(i, j)$;

(b) a state's holding over an interval, expressed by $Holds(s, i)$;

(c) a state's failing to hold within an interval, expressed by $Holds(not(s), i)$

then it is natural to allow formulae of the form

$$Meets(i, j) \wedge Holds(not(s), i) \wedge Holds(s, j)$$

which state, in effect, that the event of state s's starting to hold is instantaneous, occurring at the instant at which interval i meets interval j.

To rule out this possibility, one might simply stipulate as an axiom the formula

$$i < j \wedge Holds(not(s), i) \wedge Holds(s, j) \rightarrow$$
$$\exists u(i < u < j \wedge \neg Holds(not(s), u) \wedge \neg Holds(s, u)),$$

which postulates that between a period of s's wholly failing to hold and a later period of s's holding there must be a 'grey' interval during which s neither holds nor wholly fails to hold. Those who have set up logical systems for describing temporal phenomena have, however, conspicuously refrained from availing themselves of any such axiom. And indeed, even if they did, in most systems one might still end up with the situation expressed by the formula

$$Holds(s, j) \wedge \forall i(Meets(i, j) \rightarrow \neg Holds(s, i))$$

in which we can still recognise an event occurring at the initial instant of i, namely the onset of uninterrupted holding of s. (Just before that instant, s could indeed hold, but only in infinitely fine alternation with $not(s)$—a situation that is explicitly ruled out by, for example, [Hamblin, 1971] and [McDermott, 1982].)

Instantaneous events of the kind discussed in this section—Anscombe's 'termini'—constitute only one of a variety of phenomena which we shall need to consider under this heading. In the remainder of this paper I shall take a close look at a range of phenomena which have some claim to being called instantaneous events. My aim is to present an exhaustive account of the phenomenon of instantaneity as this appears in those logical models of time that have found favour in the temporal reasoning community.

2 A brief look at events in general

Taken as a whole, events are a pretty heterogeneous bunch. In particular, most of the events which concern us in everyday life are, at least on one level of description, complex: that is, they can be broken down in thought into simpler events composed either sequentially, in parallel, or according to a more complex set of ordering constraints. The event of my going for a swim, for example, involves my going to the pool or the beach, getting changed, entering the water, spending a while in the water, leaving the water, getting dried and dressed, and finally leaving the pool or beach for wherever I am going next. This is typical in that it can be described as a sequence of simpler events each of which is either a *transition* or a *tenure*. A **transition** is an event-type that can be specified in terms of its start state and final state: the transition is the change from the former state's holding to the latter's. For example, my getting changed is a transition from my being fully clothed to my being in my swimming things, my entering the water is a transition from my being out of the water to my being in it. A **tenure** is an event-type that can be specified in terms of a single state, the event itself consisting of that state's persisting over a period, e.g., my spending a while in the water.

It would appear that most complex events can be broken down into simple events that are either transitions or tenures. Now if an *instantaneous* event is complex, its components must all be instantaneous, and they must occur simultaneously rather than in sequence. We are thus led to consider simple instantaneous transitions and tenures. This will be done in the next section.

Our aim is to specify an event-type by giving its **occurrence conditions**, i.e., necessary and sufficient conditions for the truth of a formula of the form $Occurs(e, t)$, whose intended meaning is that an occurrence of the event-type e takes place at time t—which may in general be either an instant or an interval. For a complete reductive account of a simple event type, we require the occurrence conditions to be composed only of formulae of the form $Holds(s, t)$ and relations on times, e.g., $Meets(i, j)$[1]. Our notation will largely follow that of [Allen, 1984] as far as intervals are concerned, with natural extensions to instants as suggested in [Vilain, 1982] and [Galton, 1990].

As noted above, simple events in general may be divided into transitions and tenures. We shall refer to transitions using the notation $trans(s_1, s_2)$, where s_1 is a state that holds immediately before the transition and s_2 a state that holds immediately after, it being assumed in general that s_1 and s_2 cannot both hold together. A tenure is specified as the holding of some state, preceded and followed by times at which the state does not hold, and will be notated $ten(s)$, where s is the state whose holding characterises the tenure.

In an interval-based system such as that of [Allen, 1984], durative (non-instantaneous) forms of these event-types can be specified straightforwardly as follows:

$$Occurs(trans(s_1, s_2), i) \equiv$$
$$\exists i_1 \exists i_2(Meets(i_1, i) \wedge Meets(i, i_2) \wedge$$
$$Holds(s_1, i_1) \wedge Holds(s_2, i_2) \wedge$$
$$Holds(not(s_1), i) \wedge Holds(not(s_2), i))$$
$$Occurs(ten(s), i) \equiv$$
$$\exists i_1 \exists i_2(Meets(i_1, i) \wedge Meets(i, i_2) \wedge$$
$$Holds(s, i) \wedge Holds(not(s), i_1) \wedge$$
$$Holds(not(s), i_2)$$

3 Instantaneous Events

In a system which does not contain instants, it is not possible similarly to specify instantaneous events of any kind, since we cannot even write an expression of the form $Occurs(e, t)$ where t denotes an instant. Nonetheless, some instantaneous events must occur in a world which is describable using such a system. As already noted, if we have

$$Meets(i_1, i_2) \wedge Holds(not(s), i_1) \wedge Holds(s, i_2)$$

then an event occurs which we may describe as the inception of state s. The event occurs within any interval which overlaps both i_1 and i_2, but it does not occur *on* any such interval. In fact the only possible candidate

[1] This is exemplified for a certain class of 'motion events in [Galton, 1993].

for the time at which the event occurs is the instant at which interval i_1 meets interval i_2. If this instant is denoted t, then following [Allen and Hayes, 1989] we write $Meets\text{-}at(i_1, i_2, t)$. Allen and Hayes allow that this instant exists, in the sense that it can be constructed as a 'nest' of mutually connected intervals, but they resolutely refuse to allow properties (what we here call states) to hold at instants, and have little to say about whether they allow events to occur at instants [2].

Suppose we allow some events to occur at instants. We shall use the predicate *Occurs-at* for this. Then we can specify an instantaneous transition by the occurrence condition

$$\begin{aligned} &Occurs\text{-}at(trans(s_1, s_2), t) \equiv \\ &\quad \exists i_1 \exists i_2 (Meets\text{-}at(i_1, i_2, t) \wedge \\ &\qquad Holds(s_1, i_1) \wedge Holds(not(s_2), i_1) \\ &\qquad Holds(s_2, i_2) \wedge Holds(not(s_1), i_2)) \end{aligned}$$

Note that Allen and Hayes should be quite happy with this, since the definiens does not contain any notions absent from their system.

The above occurrence condition does not, of course, say anything about whether or not a particular type of instantaneous transition can occur. We will want to allow that it can occur for some choices of s_1 and s_2, but not for others. For example, if r_1 and r_2 are two regions of space which do not touch, and b is a body small enough to fit inside either region, then the transition

$$trans(in(b, r_1), in(b, r_2))$$

cannot be instantaneous, since in order to pass from the interior of r_1 to the interior of r_2, the body b has to cross over the space between the two regions. Later we will look in more detail at which pairs of states give rise to the possibility of instantaneous transitions.

Given their refusal to allow states to hold at instants, it would seem to be impossible, in the system of Allen and Hayes, to specify instantaneous tenures. Their refusal is admittedly not absolute, for they allow that a proposition might be said to be true at an instant so long as it is true over some interval within which that instant falls. If we use $Holds\text{-}at(s, t)$ to say that a state s holds at an instant t, this may be expressed by the equivalence

$$Holds\text{-}at(s, t) \equiv \exists i (Div(t, i) \wedge Holds(s, i)).$$

Here I use $Div(t, i)$ to mean that the instant t falls within the interval i, thereby *dividing* it into two contiguous subintervals.

I argued in [Galton, 1990] that we must also allow a state to hold instantaneously, that is, to hold at an instant without holding over any interval bounded by that instant. If we do not allow this then we cannot have continuous change. This is because a continuously changing object passes through each state only instantaneously. Thus what we require is the possibility of

$$Holds\text{-}at(s, t) \wedge \forall i (Div(t, i) \rightarrow \neg Holds(s, i)),$$

directly contradicting the Allen and Hayes position.

This will allow us to specify an instantaneous tenure by means of the occurrence condition

$$\begin{aligned} &Occurs\text{-}at(ten(s), t) \equiv \\ &\quad Holds\text{-}at(s, t) \wedge \exists i_1 \exists i_2 (Meets\text{-}at(i_1, i_2, t) \wedge \\ &\quad Holds(not(s), i_1) \wedge Holds(not(s), i_2)). \end{aligned}$$

Not all states give rise to the possibility of instantaneous tenure. We will investigate further below the circumstances under which any given state may or may not hold instantaneously.

Instantaneous tenure would appear to be possible in all and only those worlds where continuous change is possible. In fact, I would advocate the following view, which may be regarded as in some ways analogous to the view expressed by Allen and Hayes. They insist that a state may hold at an instant only by virtue of holding over an interval divided by that instant; thereby they reduce *Holds-at* to *Holds*. My view, which is motivated by the definition of the derivative in the differential calculus, is that a state may only hold at an isolated instant by virtue of states arbitrarily close to it holding during arbitrarily small intervals divided by that instant.

More exactly, a state can have instantaneous tenure only if it belongs to a continuum of states endowed with the notion of a **neighbourhood**. A neighbourhood of a state s is a state n such that, first, whenever s holds, so does n, and second, any transition from s to $not(n)$ or vice versa must involve $n \,\&\, not(s)$[3]. Then for $ten(s)$ to occur at t it is necessary that for any neighbourhood n of s, however small, there is an interval i divided by t such that n holds over i.

The paradigmatic case of instantaneous tenure involves a continuous real-valued variable v. Let s be the state $v = v_0$ in which v takes some particular value; for this to have instantaneous tenure at t, it is necessary that for any $\epsilon_1, \epsilon_2 > 0$ there is an interval i divided by t throughout which the state $v_0 - \epsilon_1 < v < v_0 + \epsilon_2$ holds. (This is simply a version of the standard mathematical definition of continuity.) The neighbourhoods of the state $v = v_0$ are all the states $v_0 - \epsilon_1 < v < v_0 + \epsilon_2$, where $\epsilon_1, \epsilon_2 > 0$. It may be that *any* world in which continuous change is possible must admit description in these terms; at any rate this has proved to be a most fruitful paradigm for mathematical physics, which essentially made no progress until mathematics had developed to the point where it could support such a paradigm[4].

An example of a world where continuous change (and hence instantaneous tenure) is not possible is a world where the facts of temporal incidence can be adequately described using Allen's logic. In such a world, whenever a state holds, it holds over an interval. When a state

[2] But note the following: 'Since intervals are the periods during which a proposition is true, the places where they meet another interval are the times when truth-values change.'

[3] The conjunction of two states is defined by the rules

$$\begin{aligned} Holds(s \,\&\, s', i) &\equiv Holds(s, i) \wedge Holds(s', i) \\ Holds\text{-}at(s \,\&\, s', t) &\equiv Holds\text{-}at(s, t) \wedge Holds\text{-}at(s', t). \end{aligned}$$

[4] Note, however, that an explicit, rigorous formulation of the paradigm did not emerge for more than a century after its first appearance—the interval between Newton and Cauchy.

holds over an interval, during that interval the world is unchanging with respect to that state. If on a neighbouring interval a different. incompatible state holds, then the transition from the first state to the second must happen discontinuously. This is the world of Zeno's arrow, which is at rest at each position it finds itself in; if one believes that motion must by nature be continuous, then in such a world motion is impossible, as Zeno stated.

Another kind of world in which there is no continuous change is one in which time is discrete. Any interval can be divided up into a finite collection of **atomic intervals**, that is, intervals which cannot themselves be divided any further. During an atomic interval no change can occur. As with Allen's world, in discrete time instantaneous tenure is not possible. In both worlds, however, instantaneous transitions occur whenever two incompatible states hold over neighbouring intervals.

In the discrete-time world, there is another phenomenon which might be mistaken for an instantaneous tenure but in fact is quite different. This is tenure over an atomic interval. It may be specified by the occurrence condition

$$\begin{aligned} &Occurs(ten(s), i_n) \equiv \\ &\quad Holds(not(s), i_{n-1}) \wedge Holds(s, i_n) \wedge \\ &\quad Holds(not(s), i_{n+1}), \end{aligned}$$

where the atomic intervals are designated $\ldots, i_{-2}, i_{-1}, i_0, i_1, i_2, \ldots$. Allen and Hayes call atomic intervals 'moments'. A tenure over an atomic interval might therefore be called **momentary** rather than instantaneous (cf. [Galton, 1984], p.65).

Discrete time could fittingly be described as *cinematographic time*, each frame of a cine-film corresponding to an atomic interval. If the atomic intervals are short enough, humans are unable to distinguish discrete time from continuous time. This is a function of the human nervous system, and it is not necessarily obvious that we should duplicate this inability to discriminate in our logical theory. (But see below, section 7.)

4 Examples of Instantaneous Events

4.1 Instantaneous events in continuous time

We shall illustrate the above ideas by considering the continuous motion of bodies in space. We shall denote by $b, b', b'', \ldots$ various extended physical bodies, and by $r, r', r'', \ldots$ various regions of space. At each instant the body b has a position $pos(b)$, which is exactly the region of space occupied by b at that instant.

Qualitative relations between regions can be expressed using the spatial calculus of [Randell *et al.*, 1992]. We shall use the following predicates taken from their system:

- $DC(r, r')$: r and r' are *disconnected*, i.e., they do not overlap and they do not share any boundary points.
- $EC(r, r')$: r and r' are *externally connected*. i.e., they do not ovelap but they share at least one boundary point.
- $PO(r, r')$: r and r' are *partially overlapping*, i.e., they have a common subregion but neither is wholly contained in the other.
- $r = r'$: r and r' are the same region.
- $TPP(r, r')$: r is a *tangential proper part* of r', i.e., r is wholly contained in r' and shares at least one boundary point with it.
- $NTPP(r, r')$: r is a *non-tangential proper part* of r', i.e., r is wholly contained in r' and shares no boundary points with it.

An example of an event which is always instantaneous is

$$trans(DC(pos(b), pos(b')), EC(pos(b), pos(b'))).$$

This is the event of bodies b and b' coming into contact; it is expressed by saying that their positions change from being disconnected to being externally connected.

The event $ten(EC(pos(b), pos(b')))$ may be instantaneous but doesn't have to be. If the bodies move apart again as soon as they have touched then it is instantaneous; but if they stay in contact for a while then it is not.

Similarly, the event

$$trans(DC(pos(b), r), PO(pos(b), r)),$$

by which body b moves from being disconnected from a region to being partly inside it, may be instantaneous or not. The only way to get continuously from $DC(pos(b), r)$ to $PO(pos(b), r)$ is by passing through $EC(pos(b), r)$. But this intermediate state may hold instantaneously or not: b may pass straight through this position, or it may stop there before proceeding.

The event

$$trans(EC(pos(b), r), TPP(pos(b), r))$$

cannot be instantaneous. This is the event of a body's entering a region: initially it is just outside the region, and after the event it is just inside it. Since the body is extended it takes time for it to cross the boundary of the region. While the event is in progress the relation $PO(pos(b), r)$ holds.

Similarly, the event

$$ten(DC(pos(b), pos(b')))$$

cannot be instantaneous. This is the event of two bodies' spending a while not touching. Immediately before the event, and immediately after it, the relation $EC(pos(b), pos(b'))$ must hold, i.e., the bodies are touching. The event thus consists of the bodies' breaking contact and moving apart from each other, then moving together again. This cannot happen instantaneously.

Thus we see that we can describe three kinds of events: those which *must* be instantaneous, those which *may* be instantaneous but need not be, and those which *cannot* be instantaneous. We have exhibited both tenures and transitions of the second and third kind, but only a transition of the first; intuitively it seems plausible that no tenure event *has* to be instantaneous, though some may be. In general, we have used our intuitive understanding of continuity to determine which category each of our examples belongs to; what we should like to do is to replace intuition by a systematic method for deriving these results.

4.2 Instantaneous and momentary events in discrete time

We shall take as our model for discrete time the sequence of moves in a game of chess (here we have discrete space as well). We should distinguish at the outset between a game of chess as a mathematical abstraction and its physical realisation as the motion of wooden pieces on a board. Any chess-player knows that the physical pieces and board are unnecessary, and it is perfectly possible for two competent players to conduct a game entirely verbally, while walking down the street, say.

Considered *in abstracto*, the game consists of a discrete sequence of states

$$s_1, s_2, s_3, \ldots, s_n,$$

each state being characterised by the distribution of pieces on the board together with such information as which player has the next move, whether each king has moved (determining whether castling is possible), and so on. Nothing that happens outside this sequence of states has any reality as far as the game is concerned. In particular, states of the game may be considered to follow on from one another without a gap. A move in the game is an instantaneous transition $trans(s_i, s_{i+1})$ from one state to the next in the sequence.

If s and s' are arbitrary game-states, then the event $trans(s, s')$ may or may not be possible, and if it is possible then it may or may not be instantaneous. It will be impossible if, for example, there are more pieces in play in state s' than in state s (pieces cannot be "untaken"). Given that the transition is possible, then it can be instantaneous if it is possible to get from s to s' in a single move, but not otherwise. But even if it *can* be instantaneous, it doesn't in general have to be: it may be possible to get from s to s' indirectly by a sequence of moves[5].

As we have already noted, instantaneous tenure is not possible in discrete time: instead we have momentary tenure, when a state persists for just one atomic period. In chess, a player *must* move when it is his turn, and therefore every occurrence of $ten(s)$, where s is a game-state, must be momentary.

We need not consider only complete game-states, however. We can introduce a predicate sq, which gives the location of a piece on a square. Thus $sq(black\text{-}queen, c5)$ is the state in which the black queen is on square C5. The event $ten(sq(black\text{-}queen, c5))$ occurs if the black queen is moved onto C5 and subsequently either moved to a different square or taken. In the latter case, this event may be momentary, but because of the nature of chess as a two-player game, it cannot be momentary in the former case: if black moves his queen to C5 and white does not take it, then two moments must elapse before black can move it away again, since it is white's turn to move next.

The transition

$$trans(sq(black\text{-}king, C5), sq(black\text{-}king, D4))$$

can be instantaneous, since it can be accomplished by a single diagonal move of the king; but it need not be, since the king can get from C5 to D4 by way of, for example, C4. On the other hand, the transition

$$trans(sq(black\text{-}king, C5), sq(black\text{-}king, C2))$$

cannot be instantaneous since it requires at least three moves to accomplish it.

[5] Note, however, that if s' is obtained from s by advancing a pawn one square, then the transition can only be instantaneous.

5 Criteria for instantaneity

The general problem confronting us may be stated as follows: given a description of an event-type e in a form such as $trans(s, s')$ or $ten(s)$, how can we determine whether the occurrences of e may, must, or cannot be instantaneous? Let us call this the **instantaneity problem**.

5.1 States of position and states of motion

In [Galton, 1990] I proposed a division of states into two categories, called **states of position** and **states of motion**. A state of position holds at the instants which limit any interval throughout which it holds, and it can hold at isolated instants (i.e., it admits of instantaneous tenure); a state of motion, on the other hand, can only hold at an instant by virtue of holding throughout some interval divided by that instant. I pointed out that Allen's interval calculus is adequate for handling states of motion but not for handling states of position.

In [Galton, 1993] I used states of position and states of motion to provide a solution to the instantaneity problem for the motion of a rigid body, insofar as this can be described in terms of transitions using the spatial relation predicates of Randell, Cui, and Cohn. The analysis makes use of the notion of **perturbation**: a state s' is a perturbation of state s so long as either of the following cases is possible:

$$Holds(s, i) \wedge Lim(t, i) \wedge Holds\text{-}at(s', t)$$
$$Holds(s', i) \wedge Lim(t, i) \wedge Holds\text{-}at(s, t)$$

where $Lim(t, i)$ says that the instant t is one of the limits (end-points) of interval i. The **perturbation principle** states that for a rigid body b and a region r, and distinct spatial relation predicates R and R', the state $R(pos(b), r)$ can only be a perturbation of state $R'(pos(b), r)$ if one of the two states is a state of motion and the other a state of position.

Events of the form $trans(R(pos(b), r), R'(pos(b), r))$, where b is a rigid body, can now be classified with respect to instantaneity as follows:

1. *Necessarily instantaneous events.* R and R' are perturbations of each other (and therefore one is a state of position and the other a state of motion).
2. *Possibly instantaneous events.* R and R' are distinct states of motion, having a common perturbation R'' (which must therefore be a state of position).
3. *Necessarily durative events.* All other cases, i.e., distinct states of motion without a common perturbation, a state of motion and a state of position which are not perturbations of each other, distinct states of position.

5.2 A new classification based on 'dominance'

It was noted in [Galton, 1993] that the two-fold distinction between states of position and states of motion is inadequate when we consider the motion of a non-rigid body. For example, $TPP(pos(b), r)$ seems to be a state of position when considered as a perturbation of $NTPP(pos(b), r)$ but a state of motion when considered as a perturbation of $pos(b) = r$. The latter perturbation does not occur, of course, if b is rigid.

I have since found a wide range of examples in which the distinction between states of motion and states of position breaks down. In all these cases, the simple binary division must be replaced by a more complex hierarchical classification based on a special relation of *dominance*. Roughly speaking, state s dominates state s' if s is a perturbation of s' and behaves like a state of position when so considered. A more exact definition is given below.

We shall say that state s **bounds** state s' so long as it is possible for s to hold at an instant which bounds an interval throughout which s' holds, i.e., it is possible to have

$$Holds\text{-}at(s,t) \wedge Lim(t,i) \wedge Holds(s',i).$$

Clearly any state trivially bounds itself so long as it can hold over an interval (which presumably all states can). Our earlier notion of perturbation is simply the symmetric closure of the bounding relation, i.e., s and s' are perturbations of each other just if one of them bounds the other.

We shall need a stronger notion of bounding. For this purpose we introduce the notion of a *substate*. A state may be regarded as a class of situations: that is, to say that a system is in state s is to specify certain aspects of the situation while leaving other aspects undetermined. Intuitively, s' is a substate of s if the set of situations which fall under s' is a subset of the class of situations which fall under s'. For example, $PO(pos(b), r)$ is a substate of $not(DC(pos(b), r))$: one way for b to fail to be disconnected from the region is for it to partially overlap the region. We can now say that s **totally bounds** s' if every substate of s (including s itself) bounds s'.

We shall say that state s **dominates** state s' so long as s totally bounds s' but s' does not bound s. Thus dominance is both irreflexive and asymmetric. In addition, if s dominates s' then s is a perturbation of s', though the reverse implication does not hold.

To illustrate these ideas, we consider a world in which we have various bodies $b_1, b_2, b_3, \ldots$ which are free to move about in space. We shall be interested in which bodies are touching one another at any given time. We shall abbreviate the states $EC(pos(b_i), pos(b_j))$ and $DC(pos(b_i), pos(b_j))$ as $e(i,j)$ and $d(i,j)$ respectively. Assuming the bodies cannot interpenetrate, each of these states is equivalent to the negation of the other.

The case of two bodies is simple: $e(i,j)$ dominates $d(i,j)$. For three bodies, things become more complex. There are altogether 8 distinct states to consider, namely

$$e(i.j) \,\&\, e(j.k) \,\&\, e(i.k)$$
$$e(i.j) \,\&\, e(j.k) \,\&\, d(i.k)$$
$$e(i.j) \,\&\, d(j.k) \,\&\, e(i.k)$$
$$e(i,j) \,\&\, d(j,k) \,\&\, d(i,k)$$
$$d(i,j) \,\&\, e(j,k) \,\&\, e(i,k)$$
$$d(i,j) \,\&\, e(j,k) \,\&\, d(i,k)$$
$$d(i,j) \,\&\, d(j,k) \,\&\, e(i,k)$$
$$d(i,j) \,\&\, d(j,k) \,\&\, d(i,k)$$

For brevity, we may refer to these as *eee*, *eed*, etc. Consider *eed*. In this state, b_i is touching b_j, which is touching b_k, but b_i is not touching b_k. Possible pertubations from this state are

- Body b_i comes into contact with body b_k, giving state *eee*. If *eed* holds at instant t, then some time must elapse before *eee* holds, during which b_i and b_k are moving towards each other. This time is limited by the first instant at which *eee* holds. Thus *eee* is a bounding state of *eed*, but not vice versa; so *eee* dominates *eed*.
- Body b_j loses contact with b_k, giving state *edd*. When this happens, there is a last moment at which *eed* holds, and then *edd* holds over an interval during which b_j moves away from b_k (and possibly for a longer period thereafter). Hence *eed* dominates *edd*.
- Body b_i loses contact with b_j, giving state *ded*. As above, *eed* dominates *ded*.
- Body b_j simultaneously loses contact with both b_i and b_k, giving state *ddd*. Here *eed* dominates *ddd*.

One might suppose that further perturbations are possible, such as 'b_i simultaneously loses contact with b_j and comes into contact with b_k, leading to state *dee*'. But a little reflection will show that at the instant this happens the state *eee* must hold: a case of instantaneous tenure. So *dee* is not a perturbation of *eed*; rather each is a perturbation of *eee*, which dominates them both.

The three-body system $\{b_i, b_j, b_k\}$ can be thought of as a combination of three independent two-body systems, namely $\{b_i, b_j\}$, $\{b_j, b_k\}$, and $\{b_i, b_k\}$. These systems are independent because whether two bodies are in mutual contact does not in general depend on their state of contact with other bodies. (In particular cases, of course, dependencies may arise as a result of extra factors not present in the general case: e.g., it is not possible for a person to be simultaneously touching both Nelson's Column and the Eiffel Tower, assuming those monuments to be fixed in their present positions.)

A general rule may be formulated for determining the dominance relations in complex systems which can be decomposed into simpler independent ones. We shall use the notation $s \preceq s'$ to mean that either s dominates s' or $s = s'$. Then the rule is

> Let the system M consist of n subsystems M_1, M_2, ..., M_3 acting independently. For $i = 1.\ldots,n$ let s_i and s_i' be two possible states of M_i. Then state $s_1 \,\&\, s_2 \,\&\, \cdots \,\&\, s_n$ of M is a perturbation of $s_1' \,\&\, s_2' \,\&\, \cdots \,\&\, s_n'$ if and only if either $s_i \preceq s_i'$ for $i = 1.\ldots.n$, or $s_i' \preceq s_i$ for $i = 1.\ldots.n$: and in the former case

$s_1 \,\&\, s_2 \,\&\, \cdots \,\&\, s_n \preceq s'_1 \,\&\, s'_2 \,\&\, \cdots \,\&\, s'_n$, while in the latter case it is the other way about.

The three-body example given above illustrates this rule in operation.

6 Dominance and instantaneity

We are now in a position to state precisely what instantaneous events are possible in continuous time.

The event $trans(s, s')$, where s and s' are incompatible states, is possible only if there exists a sequence $s = s_0, s_1, \ldots, s_n = s'$ such that, for $i = 1, \ldots, n$, s_i is a perturbation of s_{i-1} and incompatible with it. Call this a **transition sequence** from s to s'. Note that in a sense this observation is entirely trivial since an arbitrary transition from s to s' must be characterisable by one of the transition sequences s, s' and $s, not(s) \,\&\, not(s'), s'$.

If s is a perturbation of s' we get the case $n = 1$, and a direct transition is possible. For some choices of s and s' no other case occurs, that is, there are no sequences with $n > 1$. In this case $trans(s, s')$ is *necessarily instantaneous*. An example is two bodies coming into contact, i.e., $trans(d(b_i, b_j), e(b_i, b_j))$ in the notation of the previous section.

If s is a perturbation of s' and there also exists a transition sequence with $n > 1$, then the event $trans(s, s')$ is *possibly instantaneous*. An occurrence of the event which involves a direct transition from s to s' will be instantaneous, but one which involves traversing a transition sequence with $n > 1$ will not be. As an example in the three-body world, $trans(eee, ddd)$ can occur instantaneously, if all three bodies lose contact simultaneously, but it can also occur duratively, e.g., by means of the transition sequence eee, eed, edd, ddd.

If s' is not a perturbation of s then $trans(s, s')$ can be instantaneous only if there is a transition sequence s, s_1, s', where some substate of s_1 (possibly s_1 itself) dominates both s and s'; and even in this case instantaneity is merely possible and not necessary. An example from the three-body world is $trans(edd, ded)$, for which there exist three minimal transition sequences, namely

1. edd, ddd, ded,
2. edd, eed, ded,
3. edd, eee, ded.

The middle term dominates the outer terms in cases (2) and (3); thus the transition can occur instantaneously. In both cases what happens is that b_i loses contact with b_j at the same instant that it comes into contact with b_k; if sequence (3) is followed then in addition b_k makes fleeting contact with b_j just at the same instant. Note that the transition associated with sequence (1) must be durative, since it involves b_i separating from b_j and subsequently making contact with b_k.

Any other transition must either involve a transition sequence s, s_1, s' where every substate of s_1 fails to dominate at least one of s, s', or a transition sequence $s, s_1, \ldots, s_{n-1}, s'$ where $n > 2$. In the former case, since no substate of s_1 dominates both its neighbours, s_1 must hold over an interval, making the transition durative. In the latter case, the instant at which s gives way to s_1 must strictly precede the instant at which s_{n-1} gives way to s', again making the whole transition durative. We have now completely determined the possible instantaneous transitions.

We turn next to tenure events. The rule is simple to state. The tenure $ten(s)$ can be instantaneous if and only if there are states s', s'', disjoint from s though not necessarily distinct from each other, such that some substate s_1 of s dominates both s' and s''. Then the transition sequence s', s_1, s'' can occur, and it is possible (though not necessary) for state s_1, and hence state s also, to hold only for an instant: in which case we have an instantaneous occurrence of $ten(s)$. For example, in the two-body world, $ten(e(b_i, b_j))$ can occur instantaneously, but $ten(d(b_i, b_j))$ cannot. In the three-body world, the sequence ddd, edd, ddd affords the possibility of an instantaneous tenure $ten(edd)$; the sequence eed, edd, eed is also possible, but the corresponding occurrence of $ten(edd)$ is necessarily durative.

7 Some other models

I believe that the foregoing considerations exhaust the topic of instantaneous events so long as the model for time is either continuous or discrete. Not all models of time can easily be categorised in this way, however. In particular, many researchers have been attracted to models of time which are not discrete but in which there nonetheless exist atomic intervals. The system of [Allen and Hayes, 1989] is of this kind. They define a *moment* as an interval with no internal structure; the motivation is to provide times of occurrence for events such as flashes and bangs which do not possess perceptible duration but which are not instantaneous in the strict sense. What makes their system different from the discrete-time system discussed above is their axiom

$$\forall m \forall n (Moment(m) \wedge Moment(n) \rightarrow \neg Meets(m, n)).$$

Thus moments cannot be contiguous, and hence time cannot be regarded as discrete.

A momentary tenure in this system has the occurrence condition

$$\begin{aligned} &Moment(m) \rightarrow \\ &\quad (Occurs(ten(s), m) \leftrightarrow \\ &\quad \exists i \exists j (Meets(i, m) \wedge Meets(m, j) \wedge \\ &\qquad Holds(not(s), i) \wedge Holds(s, m) \wedge \\ &\qquad Holds(not(s), j))) \end{aligned}$$

Likewise, a momentary transition would be characterised by

$$\begin{aligned} &Moment(m) \rightarrow \\ &\quad (Occurs(trans(s, s'), m) \leftrightarrow \\ &\quad \exists i \exists j (Meets(i, m) \wedge Meets(m, j) \wedge \\ &\qquad Holds(s, i) \wedge Holds(s', j) \wedge \\ &\qquad Holds(not(s), m) \wedge Holds(not(s'), m))) \end{aligned}$$

Truly instantaneous events will be defined as in the other systems we have discussed, though it is not entirely clear how easy it will be to reconcile this particular model for

time with the continuity requirements for instantaneous tenure.

An alternative construction in which time is not discrete but there nonetheless exists the notion of a minimal interval is suggested by a psychological account of the subjective present. Many considerations lead to the idea that our experience of the present is not instantaneous but occupies a short interval, such that everything that happens in that interval is present to our consciousness together. This can explain how it is that we perceive motion, so long as it is neither too slow nor too fast, as a phenomenal quality *sui generis* rather than simply as a succession of different positions. On the other hand, it seems implausible to suggest that these subjective presents form a discrete series; one rather thinks of a "window" moving smoothly along a continuum[6].

One way we might attempt to model this is by picking out certain small, 'subjectively minimal', intervals as *moments*. We allow moments to meet or even overlap, and add a density requirement to the effect that given any two moments, there is a third whose beginning falls strictly between the beginning of the earlier moment and the beginning of the later moment. We also ensure that moments are all of essentially the same duration by insisting that whenever moment a begins before moment b begins then it also ends before moment b ends (i.e., the mapping from moment-beginnings to moment-endings is order-preserving).

The resulting model is not a model of the subjective experience of time alone; rather it models the way in which this is superimposed on an underlying physical continuum which contains intervals of arbitrarily short duration. This is very much in the spirit of Russell's attempt to reconcile the psychological account of time with the physico-mathematical one. We can now characterise an event as momentary so long as it falls entirely within one moment. Subjectively, we would perceive two momentary events as simultaneous so long as there is a single moment within which they both occur; this is compatible with their in fact occurring successively.

8 Conclusions

In this paper I distinguished strictly *instantaneous* events, which have zero duration, from *momentary* events which have a positive duration that is in some sense minimal. I further classified events, insofar as their occurrence conditions can be given in terms of the holding or not holding of states, into *transitions*, which are characterised in terms of the states holding immediately before and after the event, and *tenures*, which are characterised in terms of a state holding when the event actually happens, but neither immediately before nor after it. These categories can be considered in relation to both continuous and discrete models of time, resulting in a multiplication of cases to examine.

Precise occurrence conditions were given for all events arising from these considerations, using Allen's well-known interval calculus, extended to allow reference to instants. In addition, the concept of *dominance* was introduced to furnish a criterion for whether or not a given qualitative event type admits instantaneous occurrences.

I further considered, though in less detail, the possibility of a temporal model that is strictly speaking neither continuous nor discrete, instancing as examples Allen and Hayes' *moments* and Russell's account of the relation between the psychological subjective present and an underlying physical time continuum.

[6] Cf. [Russell, 1914], p. 139.

References

[Allen and Hayes, 1989] James F. Allen and Patrick J. Hayes. Moments and points in an interval-based temporal logic. *Computational Intelligence*, 5:225–238, 1989.

[Allen, 1984] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[Anscombe, 1964] G. E. M. Anscombe. Before and after. *Philosophical Review*, 73:3–24, 1964.

[Galton, 1984] Antony Galton. *The Logic of Aspect.* Clarendon Press, Oxford, 1984.

[Galton, 1990] Antony Galton. A critical examination of Allen's theory of action and time. *Artificial Intelligence*, 42:159–188, 1990.

[Galton, 1993] Antony Galton. Towards an integrated logic of space, time, and motion. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993.

[Hamblin, 1971] C. L. Hamblin. Instants and intervals. *Studium Generale*, 24:127–134, 1971.

[McDermott, 1982] Drew McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[Newton-Smith, 1980] W. H. Newton-Smith. *The Structure of Time.* Routledge and Kegan Paul, 1980.

[Randell *et al.*, 1992] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, pages 165–176, 1992. Cambridge, Massachusetts, October 1992.

[Russell, 1914] Bertrand Russell. *Our Knowledge of the External World as a Field for Scientific Method in Philosophy.* The Open Court Publishing Company, Chicago and London, 1914.

[Vilain, 1982] Marc B. Vilain. A system for reasoning about time. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-82)*, pages 197–201. The American Association for Artificial Intelligence, 1982.

On the Satisfiability Problem for Lamport's Propositional Temporal Logic of Actions and Some of Its Extensions *

Y S Ramakrishna
Theoretical Computer Science Group
Tata Institute of Fundamental Research
Homi Bhabha Road, Bombay 400 005, India
e-mail: *ysr@tcs.tifr.res.in*

Abstract

The Temporal Logic of Actions devised by Lamport is a language for proving the correctness of concurrent systems. A distinctive feature of the logic is the use of "action formulæ" to encode the "before-after" behaviour of transitions, a feature that is especially suitable for Lamport's transition-axiom method. Abadi has given a complete axiomatization for the propositional fragment of this logic, and conjectured that its validity problem may be in PSPACE. In this paper we confirm that conjecture. In fact we show that the validity problem for TLA is PSPACE-complete. For this we give an optimal automata-theoretic decision procedure for TLA. Our upper-bound result is obtained with respect to a logic which is a conservative extension of Lamport's TLA. Thus our decision procedure applies to the more restricted logic. We show how the automata-theoretic framework handles abstraction, as used in TLA, a feature considered useful for hierarchical and compositional reasoning.

1 Introduction

Lamport recently introduced a Temporal Logic of Actions, which allows reasoning with actions, and seems eminently suited for hierarchical and compositional reasoning [7]. Abadi [2] has provided a complete axiomatization for a propositional fragment of TLA. Based on the proof of completeness of his axiomatization, Abadi conjectured that the satisfiability problem for TLA may be in PSPACE. In this paper we confirm that conjecture — we show, in fact, that the problem is PSPACE-complete. We obtain our upper-bound by an optimal automata-theoretic decision procedure for TLA.

*Much of the work reported in this paper was done in the Spring of 1992, while at the Dept of Elec and Comp Eng at the University of California, Santa Barabara, CA, USA, with partial support from the NSF/ARPA grant CCR-9014382.

The decision procedure we present is in the well-known Büchi-automata-theoretic style [11, 12, 13]. Our automaton construction is inspired partially by the completeness proof of Abadi's axiomatization. However, we consider a natural syntactic extension of the language considered by Abadi. Our construction shows how the method of automata may be extended rather simply to allow satisfiability checking of logics defined with respect to both states *and* transitions, as opposed to the more usual state-based temporal logics. Once this aspect of action formulæ is taken care of, it is easy to generalize our construction to PTLA with abstraction subscripts [7], and we do so in a later section. Another advantage of the automata-theoretic framework is that it indicates that an arbitrary temporal logic, which is invariant under stuttering and is decidable by automata-theoretic methods, might be extended with actions and abstractions à la Lamport, while still retaining decidability.

The logic we present conservatively extends the propositional fragment of the Temporal Logic of Actions introduced by Lamport. Thus our decision procedure applies to the more restricted logic of Lamport. For the lower-bound, a cursory examination of our proof [10] will reveal that we do not require the full power of the extended logic to obtain PSPACE-hardness. Thus, the lower-bound result applies also to his logic.[1] Contrast this with the fact that the satisfiability problem for PTL with 'Henceforth', but no 'Until', a sub-logic of Lamport's PTLA, is NP-Complete.

The rest of this paper is organized as follows. In Section 2 we introduce the syntax, the class of intended models and the semantics for the logic. Section 3 introduces

[1] We do not attempt in this paper to justify, in any detail, the extended logic. We do feel, however, that the added expressiveness of the extended logic might allow correctness proofs to dispense with the use of prophecy variables, allowing instead such properties to be captured directly by formulæ of the logic. While we do not know if our extension properly extends the expressive power of propositional TLA, it probably does enhance succinctness of formulæ stating complex temporal properties.

some definitions and concepts that we use subsequently for describing the decision procedure. Section 4 gives the decision procedure, following the automata-theoretic paradigm [5, 13], and sketches a proof of its correctness. The complexity of the decision problem is characterized at the end of the section, relegating proof details to [10]. In Section 5 we show how the procedure of Section 4 is generalized to obtain a decision procedure for the logic with arbitrary abstraction subscripts. We conclude in Section 6 with, among other things, a general discussion of how similar techniques might be used to decide extensions of decidable temporal logics with the introduction of TLA-like features, namely actions and abstraction.

2 The Logic

2.1 Syntax

The syntax of TLA, represented by the set of well-formed formulæ (wff) of the logic, relative to a finite set $\mathcal{P}$ of primitive propositions and a disjoint finite set $\mathcal{A}$ of primitive actions, is defined by the following BNF grammar.

$$\begin{array}{lcl} f & ::= & \mathsf{true} \mid P \mid \neg f \mid f_1 \wedge f_2 \mid \Box f \mid [a] \\ a & ::= & A \mid f \mid f' \mid \neg a \mid a_1 \wedge a_2 \end{array}$$

where $P \in \mathcal{P}$ and $A \in \mathcal{A}$. As is customary, we use false as an abbreviation for $\neg$true, $f \vee g$ for $\neg(\neg f \wedge \neg g)$, $\Diamond f$ for $\neg\Box\neg f$, and $\langle a \rangle$ for $\neg[\neg a]$. We also syntactically identify $\neg\neg f$ with f. We shall refer to the syntactic category identified by f above as wff and that identified by a above as well-formed actions (wfa). Note that we have departed slightly from the syntax of Lamport's TLA by not placing subscripts on action brackets. Assume, for the present, that in every case the action formula $[a]$ is an abbreviation for $[a]_{(\mathcal{P},\mathcal{A})}$. Thus, like Abadi [2] we do not (yet) allow arbitrary abstraction subscripts in our language. We discuss, in Section 5, how arbitrary abstraction subscripts may be handled.

Modulo our remark regarding abstraction subscripts above, observe that Lamport's TLA is a sublanguage of the language we have introduced above. The following BNF grammar describes the syntax of the propositional fragment of Lamport's TLA, as used by Abadi [2].

$$\begin{array}{lcl} f & ::= & \mathsf{true} \mid P \mid \neg f \mid f_1 \wedge f_2 \mid \Box f \mid [a] \\ a & ::= & A \mid p \mid \neg a \mid a_1 \wedge a_2 \\ p & ::= & P \mid \neg p \mid p_1 \wedge p_2 \mid p' \end{array}$$

where $P \in \mathcal{P}$ and $A \in \mathcal{A}$. The difference is that in the language above, an arbitrary temporal formula cannot appear (primed or otherwise) inside action brackets, as it can in our extended version of TLA. In particular, $[[A]']$ where A is a primitive action is a legal action formula in our TLA (as is $[(\Box p)']$), but is not in Lamport's. The formula $[[A]']$ can be used to assert that at the second non-stuttering transition, if any, an A action executes. Similarly, $[(\Box p)']$ states that following the next non-stuttering transition, if any, p remains true forever. Whether this increased vocabulary increases the expressive power of the logic is at this time an open question, whose answer we conjecture is in the affirmative. The semantics of Lamport's TLA extend naturally to this larger language, giving a logic that is a conservative extension of Lamport's TLA. In the remainder of the paper, we shall refer to this larger language as TLA, and refer to the original TLA (of Lamport) as LTLA.

Notation. Throughout this paper we use the following convention for metalogical variables, with or without subscripts, unless stated otherwise : $f, g, \ldots, p, q$ for wff of TLA, a, b for wfa, A, B for primitive actions and P, Q for primitive (state) propositions. We shall use card in the conventional sense to mean the cardinality of a (finite) set.

To characterize the complexity of our decision procedure we shall need the notion of *size* of a well-formed string in our language. The size of a formula is the number of primitive propositions, primitive actions, logical connectives and action operators (brackets and priming) in the formula, This is represented by the function size.

2.2 Models

The traditional manner of defining transition-based models is to represent a computation as a sequence of alternating states and state-transitions (or events), each state being labelled with a subset of $\mathcal{P}$, representing the set of primitive propositions that hold in that state and each state-transition by a subset of $\mathcal{A}$, representing the set of actions that executed to cause the transition. In order to extend the existing automata-theoretic paradigm uniformly to such doubly labelled transition sequences, we shall modify the representation of these models slightly (and in the obvious fashion).

A TLA structure is a sequence of *clairvoyant states* where a clairvoyant state[2] is an element of $2^{\mathcal{P}} \times 2^{\mathcal{A}}$. Where a distinction needs to be made between a state in the conventional sense, representing a valuation to the primitive state propositions, and a clairvoyant state, we shall refer to the latter as a *c-state*. When there is no confusion we shall often simply say state for a c-state. Note that by treating a c-state as a tuple, we are making an implicit type-distinction between primitive propositions and primitive actions. A similar device will be used to maintain the type distinction in the automata used in the decision method. This distinction is not necessary but turns out to be a convenient device for describing the mechanics of our construction. It is easy to see that the unique c-state sequence corresponding to a conventional doubly-labelled transition sequence is obtained by merely pulling into a state the label on its outgoing arc.

[2] Clairvoyant, since the state can predict the actions to occur in the next transition.

A TLA structure such as described above is a satisfying model for a TLA formula if the formula is satisfied at the initial state of the sequence under the semantics that we outline in the next subsection.

We shall represent a model for TLA as a mapping $\mathcal{M} : \omega \rightarrow 2^{\mathcal{P}} \times 2^{\mathcal{A}}$. For technical reasons, we shall need to extend $\mathcal{M}$ so that the first component of a state is extended to a larger set of wff using the semantics that we define in the next section.

Notation. Let $\mathcal{M}$ be a TLA model as defined above. Then $\mathcal{M}(i)$ refers to the i^{th} c-state in the model, and $\mathcal{M}_\pi(i)$ and $\mathcal{M}_\alpha(i)$ represent, respectively, the set of propositions and the set of actions in that c-state – *i.e.* the two components of a c-state. Otherwise, whenever we have say a sequence σ of n-tuples, we shall abbreviate $\sigma(i)(j)$ as $\sigma_j(i)$, representing the jth component of the ith tuple in the sequence.

2.3 Semantics

A TLA formula is evaluated at a state in a model with the semantics for the boolean connectives defined in the usual manner. The temporal formula $\Box f$ is defined as usual to mean that f holds in every suffix of this sequence (a sequence is its own suffix). The formula $[a]$ is the distinctive feature of the logic, and asserts that the next *non-stuttering* transition, if any, satisfies a. Intuitively, for a primitive action A, satisfaction corresponds to the execution of that action.[3] Boolean combinations of actions have their natural meaning: $a \wedge b$ denotes simultaneous execution of a and b, $a \vee b$ denotes choice, and $\neg a$ means that an a-action did not execute.

In addition to primitive actions, the logic allows actions to be synthesized from other formulæ in the logic, by the use of a unary priming operator $'$. For instance if P is a primitive proposition, then $[P']$ asserts that P is true in the state immediately following the next non-stuttering transition. Intuitively, a transition is non-stuttering if there is an observable change in the system – either some state proposition changes its valuation or some action executes across the transition.[4] Similarly $[P]$ asserts that P is true immediately preceding the next non-stuttering transition. As a further example, consider the TLA formula $[f \wedge g']$. It requires that if the current state is not a terminal state then the formula g must hold in the next state reached as a result of a non-stuttering transition and f to hold before that transition. Thus the formula [true] always holds in any state, the formula [false] holds only in an infinitely stuttering state, and the formula $\neg$[false] holds in a state such that there is some non-stuttering transition in the future. These concepts are formalized in the following definition of the semantics of the logic.

Definition 2.1 The semantics for TLA formulæ are defined at a state $\mathcal{M}(i)$ of $\mathcal{M}$ as follows :

- $\langle \mathcal{M}, i \rangle \models \text{true}$
- $\langle \mathcal{M}, i \rangle \models P$ for $P \in \mathcal{P}$ iff $P \in \mathcal{M}_\pi(i)$
- $\langle \mathcal{M}, i \rangle \models \neg f$ iff $\langle \mathcal{M}, i \rangle \not\models f$
- $\langle \mathcal{M}, i \rangle \models f \wedge g$ iff $\langle \mathcal{M}, i \rangle \models f$ and $\langle \mathcal{M}, i \rangle \models g$
- $\langle \mathcal{M}, i \rangle \models \Box f$ iff for all $j \geq i$ $\langle \mathcal{M}, j \rangle \models f$
- $\langle \mathcal{M}, i \rangle \models [a]$ iff either for all $j \geq i$ $\mathcal{M}_\pi(i) = \mathcal{M}_\pi(j)$ and $\mathcal{M}_\alpha(j) = \emptyset$, or there exists $k > i$ such that $\mathcal{M}_\pi(i) \neq \mathcal{M}_\pi(k)$ or $\mathcal{M}_\alpha(k-1) \neq \emptyset$ such that $\langle \mathcal{M}, k-1 \rangle \| \!= a$ and for all $i < j < k$, $\mathcal{M}_\pi(i) = \mathcal{M}_\pi(j)$ and $\mathcal{M}_\alpha(j-1) = \emptyset$

In the above the $\|\!=$ relation is defined between a wfa and an element of $(2^{\mathcal{P}} \times 2^{\mathcal{A}})^\omega \times \omega$ inductively as follows :

- $\langle \mathcal{M}, i \rangle \| \!= A$ for $A \in \mathcal{A}$ iff $A \in \mathcal{M}_\alpha(i)$
- $\langle \mathcal{M}, i \rangle \| \!= f$ for f a wff iff $\langle \mathcal{M}, i \rangle \models f$
- $\langle \mathcal{M}, i \rangle \| \!= f'$ iff $\langle \mathcal{M}, i+1 \rangle \models f$
- $\langle \mathcal{M}, i \rangle \| \!= \neg a$ iff not $\langle \mathcal{M}, i \rangle \| \!= a$
- $\langle \mathcal{M}, i \rangle \| \!= a \wedge b$ iff $\langle \mathcal{M}, i \rangle \| \!= a$ and $\langle \mathcal{M}, i \rangle \| \!= b$

A formula f is *satisfiable* iff there exists a model $\mathcal{M} \in (2^{\mathcal{P}} \times 2^{\mathcal{A}})^\omega$ such that $\langle \mathcal{M}, 0 \rangle \models f$. In such a case, $\mathcal{M}$ is a *satisfying model* for f. A formula f is *valid* if every model in $(2^{\mathcal{P}} \times 2^{\mathcal{A}})^\omega$ is a satisfying model for f. A finite set S of formulæ is satisfiable (respectively, valid) iff their conjunction is.

3 Preliminaries

The automata-theoretic approach to the satisfiability problem is to construct an appropriate automaton for a formula f. This automaton takes models of the logic as input. A model is accepted by the automaton if and only if the model satisfies f. The decision procedure then consists of checking if there is *any* model that is accepted by the automaton for f. This approach to verification was first popularized by Vardi and Wolper [11, 12]. For an excellent introduction to and overview of this approach we refer the reader to [13].

3.1 Subformula Closure

As is usual for automata-theoretic decision procedures, we first define the subformula closure for a given TLA formula, whose satisfiability we intend to check. The subformula closure $\text{scl}(f)$ captures the idea that in deciding the satisfiability of a formula f, one need only consider formulæ in the set $\text{scl}(f)$.

Definition 3.1 The *subformula closure* $\text{scl}(f)$ of a formula f is the smallest set of wff such that

1. $f \in \text{scl}(f)$

[3] Lamport considers actions to be instantaneous. In that sense his actions correspond to what have been more popularly referred to as *events*.

[4] Refer, however, to Section 5 to see how observability is relativized with respect to some specific subset of actions and propositions.

2. true $\in$ scl(f)
3. $f_1 \in$ scl(f) iff $\neg f_1 \in$ scl(f)
4. if $f_1 \wedge f_2 \in$ scl(f) then $f_1 \in$ scl(f) and $f_2 \in$ scl(f)
5. if $\Box f_1 \in$ scl(f) then $f_1 \in$ scl(f)
6. for any wff f_1, if $[f_1] \in$ scl(f) or $[f_1'] \in$ scl(f) then $f_1 \in$ scl(f)
7. if $[a] \in$ scl(f) then [true] $\in$ scl(f)
8. $[a] \in$ scl(f) iff $[\neg a] \in$ scl(f)
9. if $[a \wedge b] \in$ scl(f) then $[a] \in$ scl(f) and $[b] \in$ scl(f)

The following is an easy consequence of the above definitions and follows by induction on the structure of a TLA formula.

Lemma 3.2 *For a formula f of size n,* card(scl(f)) = $O(n)$.

3.2 Hintikka Pairs and Model Extension

Most constructive decision procedures use sets of formulæ to construct the "components" of a canonical model for a given formula. The formulæ in the sets give a complete characterization of that component in terms of not only atomic formulæ, but also more complicated formulæ. The following concept of Hintikka pairs captures this notion. Note, however, that instead of constructing a canonical model for a formula, our method constructs, in fact, a canonical model-generator. This is also the case for most other linear-time temporal logics that we are aware of [9, 13]. The model-generator is canonical in the sense that every model for f can be "generated" from $\mathcal{A}_m(f)$, the automaton constructed for f. For a general discussion of these issues we refer the interested reader to the overview paper [13] by Wolper.

Definition 3.3 [HINTIKKA PAIR] A *Hintikka pair for f* is an element $\mathbf{s} = \langle \mathbf{s}_\pi, \mathbf{s}_\alpha \rangle$ of $2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}$ such that

- $\mathbf{s}_\pi$ satisfies the following conditions
  1. true $\in \mathbf{s}_\pi$ and [true] $\in \mathbf{s}_\pi$
  2. for all $f_1 \in$ scl(f), $f_1 \in \mathbf{s}_\pi$ iff $\neg f_1 \notin \mathbf{s}_\pi$
  3. for all $f_1 \wedge f_2 \in$ scl(f), if $f_1 \wedge f_2 \in \mathbf{s}_\pi$ then $f_1 \in \mathbf{s}_\pi$ and $f_2 \in \mathbf{s}_\pi$
  4. for all $\Box f_1 \in$ scl(f), if $\Box f_1 \in \mathbf{s}_\pi$ then $f_1 \in \mathbf{s}_\pi$
  5. for all $[a] \in$ scl(f), $[a] \in \mathbf{s}_\pi$ or $[\neg a] \in \mathbf{s}_\pi$
  6. for all $[a] \in$ scl(f), if $\neg[a] \in \mathbf{s}_\pi$ then $\neg$[false] $\in \mathbf{s}_\pi$
  7. for all $[a] \in$ scl(f), if $\neg$[false] $\in \mathbf{s}_\pi$ then $[a] \in \mathbf{s}_\pi$ iff $[\neg a] \notin \mathbf{s}_\pi$
  8. for all $[a \wedge b] \in$ scl(f), if $[a \wedge b] \in \mathbf{s}_\pi$ then $[a] \in \mathbf{s}_\pi$ and $[b] \in \mathbf{s}_\pi$
- $\mathbf{s}_\alpha$ satisfies the following conditions
  1. if [false] $\in \mathbf{s}_\pi$ then $\mathbf{s}_\alpha = \emptyset$
  2. if $\mathbf{s}_\alpha \neq \emptyset$ then $\mathbf{s}_\alpha = \{ A \mid A \in \mathcal{A}, [A] \in \mathbf{s}_\pi \}$

A Hintikka pair $\mathbf{s}$ is *silent* when $\mathbf{s}_\alpha = \emptyset$. The set of all Hintikka pairs for f is denoted by $\mathbf{H}(f)$.

Think of a Hintikka pair as an extended c-state of a TLA model. Observe that the first component of a Hintikka pair is *complete* (upto the elements in scl(f)) in the sense that for every $f_1 \in$ scl(f) either f_1 or $\neg f_1$ is in it. Moreover, the first component is also *propositionally consistent* in the sense that it contains precisely one of f_1 and $\neg f_1$, for any $f_1 \in$ scl(f). The second component of a Hintikka pair contains precisely the primitive actions that must happen in the next transition out of that state.

Notice that we do not have a clause for "breaking down" general temporal formulæ inside action brackets, or for looking deeper than one level into action brackets. The unwrapping of action brackets and subsequent breakdown of such formulæ is accomplished by the local automaton's transition conditions (see Clauses 4 and 5 of $\mathcal{A}_l$'s transition conditions, on the next page). The "one-level look-in" is, however, required in order to deal with primitive actions and their boolean combinations, or primed formulæ in boolean combination with other formulæ.

Definition 3.4 [EXTENSION] Given a model $\mathcal{M}$ and a formula f, the *extension* of $\mathcal{M}$ with respect to f is the function $\mathcal{M}^f : \omega \to 2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}$ such that

$$\mathcal{M}^f_\pi(i) = \{ f_1 \mid f_1 \in \mathbf{scl}(f), \langle \mathcal{M}, i \rangle \models f_1 \}$$
$$\mathcal{M}^f_\alpha(i) = \mathcal{M}_\alpha(i)$$

The following lemma states that the (extended) c-states in the extension of an arbitrary model with respect to an arbitrary formula are Hintikka in the sense of our definition above.

Lemma 3.5 *Let f be a TLA formula, $\mathcal{M}$ a model, and $\mathcal{M}^f$ its extension with respect to f. Then for every $i \in \omega$, $\mathcal{M}^f(i)$ is a Hintikka pair.*

Proof. Assume for some i that $\mathcal{M}^f(i)$ is not Hintikka. The result follows by a straightforward case analysis of the Hintikka conditions and the semantics to exhibit a contradiction. We consider here only a sample case to illustrate the argument. Assume, for instance that Condition 7 of Definition 3.3 is violated. Considering the 'if' part of the 'iff', let $\mathcal{M}^f(i)$ contain $\neg$[false], $[a]$ and $[\neg a]$ for some $[a] \in$ scl(f). By the semantics and the definition of an extension, therefore, there is some $j > i$ such that $\mathcal{M}(i) \neq \mathcal{M}(j)$. Because $\langle \mathcal{M}, i \rangle \models [a]$ we have $\langle \mathcal{M}, j-1 \rangle \| \models a$ and because $\langle \mathcal{M}, i \rangle \models [\neg a]$ we have $\langle \mathcal{M}, j-1 \rangle \| \not\models a$, a contradiction. The reverse direction is similar, and the remaining cases are also straightforward. ■

Note, however, that not every ω-sequence of Hintikka pairs corresponds to the extension of some model. This is the case since temporal formulæ in the Hintikka pairs may impose restrictions on their sequencing. The automata

that we build in the sequel operate on strings of Hintikka pairs, and detect "temporal conflicts" by their consecution and acceptance conditions.

4 Decision Procedure

We now describe the construction of the promised automaton corresponding to a formula.

The following definition of a Büchi Automaton is the standard one recast in our notation and terminology.

Definition 4.1 [BÜCHI AUTOMATON] A *Büchi Automaton* is a tuple $\langle \Sigma, \mathbf{S}, \rho, \mathbf{S}_I, \mathbf{S}_F \rangle$ where[5]

- Σ is a finite input alphabet
- $\mathbf{S}$ is a finite set of states
- $\rho : \mathbf{S} \times \Sigma \rightarrow 2^{\mathbf{S}}$ defines a transition function specifying for each state and input symbol the set of possible next states
- $\mathbf{S}_I \subseteq \mathbf{S}$ is the set of initial states
- $\mathbf{S}_F \subseteq \mathbf{S}$ is the set of accepting states.

A *run* of the automaton on an ω-string $\tau \in \Sigma^\omega$ is an ω-string $\sigma \in \mathbf{S}^\omega$ such that $\sigma(0) \in \mathbf{S}_I$ and for all $i \in \omega$, $\sigma(i+1) \in \rho(\sigma(i), \tau(i))$. The run is *accepting* iff the set $\{i \in \omega \mid \sigma(i) \in \mathbf{S}_F\}$ is infinite. An automaton in state $\mathbf{s} \in \mathbf{S}$ *consumes* an input symbol $\mathbf{t} \in \Sigma$ iff $\rho(\mathbf{s}, \mathbf{t}) \neq \emptyset$. We shall say that an automaton consumes a string if it has a run (not necessarily accepting) on that string.

We are now ready to define the construction of the automata that will be used in our decision procedure. For any formula f, the *local automaton* $\mathcal{A}_l(f)$ will be responsible for ensuring that local consistency conditions are not violated. These will essentially guarantee that, starting in a "legal" initial state, the automaton can only enter states that are consistent with its previous state. The *eventuality automaton* $\mathcal{A}_e(f)$ will be responsible for ensuring that liveness requirements are met. Where it is clear from context, we shall omit the parameterization of the automata by the formula.

Definition 4.2 Two elements $\mathbf{s}, \mathbf{t} \in (2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}})$ constitute a *stuttering pair* $\langle \mathbf{s}, \mathbf{t} \rangle$ iff $\mathbf{s}_\alpha = \emptyset$ and $\mathbf{s}_\pi \cap \mathcal{P} = \mathbf{t}_\pi \cap \mathcal{P}$.

Local Automaton

The local automaton checks the consistency of local transitions. We represent the local automaton by $\mathcal{A}_l(f) = \langle 2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}, \mathbf{H}(f), \rho_l, \mathbf{N}(f), \mathbf{H}(f) \rangle$ where

- $\rho_l : \mathbf{H}(f) \times (2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}) \rightarrow 2^{\mathbf{H}(f)}$ is the nondeterministic transition function such that $\mathbf{t} \in \rho_l(\mathbf{s}, \mathbf{i})$ iff
  1. $\mathbf{i} = \mathbf{s}$
  2. if $\Box f_1 \in \mathbf{s}_\pi$ then $\Box f_1 \in \mathbf{t}_\pi$
  3. if $\neg\Box f_1 \in \mathbf{s}_\pi$ and $f_1 \in \mathbf{s}_\pi$ then $\neg\Box f_1 \in \mathbf{t}_\pi$
  4. if $\langle \mathbf{s}, \mathbf{t} \rangle$ is not a stuttering pair then, for all $[f_1'] \in \mathbf{scl}(f)$, $[f_1'] \in \mathbf{s}_\pi$ iff $f_1 \in \mathbf{t}_\pi$
  5. if $\langle \mathbf{s}, \mathbf{t} \rangle$ is not a stuttering pair then, for all $[f_1] \in \mathbf{scl}(f)$ such that f_1 is a wff, if $[f_1] \in \mathbf{s}_\pi$ then $f_1 \in \mathbf{s}_\pi$
  6. if $\langle \mathbf{s}, \mathbf{t} \rangle$ is a stuttering pair then, for all $[a] \in \mathbf{scl}(f)$, $[a] \in \mathbf{s}_\pi$ iff $[a] \in \mathbf{t}_\pi$
- $\mathbf{N}(f) = \{\mathbf{s} \mid \mathbf{s} \in \mathbf{H}(f), f \in \mathbf{s}\}$ is the set of initial states.

Note that $\mathcal{A}_l$ is non-deterministic and, in a sense, "drives" the model-generation procedure by making choices, wherever they are available, without violating local consistency conditions. However, in any state, $\mathcal{A}_l$ accepts a unique symbol from its alphabet corresponding to that state. In particular, no symbol that is not Hintikka is ever consumed by $\mathcal{A}_l$.

Observe that for all $\mathbf{s} \in \mathbf{H}(f)$ such that $\mathbf{s}_\alpha = \emptyset$, we have $\mathbf{s} \in \rho_l(\mathbf{s}, \mathbf{s})$, *i.e.* each silent state of $\mathcal{A}_l$ has a self-loop. This, coupled with the fact that each state of $\mathcal{A}_l$ is accepting, means that the string obtained by pumping any number of copies of a silent state into an accepting string is also accepted by $\mathcal{A}_l$.

Finally, observe that our definition 4.2 of stuttering-pairdom accomplishes the check whether or not some primitive proposition has changed between two states or if some primitive action has happened. It is easy to generalize this definition to deal with arbitrary abstraction subscripts. In Section 5 we outline the requisite modifications required to obtain this generalization. Of course, all our primitive actions are uninterpreted in the sense of Abadi [2]. However, it is simple to define in TLA, interpretations for primitive actions; for instance, by demanding that $\Box([\mathit{reset}(x)] \equiv [(x = 0)'])$, where $\mathit{reset}(x)$ may be considered a primitive action and $(x = 0)$ a primitive state proposition within the propositional framework. Such interpretations would impose requirements on the states connected by a transition labelled by an action.

Eventuality Automaton

The eventuality automaton is required for checking whether eventuality requirements are fulfilled in infinite runs. This also includes the verification of the fact that in a non-terminated computation a non-stuttering step is eventually taken. The eventuality automaton is a deterministic Büchi automaton represented by $\mathcal{A}_e(f) = \langle 2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}, \{I\} \cup (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)}), \rho_e, I, 2^{\mathcal{P}} \times \{\emptyset\} \rangle$ where

- $\mathbf{E}(f) = \{\neg\Box f_1 \mid \neg\Box f_1 \in \mathbf{scl}(f)\} \cup \{\neg[\mathsf{false}]\}$
- $I \notin (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)})$ is the initial state (without any further internal structure)

[5] When we want to emphasize that the automaton is deterministic, we shall feel free to use $\langle \Sigma, \mathbf{S}, \rho, s_0, \mathbf{S}_F \rangle$, with $s_0 \in \mathbf{S}$ and $\rho : \mathbf{S} \times \Sigma \rightarrow \mathbf{S}$, instead. See, for instance, our definition of the Eventuality Automaton.

- $\rho_e : (\{I\} \cup (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)})) \times (2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}) \rightarrow (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)})$ is the deterministic transition function such that $\rho_e(\mathrm{s}, \mathrm{i}) = \mathrm{t}$ iff [6]

  1. $\mathrm{t}_0 = \mathrm{i}_\alpha \cap \mathcal{P}$
  2. if $\mathrm{s} = I$ then
     $\mathrm{t}_1 = \{ \neg\Box f_1 \mid \neg\Box f_1 \in \mathrm{i}_\pi, \neg f_1 \notin \mathrm{i}_\pi \} \cup$
     $\{ \neg[\mathsf{false}] \mid \neg[\mathsf{false}] \in \mathrm{i}_\pi, \mathrm{i}_\alpha = \emptyset \}$
  3. if $\mathrm{s} \neq I$ and $\mathrm{s}_1 = \emptyset$ then
     $\mathrm{t}_1 = \{ \neg\Box f_1 \mid \neg\Box f_1 \in \mathrm{i}_\pi, \neg f_1 \notin \mathrm{i}_\pi \} \cup$
     $\{ \neg[\mathsf{false}] \mid \neg[\mathsf{false}] \in \mathrm{i}_\pi, \mathrm{i}_\alpha = \emptyset, \mathrm{s}_0 = \mathrm{t}_0 \}$
  4. if $\mathrm{s} \neq I$ and $\mathrm{s}_1 \neq \emptyset$ then
     $\mathrm{t}_1 = \{ \neg\Box f_1 \mid \neg\Box f_1 \in \mathrm{s}_1, \neg f_1 \notin \mathrm{i}_\pi \} \cup$
     $\{ \neg[\mathsf{false}] \mid \neg[\mathsf{false}] \in \mathrm{s}_1, \mathrm{i}_\alpha = \emptyset, \mathrm{s}_0 = \mathrm{t}_0 \}$

An intuitive view of the operation of $\mathcal{A}_e$ is as follows. It carries in its first component, the set of primitive propositions that were true in the last state of the input (this explains the need for the special initial state). This is used for checking if a non-stuttering step has happened in the case where no primitive action is required to happen but the state is required to change. The second component keeps track of unsatisfied eventualities, and the requirement of another step having to be taken. Whenever the second component is empty these conditions are copied from the input into the automaton, and are discarded as and when they are satisfied, until the second component empties out once more, at which point the process is repeated. Acceptance is ensured only if the second component empties infinitely often. Note that it is sufficient that the eventuality automaton checks that some non-stuttering step happens – when such a step does indeed happen, the local automaton checks (because of our definition of Hintikka pairs) that all the actions that should have happened at that non-stuttering step did indeed happen.

Observe that $\mathcal{A}_e$ is deterministic and complete. Thus, in any state s, there is a unique next state specified for every $\mathrm{i} \in (2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}})$. It should be clear from the discussion above of the operation of $\mathcal{A}_e$ that for any state $\mathrm{s} \neq I$, if $\rho_e(\mathrm{s}, \mathrm{i}) = \mathrm{t}$, $\mathrm{s}_1 \neq \emptyset$ and $\mathrm{t}_1 \neq \mathrm{s}_1$, then $\mathrm{t}_1 \subset \mathrm{s}_1$.

Finally, observe that for every state s of $\mathcal{A}_e$ and every input i in its alphabet, if $\rho_e(\mathrm{s}, \mathrm{i}) = \mathrm{t}$ then either $\mathrm{s} = \mathrm{t}$ or $\rho_e(\mathrm{t}, \mathrm{i}) = \mathrm{t}$.

At this point we introduce a lemma which we shall need later in our correctness proof.

Lemma 4.3 *Let σ^l be an infinite run of $\mathcal{A}_l$ on the extended model $\mathcal{M}^f$, and let σ^e be the (unique) run of $\mathcal{A}_e$ on $\mathcal{M}^f$. Then for all $i > 0$, $\sigma^e_1(i) \subset \sigma^l_\pi(i)$.*

Proof. The proof is by induction on the index i of the runs σ^e and σ^l, and follows easily from the transition function of $\mathcal{A}_e$ and the fact that for every $i \in \omega$, $\sigma^l_\pi(i) = \mathcal{M}^f_\pi(i)$. ■

[6] A pair t is denoted $\langle \mathrm{t}_0, \mathrm{t}_1 \rangle$. Note that the initial state I cannot be reached from any state, so we shall feel free to assume that every reachable state has the pair structure.

Model-Checking Automaton

The so-called model-checking automaton $\mathcal{A}_m(f)$ is as usual the product $\mathcal{A}_l(f) \times \mathcal{A}_e(f)$ of the local and eventuality automata represented by $\mathcal{A}_m(f) = \langle 2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}, \mathbf{H}(f) \times (\{I\} \cup (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)})), \rho_l \times \rho_e, \mathbf{N}(f) \times \{I\}, \mathbf{H}(f) \times (2^{\mathcal{P}} \times \{\emptyset\}) \rangle$ where $(\rho_l \times \rho_e) : (\mathbf{H}(f) \times (\{I\} \cup (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)}))) \times (2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}) \rightarrow 2^{\mathbf{H}(f)} \times (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)})$ is defined by

$$(\rho_l \times \rho_e)(\langle \mathrm{s}_l, \mathrm{s}_e \rangle, \mathrm{i}) = \rho_l(\mathrm{s}_l, \mathrm{i}) \times \{\rho_e(\mathrm{s}_e, \mathrm{i})\}$$

The decision procedure now follows by simply checking for the existence of an infinite accepting path in the product automaton's state graph. The formula f is satisfiable if and only if such a path exists. A satisfying model is then simply the ω-sequence of c-states corresponding to the input labels on such a path. This sequence is actually a sequence of Hintikka pairs, but as we show in Section 4.2, the restriction of the first component of each extended c-state in the sequence to $\mathcal{P}$ gives us the required satisfying model.

We thus have our main theorem whose proof follows from the decidability of the emptiness problem for Büchi Automata, and from the Completeness and Soundness Lemmas (Lemmas 4.8 and 4.10), of Section 4.2.

Theorem 4.4 [DECISION PROCEDURE] *It is decidable whether a TLA formula f is satisfiable.*

Our decision procedure also gives us a small model theorem for TLA, which states that if a formula of TLA is satisfiable, then it is satisfiable in a model which has a small (exponential) representation.

4.1 Complexity

Let f be a TLA formula of size n and depth k. By Lemma 3.2, $\mathrm{card}(\mathbf{scl}(f)) = O(n)$. Therefore, the number of states in each of $\mathcal{A}_l$ and $\mathcal{A}_e$ is at most $2^{O(n)}$ and, therefore, so is the number of states in their product $\mathcal{A}_m$. Thus, the construction of the state transition tables for the automata may be done in $2^{O(n)}$ time, since the construction of a transition table requires checking each state against every other, and each state has $O(n)$ formulæ, each of size at most n. Since checking for emptiness of the language of a Büchi automaton is polynomial in the number of states of the automaton, the complexity of the decision procedure is $2^{O(n)}$. Thus TLA is decidable in DEXPTIME.

In fact, it is easy to prove using standard techniques, the following stronger result, the proof of which appears in [10].

Theorem 4.5 *The satisfiability problem for TLA is PSPACE-complete with respect to polynomial time reductions.*

It is easy to adapt the algorithm we gave above to obtain a model-checking algorithm for the logic, with the same complexity with respect to the input formula and linear in the size of the input model.

4.2 Proof of Correctness of the Decision Procedure

Completeness

The strategy for the completeness proof that follows is quite standard. We show that the extension with respect to f of any satisfying model for f will be accepted by $\mathcal{A}_m$. In Lemmas 4.6 and 4.7, $\mathcal{M}^f$ denotes the extension with respect to f of a satisfying model $\mathcal{M}$ for f. $\mathcal{A}_l$ and $\mathcal{A}_e$ represent, respectively, the local and eventuality automata for f as described earlier.

Lemma 4.6 *$\mathcal{A}_l$ accepts $\mathcal{M}^f$.*

Proof. Since all states of $\mathcal{A}_l$ are accepting, we need only show that there is an infinite run of $\mathcal{A}_l$ that consumes $\mathcal{M}^f$.

By Lemma 3.5 any state in $\mathcal{M}^f$ is a state of $\mathcal{A}_l$. Moreover, by construction, $\mathcal{A}_l$ can consume the input symbol $\mathcal{M}^f(n)$ iff it is in the state $\mathcal{M}^f(n)$. Thus, if there is an infinite run consuming $\mathcal{M}^f$, that run must be unique. That there is an infinite run follows by induction on the length of the run.

For the base case, since $\langle \mathcal{M}, 0\rangle \models f$, we have $f \in \mathcal{M}^f(0)$, which is, therefore, an initial state of $\mathcal{A}_l$.

For the induction step we only need to show that $\mathcal{M}^f(n+1) \in \rho_l(\mathcal{M}^f(n), \mathcal{M}^f(n))$. This follows easily from a case analysis of the transition conditions for $\mathcal{A}_l$. As a sample case, take Clause 4 of the transition function of $\mathcal{A}_l$. Assume that $\langle \mathcal{M}^f(n), \mathcal{M}^f(n+1)\rangle$ is not a stuttering pair, and $[f_1'] \in \mathcal{M}^f(n)$. The semantics (and our definition of extension) immediately give $f_1 \in \mathcal{M}^f(n+1)$. We can show similarly that each of the clauses is satisfied. ■

Lemma 4.7 *$\mathcal{A}_e$ accepts $\mathcal{M}^f$.*

Proof. Because $\mathcal{A}_e$ is deterministic and complete, as we observed before, there is a unique infinite run of $\mathcal{A}_e$ that consumes $\mathcal{M}^f$, call it σ. If σ is not accepting, then either there is no $i > 0$ with $\sigma_1(i) = \emptyset$, or there is a largest such i. In the first case, let $i = 0$. Moreover, as we observed earlier, any transition of $\mathcal{A}_e$ that takes it from one state s, with $s_1 \neq \emptyset$, to a state t, where $t_1 \neq s_1$, is such that $t_1 \subset s_1$. By the finiteness of $\mathbf{E}(f)$, there is some $k > i$, such that for all $j \geq k$, $\sigma(j) = \sigma(k)$. Thus, there is some formula $\neg\Box f_1 \in \sigma_1(j)$ for all $j \geq k$, or $\neg[\mathsf{false}] \in \sigma_1(j)$ for all $j \geq k$. In the following we consider only the latter case, since the former is essentially the same as in the case of PTL with 'always'.

By the third condition on ρ_e, therefore, $\mathcal{M}_\alpha^f(j) = \emptyset$, $\neg[\mathsf{false}] \in \sigma_1^e(j)$ and $\sigma_1^e(j) = \sigma_1^e(j+1)$ for all $j \geq k$. Using Lemma 4.3, $\neg[\mathsf{false}] \in \sigma_\pi^l(j)$, where σ^l represents the accepting run of $\mathcal{A}_l$ on $\mathcal{M}^f$. Moreover, because of the definition of $\mathcal{A}_l$ and $\mathcal{A}_e$, it follows that $\sigma_\alpha^l(j) = \emptyset$ and $\sigma_\pi^l(j) \cap \mathcal{P} = \sigma^l(j+1) \cap \mathcal{P}$ for all $j \geq k$. Therefore, for all $j \geq k$, we have $\langle \mathcal{M}, j\rangle \models \neg\mathsf{false}$, $\mathcal{M}(j) = \mathcal{M}(j+1)$ and $\mathcal{M}_\alpha(j) = \emptyset$ for all $j \geq k$. The semantics then yield an immediate contradiction. ■

The completeness of our decision procedure follows immediately from Lemmas 4.6 and 4.7, and the definition of $\mathcal{A}_m$.

Lemma 4.8 [COMPLETENESS] *For any formula f, if f is satisfiable, then the language accepted by $\mathcal{A}_m(f)$ is nonempty.*

Soundness

We want to show that given a string in the language of $\mathcal{A}_m$ we can construct from it a satisfying model for f. Accordingly, let σ be a string accepted by $\mathcal{A}_m$ and define the sequence $\mathcal{M} \in (2^{\mathcal{P}} \times 2^{\mathcal{A}})^\omega$ by

$$\begin{aligned} \mathcal{M}_\pi(i) &= \{P \mid P \in \sigma_\pi(i) \cap \mathcal{P}\} \\ \mathcal{M}_\alpha(i) &= \sigma_\alpha(i) \end{aligned}$$

We first prove the following stronger lemma.

Lemma 4.9 *For any $i \in \omega$ and $f_1 \in \mathrm{scl}(f)$, $f_1 \in \sigma_\pi(i)$ iff $\langle \mathcal{M}, i\rangle \models f_1$.*

Proof. The proof is for an arbitrary i by induction on the inclusion order induced by scl on the formulæ in $\mathrm{scl}(f)$.[7] The base case of primitive propositions follows by construction, and the case of primitive action formulæ is also straightforward.

For the induction step we consider the sample case of $f_1 = [f_2']$, the other cases following along rather similar lines.

For the forwards direction, let $[f_2'] \in \sigma_\pi(i)$. We consider two cases: either $[\mathsf{false}] \in \sigma_\pi(i)$ or $[\mathsf{false}] \notin \sigma_\pi(i)$. For the case of $[\mathsf{false}] \in \sigma_\pi(i)$, by the induction hypothesis $\langle \mathcal{M}, i\rangle \models [\mathsf{false}]$, whence the semantics immediately yield $\langle \mathcal{M}, i\rangle \models [f_3]$ for any f_3, and in particular for $f_3 = f_2'$. For the case of $[\mathsf{false}] \notin \sigma_\pi(i)$, since $\sigma(i)$ is Hintikka, $\neg[\mathsf{false}] \in \sigma_\pi(i)$, whence the induction hypothesis and the semantics yield the existence of a smallest $k \geq i$, such that $\mathcal{M}(k) \rightarrow \mathcal{M}(k+1)$ is a non-stuttering transition. The transition conditions of $\mathcal{A}_l$ ensure that $[f_2'] \in \sigma_\pi(k)$ at least until that k. Moreover since $\mathcal{M}(k) \rightarrow \mathcal{M}(k+1)$ is a non-stuttering transition, it follows by construction that $\langle \sigma(k), \sigma(k+1)\rangle$ is a non-stuttering pair. The transition conditions for $\mathcal{A}_l$ then ensure that $f_2 \in \sigma_\pi(k+1)$, whence the induction hypothesis yields $\langle \mathcal{M}, k+1\rangle \models f_2$, so that $\langle \mathcal{M}, i\rangle \models [f_2']$.

The backwards direction is also similarly straightforward. ■

Since $f \in \sigma_\pi(0)$, as a corollary to the above lemma, we immediately have

Lemma 4.10 [SOUNDNESS] *For any formula f, if the language accepted by $\mathcal{A}_m(f)$ is nonempty, then f is satisfiable.*

[7] Consider the (irreflexive) partial order relation $\sqsubset$, between formulæ, defined by $f_1 \sqsubset f_2$ iff $\mathrm{scl}(f_1) \subset \mathrm{scl}(f_2)$.

5 Handling Abstraction

We show how the decision procedure we gave earlier may be simply extended to handle arbitrary abstraction subscripts. Recall that in our previous syntax $[a]$ was a shorthand for $[a]_{(\mathcal{P},\mathcal{A})}$, and meant that the action a holds on the next non-stuttering transition. Relativizing stuttering now to arbitrary subsets $\mathsf{P} \subseteq \mathcal{P}$ and $\mathsf{A} \subseteq \mathcal{A}$, the formula $[a]_{(\mathsf{P},\mathsf{A})}$ is true precisely when a holds on the next transition across which some proposition in P changes value or some action in A executes. In effect, the formula abstracts away from all the intermediate transitions which do not modify the values of the propositions in P and across which no action in A occurs. This ability to abstract away from the details of the computation of a system is, in some sense, equivalent to *hiding* and provides a succinct and powerful mechanism for doing hierarchical proofs of correctness of systems, and (together with logical implication) of proving that one system implements another. For a detailed discussion of this and other related issues, we refer the reader to, for instance, [7, 1, 3]. In particular, in [3] the construct is used to telling effect for succinctly expressing transition invariants.

In the remainder if this section, we first give a syntax and semantics for abstraction, then present the modifications necessary to the constructions given earlier. While we do motivate these modifications with informal explanations, we do not here provide a complete proof of correctness of the modified decision procedure. It is easy to see, however, that all our earlier theorems and lemmas remain valid for the language with abstraction and that the proofs now follow essentially along the lines of those we gave earlier.

5.1 Syntax and Semantics

The syntax now consists of using action brackets subscripted with a pair (P,A), with $\mathsf{P} \subseteq \mathcal{P}$ and $\mathsf{A} \subseteq \mathcal{A}$ indicating the subset of state propositions and primitive actions which are observable.[8] In the grammar for wff, the following obvious modification affects this extension:

$$f \quad ::= \quad P \mid \neg f \mid f_1 \wedge f_2 \mid \Box f \mid [a]_{(\mathsf{P},\mathsf{A})}$$

where $P \in \mathcal{P}$, $\mathsf{A} \subseteq \mathcal{A}$, $\mathsf{P} \subseteq \mathcal{P}$.

For the semantics, only the rule for action-bracketted formulæ requires modification. The remaining cases stay unchanged for the larger language.

[8] Naturally, one may require for $[f]_{Abs}$ that any propositions or actions mentioned in f should appear in Abs, since formulæ which violate this requirement would be quite unintuitive and probably not very useful in practice. However, in the interests of generality, we do not make this restriction. We note, however, that even if one were to make this restriction, the resulting syntax would still be closed under our subformula formation rules, so that the logic would retain the desirable subformula property. The complexity of the decision procedure is also independent of such a restriction.

- $\langle \mathcal{M}, i \rangle \models [a]_{(\mathsf{P},\mathsf{A})}$ iff either,
  - for all $j \geq i$, $\mathcal{M}_\pi(i) \cap \mathsf{P} = \mathcal{M}_\pi(j) \cap \mathsf{P}$ and $\mathcal{M}_\alpha(j) \cap \mathsf{A} = \emptyset$, or
  - there exists $k > i$, such that $\mathcal{M}_\pi(i) \cap \mathsf{P} \neq \mathcal{M}_\pi(k) \cap \mathsf{P}$ or $\mathcal{M}_\alpha(k-1) \cap \mathsf{A} \neq \emptyset$ such that $\langle \mathcal{M}, k-1 \rangle \| \models a$ and, for all $i < j < k$, $\mathcal{M}_\pi(i) \cap \mathsf{P} = \mathcal{M}_\pi(j) \cap \mathsf{P}$ and $\mathcal{M}_\alpha(j-1) \cap \mathsf{A} = \emptyset$

5.2 Subformula Closure, Hintikka Pairs and Extension

Subformula Closure

In the definition of subformula closure, Clauses 7,8,9 of Definition 3.1 are modified as follows.

8. if $[a]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$ then $[\mathsf{true}]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$
9. $[a]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$ iff $[\neg a]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$
10. if $[a \wedge b]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$ then $[a]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$ and $[b]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$

It is not difficult to see that the cardinality of the subformula closure remains unchanged. Thus, the complexity of the decision problem also remains essentially unchanged.

Hintikka Pairs

The obvious modification of Definition 3.3 appears below. The definition below assumes the new definition of $\mathrm{scl}(f)$ as given above.

Definition 5.1 [HINTIKKA PAIR WITH ABSTRACTION] A *Hintikka pair for* f is an element $\mathsf{s} = \langle \mathsf{s}_\pi, \mathsf{s}_\alpha \rangle$ of $2^{\mathrm{scl}(f)} \times 2^{\mathcal{A}}$ such that

- s_π satisfies the following conditions
  1. $\mathsf{true} \in \mathsf{s}_\pi$ and, for all $[\mathsf{true}]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$, $[\mathsf{true}]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$
  2. for all $f_1 \in \mathrm{scl}(f)$, $f_1 \in \mathsf{s}_\pi$ iff $\neg f_1 \notin \mathsf{s}_\pi$
  3. for all $f_1 \wedge f_2 \in \mathrm{scl}(f)$, if $f_1 \wedge f_2 \in \mathsf{s}_\pi$ then $f_1 \in \mathsf{s}_\pi$ and $f_2 \in \mathsf{s}_\pi$
  4. for all $\Box f_1 \in \mathrm{scl}(f)$, if $\Box f_1 \in \mathsf{s}_\pi$ then $f_1 \in \mathsf{s}_\pi$
  5. for all $[a]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$, $[a]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$ or $[\neg a]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$
  6. for all $[a]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$, if $\neg[a]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$ then $\neg[\mathsf{false}]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$
  7. for all $[a]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$, if $\neg[\mathsf{false}]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$ then $[a]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$ iff $[\neg a]_{(\mathsf{P},\mathsf{A})} \notin \mathsf{s}_\pi$
  8. for all $[a \wedge b]_{(\mathsf{P},\mathsf{A})} \in \mathrm{scl}(f)$, if $[a \wedge b]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$ then $[a]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$ and $[b]_{(\mathsf{P},\mathsf{A})} \in \mathsf{s}_\pi$

- for all $[\mathsf{false}]_{(A,P)} \in \mathbf{scl}(f)$, if $[\mathsf{false}]_{(A,P)} \in s_\pi$ then $s_\alpha \cap A = \emptyset$

The only point worth noting above is that when $s_\alpha \neq \emptyset$ then, unlike before, the contents of s_α are not now uniquely determined by the contents of s_π alone. Essentially the transition context will now supply our automaton with the requisite information to make this choice. This, as we shall see below, will now appear in the local automaton's transition conditions.

Extension

The definition of extension remains unchanged modulo the new definition of $\mathbf{scl}(f)$. Lemma 3.5 goes through with the obvious modifications to its proof.

5.3 Automata Construction and Complexity

We first generalize our definition of stuttering-pairdom by relativizing it with respect to an abstraction subscript: Two elements $s, t \in (2^{\mathbf{scl}(f)} \times 2^{\mathcal{P}})$ constitute a *stuttering pair* $\langle s, t\rangle$ *with respect to* (P, A), where $A \subseteq \mathcal{A}$ and $P \subseteq \mathcal{P}$, iff $s_\alpha \cap A = \emptyset$ and $s_\pi \cap P = t_\pi \cap P$.

With this the modifications required in the construction of $\mathcal{A}_l$ and $\mathcal{A}_e$ are obvious.

Local Automaton

We reproduce below the modified definition of the local automaton $\mathcal{A}_l(f) = \langle 2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}, \mathbf{H}(f), \rho_l, \mathbf{N}(f), \mathbf{H}(f)\rangle$ where

- $\rho_l : \mathbf{H}(f) \times (2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}) \rightarrow 2^{\mathbf{H}(f)}$ is the nondeterministic transition function such that $t \in \rho_l(s, i)$ iff
  1. $i = s$
  2. if $\Box f_1 \in s_\pi$ then $\Box f_1 \in t_\pi$
  3. if $\neg\Box f_1 \in s_\pi$ and $f_1 \in s_\pi$ then $\neg\Box f_1 \in t_\pi$
  4. for all $[f_1']_{(P,A)} \in \mathbf{scl}(f)$, if $\langle s, t\rangle$ is not a stuttering pair with respect to (P, A), then $[f_1']_{(P,A)} \in s_\pi$ iff $f_1 \in t_\pi$
  5. for all $[f_1]_{(P,A)} \in \mathbf{scl}(f)$ such that f_1 is a wff, if $\langle s, t\rangle$ is not a stuttering pair with respect to (P, A), then, if $[f_1]_{(P,A)} \in s_\pi$ then $f_1 \in s_\pi$
  6. for all $[a]_{(P,A)} \in \mathbf{scl}(f)$, if $\langle s, t\rangle$ is not a stuttering pair with respect to (P, A), then $[a]_{(P,A)} \in s_\pi$ iff $[a]_{(P,A)} \in t_\pi$
  7. for all $[A]_{(P,A)} \in \mathbf{scl}(f)$ such that $A \in \mathcal{A}$, if $\langle s, t\rangle$ is not a stuttering pair with respect to (P, A), then $A \in s_\alpha$ iff $[A]_{(P,A)} \in s_\pi$
- $\mathbf{N}(f) = \{ s \mid s \in \mathbf{H}(f), f \in s \}$ is the set of initial states.

The last clause defining the transition function now ensures that the set of actions that occur across the transition are indeed precisely those that are "required" by s_π, something that had been enforced by the Hintikka conditions in the logic without abstraction subscripts.

Eventuality Automaton

The following modified definition takes care of abstraction subscripts. The eventuality automaton is a deterministic Büchi automaton represented by $\mathcal{A}_e(f) = \langle 2^{\mathbf{scl}(f)} \times 2^{\mathcal{P}}, \{I\} \cup (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)}), \rho_e, I, 2^{\mathcal{P}} \times \{\emptyset\}\rangle$ where

- $\mathbf{E}(f) = \{ \neg\Box f_1 \mid \neg\Box f_1 \in \mathbf{scl}(f) \} \cup \left\{ \neg[\mathsf{false}]_{(P,A)} \;\middle|\; \neg[\mathsf{false}]_{(P,A)} \in \mathbf{scl}(f) \right\}$
- $I \notin (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)})$ is the initial state
- $\rho_e : (\{I\} \cup (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)})) \times (2^{\mathbf{scl}(f)} \times 2^{\mathcal{A}}) \rightarrow (2^{\mathcal{P}} \times 2^{\mathbf{E}(f)})$ is the deterministic transition function such that $\rho_e(s, i) = t$ iff
  1. $t_0 = i_0 \cap \mathcal{P}$
  2. if $s = I$ then $t_1 = \{ \neg\Box f_1 \mid \neg\Box f_1 \in i_\pi, \neg f_1 \notin i_\pi \} \cup \left\{ \neg[\mathsf{false}]_{(P,A)} \;\middle|\; \begin{array}{l} \neg[\mathsf{false}]_{(P,A)} \in i_\pi \\ i_\alpha \cap A = \emptyset \end{array} \right\}$
  3. if $s \neq I$ and $s_1 = \emptyset$ then $t_1 = \{ \neg\Box f_1 \mid \neg\Box f_1 \in i_\pi, \neg f_1 \notin i_\pi \} \cup \left\{ \neg[\mathsf{false}]_{(P,A)} \;\middle|\; \begin{array}{l} \neg[\mathsf{false}]_{(P,A)} \in i_\pi \\ i_\alpha \cap A = \emptyset \\ s_0 \cap P = t_0 \cap P \end{array} \right\}$
  4. if $s \neq I$ and $s_1 \neq \emptyset$ then $t_1 = \{ \neg\Box f_1 \mid \neg\Box f_1 \in s_1, \neg f_1 \notin i_\pi \} \cup \left\{ \neg[\mathsf{false}]_{(P,A)} \;\middle|\; \begin{array}{l} \neg[\mathsf{false}]_{(P,A)} \in s_1 \\ i_\alpha \cap A = \emptyset \\ s_0 \cap P = t_0 \cap P \end{array} \right\}$

Complexity

The decision procedure still runs in DEXPTIME, and it can be shown similar to the case for the logic without general abstraction, that the extended logic is still in PSPACE, so that Theorem 4.5 continues to hold for the more general logic.

6 Conclusion

We have presented a decision procedure for a Temporal Logic of Actions which conservatively extends the one introduced by Lamport. Automata-theoretic methods have already yielded substantial benefits in not only obtaining decision methods for new temporal logics, but also helping to unify temporal logic decision procedures under a

powerful common framework. The present work is further evidence of the flexibility and intuitive breadth of this framework.

The work we have presented may be extended in several directions. Firstly, it is possible to extract from the decision procedure a tableau-proof system for PTLA with abstraction. Secondly, using (a slight generalization of) the methods of [8],[9, Chapter 4], the decision procedure presented here may be combined with decidable quantifier-free theories to solve the decision problem for a quantifier-free TLA with assertions from these theories mentioning both rigid and flexible variables. This would be an approach different from the one taken by [6] and [14] in placing greater emphasis on an automatic decision procedure for a very large fragment of full TLA. Thirdly, TLA has been used to verify real-time properties in [3]. It would be interesting to see how far such real-time reasoning could be automated, in view of the recent rapid strides in real-time verification made possible by the real-time counterparts of Büchi Automata devised by Alur and Dill [4].

Another possible direction for future work in TLA might be to extend the notion of abstraction to actions, so that actions primitive at one level may be viewed as appropriate compositions of finer actions at a lower level. For this, the extensive work on action refinement in process algebra might prove useful.

PTLA may be considered an extension of PTL(□)[9] with actions and abstraction. Similarly, using the decision procedure given above, we think that it might be possible to devise a decision procedure for similar extensions of any (stuttering-invariant) temporal logic decidable by reduction to the emptiness problem for Büchi Automata. It would be an interesting exercise to show that this indeed is the case. Moreover, it appears (see, for instance, [9]) that such an extension would be decidable with no complexity penalty if the base logic were already at least PSPACE-hard, a property true of most linear temporal logics in use today.

Finally, there are expressiveness questions[10] regarding (propositional) LTLA and TLA that remain open. While it is easy to show that LTLA is already strictly more expressive than PTL(□), the only non-trivial expressiveness result known to this author is that, barring primitive actions, PTL($\mathcal{U}$) is at least as expressive as the extended version of TLA examined in this paper.

Acknowledgements. I am grateful to Laurie Dillon, Michael Melliar-Smith and Louise Moser, for their comments on an early draft of this paper.

References

[1] Abadi M, Lamport L, *The Existence of Refinement Mappings*, Proc. IEEE LICS 1988, pp 165-175.

[2] Abadi M, *An Axiomatization of Lamport's Temporal Logic of Actions*, Proc. CONCUR, 1990, LNCS 458, pp 57-69; see also TR 65, DEC SRC, October 1990 (revised March 1993).

[3] Abadi M, Lamport L, *An Old-Fashioned Recipe for Real Time*, Proc REX Workshop "Real-Time: Theory in Practice," 1991, LNCS 600, pp 1-27.

[4] Alur R, Dill D, *Automata for Modelling Real-Time Systems*, Proc. 17th ICALP, 1990, LNCS 443, pp 322-335.

[5] Emerson E A, *Temporal and Modal Logic*, in "Handbook of Theoretical Computer Science," Volume B (Formal Models and Semantics), editor J van Leeuwen, The MIT Press, Cambridge, 1990, pp 789-840.

[6] Engberg U, Grønning P, Lamport L, *Mechanical Verification of Concurrent Systems with TLA*, Proc. 4th CAV, 1992, LNCS, pp 48-60.

[7] Lamport L, *The Temporal Logic of Actions*, TR 79, DEC SRC, December 1991; see also TR 57, DEC SRC, April 1990.

[8] Plaisted D, *A Decision Procedure for Combinations of Propositional Temporal Logics and Other Specialized Theories*, J. Automated Reasoning, **2** (1986), pp 171-190.

[9] Ramakrishna Y S, *Interval Logics for Temporal Specification and Verification*, PhD Thesis, Dept of Elec and Comp Eng, University of California, Santa Barbara, 1993.

[10] Ramakrishna Y S, *On the Satisfiability Problem for Lamport's Propositional Temporal Logic of Actions and Some of Its Extensions*, TIFR Technical Report CS-94-1, April 1994.

[11] Vardi M, Wolper P, *Automata-Theoretic Techniques for Modal Logics of Programs*, J. CSS, **32** (2), 1986, pp 183-210.

[12] Vardi M Y, Wolper P, *An Automata-Theoretic Approach to Automatic Program Verification*, Proc. 1st IEEE LICS, 1986, pp 183-221.

[13] Wolper P, *On the Relation of Programs and Computations to Models of Temporal Logic*, Proc. Conf. Temporal Logic in Specification, 1987, LNCS 398, pp 75-123.

[14] von Wright J, Långbacka T, *Using a Theorem Prover for Reasoning about Concurrent Algorithms*, Proc. 4th CAV, 1992, LNCS, pp 61-83.

[9] but no 'next'.

[10] All our observations here are with respect to PTL without the 'next' operator.

Parameterized Evaluation of CTL-X formulae

Vernier Isabelle
Labo. MASI
Université Pierre et Marie Curie
4, place Jussieu
75252 Paris Cedex 05
Email : vernier@masi.ibp.fr
Tel : (33-1) 44-27-71-04

Abstract : Temporal formulae of languages like LTL, CTL-X, CTL, ... may be efficiently verified for systems described by finite automata. However these methods require to fix the different parameters of the system such as the number of processes or resources. Therefore new algorithms must be developed to study parallel programs with parameters. We present such an algorithm. The systems we study can be decomposed into two synchronized automata. One of the two automata represents the behaviour of a finite but unknown number of processes and the other one represents the control of these processes. Many systems can be modelled this way. The verification algorithm is divided into two algorithms. The first algorithm tries to build a parameterized state graph independent of the number of processes. Then, if it ends with success, the second algorithm can evaluate any formula of CTL-X.

Key words : temporal logic, synchronization, reachability graph, parallel programs, model checking, parameterized state graph.

1. Introduction

The study of parallel programs requires to specify and verify parallel behaviours and properties. Temporal logic offers, among others, these two possibilities. Simple temporal logic formulae express the usual properties of parallel programs [Lamport 83], [Emerson 88]. The properties of finite state programs are checked by efficient inspection of the reachability graph [Lichtenstein 85], [Clarke 86], [Emerson 89], [Vergauwen 93].

However parallel and distributed programs usually involve a finite but unknown number of processes communicating through a control structure. Thus the usual verification methods are no more applicable to obtain a complete verification. We need to verify properties on such programs independently of the number of communicating processes. Several authors have already proposed solutions to overcome this problem.

In [Clarke 87] and [Sistla 87], algorithms for temporal model checking concerning programs with many similar processes are introduced. In the first paper, they consider tree logic. The basic idea is to find a sufficient number of processes to have the full behaviour of the program. Then the usual verification methods are applied. However the search of the "sufficient number" is partially automatic. In the second paper, automatic verifications are obtained by restricting either the model or the formulae language. In [Wolper 89], an alternative approach is proposed where induction on formulae is used to verify that a given invariant remains true if the number of processes is increased. There is no way to find such an invariant and to decide if it exists. We must have the intuition of the useful invariant.

Our work is close to the one presented in [Clarke 87]. Our model is composed of two synchronized automata. One represents the behaviour of the processes and the other the behaviour of the control. The model can be instantiated by the number of processes. Then, it represents the program for the fixed number of processes. We give in [Vernier 93.a] an algorithm to directly build a parameterized state graph where each state represents a family of states of an ordinary computation. The construction procedure may fail; however in practice it usually terminates with success. Furthermore this graph is a high-level representation of the behaviour of the program. With this parameterized state graph, we immediately obtain the reachability set for any large enough number of processes. The algorithm presented in this paper is a parameterized temporal logic verification algorithm. We verify CTL-X formulae on the parameterized state graph. We have adapted the decision procedures presented in [Clarke 86] to take into account the specific features of the parameterized states.

The remainder of the paper is organized as follows. In section 2, we present the model we use for parameterized parallel computations. In section 3, we give a general survey of the parameterized state graph building algorithm. Section 4 presents an algorithm to evaluate temporal logic formulae on

the parameterized state graph. We conclude in section 5 and we present the future extensions of our work.

2. A Model for Parameterized Parallel Computations

Many parallel algorithms proposed to ensure critical sections, distributed termination or to establish synchronization states are based on the (symmetrical) definition of a finite but indefinite number of processes communicating by some kind of control structure (shared variables, message passing, ...). These algorithms are supposed to be correct whatever the number of processes. However one seldom find formal proofs of such algorithms. In this section we define a model that is suitable to lead to a formal proof independent of the number of processes.

The model we define is called PPCM (Parameterized Parallel Computations Model). It is composed of two automata: one representing the behaviour of the processes and the other the behaviour of the control structure. The control structure allows access to resources or communication between the processes. A PPCM is a family of parallel programs, one per cardinality of processes. Each automata is defined by a set of states, a set of transitions and two flow relations between states and transitions. Each transition has one source state and one target state. The backward flow relation associates to each transition its source state and the forward flow relation its target state.

Definition 2.1 *PPCM : a Parameterized Parallel Computations Model*

A parameterized parallel computations model P = $<G_p, G_c>$ is defined by:

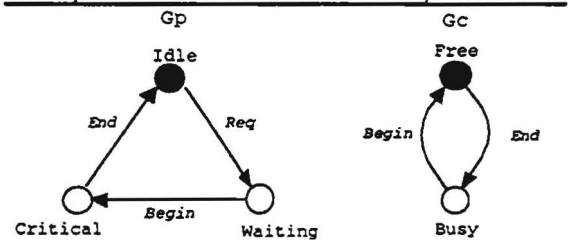
• The process automaton $G_p = <S_p, T_p, \Gamma_p^-, \Gamma_p^+, start_p>$ where :
- S_p is the set of states of a process,
- T_p is the set of transitions of a process,
- Γ_p^- (resp. Γ_p^+) a function from T_p to S_p is the backward (resp. forward) flow relation between states,
- $start_p \in S_p$ is the initial state of a process,

• The control automaton $G_c = <S_c, T_c, \Gamma_c^-, \Gamma_c^+, start_c>$ where :
- S_c is the set of states of the control,
- T_c is the set of transitions of the control,
- Γ_c^- (resp. Γ_c^+) a function from T_c to S_c is the backward (resp. forward) flow relation between states of the control,
- $start_c \in S_c$ is the initial state of the control.

• $S_p \cap S_c = \emptyset$ but $T_p \cap T_c$ is generally non empty and represents the synchronization transitions.

A PPCM does not enable direct synchronization between two processes. However this extension is straightforward.

Example 1: *A critical section between processes*



The states are represented by the circles and the transitions by the arcs. The blackened states are the initial ones. Initially all the processes are idle. A process submits a request and waits until the control authorizes it to begin its critical section. At the end of the critical section, the control can accept new requests. {Begin, End} is the synchronization transitions set.

We now state the definition and the semantics of an instantiated PPCM that is a program with a fixed number of processes. If n is the number of processes, the instantiated PPCM is composed of n copies of G_p and one copy of G_c. The synchronizations can only be executed between one copy of G_p and G_c.

Definition 2.2 *Instantiated PPCM*

Let P = $<G_p, G_c>$ be a PPCM and n be a positive integer (the number of processes), then P[n] = $<S[n], R[n], start[n]>$, the instantiated PPCM, is the parallel program defined by:

• S[n] is the set of possible states defined by: $S[n] = S_p^n \times S_c$.
An item s of S[n] is denoted by a tuple $s = <s_1, ..., s_n, s_c>$. In the rest of the paper an item s is called an ordinary state since it is not parameterized.

• $start[n] \in S[n]$ is the initial ordinary state defined by: $start[n]=<start_p,..., start_p, start_c>$

• R[n] is a relation between ordinary states of S[n] defined by:
s R[n] s' where s, s' $\in$ S[n] if and only if $\exists t \in T_p \cup T_c$ with
(1)
- $t \in T_p \Rightarrow \exists i \leq n$ with $s_i = \Gamma_p^+(t)$ and $s_i' = \Gamma_p^+(t)$ and $\forall j \neq i, s_j = s_j'$
- $t \notin T_p \Rightarrow \forall j \leq n, s_j = s_j'$
(2)
- $t \in T_c \Rightarrow s_c = \Gamma_c^-(t)$ and $s_c' = \Gamma_c^+(t)$
- $t \notin T_c \Rightarrow s_c = s_c'$.

Additional remarks

- S[n] defines the set of possible states but not the set of reachable states. The accessibility set is the transitive closure of the relation R[n] applied to $start_n$.

- R[n] defines the semantics of the program. A step of the program can be done by a process "i", the control or simultaneously both if the transition t belongs to T_p and T_c.

Example 2: *A computation of a parallel program*

Let P be the program of the example 1. Then a possible computation of P[2] is given by:
- From <Idle, Idle, Free> to <Waiting, Idle, Free> using transition Req activated by process 1
- From <Waiting, Idle, Free> to <Waiting, Waiting, Free> using transition Req activated by process 2
- From <Waiting, Waiting, Free> to <Critical, Waiting, Busy> using transition Begin activated by process 1 and the control
...

The aim of this paper is to check, when possible, properties of the programs P[n] for all values of n. At first we look at the reachability problem and this is the subject of the next section.

3. Parameterized State Graph

To be sure that a program, modelled by a PPCM, verifies some properties expressed as temporal logic formulae, we need first to build its reachability graph.

To study parameterized models, we have to build a parameterized reachability graph that represents all the reachable states and all the possible computations of the model. The graph must not represent particular cases of behaviour.

In many cases, the behaviour of the programs P[n] may be particular for some but a finite number of values of n. Mutual exclusion programs for n processes may have a special behaviour if there is only one process. We want to study the program when the value of n does not represent a special behavioural case. Usually, there is a minimum necessary number of processes to have no special behaviour of the program. This minimal bound is computed during the building of the parameterized state graph. As we do not represent special behaviours in the graph, we forbid the executions of transitions that imply a value assignment to n.

We briefly present here the features of the parameterized state graph. In [Vernier 93.a], the parameterized state graph building algorithm is described.

3.1 Parameterized States

Each parameterized state of the reachability graph represents a set of ordinary states for any P[n]. The key point of the representation is to associate an integer called a marking and a relational operator "=" or "≥" to each state of S_p. The intended meaning is the following:

- If "=x" is associated to a state, then exactly x processes are in this state.
- If "≥x" is associated to a state, then at least x processes are in this state.

These attributes are also used for the states of S_c. As there is only one control process we require that:

- only equality can be used,
- the marking can be either 0 or 1,
- there is exactly one state for which the marking is 1.

The used notation is close to the one presented in [Emerson 90]. In this paper, they are focusing on testing satisfiability rather than model checking.

The two attributes are defined by two functions associated with a parameterized state. If ps is a parameterized state then ps.OP and ps.M denote respectively the operator and the marking functions.

Notations

- We denote a parameterized state by $\{s\ OP(s)\ M(s)\}_{s \in S_p \cup S_c}$, forgetting all the states s for which OP(s) is "=" and M(s) is 0. Moreover the only marked state of S_c is represented without its relation and its marking (necessarily equal to "=" and 1).

- When necessary, we prefix M and OP by the parameterized state like for instance ps.OP(s).

Example 3: *Parameterized states for the critical section model*

The initial parameterized state of the critical section model is given by: {Idle≥1, Free} which denotes that the control is in state Free and all the processes are in state Idle.
The following parameterized state denotes that one process is in its critical section, two others are waiting for the critical section and the rest of the processes are in state idle: {Idle≥1, Waiting=2, Critical=1, Busy}.

The relation between ordinary and parameterized states is intuitive. An ordinary state belongs to a parameterized one if the distribution of processes among states fulfils the relations expressed by the parameterized state. A parameterized state ps is included in an other one, ps', if and only if the set of ordinary states represented by ps is included in

the set of ordinary states represented by ps'. This is noted ps $\subseteq$ ps'. If ps is not equal to ps', then ps $\subset$ ps'.

Example 4: *Correspondence between states*

Let ps be the following parameterized state:
{Idle≥1,Waiting=2,Critical=1,Busy} then in P[5],
os = <Idle, Waiting, Waiting, Critical, Idle, Busy> belongs to ps.

In the previous example if we consider P[3], ps represents none of its ordinary states. This correspondence between states shows that a parameterized state, ps, represents a set of ordinary states of P[n], {os}, only if n is greater than a bound defined by M. This bound is equal

to $\sum_{s \in Sp} M(s)$.

3.2 Parameterized Transitions

To build the parameterized reachability graph, we need to define parameterized transitions between parameterized states. This definition must be consistent with the definition of a transition between ordinary states. A transitions can be executed from a parameterized state ps if and only if it can be executed from each ordinary state represented by ps. For each ordinary state represented by ps, its successor state, after the execution of a transition t, is represented by one successor parameterized state of ps, after the execution of the same transition. If ps' is a successor of ps by transition t, then for each ordinary states, os', represented by ps', there is an ordinary state, os, represented by ps such that os' is the successor of os by transition t. Roughly speaking, a parameterized transition from a parameterized source state to a parameterized target state is possible if:

- there is enough marking in the source state.

- the target state is obtained by consistent decreases and increases.

Let ps and ps' be two parameterized states. If we want to execute the transition Req from parameterized state ps, there must be at least one process in state Idle, e.g., ps.M(Idle) ≥ 1. If ps' is the reached parameterized state when the transition Req is executed then ps'.M(Idle) = ps.M(Idle)-1 and ps'.M(Waiting) = ps.M(Waiting)+1 since by transition Req one process moves from state Idle to state Waiting.

However when a state s fulfilling s ≥ 1 is decreased, two cases may happen:

- only one process was in state s, then in the target state one have s = 0. As the number of processes is parameterized, this is possible only if another state s' fulfils s'≥j. Otherwise it would assign a value to the parameter n.

- at least two processes were in state s, then in the target state one have s ≥ 1.

The parameterized transitions include these two cases that can simultaneously happen.

Example 5: *Transitions between states*

Let ps be the following parameterized state:
{Idle≥1, Waiting≥2, Free}
Then ps leads to ps_1 by transition begin where:
ps_1 = {Idle≥1, Waiting≥1, Critical=1, Busy}.
Also ps leads to ps_2 and ps_3 by transition Req where:
ps_2 = {Idle≥1, Waiting≥3, Free} and ps_3 = {Waiting≥3, Free}.

3.3 Building of the Parameterized State Graph

Since we are given an initial parameterized state and a transition rule, it should be easy to develop an accessibility tree (or graph). However such a structure would be generally boundless. Thus we must take some decisions during the building of the tree. The procedure avoiding the infinite development is similar to the one of Karp and Miller [Karp 69] [Finkel 90] and we just give here an informal presentation.

Let ps and ps' be two parameterized states, ps ≤ ps' if for each place p:

- ps.M(p) ≤ ps'.M(p) and

- ps.OP(p) ≤ ps'.OP(p) (we state that "=" < "≥").

As soon as we find two states ps and ps' on the same branch of the reachability tree such that ps ≤ ps', three cases may happen:

(a) ps = ps', so we stop the development of this branch.

(b) ps ≠ ps' and the sequence of transitions that leads from ps to ps' is identifiable as "a state change from s to s' for one process", then we transform ps' in ps except for s' where ps.OP(s') is set to "≥". Moreover if ps.M(s')=0 then ps'.M(s)=1.

(c) otherwise we stop the whole building : this is the failure case of the algorithm.

A sequence of transitions is "a state change from s to s' for one process" if after its execution:

- one process has moved from s to s',
- all the other processes and the control are in the same state.

The change of operator "=" to "≥" is justified by the fact that this sequence can be iterated as long as there are processes in state s. After each execution of the sequence, a process has moved

from the state s to s'.

In practice, case(c) is rarely encountered. It is mainly met when the execution of sequences is dependent on some properties of the parameter, e.g., if the behaviour of the algorithm is different when the number of processes is odd or even. In such cases, we can not represent in a single graph the behaviour of the algorithms whatever the number of processes.

Let us show how the building works on one branch.

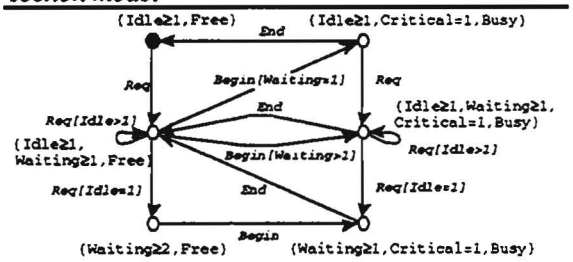
Example 6: *A branch of the accessibility tree*

We start from the initial parameterized marking
pstart = {Idle≥1, Free}
We can use transition Req and obtain
$pstemp_1$ = {Idle≥1, Waiting=1, Free }
$pstemp_1$ is greater than pstart and we can identify Req to a process change from Idle to Waiting so we change $pstemp_1$ to:
ps_1 = {Idle≥1, Waiting≥1, Free}
We can use the transition Req and obtain
$pstemp_2$ = {Idle≥1, Waiting≥2, Free}
Again we transform $pstemp_2$ but this time $ps_2 = ps_1$ so we stop the building of this branch.

We do not compute the successor states of a state ps if there exists a parameterized state ps', already computed, such that ps is included in ps'. While computing the successor states of ps' we also compute those of ps.

The termination of our algorithm is now ensured by the order properties of $\mathbb{N}^k$. So if we get success we identify the nodes ps with ps' if ps $\subset$ ps'. Therefore, we obtain a parameterized state graph.

Example 7: *The parameterized graph of the critical section model*

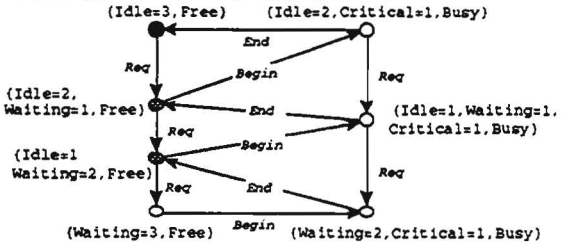


This parameterized state graph represents the reachability states of P[n] with $n \geq 3$.

Let P be a PPCM, let G be the parameterized graph of P (if the building was successful). G represents the behaviour of P[n] if and only if n is greater than n_0 where n_0 is the maximum of $\Sigma ps.M(s)$ for ps $\in$ G and s $\in S_p$. Therefore, for each $n \geq n_0$, G can be unfolded to G[n], the state graph of P[n].

Example 8: *Unfolding of the parameterized state graph for n=3: G[3]*

The gray states are represented by the same parameterized state {Idle≥1, Waiting≥1, Free} in the parameterized state graph of the example 7.



For each execution sequence of G[n], $n \geq n_0$, there is an equivalent possible sequence in G. For each possible sequence of G, there is an equivalent sequence in G[n], $\forall n \geq n_0$. Therefore, the result of the verification of properties on G is valid for each G[n], $n \geq n_0$. The proof is given in [Vernier 94].

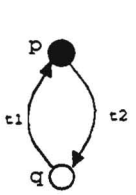
3.4 Detection of "real" circuits

An important feature of the CTL formulae evaluation algorithms is the detection of "real" circuits. They allow to find executions that may infinitely be repeated. By "real" circuits we mean circuits that appear in the graphs G[n]. While the computation of the parameterized state graph, the built circuits are not always "real".

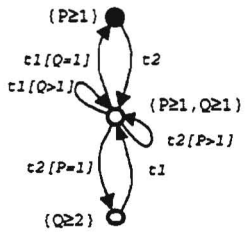
On the graph of the example 7, all the "real" circuits are already computed.

Example 9: *Parameterized state graph without "real" circuits*

The following PPCM represents the behaviour of processes that can be in two states and move from one to the other without constraints.



PPCM



Parameterized State Graph

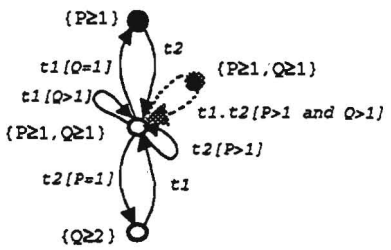
The two circuits of the graph are t1 and t2 from the parameterized state {P≥1,Q≥1} to itself. They can be executed only when P>1 and Q>1. A real circuit can be composed of these two transitions and it corresponds to the execution of the transition t1 (resp. t2) a λ_1 (λ_2) number of times. The system of equations we have to solve represents the change of the number of processes in the places P and Q when the supposed circuit is executed. If there is a

strictly positive integer solution then there is a real circuit that leads from {P≥1,Q≥1} to itself. The system of equations is:

state P: $\lambda_1 - \lambda_2 = 0$

state Q: $\lambda_2 - \lambda_1 = 0$

This equation has a solution. Therefore, there is a "real" circuit, holding t1, that leads from {P≥1,Q≥1} to itself. This circuit can be executed only if P>1 and Q>1. The parameterized state graph with real circuits is as follows :



Parameterized State Graph with "real" circuits

The set of the real circuits can be computed from the initial circuits by solving positive integer equations [Ginsburg 64].

4. Parameterized Evaluation of CTL-X Formulae

Everyone that models a system has some ideas of the supposed properties of the system. To express these properties, temporal logic formulae are an appropriate language [Lamport 83], [Emerson 88]. We have chosen CTL-X.

CTL-X is a temporal logic reasoning about states and their successors. The existing algorithms are depth-first search algorithms [Clarke 86], [Vergauwen 93]. They are based on the following principle: they do not iterate the evaluation procedures on nodes of the graph that have already been visited. It is useful to visit one time a node to know if it verifies a formula. This last property is not respected for the nodes of the parameterized state graph. Each parameterized state defines a set of ordinary one. Therefore, the examination of new successors of an already visited parameterized state may allow to detect new ordinary states that verify the formula. To solve this problem, the algorithm attaches to each node a condition that defines the subset of ordinary states that fulfil the formula. When visiting a node, the subset is kept up to date. If it is stable, we do not continue the examination from this node.

4.1 CTL-X

We have chosen a branching time logic since it is more adapted as a linear to the structure of the state graph and to the properties we want to verify [Pnueli 85], [Emerson 86].

Although CTL-X has a more restrictive power than other languages (e.g., CTL*), we have chosen it for the following reasons:

• The verification of a CTL-X formula on a state graph has a complexity linear in the size of both the formula and the graph.

• Although CTL-X is a simple language, it is sufficient to express the usual properties such as safeness (i.e. a property f must be verified at each reachable state which is written ($\forall G$ f) in CTL-X) and liveness (i.e. from each state which verify g we can always reach a state that verify h which is written ($\forall G(g \Rightarrow \forall F\ h)$) in CTL-X).

The version of CTL we use is named CTL-X and a formula is equal to:

- one of the two constants: True or False,
- a basic formula,
- a formula obtained by connectors And, Or, Not applied to formulae of CTL-X,
- a formula obtained by connectors $\forall G$, $\forall F$, $\forall U$, $\exists G$, $\exists F$, $\exists U$ applied to formulae of CTL-X.

The basic formulae define the set of properties that can be verified at a parameterized state without inspection of its successors. Therefore they have the following syntax and semantics:

- {s ≥ 1}, means that at least one process is in state s,
- {s = n}, means that all the identical processes are in state s,
- {s op j}, $j \in \mathbb{N}$ and op {=, >} means that the number of processes in state s respects the relation "op j".

A parameterized state ps verifies:

- {s ≥ 1} only if ps.M(s) ≥ 1,
- {s = n} only if ps.OP(s) = "≥" and $\forall s' \neq s$, ps.M(s')=0,
- {s op j} only if
  ps.OP(s) = "=" and ps.M(s) op j or
  ps.OP(p) ="≥" and $j+\sum ps.M(p') \leq n_0$, this condition ensures that the formula is verified whatever the possible value of the parameter is.

The verification, at a parameterized state ps, of a formula with temporal operator depends on its successors. The next paragraph states the semantics of the temporal operators.

- $\forall$ means for any path starting from the current state...
- $\exists$ means there is some path starting from the current state...
- Gp means ...such that p is true for all states of the path
- Fp means ...such that p is true for some state of the path
- pUq means ...such that p is true on the path until q becomes true and q will eventually becomes true

The operators G and F are abbreviations of formulae written with the U operator.

We do not consider the X (successor state) operator since it allows to count the number of processes. Let F be the formula $\forall G(Idle \geq 1 \Rightarrow \exists X(Idle=0))$. F is verified if each time there is a process in state Idle, one of the successor states is such that there is no Idle process. Considering our example, F can be verified only if there is one process. Such formulae may represent special behaviours of the programs.

Example 10: *Some simple formulae about the critical section model*

Two processes are never simultaneously in critical section:
$\forall G$ ({Critical $\leq$ 1})
Requests imply services:
$\forall G$({Waiting $\geq$ 1} $\Rightarrow \forall F$({Critical $\geq$ 1})

4.2 Conditions on parameterized states

It may possibly be that two ordinary states, held by the same parameterized state, do not verify the same formulae. This case may happen only if from one ordinary state it is possible to empty a state while from the others it is not possible. Consider the parameterized state {Idle≥1, Waiting≥1, Free}, the ordinary states such that Idle>1 may not verify the same formulae as those where Idle=1. The condition, Cond(ps,f), defines the subset of ordinary states, included in ps, that verify f. The subsets can not be differentiated by the marking of the control states. The conditions concern only the states of the processes automaton. A condition is:

- an elementary condition Cond(ps,f) = Ce where
  - Ce = All (all the ordinary states included in ps verify f) or
  - Ce = None (no ordinary state included in ps verify f) or
  - Ce is a pseudo parameterized state where the operator associated to each place can be "=", ">" or "≥". In a condition, the states that do not appear are not constraint. The number of processes in these states depends only of the parameterized state with which the condition is associated. If ps={Idle≥1, Waiting≥1, Free} and Ce(ps,f)={Idle>1} then the ordinary states, included in ps, with more than one process in state Idle verify f. We can extend Ce(ps,f) to {Idle>1, Waiting≥1, Free}. A condition Ce(ps,f) such that $\sum Ce(ps,f).M(p) > n_0$ is not compatible with the parameterized state graph. Otherwise, Ce is not equal to None does not mean that for each parameter value, there is at least one ordinary state represented by ps that verifies the formula f. We want to test if a distribution of processes verifies a formula whatever the value of the parameter is.
- an union of elementary conditions

  Cond(ps,f) = $\cup Ce_i$ (if $\cup Ce_i$=ps then Cond(ps,f)=All).

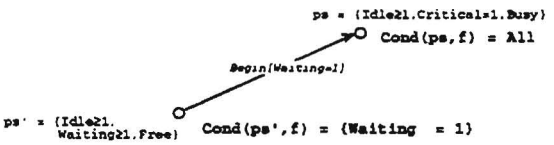
For the verification algorithm, we need to know how to compute the set of ordinary states included in ps that allow to reach the ordinary states of ps' that verify a condition, Cond, when ps$\xrightarrow{t}$>ps'. This set, Trans(Cond,ps,t,ps'), is a condition defined as follows and is only computed when Cond ≠ None:

- let s and s' be the two states of S_p such that $\Gamma_p^-(t)=s$ and $\Gamma_p^+(t)=s'$,
- Cond = All then Trans(Cond,ps,t,ps') = All
- Cond = $\cup Ce_i$ and
  Trans(Cond,ps,t,ps') = $\cup$ (ps $\cap$ C'e_i) where:
  - $\forall Ce_i$
    if s appears in Ce_i then C'e_i.M(s) = Ce_i.M(s)+1, C'e_i.OP(s) = Ce_i.OP(s),
    if s' appears in Ce_i then C'e_i.M(s) = Ce_i.M(s)-1, C'e_i.OP(s') = Ce_i.OP(s'),
    $\forall s'' \notin \{s,s'\}$, C'$e_i$.M(s") = Ce_i.M(s"), C'e_i.OP(s") = Ce_i.OP(s"),
- if t empties s then Trans(Cond,ps,t,ps') = Trans(Cond,ps,t,ps') $\cap$ {s=1},
- if t does not empty s then Trans(Cond,ps,t,ps') = Trans(Cond,ps,t,ps') $\cap$ {s>1}.

The set $Trans^{-1}$(Cond,ps,t,ps') defines the ordinary states of ps' that are reached by ordinary states of ps that verify Cond when ps$\xrightarrow{t}$>ps'. We just notice that Trans(Trans^{-1}(Cond,ps,t,ps'),ps,t,ps')=Cond.

Example 11

We consider the following restriction of the parameterized state graph of the example 7 and the formula
f = $\exists$(True U {Waiting=0}).



The parameterized state ps verifies the basic formula {Waiting=0} therefore Cond(ps,f)=All. When we restrict to this part of the graph Cond(ps',f) = Trans(All,ps',Begin[Waiting=1],ps) $\cap$ All = {Waiting=1} $\cap$ All = {Waiting=1}. All the ordinary states of ps' such that one process is in state Waiting verify f.

4.3 Algorithm

The algorithm to evaluate CTL-X formulae on a parameterized state graph is similar to the one presented in [Clarke 86]. We have defined new procedures to evaluate formulae with temporal operators as $\forall[f_1 \, U \, f_2]$ and $\exists[f_1 \, U \, f_2]$. For place constraints, we only present the procedure to evaluate formulae as $\forall[f_1 \, U \, f_2]$ while f_1 and f_2

have already been evaluated.

With each parameterized state and formula is associated a condition that defines the subset of ordinary states that verify the formula. We use an array of conditions (Marked) that associates with each parameterized state the set of ordinary markings that have already been visited. We also use a stack (ST) to store the parameterized states visited along a path and the path. When we push a parameterized state, we also push the transition that leads to it (if there is one). This way it is possible to know if some parameterized states are linked by a circuit ("real" or "false"). The function RealCircuit(ST,ps) returns the value true if the circuit between ps and ps is or hides a "real" one. The function Stacked(ST,ps) returns the value true if ps is in the stack ST.

```
procedure AU(f)
{f = ∀[f1 U f2]}
begin
   for all ps ∈ G do
      begin
         Marked(ps) := Cond(ps,f2);
(a)      Cond(ps,f) := Cond(ps,f1)∪Cond(ps,f2)
         Modif(ps) := false;
         Liste(ps) := Ø;
      end
   for all ps ∈ G do
      if Marked(ps)<>All then VerifAU(f,ps,All)
end
```

The procedure VerifAU(f,ps,ens) checks the formula f at ordinary states of ps that verify the condition ens. While the execution of this procedure, the ordinary states included in ps that verify f_1 and not f_2 are deleted from Cond(ps,f). When the procedure ends, then Cond(ps,f) ∩ ens gives the ordinary states of ens that verify f. Modif(ps) indicates if the condition associate with ps has to be modified. A modification is due to the detection of a state change sequence, from p' to p", that leads from ps to ps. The set Liste(ps) holds the places p' of the state change sequences that have been detected.

```
VerifAU(f,ps,ens)
begin
   if (ens ⊆ Marked(ps)) and not(Stacked(ST,ps))
      then return;                              {1}
   if Stacked(ST,ps) then
      if RealCircuit(ST,ps) then                {2}
         begin
(b)         Cond(ps,f) := Cond(ps,f) - ens;
            return
         end
      else                                      {3}
         { there is a state change sequence from
         p' to p" }
         if (Σens.M(p) > n0) then
            begin
               Modif(ps):=true
               Liste(ps):=Liste(ps)∪{p'}
            end
      return;
```

{In case (1), the ordinary states have already been studied. In case (2), ps is linked to itself by a "real" circuit, therefore there is a loop of ordinary states that never fulfil f_2. In case (3), the circuit is a "false" one, we can not say, at this step, if the states defined by ens verify f or not. We only know that the executed sequence can be done again and then the conditions associated to the place p' have to be modified}.

```
   Marked(ps) := Marked(ps) ∪ ens;
   ens2 := Cond(ps,f1) ∩ ens;
   {set of states of ens that may verify f}

   Push(ST,ps);
   ens3 := Ø;
   for all ps' such that ps-t->ps' do
      begin
         ens4 := Trans-1(ens2,ps,t,ps');
         {set of states of ps' that are reached
         by states of ens2 with the transition t}

         ens3 := ens3 ∪ ens4;
         {set of states reached by states of ens2}

         VerifAU(f,ps',ens4);
(c)      Cond(ps,f) := Cond(ps,f)-
               Trans(ens4- Cond(ps',f),ps,t,ps');
         {we remove from Cond(ps,f) the states
         of ens2 that lead to states of ens4 that
         do not verify f}
         if Cond(ps,f)=None then
            begin
               Pop(ST);
               return
            end
      end;

      Cond(ps,f) := (Cond(ps,f) -
      (ens2-Trans(ens3,ps,t,ps')))∪Cond(ps,f2);
      {we remove from Cond(ps,f) the states of
      ens2 that do not verify f2 and have no
      successor}

      if Modif(ps) = true then
         begin
            { Cond(ps) = ∪Cei }
            for each p in Liste(ps)
               For each Cei, Cei.OP(p) = "≥"
         end
   Pop(ST);
end;
```

To define the set of ordinary states of ps that verify $\exists[f_1 \text{ U } f_2]$ we study all the successors of ps and

add to Cond(ps,f) the states that verify f_1 and that lead to one that verify $\exists[f_1 \, U \, f_2]$. The algorithm is, roughly speaking, the same as for $\forall[f_1 \, U \, f_2]$ apart from the lines a,b and c.

(a') Cond(ps,f) := Cond(ps,f_2);
(b'), (d') we remove these lines;
(c') Cond(ps,f) := Cond(ps,f) ∪
Trans(Cond(ps',f),ps,t,ps');

Property: The conditions Cond(ps,f) = $\cup Ce_i$ are such that $\sum Ce_i.M(s) \leq n_0$. Otherwise, the condition Ce_i does not appear in Cond. It is not compatible with the parameterized state graph and the bound n_0.

This property ensures that the verification algorithm ends.

5. Conclusion

In this paper, we have introduced a model to reason about parallel algorithms including a finite but unknown number of processes. Then we have presented a procedure to obtain a parameterized representation of the reachability set. This procedure fails when we are not able to represent, with a single graph, the behaviour of the model whatever the value of the parameter is. The success of this procedure often happens in real cases. We have developed a procedure to evaluate the formulae of the language CTL-X on the parameterized state graph.

Compared with the work presented in [Clarke 87], we also define a sufficient number to have the full behaviour of the parameterized program. Clarke and Grümberg do not find this bound automatically. They do not specify how to determine if such a bound exists. In our work we find a bound automatically during the building of the parameterized state graph. Another interest of our work is that we do not compute the reachable states only for the program with the sufficient number of processes. We compute the reachable states for programs with any number of processes greater or equal to the bound.

Further works will deal with the possible extensions of the model. The model presented in [Vernier 93.b] allows to distinguish the behaviour of one process. This distinction offers two possibilities: to prove fairness properties (e.g., if a process wants to access critical section it will success) and to model problems where one process has a specific behaviour. This distinguished process can be dynamically changed. The parameterized state graph building algorithm and the evaluation algorithm work on this model. A lack of this model is the inability to consider order on processes. It will allow to model algorithms that assume that processes are on a ring and can communicate only with their neighbours. The difficulty lies in the building of the parameterized state graph. We have to be able to keep links between processes.

Another improvement of our work is the consideration of more complex formulae languages such as CTL*, FCTL [Emerson 85]. The model checkers that consider such logics have to be able to compute strongly connected components of the state graph. Therefore we will be able to deal with these logics if we are able to detect strongly connected components on the parameterized state graph.

References

[Clarke 86] Clarke E.M., Emerson E.A., Sistla J., *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification*, ACM Transactions on Programming languages and systems, vol. 8, n°2, April 1986, pages 244-263.

[Clarke 87] Clarke E.M., Grümberg O., *Avoiding The State Explosion Problem in Temporal Logic Model Checking Algorithms*, Proc. 6th ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, August 1987, pages 244-303.

[Emerson 85] Emerson E.A., Lei C., *Modalities for model checking: Branching time strikes back*, Proc. 12th Annual Symposium on Principles of Programming Languages, New-Orleans, Louisiana, January 1985, pages 84-96.

[Emerson 86] Emerson E.A., Halpern J.Y., *'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time*, Journal of ACM, vol. 33, n°1, 1986, pages 151-178.

[Emerson 88] Emerson E.A., Srinivasan J., *Branching Time Temporal Logic*, Branching Time and Partial Order in Logics and Models for Concurrency, Workshop on Linear Time, LNCS 354, 1988, pages 123-172.

[Emerson 89] Emerson E.A., Sadler T., Srinivasan J., *Efficient Temporal Reasoning*, Proc. of the 16th ACM Symposium of Principles of Programming Languages, Austin, Texas, 1989, pages 166-178.

[Emerson 90] Emerson E.A., Sadler T., *A Decidable Temporal Logic to Reason about Many Processes*, Proc. 9th ACM Symposium on Principles of Distributed Computing, 1990, pages 233-245.

[Finkel 90] Finkel A., *A minimal Coverability Graph for Petri Nets*, Proc. 11th International Conference on Application and Theory of Petri Nets, Paris, France, June 1990, pages 233-245.

[Ginsburg 64] Ginsburg S., Spanier E., *Bounded Algol-like languages*, Trans. Amer. Math. Soc. 113, 1964, pages 333-368.

[Karp 69] Karp R.M., Miller R.E., *Parallel Program Schemata*, Journal of Computer science, vol. 3, n°2, May, 1969.

[Lamport 83] Lamport L., *What good is temporal logic?*, Proc. IFIP 9th World Computer Congress, Paris, France, 1983, Information processing 83, R.E.A. Mason editor, North-Holland, pages 657-668.

[Lichtenstein 85] Lichtenstein O., Pnueli A., *Checking That Finite State Concurrent programs*

Satisfy Their Linear Specification, Proc. 12th ACM Symposium of Principles of Programming Languages, New-Orleans, Louisiana, January 1985, pages 97-107.

[Pnueli 85] Pnueli A., *Linear and branching structures in the semantics and logics of reactive systems*, Proc. 12th International Colloquium on Automata, Languages and Programming, Springer Verlag, 1985, pages 15-32.

[Sistla 87] Sistla A.P., German S.M., *Reasoning with Many Processes*, Proc. Symposium on Logic in Computer Science, Ithaca, New-York, June 1987, pages 138-152.

[Vergauwen 93] Vergauwen B., Lewi J., *A Linear Model Checking Algoritm for CTL*, Proc. 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 1993, LNCS 715, Springer Verlag, pages 447-461.

[Vernier 93.a] Vernier I., *Parameterized Reachability Graph for Parametrized Petri Nets*, Internal report, 1993.

[Vernier 93.b] Vernier I., *Specification And Verification of Parametrized Parallel Programs*, Proc. 8th International Symposium on Computer and Information Sciences, Istanbul, Turkey, November 1993, pages 622-625.

[Vernier 94] Vernier I., Evaluation Paramétrée de Formules de Logique Temporelle, Internal report, IBP, Laboratoire MASI, to appear.

[Wolper 89] Wolper P., Lovinfosse V., *Verifying Properties of Large Sets of Processes with Network Invariants*, Proc. International Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France 1989, LNCS 407, pages 68-80.

Final Model Semantics for Normal Default Theories*

Joeri Engelfriet and Jan Treur

Free University Amsterdam, Department of Mathematics and Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
email: joeri@cs.vu.nl, treur@cs.vu.nl

1 Introduction

An important characteristic of default reasoning is that usually there are different lines of reasoning possible, each leading to a consistent set of conclusions. In default logic these conclusion sets are descibed by (Reiter) extensions. In common examples this leads to a variety of (mutually contradictory) extensions. In logic one is used to express semantics in terms of models that represent consistent descriptions of (conclusions about) the world and semantic entailment relations based on a specific class of this type of models. These notions lead to essential problems in describing alternative conclusion sets for default reasoning. Sometimes one introduces sceptical entailment (what is true in all conclusion sets) or credulous entailment (what is true in some conclusion set). From a semantic point of view both notions are quite unsatisfactory. They give no description of the essential branching character of default reasoning. Instead they try to ignore or eliminate the branching by only considering global upper and lower limits for conclusion sets. Often these limits leave open a very large space in between them. For a particular line of reasoning they give hardly any information.

In this paper we formalize default reasoning processes (based on normal default theories) by temporal models. This enables us to integrate process aspects of the reasoning in the semantics in an explicit manner. The branching character of these reasoning processes is described by branching time temporal models. One line of reasoning corresponds to a linear time model: a branch in the tree of all possibilities. In one branching time model more than one line of reasoning (and the resulting conclusion sets) can be represented (even though they are mutually contradictory).

We show that under some finiteness condition each normal default theory has a unique (up to isomorphism) final branching time model that indeed covers all possible lines of reasoning, and that the semantics of the theory can be defined on the basis of this unique final model. As an aside we show how sceptical and credulous entailment relations can be defined as well on the basis of this final model.

The idea behind our temporal interpretation of default reasoning is the following. Suppose we have an automated reasoning process which can reason using defaults. If at a certain point in time it wants to apply a default rule, it must be known that the justification of this rule remains consistent in the future of the reasoning process. Therefore the justifications of default rules involve an essential temporal element in default reasoning. The reasoning process can be formalized by describing it as a branching sequence of *information states*, which are descriptions of the (partial) knowledge the process may have deduced up until a certain moment in time. This justifies the use of a temporal partial logic to describe reasoning processes. Our results support the claim that it is more fruitful to focus on formal semantics for non-monotonic logic where (temporal) aspects of the process of reasoning and the resulting conclusions are both integrated in an explicit manner. This in contrast to semantic approaches that intend to focus only on the latter, in an attempt to abstract from the former. Among the authors that also defend the claim are [11], [9].

2 Branching Time Temporal Partial Logic

As no substantial literature on the combination of temporal and partial logic is known to the authors, and the few approaches combining partial and modal logic as known (e.g., [14]) do not serve our purposes, we had to design our own branching time temporal partial logic. In this section we introduce the temporal partial logic that we have defined to satisfy our requirements. We shall first describe partial (strong Kleene) logic. The language will consist of all classical propositional formulae of a certain signature Σ, an ordered sequence of atom names. By $\mathbf{At}(\Sigma)$ we denote the set of atoms of Σ and by $\mathbf{Lit}(\Sigma)$ the set of literals. The formulae in propositional logic based on Σ will be called propositional formulae.

Definition 2.1 (partial model)
a) A *partial model* $\mathbf{M}$ of this signature is an assignment to each atom of Σ of a truth value $\mathbf{0}, \mathbf{1}$ or $\mathbf{u}$ standing for *false*, *true* and *undefined* respectively. If an atom $\mathbf{a}$ is false in $\mathbf{M}$, then the literal $\neg\, \mathbf{a}$ is true in $\mathbf{M}$ and vice versa. If $\mathbf{a}$ is unknown, so is $\neg\mathbf{a}$.

* This work has been carried out in the context of SKBS and the ESPRIT III Basic Research project 6156 DRUMS II.

b) The *ordering of truth values* is defined by $u \leq 0$, $u \leq 1$, $u \leq u$, $0 \leq 0$, $1 \leq 1$. We call the model N a *refinement* of the model M, denoted by $M \leq N$, if for all atoms a it holds: $M(a) \leq N(a)$.
c) For a set of formulae K, by $Lit(K)$ we denote all literals in K. For any partial model M we define $Lit(M) = \{ L \in Lit(\Sigma) \mid L \text{ is true in } M \}$. For any consistent set of literals S the unique partial model N with $Lit(N) = S$ will be denoted by $<S>$.
d) If no atom has truth-value u in a model M, then M is called *complete*.

One can define the truth value of an arbitrary propositional formula in a partial model by induction on the structure of the formula, where we will use the Strong Kleene definition for the connectives (e.g., see [5], [12]). By $M \models^+ \alpha$, $M \models^- \alpha$ and $M \models^u \alpha$ we will denote that α is true, false or undefined respectively.

We will now temporalize (see [10]) these models to partial temporal models, based on some flow of time.

Definition 2.2 (flow of time)
A *flow of time* is a pair $(T, <)$ where T is a non-empty set of time points and $<$ is a binary relation over T, called the *immediate successor relation*. Here for s, t in T the expression $s < t$ denotes that t is an (immediate) *successor* of s, and that s is an (immediate) *predecessor* of t. We want to consider forward branching structures: $(T, <)$ viewed as a graph has to be a *forest*, that is a disjoint union of trees. Furthermore the transitive (but not reflexive) closure $\ll$ of $<$ is introduced. A flow of time is called *linear* if $\ll$ is a total ordering. A *branch* in a forest is a branch of any of its trees. We call an element s of T *minimal* if it is a root of one of the trees, and we call it the *root* of T if it is the unique minimal element (then T is a tree).

Definition 2.3 (Partial temporal model)
Let Σ be a signature and $(T, <)$ a flow of time.
a) A (propositional) *partial temporal model* of signature Σ and flow of time $(T, <)$ is a triple $(M, T, <)$ where $(T, <)$ is a flow of time and M is a mapping

$$M : T \times At(\Sigma) \rightarrow \{0, 1, u\}$$

If no confusion is expected we will often denote a partial temporal model by M. Moreover, instead of t is a time point in $(M, T, <)$ we sometimes say t is a time point in M (or simply t in M), with meaning $t \in T$. If a is an atom, and t is a time point in T, and $M(t, a) = 1$, then we say that in this model M *at time point* t *the atom* a *is true*. Similarly we say that *at time point* t *the atom* a is *false*, respectively *undefined*, if $M(t, a) = 0$, respectively $M(t, a) = u$.
b) If M is a partial temporal model, then for any fixed time point t the partial model $M_t : At(\Sigma) \rightarrow \{0, 1, u\}$ (the *snapshot at time point* t) is defined by M_t: $a \mapsto M(t, a)$.
We sometimes will use the notation $(M_t)_{t \in T}$ where each M_t is a partial model as an equivalent description of a partial temporal model M. We call a partial temporal model *complete* if no u is assigned by it.
c) If K is a set of propositional formulae for the signature Σ, a temporal model M of signature Σ is called *a model of* K if all formulae of K are true in M_t (according to the Strong Kleene semantics) for all $t \in T$. This is denoted by $M \models^+ K$.
d) The *refinement relation* $\leq$ between partial temporal models is defined by: $M \leq N$ if they have the same flow of time and $M(t, a) \leq N(t, a)$ for all time points t and atoms a.
e) In a straightforward manner the notion of sub-ft of flows of time is inherited to the notion of *submodel*.

Because the partial temporal models we will use have a more differentiated structure towards the future than towards the past, we assume the following temporal operators. In these definitions, $(M, t) \models^+ \alpha$ means that in the model M at time point t the formula α is true, $(M, t) \models^- \alpha$ that it is false and $(M, t) \models^u \alpha$ that it is undefined. By the sign $\not\models$ we denote that the concerning relation is not the case. Definitions 2.4 and 2.5 below should be read in relation to each other.

Definition 2.4 (temporal operators and their semantics)
Let a formula α, a partial temporal model M, and a time point $t \in T$ be given, then:

a) $(M, t) \models^+ \exists F\alpha \Leftrightarrow \exists s \in T\ [t \ll s \ \&\ (M, s) \models^+ \alpha]$
$(M, t) \models^- \exists F\alpha \Leftrightarrow (M, t) \not\models^+ \exists F\alpha$
b) $(M, t) \models^+ \forall F\alpha \Leftrightarrow$ **for all branches including t there exists an s in that branch such that** $[t \ll s \ \&\ (M, s) \models^+ \alpha]$
$(M, t) \models^- \forall F\alpha \Leftrightarrow (M, t) \not\models^+ \forall F\alpha$
c) $(M, t) \models^+ \forall G\alpha \Leftrightarrow \forall s \in T\ [t \ll s \Rightarrow (M, s) \models^+ \alpha]$
$(M, t) \models^- \forall G\alpha \Leftrightarrow (M, t) \not\models^+ \forall G\alpha$
d) $(M, t) \models^+ \exists G\alpha \Leftrightarrow$ **there exists a branch including t such that for**

			all s in that branch [t « s ⇒ (M, s) ⊨⁺ α]
	$(M,t) \models^- \exists G\alpha$	⇔	$(M,t) \not\models^+ \exists G\alpha$
e)	$(M,t) \models^+ P\alpha$	⇔	$\exists s \in T\ [s « t\ \&\ (M,s) \models^+ \alpha]$
	$(M,t) \models^- P\alpha$	⇔	$(M,t) \not\models^+ P\alpha$
f)	$(M,t) \models^+ C\alpha$	⇔	$(M,t) \models^+ \alpha$
	$(M,t) \models^- C\alpha$	⇔	$(M,t) \not\models^+ C\alpha$

Now we can make new formulae using conjunctions, negations and implications and these temporal operators. From now on the word *(temporal) formula* will be used to denote a formula possibly containing any of the new operators, unless stated otherwise. We call a subformula *guarded* if it is in the scope of an operator. A *purely temporal formula* is one in which all objective subformulae are guarded. Formulae without temporal operators are called *propositional*.

Definition 2.5 (Interpretation of temporal formulae in a partial temporal model)
Let Σ be a signature, let M be a partial temporal model for Σ, and $t \in T$ a time point.

a) For any propositional atom $p \in At(\Sigma)$:

$(M,t) \models^+ p$	⇔	$M(t,p) = 1$
$(M,t) \models^- p$	⇔	$M(t,p) = 0$

b) For a formula of the form $\exists F\alpha$, $\forall F\alpha$, et cetera, see Definition 2.4

c) For any two temporal formulae φ and ψ:

(i)	$(M,t) \models^+ \varphi \wedge \psi$	⇔	$(M,t) \models^+ \varphi$ and $(M,t) \models^+ \psi$
	$(M,t) \models^- \varphi \wedge \psi$	⇔	$(M,t) \models^- \varphi$ or $(M,t) \models^- \psi$
(ii)	$(M,t) \models^+ \varphi \rightarrow \psi$	⇔	$(M,t) \models^- \varphi$ or $(M,t) \models^+ \psi$
	$(M,t) \models^- \varphi \rightarrow \psi$	⇔	$(M,t) \models^+ \varphi$ and $(M,t) \models^- \psi$
(iii)	$(M,t) \models^+ \neg\varphi$	⇔	$(M,t) \models^- \varphi$
	$(M,t) \models^- \neg\varphi$	⇔	$(M,t) \models^+ \varphi$

d) For any temporal formula φ:

$(M,t) \not\models^+ \varphi$	⇔	$(M,t) \models^+ \varphi$ does not hold
$(M,t) \not\models^- \varphi$	⇔	$(M,t) \models^- \varphi$ does not hold
$(M,t) \models^u \varphi$	⇔	$(M,t) \not\models^+ \varphi$ and $(M,t) \not\models^- \varphi$

e) For a partial temporal model M, by $M \models^+ \varphi$ we mean $(M,t) \models^+ \varphi$ for all $t \in T$ and by $M \models^+ K$ we mean $M \models^+ \varphi$ for all $\varphi \in K$, where K is a set of formulae possibly containing any of the defined operators.

Notice that essentially the formulae of the form $O\alpha$ with O one of our temporal operators are two-valued: the truth value of $O\alpha$ can only be true or false; the truth value undefined is excluded. The definition of the operators could be altered to allow for a value undefined as well (e.g., see [14], p. 46), but that does not serve our purposes in view of the requirements we have.

If an element t lies in a tree with root r the length of the unique path of r to t is called the *depth* of the element t. This defines a mapping from the flow of time to the natural numbers; in the case of a branch this mapping is a successor relation isomorphism. We can identify a branch with a linear flow of time.

What is still interesting about a reasoning process, is of course its set of final conclusions. To be able to talk about *final* conclusions, we have to assume that the reasoning is *conservative*, which means that once a fact is established, it will remain true in the future of the reasoning process. In that case a fact is a final conclusion of a process if it is established at the branch representing the process at any point in time. So, besides reasoning paths also the conclusions they result in are defined in a branching time model in the following manner:

Definition 2.6 (limit models of a conservative model)
Let M be a partial temporal model. Then M is *conservative* if $M_t \leq M_s$ whenever $t « s$. The *limit model of a branch* $B = (T', <')$ of M, denoted by $\lim_B M$, is the partial model with for all atoms p:

(i) $\lim_B M \models^+ p$	⇔	$\exists t \in T'$: $(M,t) \models^+ p$
(ii) $\lim_B M \models^- p$	⇔	$\exists t \in T'$: $(M,t) \models^- p$

Notice that p is undefined in $\lim_B M$ if and only if p is undefined in B for all $t \in T'$.

3 Interpreting Default Logic in Temporal Partial Logic

We will first give a brief overview of Reiter's default logic, restricted to a propositional language. A *normal default rule* is an expression of the form $(\alpha:\beta)/\beta$, where α and β are propositional formulae. A *default theory* Δ is then a pair $<W,D>$ where W is a set of sentences (the *axioms* of Δ), and D a set of default rules. We will not give Reiter's original definition of an extension (see [2], [13]), but a slight variation of it which in [6] has been shown to be equivalent.

Definition 3.1 (Reiter Extension)
Let $\Delta = <W, D>$ be a default theory of signature Σ, and let E be a consistent set of sentences for Σ. Then E is a Reiter extension of Δ if $E = \bigcup_{i=0}^{\infty} E_i$ where $E_0=Th(W)$, and for all $i \geq 0$

$$E_{i+1} = Th(E_i \cup \{ \beta \mid (\alpha : \beta) / \beta \in D, \alpha \in E_i \text{ and } \neg\beta \notin E \})$$

If E is a Reiter extension, then throughout the paper by E_i we will denote the subsets of E as defined in this lemma. We will restrict the defaults to be *based on literals*, which means that all formulae in a default rule have to be literals. This is not an important hindrance, since if an arbitrary formula φ occurs in a default rule, one could extend the signature with a new atom a_φ, substitute this for φ and add a rule $\varphi \leftrightarrow a_\varphi$ to W. It can be shown that the adapted theory has extensions which are the same as the extensions of the original default theory if from the former the formulae containing the new atom are omitted. The restriction to literals is necessary to be able to describe the E_i by partial models.

We will establish an interpretation mapping from default theories to temporal theories. Under this interpretation the Reiter extensions of a default theory and partial temporal models which obey a number of rules correspond to each other. The correspondence we are aiming at will be such that the literals true in a branch B of the temporal model at depth i will be exactly those literals which are element of E_i and the literals in E will be those true in the limit model of a branch B in M:

$$Lit(B_i) = Lit(E_i) \text{ and } Lit(\lim_B M) = Lit(E).$$

We will investigate what requirements should be imposed on the partial temporal model M. Firstly, since the E_i are non-decreasing and are defined for any $i \in \mathbb{N}$, our model should be conservative and have only infinite branches. If we define

$$C' = \{ P(L) \rightarrow C(L) \mid L \text{ a literal} \} \cup \{ \exists F \text{ true} \}$$

it can easily be shown that $M \models^+ C'$ if and only if M is conservative and has only infinite branches. Next we will try to see which rules will ensure in the model the effect of application of the default rules. To this end we have to look at how a default rule is used in our definition of an extension. The meaning of a normal default rule $(\alpha: \beta) / \beta$ is that if $\alpha \in E_i$, and $\neg\beta \notin E$, then β has to be in E_{i+1}, and consequently in E_j for all $j > i$. The requirement $\neg\beta \notin E$ is equivalent to $\neg\beta \notin E_i$ for all $i \in \mathbb{N}$, and as the sets E_i are non-decreasing, it is equivalent to $\neg\beta \notin E_j$ for all $j > i$. If we want to enforce a corresponding effect of the use of defaults in our temporal model, we have to make sure that at all times, if α has become true in the past, and there is at least a reasoning path where $\sim\beta$ is not true at any point in the future, then β has to be true in the current state. This leads us to the rule:

$$P\alpha \wedge \neg \forall F \sim\beta \rightarrow C\beta$$

which has to be true in the model at all time points. As the E_i are closed under logical entailment, a corresponding model will have to be semantically closed:

Definition 3.2 (semantically closed)
a) Let K be a set of propositional formulae, and let M be a partial model. We call M *consistent* with K if there exists a complete model of $Lit(M) \cup K$. The model M is called *semantically closed* with respect to K if for any literal L it holds:

$$Lit(M) \cup K \models L \Rightarrow M \models^+ L,$$

where $\models$ is the standard semantic consequence relation (for complete models).
b) A partial temporal model M is *semantically closed* with respect to K if all its states M_t are.

Next we point out how to find a set of rules W', such that the models of this set are those which are semantically closed with respect to W. The method to construct such a theory W' is an adapted form of the method to construct rule-based knowledge bases satisfying a given functionality (see [15]). The semantically closedness restriction says that in a situation where certain literals true in a model M, together with W, can be used to infer another literal, this literal should be true in M. Now we can just rigorously describe each of these situations by a rule. So for each subset of literals F which can be used to infer a literal L using W, we make a rule expressing that if all literals in F are true, L should be true too. The case when F is empty (when a literal can be inferred from W alone) has to be taken care of separately:

$W' = \{ C(L) \mid L \text{ literal}, W \models L \} \cup$
$\{ C(con(F)) \rightarrow C(L) \mid L \text{ literal}, F \neq \emptyset \text{ a finite set of literals with } F \cup W \models L \}$

where the formula **con(F)** is the conjunction of the elements of **F**.

Now we can define a temporal interpretation of Δ as a temporal theory associated to Δ. Temporal models of Δ are models of this temporal theory. As we do not want any extra conclusions in the corresponding model than those which have to be drawn, we will take the minimal models with respect to $\leq$.

Definition 3.3 (temporal interpretation of a normal default theory)
Let $\Delta = <W, D>$ be a normal default theory of signature Σ. Define

$C' = \{ P(L) \rightarrow C(L) \mid L \in Lit(\Sigma) \} \cup \{ \exists F \text{ true} \}$
$D' = \{ P\alpha \wedge \neg \forall F \neg\beta \rightarrow C\beta \mid (\alpha : \beta) / \beta \in D \}$
$W' = \{ C(L) \mid L \text{ literal}, W \models L \} \cup$
$\{ C(con(F)) \rightarrow C(L) \mid L \text{ literal}, F \neq \emptyset \text{ a finite set of literals with } F \cup W \models L \}$

a) The *temporal interpretation* of Δ is the temporal theory $T_\Delta = C' \cup D' \cup W'$.
b) If a (branching time) partial temporal model **M** of signature Σ with a forest as its flow of time satisfies $M \models^+ T_\Delta$ then we call **M** a *temporal model of* the default theory Δ.
c) The temporal model **M** of Δ is called *minimal* if for every model **C** of T_Δ with $C \leq M$ it holds $C = M$.

In a previous publication [6] it has been shown how a linear time partial temporal model of a normal default theory can describe one line of reasoning (i.e., can play in a sense the role of a Reiter extension of the default theory). In [6] we gave a treatment restricted to the linear time case, but there also non-normal defaults were covered (we restrict ourselves here to the normal case).

Theorem 3.4
Let $\Delta = <W, D>$ be a normal default theory with **D** based on literals.
a) If **M** is a minimal linear time temporal model of Δ, then $Th(Lit(\lim_M M) \cup W)$ is a Reiter extension **E** of Δ. Moreover, $E_i = Th(Lit(M_i) \cup W)$ for all $i \in \mathbb{N}$.
b) If **W** is consistent and **E** a Reiter extension of Δ, then the partial temporal model **M** defined by $M = (< Lit(E_i) >)_{i \in \mathbb{N}}$ is a minimal linear time temporal model of Δ with $Lit(\lim_M M) = Lit(E)$.

As maximal branches of a branching time temporal model are linear time models it seems reasonable to use Theorem 3.4 to establish a correspondence between these branches and Reiter extensions. In next sections we will show that this indeed can be worked out technically.

4 Branching time models for normal default theories

In this section we apply the algebraic techniques developed in [7] to the model theory of the temporal translation of normal default theories. We will assume all partial temporal models forests.

Definition 4.1 (homomorphism)
A mapping $f : T \rightarrow T'$ is called a *homomorphism* of **M** to **M'** if
(i) $s < t \rightarrow f(s) < f(t)$ for all $s, t \in T$
(ii) $M(s) = M'(f(s))$ for all $s \in T$
(iii) If **s** is a minimal element of **T** then **f(s)** is minimal element of **T'**

Definition 4.2 (persistency)
Let $f : M \rightarrow M'$ be a homomorphism.
The *forward persistency* property for a formula φ (under **f**) is defined by
$M, t \models^+ \varphi \Rightarrow M', f(t) \models^+ \varphi$
for all time points **t** in **T**.
The *backward persistency* property for a formula **A** (under **f**) is defined by
$M, t \models^+ \varphi \Leftarrow M', f(t) \models^+ \varphi$
for all time points **t** in **T**.

In [7] an overview of results on persistency is given. Here we confine ourselves to the following:

Proposition 4.3

For any default theory Δ its temporal interpretation T_Δ is backward persistent under homomorphisms from models with successor existence.

The following definition summarizes results of [7].

Definition 4.4 (closed model)
A model **M** is called *closed* if one of the following equivalent conditions is satisfied:

(i) For all **t, u** minimal elements or with a common immediate predecessor $M_t = M_u$ implies $t = u$.
(ii) Every homomorphism $f : M \to M'$ is injective
(iii) For every **M'** there is at most one homomorphism $f : M' \to M$

In the class **BT** of all branching time models we distinguish two subclasses, namely **LT**, the class of linear time models and **CL**, the class of closed models. Since it is easy to establish that linear time models are closed we have

$$LT \subset CL \subset BT$$

There are other connections as well. Every branching time model can be mapped by a surjective homomorphism onto a closed one (its closure). Moreover, all maximal branches in a branching time model are linear models, and together they cover the whole flow of time.

Proposition 4.5
Let Δ be a normal default theory and **M** a temporal model of Δ.
a) Every maximal branch of **M** is a linear time model of Δ.
b) **M** is a minimal temporal model of T_Δ if and only if every maximal branch of **M** is a minimal temporal model of Δ.

Let $\mathbb{LT}(M)$ denote the set of maximal branches in **M**, and let $\mathbb{E}(\Delta)$ be the set of all consistent Reiter extensions of Δ. Applying Theorem 3.4 and Proposition 4.5 (and noticing that a linear model can be mapped in at most one way to a closed **M**) yields the following theorem. Notice here that we denote the partial model based on (i.e., making true just) a set of literals **L** by $<L>$.

Theorem 4.6
Let Δ be a normal default theory and **M** a closed minimal temporal model of Δ.
There is a one to one correspondence between the set $\mathbb{LT}(M)$ of maximal branches of **M** and a set of Reiter extensions of Δ. More precisely, there is an injective mapping

$$\Phi : \mathbb{LT}(M) \to \mathbb{E}(\Delta)$$

defined by

$$\Phi(B) = Th(W \cup Lit(\lim_B M))$$

Furthermore, for every $i \in \mathbb{N}$ it holds

$$\Phi(B)_i = Th(W \cup Lit(B_i))$$

If

$$\Psi : \mathbb{E}(\Delta) \to \mathbb{LT}(M)$$

is defined by

$$\Psi(E) = (<Lit(E_i)>)_{i \in \mathbb{N}}$$

then

$$\Psi(\Phi(B)) = B$$

for all **B** in $\mathbb{LT}(M)$.

From this correspondence for $i = 0$ it follows that any two maximal branches **A** and **B** have the same initial information state:

$$A_0 = <Lit(\Phi(A)_0)> = <Lit(Th(W))> = <Lit(\Phi(B)_0)> = B_0$$

Because **M** is closed it follows they are related to the same time point. This means that all minimal elements of **M** coincide. Therefore we have:

Corollary 4.7
Let Δ be a normal default theory.
All closed minimal temporal models of Δ have flows of time that are trees.

We now established results for given closed minimal models for the temporal interpretation of a normal default theory. In the next section we will show how to build closed minimal models from minimal linear time models.

5 Final minimal temporal models of normal default theories

We need to impose a finiteness requirement on default theories to exclude pathological examples. Roughly spoken, we require that if a set of formulae can be approximated arbitrarily close by a set of (Reiter) extensions, then it is itself a Reiter extension.

Definition 5.1 (extension complete)
Let Δ be a default theory.
a) We call a chain of sets of formulae
$S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots$
approximated by a (Reiter) extension E of Δ *up to depth* n if for all $i \leq n$ it holds $S_i = E_i$ (where the E_i are as in Definition 3.1).
We call the chain $(S_k)_{k \in \mathbb{N}}$ *approximated by a set* of Reiter extensions R of Δ if for every $n \in \mathbb{N}$ there is an $E \in R$ such that $(S_k)_{k \in \mathbb{N}}$ is approximated by E up to depth n.
b) We call Δ (Reiter) *extension complete* if for any chain of sets of formulae $(S_k)_{k \in \mathbb{N}}$ that is approximated by a set of (Reiter) extensions R of Δ, its union $\bigcup_{k \in \mathbb{N}} S_k$ is a Reiter extension E with (where the E_i are as in Definition 3.1) $S_i = E_i$ for all i.

An example of a normal default theory that is not extension complete is the following. Take the propositional atoms b, $a_0, a_1, \ldots$ (infinitely many), and $\Delta = \langle W, D \rangle$ with

$$\begin{aligned} W &= \{a_0\} \cup \{ b \rightarrow a_i \mid i \in \mathbb{N}\} \\ D &= \{ : b / b\} \cup \\ &\quad \{ a_i : a_{i+1} / a_{i+1} \mid i \in \mathbb{N}\} \cup \\ &\quad \{ a_i : \neg a_{i+1} / \neg a_{i+1} \mid i \in \mathbb{N}\} \end{aligned}$$

This normal default theory has infinitely many extensions:

$$E^{(0)} = Th(W \cup \{ b \})$$

and for each $n \in \mathbb{N}$

$$E^{(n)} = Th(W \cup \{ a_i \mid i \leq n \} \cup \{ \neg a_i \mid i > n \}$$

The chain of sets S_k defined by

$$S_k = Th(W \cup \{ a_i \mid i \leq k \})$$

is approximated by the extensions $E^{(n)}$ but its union is even not an extension (it is only included in an extension). Therefore this default theory Δ is not extension complete. In finite cases such phenomena cannot occur:

Proposition 5.2
Every default theory with a finite set of defaults is extension complete

We recall from [7] the following definition of joint closure.

Definition 5.3 (joint closure)
Let I be any index set and let $(M^i)_{i \in I}$ be a collection of models
a) Let $(a_i)_{i \in I}$ be a collection of homomorphisms $a_i : M^i \rightarrow N$.
We call $(a_i)_{i \in I}$ *jointly epic* if for every t in N there exists an i and an s in M^i such that $a_i(s) = t$.
b) We call $(C, (a_i)_{i \in I})$ with $a_i : M^i \rightarrow C$ a homomorphism for each i in I a *joint (epic) closure* for $(M^i)_{i \in I}$ if
(i) $(a_i)_{i \in I}$ are jointly epic.
(ii) For every model D with jointly epic homomorphisms $b_i : M^i \rightarrow D$ there exists a homomorphism $c : D \rightarrow C$ such that $ca_i = b_i$ for all $i \in I$.
c) A joint closure of a single M is called an *(epic) closure* of M.

Any joint closure is closed (see [7]). It has been established in [7] that every collection of models has a joint closure, and that there is a canonical construction for it. The idea of the construction is to take the coproduct (disjoint union) of the flows of time, and making that closed by forming a (surjective) homomorphic image that identifies the nodes that a) are minimal or have a common immediate predecessor and b) have the same information state.

Proposition 5.4
a) Every collection of partial temporal models $(M^i)_{i \in I}$ has a unique (up to isomorphism) joint closure. It is closed, and is denoted by $\mathbf{jcl}((M^i)_{i \in I})$.

b) If φ is forward persistent under surjective homomorphisms, then

$M^i \models^+ \varphi$ for all $i \in I$ $\Rightarrow$ $jcl((M^i)_{i \in I}) \models^+ \varphi$.

If φ is backward persistent under injective homomorphisms, then

$jcl((M^i)_{i \in I}) \models^+ \varphi$ $\Rightarrow$ $M^i \models^+ \varphi$ for all $i \in I$.

Proposition 5.5

Let Δ be an extension complete normal default theory and let M be a minimal model of T_Δ. If $f:M \to M'$ is a surjective homomorphism, then M' is a minimal model of T_Δ.

Theorem 5.6

Let Δ be an extension complete normal default theory and M a partial temporal model.
The following are equivalent:

(i) M is a closed minimal temporal model of Δ

(ii) M is the joint closure of a set S of minimal linear time models of Δ

Definition 5.7 (final minimal model)

The model F is called a *final minimal temporal model* of Δ if it is a minimal temporal model of Δ and for each minimal temporal model M of Δ there is a unique homomorphism $f : M \to F$.

If a final minimal temporal model exists it is unique up to isomorphism; it is denoted by F_Δ. We have the following main result on the existence of final minimal temporal models of normal default theories. It can be proven by taking the joint closure of all minimal linear time models of Δ and applying Theorem 5.6 and Theorem 4.6.

Theorem 5.8

Let Δ be a normal default theory.

a) If Δ is extension complete, then there exists a unique final minimal temporal model F_Δ of Δ.

b) Suppose a final minimal temporal model F_Δ of Δ exists.

Then F_Δ is the joint closure of all minimal temporal models of Δ, and also of all minimal linear time models of Δ.
For every minimal temporal model of Δ there is a unique homomorphism into F_Δ; for closed minimal temporal models of Δ this homomorphism is injective.
There is a one to one correspondence between the set $\mathbb{LT}(F_\Delta)$ of maximal branches of F_Δ and the set $\mathbb{E}(\Delta)$ of all Reiter extensions of Δ. More precisely, there is a bijective mapping

$$\Phi : \mathbb{LT}(F_\Delta) \to \mathbb{E}(\Delta)$$

defined by

$$\Phi(B) = Th(W \cup Lit(\lim_B F_\Delta))$$

Furthermore, for every $i \in \mathbb{N}$ it holds

$$\Phi(B)_i = Th(W \cup Lit(B_i))$$

If

$$\Psi : \mathbb{E}(\Delta) \to \mathbb{LT}(F_\Delta)$$

is defined by

$$\Psi(E) = (< Lit(E_i) >)_{i \in \mathbb{N}}$$

then

$$\Psi(\Phi(B)) = B \quad \text{and} \quad \Phi(\Psi(E)) = E$$

for all B in $\mathbb{LT}(F_\Delta)$ and E in $\mathbb{E}(\Delta)$.

Definition 5.9

We can define the following minimal semantic entailment relations (where Δ is any extension complete normal default theory):

$\Delta \models^m_{CL} \varphi \Leftrightarrow \forall M$ [M is a minimal model of Th & $M \in CL \Rightarrow M \models^+ \varphi$]

$\Delta \models^m_{LT} \varphi \Leftrightarrow \forall M$ [M is a minimal model of Th & $M \in LT \Rightarrow M \models^+ \varphi$]

$\Delta \models^m_{FI} \varphi \Leftrightarrow \forall M$ [M is a minimal model of Th & M final $\Rightarrow M \models^+ \varphi$]

There are some apparent logical relations :

$$\Delta \models^m_{CL} \varphi \Rightarrow \Delta \models^m_{LT} \varphi$$

$$\Delta \models^m_{CL} \varphi \Rightarrow \Delta \models^m_{FI} \varphi$$

As a consequence of the structure theorems above we are able to establish the following theorem that gives more precise connections between the different semantic consequence relations.

Theorem 5.10
Let T_Δ be the temporal interpretation of an extension complete normal default theory Δ and ϕ any formula.
a) If ϕ is forward persistent under surjections, then
$\Delta \models^m_{CL} \varphi \quad \Leftrightarrow \quad \Delta \models^m_{LT} \varphi$
b) If ϕ is backward persistent under injections, then
$\Delta \models^m_{CL} \varphi \quad \Leftrightarrow \quad \Delta \models^m_{FI} \varphi$
c) If ϕ is propositional, then
$\Delta \models^m_{CL} \forall F \varphi \quad \Leftrightarrow \quad \Delta \models^m_{LT} \forall F \varphi \quad \Leftrightarrow \quad \Delta \models^m_{FI} \forall F \varphi$

We will show in Theorem 5.12 how these formulae $\forall F \varphi$ are related to sceptical entailment.

A final minimal model F_Δ of a normal default theory Δ gives an overview of both all possible reasoning paths from a default theory (the branches) and the resulting conclusion sets (the limit models). Therefore in principle it contains all information that is relevant for an intended semantics. As a special case also sceptical and credulous entailment relations can be based on this final model.

Definition 5.11 (sceptical and credulous entailment)
Let F_Δ be the final minimal temporal model of an extension complete normal default theory Δ and let φ be a propositional formula.
a) We define the *sceptical entailment relation* by:
$\Delta \models_{scep} \varphi$ if for all maximal branches B of F_Δ there is a t on B such that $F_\Delta, t \models^+ \varphi$
b) We define the *credulous entailment relation* by:
$\Delta \models_{cred} \varphi$ if there is any t such that $F_\Delta, t \models^+ \varphi$
c) We define F_Δ^ω as the set of the limit models of all maximal branches of F_Δ, i.e.,
$F_\Delta^\omega = \{ \lim_B F_\Delta \mid B \text{ maximal branch of } M \}$

Theorem 5.12
Let F_Δ be the final minimal model with root r of an extension complete normal default theory Δ and let φ be a propositional formula.
a) The following are equivalent:
(i) $\Delta \models_{scep} \varphi$
(ii) $F_\Delta^\omega \models^+ \varphi$
(iii) $(F_\Delta, r) \models^+ \forall F \varphi$
(iv) $(L, s) \models^+ \forall F \varphi$ for every minimal linear time model L of Δ with root s
(v) $(M, s) \models^+ \forall F \varphi$ for every closed minimal temporal model M of Δ with root s
b) The following are equivalent:
(i) $\Delta \models_{cred} \varphi$
(ii) $\lim_B F_\Delta \models^+ \phi$ for some maximal branch B
(iii) $(F_\Delta, r) \models^+ \exists F \varphi$
(iv) $(L, s) \models^+ \exists F \varphi$ for some minimal linear time model L of Δ with root s
(v) $(L, s) \models^+ \forall F \varphi$ for some minimal linear time model L of Δ with root s

6 Conclusions

In other approaches to semantics for default logic (e.g. [3], [9], [16]) usually a preference relation on classes of models is defined. Maximal chains in these relations correspond in a sense to our partial temporal models, where our limit model plays the role of a minimal element of such a chain. The definition of these relations are often not intuitive and usually hide a (branching) process of default reasoning, which is made explicit in our models.

In this paper we have given a temporal interpretation to the notion of a justification in a default rule. This led us to a translation of normal default theories into temporal theories, and to a one-to-one semantical correspondence between the Reiter extensions of such a default theory and the maximal branches of minimal partial models of its temporal translation. To this end we designed a branching time temporal partial logic. As another main result we established that for any normal default theory satisfying a finiteness condition (extension complete) there exists one unique final model where in fact all intended semantics are concentrated.

This work enables one to use concepts from temporal logic to integrate process aspects in the study of formal semantics for default reasoning. We share the view also put forward in [11], [9] that integrating such dynamics in the

semantics is more transparent and fruitful then trying to abstract from them. We think that our work as presented in the current paper (and [6], [8]) contributes to the operationalization of this view.

Acknowledgements

Discussions about the subject with Johan van Benthem, Wiebe van der Hoek and John-Jules Meyer played a stimulating role in the development of the material presented here. This work has been carried out in the context of SKBS and the ESPRIT III Basic Research project 6156 DRUMS II.

References

1. J.F.A.K. van Benthem, The logic of time : a model-theoretic investigation into the varieties of temporal ontology and temporal discourse, Reidel, Dordrecht, 1983
2. P. Besnard, An Introduction to Default Logic, Springer Verlag, 1989
3. P. Besnard, T. Schaub, Possible Worlds Semantics for Default Logics, to appear in Fundamenta Informaticae
4. H. Bestougeff, G. Ligozat, Logical tools for temporal knowledge representation, Ellis Horwood, 1992
5. S. Blamey, Partial Logic, in: D. Gabbay and F. Guenthner (eds.), Handbook of Philosophical Logic, Vol. III, 1-70, Reidel, Dordrecht, 1986.
6. J. Engelfriet, J. Treur, A Temporal Model Theory for Default Logic, in: M. Clarke, R. Kruse, S. Moral (eds.), Proc. 2nd European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty, ECSQARU '93, Springer Verlag, 1993, pp. 91-96
   Extended version: Report IR-334, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science, 1993, pp. 38
7. J. Engelfriet, J. Treur, Relating linear and branching time temporal models, Report, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science, 1994
8. J. Engelfriet, J. Treur, Temporal Theories of Reasoning, Proc. JELIA-94, Springer Verlag, 1994
9. D.W. Etherington, A Semantics for Default Logic, Proc. IJCAI-87. Also in: Reasoning with Incomplete Information, Morgan Kaufmann, 1988
10. M. Finger, D. Gabbay, Adding a temporal dimension to a logic system, Journal of Logic, Language and Information 1 (1992), pp. 203-233
11. D.M. Gabbay, Intuitionistic basis for non-monotonic logic, In: G. Goos, J. Hartmanis (eds.), 6th Conference on Automated Deduction, Lecture Notes in Computer Science No. 138, Springer Verlag, 1982, pp.260-273
12. T. Langholm, Partiality, Truth and Persistence, CSLI Lecture Notes No. 15, Stanford University, Stanford, 1988.
13. R. Reiter, A logic for default reasoning, Artificial Intelligence 13, 1980, pp. 81-132
14. E. Thijsse, Partial logic and knowledge representation, Ph.D. Thesis, Tilburg University, 1992
15. J. Treur, Completeness and definability in diagnostic expert systems, Proc. European Conf. on AI, ECAI-88, München, 1988, pp. 619-624.
16. F. Voorbraak, Preference-based semantics for nonmonotonic logics, in Proc. IJCAI-93, Morgan Kaufmann, 1993, pp. 584-589

Situated Reasoning with Temporal Anaphora

Alice ter Meulen

Indiana University

Logic Group

atm@phil.indiana.edu

Abstract

This paper presents a dynamic logic to model our reasoning in ordinary English about temporal relations between events described by a narrative text. In an interpretation of a sequence of sentences we obtain descriptive, aspectual and perspectival information. Finite labelled trees, called Dynamic Aspect Trees (DATs), with two node sorts represent these three kinds of information in an integrated way, leading to a new notion of a *chronoscope*. They are interpreted in event-structures which satisfy the semantic constraints based on inferential properties of progressive, perfect and past tense inflections. Aspectual information controls how such DATs are updated, which may or may not affect the perspective. The notion of *situated inference* models the context-dependent reasoning about the flow of time. Aspectual information is claimed to be logical in nature, arguing against deferring temporal reasoning to a general default inference system.

1. *Introduction*

When we describe something that happened, we make, quite unreflectively, certain choices in inflecting the verbs for tense and aspect. Even though we hardly pay any attention to it, the inflection chosen contributes information used in our reasoning about the described episode. To characterize this linguistic competence in using tense and aspect, linguistic theory should model not only how the temporal information given by a text is dynamically represented, but also how further information may be extracted at any point from such representations by inference. An account of temporal reasoning in natural language needs to formulate under what conditions a conclusion may be derived from a given representation with a progressive, a perfect or a simple past verbal inflection. Such rules depend not only on the inflection with which the descriptive information was initially given, but also on the order in which it was presented in the text.

Whatever form these representations take, they should be suitable for defining such 'situated' inferences in terms of their configurational properties. This paper proposes that finite labelled trees constitute a clear and flexible medium for such representations of temporal information, integrating the descriptive, aspectual and perspectival information used in reasoning. Reasoning about change and time is inherently context-dependent, as the flow of information is simultaneously *in* time as well as *about* time. For this reason, it provides a paradigmatic domain for linguistic applications of situated inference, i.e. reasoning in natural language about the temporal relations that exist between events we have information about and our own situation. Such reasoning about time is dependent upon specific syntactic and semantic properties of the linguistic form in which the information is presented. In particular, the temporal order in which the information is given matters significantly, as is widely recognized in the literature. The aspectual class of the description also contributes essential control information. The classification of the descriptive information contained in an inflected clause into aspectual classes is a complex process. It is partly lexically driven, partly by the argument-structure, but also depends on the presuppositions of both the locally given and the new information and, ultimately also on core logical relationships. Quantificational information and negative information is typically represented in a sticker, as static information that does not change the context, but constrains the set of possible dynamic updates. The representation of the temporal information should depend on all such parameters in the context. In this paper we take for granted the mapping from English surface structure into aspectual classes.[1]

Discourse Representation Theory (Kamp and Rohrer, 1983 and Kamp an Reyle, 1993) has provided a dynamic account of the interaction between tense and aspectual class in creating temporal anaphora between sequences of sentences. It relies crucially on the notion of a reference-time that constrains the temporal inclusion and precedence between reference markers for the described states and events. The dynamic context-shifting of the reference-time is triggered only by the aspectual classes of accomplishments and achievements, representing events that finish, rather than end, and that can be modified by container adverbial phrases, or certain manner adverbials (Dowty, 1979). Descriptions of events that end, i.e. activities, at times also called 'processes', and all different ways of describing states do not affect the reference time, but simply temporally include the given one. The order in which the stative information is given, is not reflected in the DRS-conditions, nor how stative information behaves, when the reference-time changes.

Various linguistic forms may be used to give stative information: statements with progressive and perfect inflection, those with main verbs *be* or *have* , lexical statives, attitude verbs, and those containing generic NPs referring to kinds. Only some of these forms are sensitive to the textual order. In reasoning about temporal relations we often use relations between clauses containing stative information given at different points in the text. We exploit the logical differences between activities that may end and states described by perfect tenses which, once they started, continue for ever. An account of temporal reasoning should do justice to the fact that certain ways of giving stative information indicate that it persists through changes in context. Of course, knowledge of the world may in addition constrain the temporal relations, as causal structure is directly related to temporal relations. But in this paper we attempt to maintain a clear distinction between knowledge of language and knowledge of the world. Semantic theory should only account for the first, assuming lexical meaning is fixed. In this paper we use the framework of Dynamic Aspect Trees (Seligman & ter Meulen, 1993, and ter Meulen, 1995) to characterize when situated conclusions with different verbal inflections are valid in dynamic interpretation.

[1] See ter Meulen (1995) for a detailed account of how aspectual class is computed for any sentence. This book provides all requisite technical details of the DAT-system.

2. Dynamic Aspect Trees and Chronoscopes

This section introduces the dynamic representation of temporal anaphora in Dynamic Aspect Trees. Examples are discussed at first informally, to illustrate their workings and connect the English clauses to their representation in DATs. Further formalization of their syntax and semantics characterize the system more precisely, so that the notion of situated entailment can be proper defined.

2.1 Examples

DATs represent the descriptive, aspectual and perspectival information that results from our interpretation of English discourse or texts. We assume that the linguistic input is parsed into basic constituent structure, where the lexical expressions in the structured strings are assigned syntactic categories.The first example of how a DAT is constructed is based on the sequence of sentences in (1).

(1) A car hit the fence. The driver was killed. The police arrived.

The interpretation of (1) starts by designing a DAT representing the source-node as a closed node (plug) dominated by the root, an open node (hole). The source, from where the information was issued, is constrained to remain the right-most node in every extension of this DAT. The first sentence *A car hit the fence* contains the descriptive information captured in the type

«***HIT***, *x* «***CAR***, *x*, + », *y* «***FENCE***, *y*, + »+ »

Both indefinite and definite NPs are represented as restrictions on parameters that are arguments of the ***HIT***-relation. The definiteness of *the fence* requires that the parameter *y* is identified with an old parameter already used before this point of the interpretation, possibly identified with an appropriate object in the source-situation. This is not made explicit here in the type above for simplicity, but it is straightforward to incorporate such a definiteness requirement as a constraint. Its simple past tense induces the constraint that the event described precedes the event of issuing of the information. In the DAT this is represented by constructing a branch to the left of the branch containing the source node and labelling that node with the type representing the descriptive information. The node itself is closed, representing a plug. The DAT now contains three nodes, and the current node, the last one constructed, is the plug labelled by the type representing the descriptive information from the first sentence. It indicates that further information will not be encoded as a type labelling a node dependent on the current one, but should label a new right sister node depending on the root. The DAT resulting from the interpretation of the first sentence of (1) is displayed in Fig. 1.

Figure 1. First partial DAT for (1)

The descriptive information expressed in the second sentence in (1) *the driver was killed* is represented by the type

«***KILL***, *z*, *v*«***DRIVE***, *v*, *x*, + », + »

The passive clause is first converted to its active form and represented as a relation between two parameters, the second restricted to refer to the driver of the car introced in the interpretation of the first sentence. Again the definiteness of *the driver* should be made explicit by an identity condition in its type that constrains its reference to corefer with a suitable given parameter. The parameter *x* is picked up as the second parameter of the relation *drive* , to constrain the driver to be the driver of the car that hit the fence. The type labels a new node in the DAT that now becomes the current node, another plug, that is a right sister to the former current one. The resulting DAT is given in Fig. 2. below.

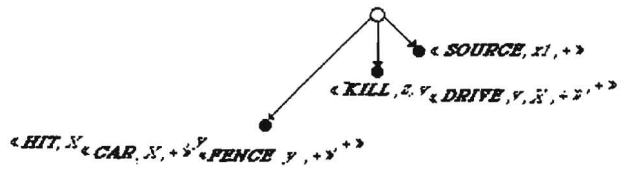


Figure 2 Second partial DAT for (1)

Note that the past tense of the second sentence does not contribute any temporal referential force to the interpretation of the second sentence. Its semantic role is merely to constrain its reference to an event that precedes the source. The context represented in the DAT and its current node determine how the event referred to by the second sentence is located in time with respect to the events already represented in the DAT.

The last sentence in (1) *the police arrived* is now interpreted. It is represented by a third closed node, sister to the current one and to the left of the source. Its label is

« ***ARRIVE***, *w*« ***POLICE***, *w*, + » , + »

The resulting DAT represents this event as a plug as well. The node is located as a right sister to the current one, and a left sister to the source node, accommodating the past tense constraint. Fig.3 contains the final DAT for (1).

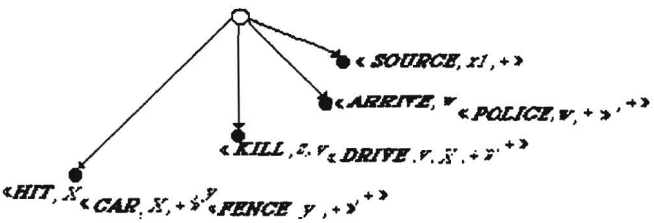


Figure. 3. Complete DAT for (1.1)

An important difference in the dynamic force of simple past tense versus stative perfect tense can now be clarified. Had (1) contained a perfect tense second clause, *the driver had been killed*, the DAT for the resulting (1') would be different. Perfect tenses are represented by the type *PERF* « *T*, + » appended as sticker to the label of either the current node, if it is a plug, or to the next node introduced, if the current node is a hole in the given DAT. Perfect tense clauses are descriptions of perfect states and hence do not introduce new nodes that make the DAT grow. The resulting DAT is given in Fig. 4.

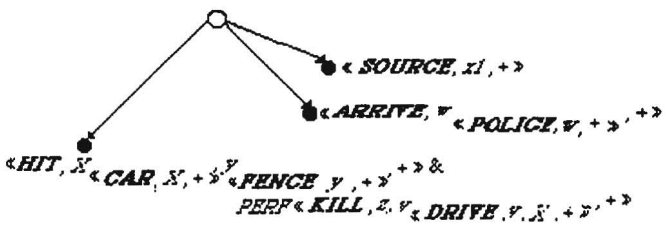


Figure .4 DAT for (1')

The DAT in Fig. 4 above contains a conjoined label on the left most node, which represents the information that at the

time the car hit the fence the driver had already been killed. Despite their static nature, it still matters when perfect tense clauses are given in a text. For the DAT representing the text where the perfect clause comes last, differs from Fig. 4. in which node carries the information that the driver has been killed.

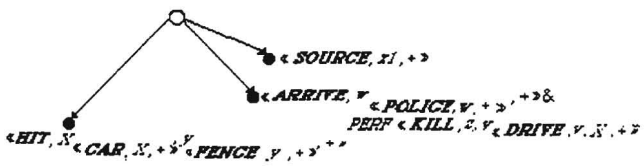


Figure .5.
From fig. 4, but not from fig. 5, we infer

(2) The driver was killed before his car hit the fence and before the police arrived

If what happened is correctly represented in fig. 5, the killing of the driver could well have taken place after the car hit the fence, but before the police arrived. That course of events is not a possible interpretation for (1'), as the perfect tense requires the event that caused the perfect state to arise to have ended before the state was described. I.e. killing the driver must have ended before the car hit the fence, leaving wide open the question what killed him.

2.2 Chronoscopes

In shifting perspective by introducing a new sister to the given current node, you back up to the lowest node compatible with the new information, ensuring also that the new information is compatible with any higher node on the path to the root of the DAT. Such paths connecting a given node to its root in DATs are called *chronoscopes,* very useful instruments in temporal reasoning with DATs. A chronoscope is called the *current chronoscope* just in case it contains the current node. Compatibility of information is partly lexically determined, partly by the entailment relations between types within the same chronoscope. Obviously a type with a positive polarity is incompatible with the type constituted of the same relation and arguments, but containing a negative polarity. As a relation between types, compatibility is obviously reflexive, symmetric and transitive. Furthermore, if T is compatible with T' and with T'' independently, and T' is compatible with T'', then T must be compatible with T'&T''. Backing up in a DAT to create a new chronoscope is a recursive process. It terminates at the lowest ancestor node of the given current node labelled with compatible information, while plugging its child carrying information that was still incompatible. The creation of a new chronoscope and shifting the perspective requires a certain amount of reasoning, checking of compatibility of information and various inferential other processes.

2.3 Construction Rules

Based on these examples of how DATs get constructed by the interpretation of a text, the following rules may now be formulated.

Rules for DAT-representation

1. If the new information is a state-type, append the type as a sticker to the current node, if it is a plug, and append it to the next node, if the current node is a hole.
2. If the new information is a transition from start to end, introduce a new hole, make it the current node and label it with the new type.
3. If the new information is a transition from start to finish, introduce a plug, make it the current node and label it with the new type.
4. If the extension of the DAT resulting from applying rules 1-4 is inconsistent, then plug the closest dominating node necessary to remove the inconsistency, make it the current node and reapply the rules.

The nature of the current node determines, as we have seen, where the new node is located. For future reference, these rules are stated here as well.

Rules for introducing new nodes

(i) Hole-rule: if the current node is a hole, represent the new node as child of it.
(ii) Plug rule: if the current node is a plug, represent the new node as sister to it.

Plugging of an internally structured node hides its internal structure, rendering it inaccessible in the shifted perspective to the new current chronoscope. The linguistic effect of this plugging up is that all the information labelling the inaccessible nodes must be reported by perfect tense, unless they are represented in simple stickers that may be transmitted to a node in the new current chronoscope. Plugs, terminating a chronoscope, are self-dual , they have no internal structure in the given perspective.

3.Reasoning with DATs

In reasoning with a DAT we rely on a fundamental property of the situations that support them, called *persistence.* If a situation s_1 supports DAT_1 , for instance, and DAT_2 grows out of DAT_1 by application of the rules formulated above, and s_2 supports DAT_2 , then s_2 supports DAT_1. Growing DATs models the accumulation of information in processing natural language input. Persistence means that the information represented in an earlier stage of construction of a DAT is not lost, when the interpretation is continued, although it may not remain accessible from any later current node. The way the available information may be expressed using natural language in conclusions depends on where the current node is in the DAT and its relation to the node labelled with the information the conclusion is based on. The given DAT may be affected by incorporating new information, not merely by growing new nodes, but also by plugging up holes or unplugging plugs. The interpretation defines a non-monotonic relation between DATs.

The list of inferences based on (1) is given below in (3).

(3) inferences from (1.1) (* indicates invalid inference)
a. The car hit the fence before the driver was killed
b. *The car hit the fence when/after the driver was killed
c. The driver was killed when/after the car had hit the fence
d. The car had hit the fence before/when the driver was killed
e. *The driver had been killed after/while the car hit the fence
f. The police arrived after the driver was killed
g. The police arrived when the driver was dead
h. The police arrived when the car had hit the fence
i. The car having hit the fence, the police arrived

The completed DAT for (1) was given in Fig. 3. Its current node is the one, carrying the information that the police arrived, immediate left sister of the source. The conclusions also describe the current node, if the argument is valid. In this sense, reasoning with the information represented in a DAT is always dependent upon the current node, and this makes the temporal reasoning a form of situated reasoning. The current node is the point of departure, as it were, in the process of attempting to verify the conclusion. For (3a) to

be true, given this DAT for (1), there must be a node in the current chronoscope dominating the two nodes labelled with the information expressed by simple past clauses, linked by the temporal adverbial *before*. This adverbial constrains the relation between these nodes with this common ancestor: the node representing the car hitting the fence should be a left descendant of the common ancestor relative to the node representing the driver getting killed. This is in fact the case in Fig. 3: the common ancestor is the root, its left descendant is indeed the node representing the car hitting the fence relative to the node representing the killing of the driver. If the clause-linking adverbial had been *when* or *after*, as in (3b), the conclusion would not follow from the DAT in Fig.3, as they would require the hitting node respectively to be in the same chronoscope as the killing node or be a right descendant of its common ancestor.

In (3 c-e) the perfect clause describes the enduring state that began after the hitting ended. That state is represented by a sticker appended to the current node, when we interpret the conclusion. The adverb *when* requires that the related nodes be in the same chronoscope. Using *after* as temporal connector, the perfect state is required to have started before action reported in the simple past tense clause started. Using *before/when* as in (3 d) is similar, when preceded with descriptions in perfect tense. But, as evident in (3 e), to the label of the current node you cannot append a sticker describing the state resulting from a later event.

Verifying a conclusion adds the label *PERF* (T) to the current node and searches for a non-empty intersection of the current chronoscope with a chronoscope containing a left descendant node labelled T. The DAT representation suggests a rather simple systematic and always terminating search-algorithm: back up to the first node dominating the current node and see whether it is contained in any chronoscope containing the desired node; if so, the conclusion is verified, if not, back up to the next higher node in the current chronoscope, repeat until the entire DAT is searched. If you have not been able to verify the conclusion, the argument is invalid. In verifying (3 f-i), the current node is labelled with the information that the police arrived. All that needs verifying is that it is a right descendant of a common ancestor of the node representing the killing (3 f). The causal consequences of that node, i.e. the state of the driver being dead, are stickers portable within the current chronoscope (3 g), and the current chronoscope intersects with the chronoscope containing the node that the car hit the fence (3 h). For the gerundive perfect conclusion in (3 i) we need to verify that the perfect type labels a node in the current chronoscope. For *PROG* (T) conclusions, it is required that an ancestor of the current node, i.e. in the current chronoscope, is labelled T or *PROG* (T) again.

These examples illustrated how inferences are made using a DAT representation of the information given in the premises. Reasoning with a DAT consists of systematic operations or procedures to verify conclusions from its current node. This operationalization of inferential processes clarifies what is gained by such graphic representations of information over pure symbolic logical form representations. The form of a DAT and the location of its current node steers the verification of the conclusion by providing a systematic search procedure, guaranteed to terminate either in a verification or in a rejection of the conclusion. Drawing a conclusion never add a new node to the given DAT for its premises, but the types corresponding to the conclusion is appended to the current node from which the verification departs. In the next section the intended semantic interpretation of DATs is presented in terms of embeddings into event-structures. This leads to the proper characterization of situated entailment as a non-monotonic relation between premises and conclusion of an inference in terms of the embeddings of the DAT that represents the information contained in the premises.

In order to consider DATs as structured objects in their own right, we should determine what forms they may take by specifying the syntactic conditions on their wellformedness. DATs are structured semantic objects constrained by certain formation principles; they are not expressions of any language that encode the 'logical form' of the natural language input into a language designed to represent its meaning.

Types that label the nodes of a DAT are formed by the following rules, recursively specifying the class of types *TYPE*.

Definition of TYPE

(i) T is a *basic* type in *TYPE* iff. T is a sequence of an n-ary relation $\boldsymbol{R}$, n objects $\boldsymbol{a_1},..., \boldsymbol{a_n}$, and a positive or negative polarity + or -.

(ii) T is a *parametric* or *parametrized* type in *TYPE* iff. T is a basic type in which a relation or an object is replaced by a relation parameter R or an object parametera_i. respectively.

(iii) if T is in *TYPE* and x is an object parameter then x_T is a restricted object parameter. If T contains x_T , T is a *restricted parametrized type* in *TYPE*.

(iv) if x is a parameter and T is in *TYPE* , then $[x \mid T\ ]$ is a *role*. If T is a restricted parametrized type, all parameters in the restriction must occur in the role-type to the left of |.

(v) if T and T' are in *TYPE*, then so is the *conjoined type* «T &T' » and the *conditional type* «$T \Rightarrow T'$ ».

(vi) closure.

There is a fundamental notion, encountered already above, of *compatibility* of types. Any two types are compatible iff. there is a situation that supports both within the same chronoscope. Obviously, any type is compatible with itself, and incompatible with its negative counterpart with a negative polarity or any of its entailments. Knowledge of the world supplies a host of basic (in)compatibility relations between types, that may be encoded as constraints in the lexicon. Two sets of types are compatible iff. their union is compatible; and a single type T is compatible with a set S of types just in case $S \cup \{T\}$ is compatible.

A DAT consists of :

(i) a finite set of nodes, $N = \{n, n', ...,n_m\}$

(ii) a function δ_N from N to N^*, where N^* is the set of non-repeating finite sequences of N, assigning to each node n a sequence of nodes, its children or immediate dependents $\delta_N(n)$.

(iii) a function π_N from N to N, assigning an arrow pointing to a node n from its immediately dominating node, its parent $\pi_N(n)$.

(iv) a subset H_N of N, the *H*oles; $N - H_N$ is the set P_N of plugs.

(v) a function α_N from N to the powerset of *TYPE*, assigning to each node a set of types.

(vi) a distinguished node c_N in N, the current node, and another distinguished node s_N in P_N, the source node.

such that

1. $\forall n, n'$ in N, n is in $\delta_N(n')$ iff. $\pi_N(n) = n'$
2. There is one and only one node, the root, that is its own ancestor (n is an ancestor of n' iff.$\exists n_1, ...,n_m$ $(m \geq 2)$ such that $n_i = \pi_N(n_{i+1})$, $n = n_1$ and $n' = n_m$).
3. The source node is the terminal plug of the right most chronoscope.
4. The set of all types labelling the ancestors of a node n (i.e. $\bigcup\{\alpha_N(n') \mid n'$ is an ancestor of $n\}$) is compatible with those types labelling n itself.

Information accumulation is simulated by a growing DAT. It is defined as an ordering on DATs as follows: $D_1 < D_2$ = $D_1 \cup \{c_{D2}\}$ and either
1. (Hole rule) c_{D1} is in H_{D1} and each of δ_{D2} ,π_{D2} , H_{D2} and α_{D2} extends the corresponding function of D_1 with the single exception that δ_{D2} (c_{D1})= < c_{D2} > , or
2. (Plug rule) c_{D1} is in P_{D1} and each of δ_{D2} ,π_{D2} , H_{D2} and α_{D2} extends the corresponding function of D_1 with the single exception that δ_{D2} (π_{D1} (c_{D1})) is obtained by appending c_{D2} to the end of δ_{D1} (π_{D1} (c_{D1})), or
3. (Filler rule) α_{D2} (c_{D2}) is incompatible with the types assigned to the ancestors of c_{D1} ; and $\exists$ $D'_1 < D_2$ with the same nodes as D_1 and the same functions δ ,π and α but with $c_{D'1}$ as ancestor of c_{D1} and $c_{D'1}$ is not in $H_{D'1}$.
4. (Sticker rule) c_{D1} is in N_{D1} and each of δ_{D2} ,π_{D2} , N_{D2} is identical to its corresponding function of D_1, but α_{D2} extends the corresponding function of D_1 and c_{D1} = c_{D2} , if c_{D1} is in P_{D1} , and otherwise α_{D2} is extended into α_{D3} after application of the hole or filler rule.

Define $\ll$ to be the transitive closure of $<$, i.e. $D \ll D'$ iff. there are D_1 ,..., D_n such that $D_i < D_{i+1}$,$D = D_1$ and $D' = D_n$.
Let 0 be the DAT with a single node o, δ_0 (o) = < o >, π_0 (o) = o, H_0 (o) = o, c_0 (o) = o, α_0 (o) = $\emptyset$. The class of *wellformed* DATs consists of only those DATs D such that $0 \ll D$.

DATs are intended to be interpreted in event-structures, consisting of events with their natural temporal inclusion and precedence ordering and constrained by some additional temporal conditions.
A *DAT-frame* consists of a set of events E ordered by temporal inclusion $\rightarrow$ ($x \rightarrow y$ - y is a temporal part of x) and temporal precedence < ($x < y$ - x occurs before y) , with an assignment to each T in *TYPE*, of a set of events [[T]] , the *extension* of T, such that the temporal inclusion is a partial order, precedence is a strict partial order and their interaction is constrained by:
- *monotonicity*: if $y \rightarrow x$ and $y < z$ then $x < z$
- *convexity*: if $x < y < z$ and $u \rightarrow x$ and $u \rightarrow z$ then $u \rightarrow y$

DATs are interpreted in such event frames by embeddings, mapping nodes to events preserving the temporal relations and satisfying certain additional conditions.

Definition of embedding
A function f mapping a DAT into a model M is an *embedding* when:
(i) for every arrow π_N (n) $\rightarrow n$, $f(\pi_N (n)) \rightarrow f(n)$
(ii) if n commands n' , then $f(n) < f(n')$
(iii) $f(n) \models T$, where T is the type labelling n.

The event-structures that are suitable models for temporal reasoning must in addition satisfy at least the following constraints.

Semantic constraints for temporal reasoning

(1) maximality of event-types
if $e \models T$ and e is part of e' and $e' \models T$ then $e = e'$
(2) downwards persistence for states/stickers
if $e \models T$, T is a sticker and e ' is part of e ,then $e' \models T$
(3) if T is part of T' then$\forall e$ if $e \models PROG(T)$,
then $e \models PROG(T')$
(4) $e \models PERF(T)$ iff. $\exists e'$ $< e$, $e' \models T$
(5) if $e \models T$ and e ' is part of e then e ' $\models PROG(T)$
(6) if $e \models PROG(T)$ and T is classified as an ACTIVITY,
then $\exists e$ ' ,e '$\rightarrow e$ and e '$\models T$
(7) if $e < e$ ' then $e <$ start(e ')
(8) $e \models$ ***START*** (T) iff. $\exists e'$ $e' \models T$ and e = start(e')
(9) $\forall e$ if$e \models PROG(T)$ then $e \models PERF($***START*** $(T))$

Maximality (1) ensures that events are individuated as individuals, packing the largest chunk of the event-type. The downwards persistence of stickers intuitively corresponds to the fact that any part of a state of a certain type must be a state of that same type. This condition has received a number of different names in the linguistic literature, of which 'homogeneity condition' and 'downward closure' are most familiar.The third constraint requires that the natural part whole relation between types is preserved between events where such types are going on. The fourth constraint requires that a perfect state is preceded by the event which caused it. The fifth one ensures that any part of an event of type T is one at which T is in progress, with the *caveat* , that this means only that it has been started and not yet been ended or finished. In such event structures interruptions of ongoing actions, i.e. transitions from stopping to resuming the action, are admitted, without affecting the support of the corresponding progressive stickers.

The sixth constraint requires a bit more discussion, for it ensures that the imperfective 'paradox' puzzle is resolved: if an event supports a progressive sticker embedding an activity type, a transition from *start* to *end*, any part of it supports the corresponding simple type. This classical puzzle for any account of the semantics of tense and aspect is simply resolved by requiring DATs to be embedded in event-structures that meet this semantic constraint. Given the definition of situated entailment, a simple past tense conclusion cannot be a situated entailment from merely a progressive past premise. The vast body of literature on the so-called "imperfective paradox" has been concerned with static logical entailment based only on truth-conditions, often defined in an interval semantics. This puzzle - for it is by no means paradoxical - is illustrated in (4).

(4)
a. Jane was wandering around => Jane wandered around
b. Jane was reading this book ≠> Jane read this book

In DAT-representation the semantics of the progressive is not defined compositionally in terms of the interpretation of the embedded event-type, as is commonly attempted in possible world semantics, resorting to completions or culminations of accomplishments in inertia worlds where actual interruptions had not taken place. This modal line of accounting for the imperfective paradox is fraught with difficulties in explaining just how much variation to permit in the relevant inertia worlds on the way the event is continued, or in the past leading up to the actual one. Any long term process or action must allow different 'natural' interruptions, often necessary to satisfy the presuppositions of continuing the action. Writing a book certainly requires interruptions, stopping to rest, in order to be able to resume and ultimately finish it. Ordinarily, one would not have written a different book, if an interruption had taken place somewhat earlier than it did or not at all. Interruptions are disregarded for the internal constituency of an event. We ordinarily measure duration of the event after its completion by gauging the time elapsed between its unique start and finish. Of course, refinement of perspective and the corresponding internal structure of the representation of that event, may provide always more detailed information and more precise estimate of how much time was actually spent writing.
The remaining three constraints are concerned with the internal structure of events. The one in (7) requires the obvious relation between precedence and starting point; (8) ensures the relation between start-events and the aspectual verb *start*. The last constraint (9) simply requires that an

event in progress must have been started - an assumption discussed in the previous chapter.

A sequence of sentences or text is *interpretable* in an event-model iff. there is an embedding of a DAT, constructed by the rules in an interpretation of this text, into that event-model. The text *describes* part of the model via the embedding of its DAT; defined below.

Let D be a DAT and f an embedding of it into an event structure, D *describes* f iff.
for all nodes n in D and all types T
if T labels n in D then $f(n) \models T$ in the event structure

That part of the event-model providing values for the described embedding is called the *described episode.* A text is *true* just in case the episode the DAT describes is part of the world. A true text is called *factual* ; a text that is interpretable, but not true, is called *fictional.* This allows for the possibility that interpretable texts may consist of factual and fictional parts.

The traditional notion of logical entailment is a static truth-preserving relation between the set of premises and the conclusion of an inference. But we have realized now that the information obtained by interpreting the premisses of an argument is about an entire episode, whereas the conclusion is concerned only with a part of that episode corresponding to the current node of a DAT representing the premises. In reasoning with DATs, the conclusion describes an event via the embedding of the current node in a DAT representing the information obtained by processing its premises. The premises together describe the entire episode, containing the event described by the conclusion. To capture this context-dependence of temporal reasoning, the notion of situated entailment in DATs is defined as follows.

Definition of situated entailment
Let D be a DAT for the premises $T_1, ..., T_n$ and let c be its current node, then $T_1, ..., T_n \vdash T$ iff. for all event-structures S and all embeddings f of D into S, if $T_1, ..., T_n$ describes $f(D)$, then $f(c)$ is of type T.

The DAT-representations mix control information with descriptive information and perspectival information, and capture situated entailment as a relation between a DAT, its current node and its embeddings into event-structures. Mixing such different kinds of information is useful in representing the content of a text.

4. Conclusion

In conclusion, some of the logical issues of DAT-representation are considered. In particular, we need to characterize what kinds of inferences are monotonic, what kinds are not, in terms of DAT-updates and chronoscopes. Since situated inference is a three-place relation between the given information, its DAT-representation and a conclusion, some of the structural rules must be non-monotonic. For instance, if you add a node to a given DAT, the current node is always reset. A situated conclusion valid on the current node in a given DAT need not be valid any more after an update, adding additional information to the premises. Below, (5) contains an illustration of a situated inference which is not preserved after an update, and (6) one which is preserved no matter what additional information is incorporated into its DAT representation.

(5) The car was moving off the road. It hit a fence. The driver was killed
|= The car was moving off the road
(6) The car hit the fence. The driver was killed. |= The car had hit the fence

If we add the information *The car exploded* to the premises in (5), the conclusion is not any longer a valid situated inference at the new plug introduced for the explosion. In general a past progressive conclusion is only portable into a new chronoscope, if either a higher node in the chronoscope is labeled with the progressive sticker, or the descriptive information presupposes or entails that the action described by the progressive is still continuing.
If we add the information *The car exploded* to the premises in (6), the conclusion is still a valid situated inference at the new plug. In general, once a perfect sticker is a valid conclusion at some node in a DAT, any node introduced later may be labeled with the same perfect sticker. Information expressed in the perfect is retained no matter what additional information is represented in the DAT later. Perfect stickers can be imported into any later chronoscope. The chronoscope containing the event that caused the perfect state is a treshold for the domain in which the perfect sticker is freely transmitted. Using the perfect may hence be understood as a natural way to make descriptive information portable and persistent.
In general a simple past conclusion is a valid situated inference if its type labels current node or if it is a simple sticker portable to current node. The portability conditions associated with the different stickers can be considered structural rules of proof, appealing to configurational structure in DAT, coding inferential behavior as a matter of form.
More questions still remain. I provide the following list for future action:
1) How can two DATs be merged? Is that process is just unification?
2) Can situated inference be defined in a multidimensional modal logic? Conjecture: need four dimensions.
3) How do DATs constrain NP anaphora?
4) Is interpretation independent and prior to inference, or do the two processes go hand in hand?
5) How does a DAT differentiate between asserted information, presupposed information and inferred information?

Such questions may inspire others to study our reasoning in natural language in requisite detail, adopting these DAT representations as tools. DATs lend themselves naturally for generalizations to represent information from multiple sources or to study perspectival refinement as a systematic operation on information admitting internal restructuring of given information when certain conditions are satisfied. Some of these issues are addressed in ter Meulen (19950, others remain for future research.

References

Asher, N. and A. Lascarides: 1993, 'Temporal Interpretation, Discourse Relations and Commonsense Entailment', *Linguistics and Philosophy*.. 16.5 ,437-493.

Benthem, J. van : 1983, *The Logic of Time*. Kluwer Academic Publ., Dordrecht.

Cooper, R. : 1986, 'Tense and discourse location in Situation Semantics', *Linguistics and Philosophy* 9, 17-36.

Dowty, D. : 1979, *Word Meaning and Montague Grammar*.Dordrecht: Reidel Publ. Co..

Dowty, D.: 1982, 'Tenses, time adverbials and compositional semantic theory', *Linguistics and Philosophy* 5, 23-33.

Galton, A. : 1984, *The Logic of Aspect. An axiomatic approach*. Oxford: Clarendon.

Hinrichs, E. : 1985, *A compositional semantics for Aktionsarten and NPreference in English*. Ph.D. dissertation, Ohio State University, Dept. of Linguistics.

Hinrichs, E. : 1986, 'Temporal anaphora in discourses of English', *Linguistics and Philosophy* 9, 63-82.

Hinrichs, E.: 1988, 'Tense, quantifiers and contexts', *Computational Linguistics* 14, 3-14.

Kamp, H. : 1979, 'Events, instants and temporal reference', in: Bäuerle, R. et al. (eds.), *Semantics from different points of view*. Berlin: Springer Verlag, 376-417.

Kamp, H.: 1980, 'Some remarks on the logic of change', in Chr. Rohrer (ed.), *Time, Tense and Quantifiers*. Tubingen, Niemeyer Verlag, 135-179.

Kamp, H. and U. Reyle : 1993, *From Discourse to Logic*. Kluwer Academic Publishers, Dordrecht.

Kamp, H. and Chr. Rohrer : 1983, 'Tense in texts', in: Bäuerle, R. et al. (eds.), *Meaning, Use and Interpretation of Language*. Berlin: de Gruyter, 250-269.

Moens, M. and M. Steedman: 1988, 'Temporal ontology and temporal reference', *Computational Linguistics* 2, 15-28.

Mourelatos, A.: 1978, 'Events, states and processes', *Linguistics and Philosophy* 2/3, 415-434.

Parsons, T.: 1989, 'The progressive in English: events, states and processes', *Linguistics and Philosophy* 12, 213-242.

Parsons, T.: 1990, *Events in the semantics of English: a study in subatomic semantics*. MIT Press, Cambridge.

Partee, B. : 1984, 'Nominal and temporal anaphora', *Linguistics and Philosophy* 7, 243-286.

Seligman, J. and A. ter Meulen : 1992, 'Dynamic aspect trees'. *Logic at Work*, CCSOM, U. of Amsterdam. [Kluwer publ forthcoming]

Shoham Y.: 1988, *Reasoning about Change*. MIT Press, Cambridge.

ter Meulen, A. : 1995, *Representing Time in Natural Language. The dynamic interpretation of tense and aspect*. Bradford Books, MIT Press, Cambridge.

Vlach, F.: 1993, 'Temporal adverbials, tenses and the perfect', *Linguistics and Philosophy* 16:3, 231-283.

Webber, B.: 1988, 'Tense as a discourse anaphor', *Computational Linguistics* 14, 61-73.

A Metric and Layered Temporal Logic for Time Granularity, Synchrony and Asynchrony

Angelo Montanari
Dipartimento di Matematica e Informatica, Università di Udine
Via Zanon, 6 - 33100 Udine, Italy - e-mail: montana@dimi.uniud.it

Abstract

The paper presents a logical language for specifying a wide-ranging class of real-time systems: the systems whose components have dynamic behaviour regulated by very different time constants (granular systems). In [6, 23] we proposed a metric and layered temporal logic (MLTL) extending topological temporal logics with contextual and projection operators to deal with *time granularity*. It allows one to build granular system specifications by referring to the 'natural' scale in the specification of any component and by properly constraining interactions between components. However, it confines itself to the treatment of the class of *synchronous* granular systems, whose components are temporally qualified with respect to the same clock, and cannot be used to specify *asynchronous* granular systems, whose components are temporally qualified with respect to different clocks. In this paper we revise and extend MLTL to support the specification of synchronous and asynchronous interactions among the components of granular systems in a uniform logical framework[1].

1 Introduction

Logic-based methods for representing and reasoning about temporal information have proved to be highly beneficial in several areas of software engineering [17, 31]. We are interested in their application during the specification phase of real-time software production. Temporal properties play a major role in the specification of reactive and concurrent software systems that operate in real-time. They constrain the interactions between different components of the system as well as between the system and its environment, and minor changes in the precise timing of interactions may lead to radically different behaviors [22]. *Temporal logic* can be successfully used for modeling and analysing the desired behavior of reactive and concurrent systems. It supports semantic model checking, to verify consistency of specifications, and to check positive and negative examples of system behavior against specifications, as well as pure syntactic deduction, to prove desired properties of the system [11]. Furthermore, it makes it possible a (semi-)automatic generation of programs meeting certain specifications of desired temporal behaviour [18]. Unfortunately, most common representation languages in the area of formal specifications, are inadequate for real-time applications, because they lack an explicit and quantitative representation of time. To cope with real-time aspects several extensions of existing formalisms for modelling temporal information have been proposed in the literature, e.g. [12, 29]. In this paper, we focus on *metric temporal logics* [2, 6, 8, 16, 21], which make it possible to express both qualitative and quantitative temporal properties in a uniform framework, and extend them to deal with a wide-ranging class of real-time systems: the systems whose components have dynamic behaviour regulated by very different - even by order of magnitude - time constants (hereinafter granular systems).

As pointed out in [3], the importance of being able to provide and relate temporal representations at different 'grain levels' of the same reality is widely recognized in several areas of computer science. For instance, time granularity or related concepts have been discussed in [7, 13, 19, 24, 33]. Nevertheless, there is still a lack of a systematic framework for time granularity. In [6, 23], we proposed a temporal logic that extends topological temporal logics with contextual and projection operators to deal with time granularity, and is endowed with a model-theoretic semantics and a sound axiomatization. However, it confines itself to the treatment of the class of *synchronous* granular systems, where each part (component) of a specification (system) is temporally qualified with respect to the same clock, and cannot be used to specify asynchronous real-time systems, where different parts (components) of a specification (system) are temporally qualified with respect to different clocks.

This paper presents a metric and layered temporal logic within which synchronous and asynchronous interactions among the components of granular systems can be modeled in a uniform way [25]. Synchronizing asynchronous processes means relating their different clocks. If we omit relativistic effects, the relation between two

[1] Current address: Institute for Logic Language and Computation, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands - email: angelo@fwi.uva.nl (until November 30, 1994).

clocks is completely defined by setting their relative *phase displacement*. From a model-theoretic point of view, this means that models of asynchronous real-time systems are parametric with respect to phase displacements. As far as asynchronous components are associated with the same temporal domain, their interactions can then be easily modeled by introducing phase displacement parameters. We deal with the general case of interactions between synchronous and asynchronous components associated with different temporal domains.

The paper is organized as follows. Section 2 presents syntax, semantics and axiomatization of the basic temporal logic for synchronous granular systems. Section 3 contrasts synchronous and asynchronous interpretations of assertions, proposing a solution to the so-called *alignment problem* of temporal domains. Section 4 defines syntax, semantics and axiomatization of the extended language. Finally, conclusions provide an assessment of the work, describe on-going research and discuss open issues.

2 The basic logical formalism

The basic language is a many-sorted first order logic augmented with temporal operators and a metric on time [6, 23]. It replaces the 'flat' temporal domain of standard temporal logics with a temporal universe consisting of a finite set of disjoint and differently grained temporal domains. Such a temporal universe identifies the temporal domains relevant to a granular system specification and defines the relations between instants belonging to different domains.

To qualify formulae with respect to the temporal universe, the logic is provided with a *contextual operator* that identifies the domain(s) a given formula refers to. Within each temporal domain, it is then possible to talk about truth and falsity of formulae at time instants different from the current one that is left implicit by means of a *displacement operator*. Furthermore, a *projection operator* can be used to constrain the relationships between formulae associated with differently grained domains. The combined use of these three operators allows one to represent a granular system by properly connecting a set of differently grained formulae. In the simplest case, such a representation consists of the logical composition of a number of formulae referring to different temporal domains. In more complex cases, the projection operator can be used to deal with nested quantifications of differently grained temporal displacements, or to specify the composition of differently grained temporal displacements. Finally, the logic is provided with a *projection rule* that, given the truth value of a formula with respect the domain it explicitly refers to, allows us to constrain its truth value with respect to any other domain.

Let us now briefly describe syntax, semantics and axiomatization of the basic language. An extensive presentation, together with formal definitions and proofs of stated results, can be found in [23, 25].

2.1 The syntax of the language

In this section, we introduce alphabet, terms and formulae of the basic logical language. The alphabet of the language includes *sorts*, *variables*, *constants*, *functions*, *predicates* and *logical constants*. The set of sorts includes two particular sorts: (i) the temporal sort s_T, which denotes the set of values of temporal displacements; (ii) the contextual sort s_C, which denotes the set of domains into which the temporal universe is partitioned. All constant, function and predicate symbols are typed as well as variables. We also assume that constants, functions and predicates are time-dependent, while variables are time-independent. It can be useful, however, to constrain a subset of the set of constants, functions and predicates to be time-independent. The set of logical constants includes the usual propositional connectives $\neg$ and $\supset$, the quantifier $\forall$, the parametrized temporal displacement operator ∇_α, where α is of sort s_T, the parametrized contextual operator ∇^A, where A is a context, and the projection operator $\downarrow$. The syntax of the language is given by inductively defining its terms and formulae. Terms are defined in a mutually recursive fashion as usual. Moreover, we generalize the notion of time independency from constants and functions to terms and assume, for simplicity, that all context terms are time independent. The set of formulae includes atomic formulae, equalities between terms of the same sort, and the formulae $\neg\mathcal{F}$, $\mathcal{F} \supset \mathcal{G}$, $\forall x\mathcal{F}$, $\nabla_\alpha\mathcal{F}$, $\nabla^A\mathcal{F}$, and $\downarrow \mathcal{F}$, where $\mathcal{F}$ and $\mathcal{G}$ are formulae, x is a variable, α is a temporal term, and A is a context term. The other propositional connectives $\wedge, \vee, \equiv$, the existential quantifier $\exists$, the dual operators Δ_α, Δ^A, and $\diamond$, and the shorthands $\top$ (for *true*) and $\bot$ (for *false*) are defined as usual. The displacement operator $\nabla_\alpha\mathcal{F}$ allows us to evaluate the formula $\mathcal{F}$ at time instants at distance α from the current one. If there is not such a time instant, the formula conventionally evaluates to true. The contextual operator ∇^A restricts the evaluation of $\mathcal{F}$ to time instants belonging to the context A only, so that $\nabla^A\mathcal{F}$ conventionally evaluates to true outside the context A. Finally, the projection operator $\downarrow$ allows us to evaluate $\mathcal{F}$ at time instants the current one projects. The formula $\downarrow\mathcal{F}$ evaluates to true if $\mathcal{F}$ is true at each time instant the current instant projects on.

2.2 The semantics of the language

2.2.1 The temporal universe

A temporal universe $\mathcal{T}$ is the union of a *finite* set of disjoint *temporal domains* $\{T_1, .., T_n\}$. The set of domains constitutes a partition of the temporal universe, and it is totally ordered on the basis of the degree of fineness (coarseness) of its elements. Let $\prec$ be such a *granularity* relation. As an example, consider the temporal universe including *years*, *months*, *weeks* and *days*. The domains are ordered by granularity as follows: *years* $\prec$ *months* $\prec$ *weeks* $\prec$ *days*. Formally, a *granularity* relation on $\{T_1, .., T_n\}$ is a total ordering $\prec$ such that $T_i \prec T_j$, for $1 \leq i < n-1$ and $i < j \leq n$. We also define a finer relation on $\{T_1, .., T_n\}$ modeling a natural notion of inclusion between domains. Such a relation allows us to rule out domains like *weeks* whose elements can

'overlap' elements of coarser domains (a week can overlap two consecutive months or years). Let $\sqsupseteq$ be such a *disjointedness* relation. With respect to the previous example, domains are ordered by disjointedness as follows: $years \sqsupseteq months, months \sqsupseteq days, weeks \sqsupseteq days$, plus the pair $x \sqsupseteq x$, for each domain x. Formally, we define the disjointedness relation $\sqsupseteq$ as the reflexive closure of a partial ordering relation $\supset$ on $\{T_1, .., T_n\}$ such that, for all i, j, if $T_i \supset T_j$, then $i < j$.

Each domain T_i is *discrete* and *metric*. Formally, let $D_{T_1}, .., D_{T_n}$ be n (not empty) linearly ordered domains of temporal displacements over $T_1, .., T_n$, respectively. For each D_{T_i} a structure $(D_{T_i}, <, \bar{\oplus}, \bar{}, d_0)$ is defined, where $<$ is a total order, $\bar{\oplus}$ is a binary function of *displacement composition*, $\bar{}$ is a unary function of *inverse displacement*, and d_0 is the *zero displacement* constant. The function $\bar{\oplus}$ is *commutative* and *associative*, and the constant d_0 acts as its *zero element* ($d_\alpha \bar{\oplus} d_0 = d_\alpha$). Furthemore, the composition of a displacement and its inverse results in the zero displacement ($d_\alpha \bar{\oplus} \bar{d_\alpha} = d_0$). Finally, $\bar{\oplus}$ is *injective* ($d_\alpha \bar{\oplus} d_\beta = d_\alpha \bar{\oplus} d_\gamma \supset d_\beta = d_\gamma$) and, for any ordered pair of displacements d_α, d_β, there exists a *difference displacement* d_γ such that $d_\beta \bar{\oplus} d_\gamma = d_\alpha$.

Furthermore, for each domain T_i, we define a *metric relation* $+ : T_i \times D_{T_i} \times T_i$ satisfying the following properties:

1. *quasi-functionality*:
   $\forall i, j, j', d_\alpha((+(i, d_\alpha, j) \wedge +(i, d_\alpha, j')) \supset j = j')$
2. *reflexivity*:
   $\forall i + (i, d_0, i)$
3. *transitivity*:
   $\forall i, j, k, d_\alpha, d_\beta((+(i, d_\alpha, j) \wedge +(j, d_\beta, k)) \supset +(i, d_\alpha \bar{\oplus} d_\beta, k))$
4. *euclidity*:
   $\forall i, j, t_k, d_\alpha, d_\beta((+(i, d_\alpha, j) \wedge +(i, d_\alpha \bar{\oplus} d_\beta, k)) \supset +(j, d_\beta, k))$

The *quasi-functionality* of $+$ with respect to the first two arguments allows us to interpret it as a *partial function*, that is, for all i, d_α, if j exists such that $+(i. d_\alpha, j)$ then it is unique. Moreover, under the further assumptions that $+$ is *totally connected* ($\forall i, j \exists d_\alpha + (i, d_\alpha, j)$) and *quasi-functional* with respect to the first and third arguments ($\forall i, j, d_\alpha, d_\beta(+(i, d_\alpha, j) \wedge +(i, d_\beta, j) \supset d_\alpha = d_\beta)$), an ordering relation $<_{T_i}$ over T_i, based on the ordering relation $<$ over D_{T_i} can be defined as follows:

$$i <_{T_i} j =_{def} +(i, d_\alpha, j) \wedge 0 < d_\alpha$$

In such a case, the quasi-functionality of $+$ with respect to the first two arguments allows us conclude that temporal domains are *linearly ordered*.

Finally, we define a *projection* relation that, for each pair of domains T_i and T_j, with $T_i \prec T_j$, links any instant i in T_i to each of its component j in T_j and any instant j' in T_j to each instant i' in T_i of which it is a component. Formally, for any ordered pair of domain T_i, T_j, and each i in T_i, let $\lceil C_F(i, T_i, T_j) \rceil$ be the number of j in T_j such that $i \rightarrow j$, where C_F is a function with signature $C_F : T_i \times \{T_1, ... T_n\} \times \{T_1. ... T_n\} \rightarrow Q$ called the *conversion factor* between the granularities of T_i and T_j with respect to i. The relation $\rightarrow : \mathcal{T} \times \mathcal{T} = \bigcup_{i,j=1,..,n} T_i \times T_j$ satisfies the following basic properties: (i) every time instant projects on itself (*reflexivity*); (ii) if i downward (upward) projects on j, then j upward (downward) projects on i (*symmetry*); (iii) if $T_i \sqsupseteq T_j \sqsupseteq T_k$ and i of T_i projects on j of T_j and j projects on k of T_k, then i projects on k (*downward transitivity*); (iv) if $T_i \sqsupseteq T_k \sqsupseteq T_j$ and i of T_i projects on j of T_j and j projects on k of T_k, then i projects on k (*downward/upward transitivity*); (v) the 'ordering' of time instants is preserved by the projection relation; more precisely, for each T_i and T_j we require that the projection intervals are 'ordered' but possibly meet, while, for pairs of domains ordered by disjointedness, we require the stronger property that projection intervals are disjoint (*order preservation*); (vi) the projection relation maps an instant into an interval of contiguous instants on a given domain, and from the definition of conversion factor, it follows that such an interval consists of $\lceil C_F(i, T_i, T_j) \rceil$ instants, where $\lceil C_F(i, T_i, T_j) \rceil$ is the conversion factor between T_i and T_j with respect to i (*contiguity*). We can also require that the projection satisfies the property of *homogeneity* stating that, for each pair of disjoint domains of the temporal universe, there exists a *constant* conversion factor expressing the numerical relationship between their granularities. For the sake of simplicity, hereinafter we assume $\rightarrow$ to be homogeneous. However, most results we are going to present can be easily generalized to non-homogeneous temporal structures.

2.2.2 The temporal structure

The semantics of the language is based on the concept of *temporal structure* that allows us to derive the notions of state and valuation function. The *state* is an assignment of suitable values to constants, functions and predicates at each time instant of the temporal universe. The *valuation function* is an assignment of a value to terms and formulae at each time instant of the temporal universe. Formally, a *temporal structure* S is a triplet $S = (\mathcal{D}. \mathcal{T}. \Im)$ where $\mathcal{D}$ is a family of non-empty sets called the *domain of interpretation* including n domains $D_{T_1}, ... D_{T_n}$ over which temporal temporal terms are interpreted and the set of domains of the temporal universe $\{T_1, .., T_n\}$ over which context terms are interpreted; $\mathcal{T}$ is the *temporal universe* over which are defined a projection relation $\rightarrow$ and n relations $+$ (as many temporal domains as there are); $\Im$ is the *interpretation function* that assigns a value to variables x, constants c, functions f and predicates p. The value of a constant c, a function f and a predicate p at a time instant i of $\mathcal{T}$ are denoted by $\Im_i(c), \Im_i(f)$ and $\Im_i(p)$, respectively. The set of function symbols includes the *interpreted* function symbols C_F which denote the conversion factor function. For any i belonging to T_i and any context B such that $\Im(B) = T_j$, $\Im_i(C_F(B)) = C_F(i, T_i, T_j)$. S defines a set of interpretations that differ one from each other in the time instant of the temporal domain they assign to the implicit current instant. On the basis of $\Im$, we can give a value to each term and formula of the language at each time instant i of $\mathcal{T}$:

Definition 2.1 (Satisfiability)

if x is a variable then $\Im_i(x) = \Im(x)$;
if f is a n-ary function and $t_1, .., t_n$ are terms then
$\Im_i(f(t_1, .., t_n)) = \Im_i(f)(\Im_i(t_1), .., \Im_i(t_n))$;
if p is a n-ary predicate and $t_1, .., t_n$ are terms then
$\Im_i(p(t_1, .., t_n)) = true \Longleftrightarrow$
$(\Im_i(t_1), ..., \Im_i(t_n)) \in \Im_i(p)$;
$\Im_i(t_1 =_s t_2) = true \Longleftrightarrow \Im_i(t_1) = \Im_i(t_2)$
where $=$ is the identity relation in D_s;
$\Im_i(\nabla_\alpha \mathcal{F}) = true \Longleftrightarrow \Im_j(\mathcal{F}) = true$
for each j such that $+(i, \Im_i(\alpha), j)$
$\Im_i(\nabla^A \mathcal{F}) = true \Longleftrightarrow$ if $i \in \Im_i(A)$ then $\Im_i(\mathcal{F}) = true$;
$\Im_i(\ \Box\ \mathcal{F}) = true \Longleftrightarrow \Im_j(\mathcal{F}) = true$
for each j such that $i \rightarrow j$.

The cases for the logical connectives and quantifiers are just as in classical first-order predicate logic and therefore are omitted here.

The notions of temporal satisfiability and temporal validity of formulae with respect to a temporal structure are defined as follows. A formula $\mathcal{F}$ is said *locally temporally satisfiable* with respect to a temporal domain T_i of a temporal structure S if and only if S evaluates to true in at least one instant of T_i. A formula $\mathcal{F}$ is said *locally temporally valid* with respect to a temporal domain T_i of a temporal structure S if and only if S evaluates to true in every instant of T_i. A formula $\mathcal{F}$ is said *locally temporally invariant* if and only if it is locally temporally unsatisfiable or locally temporally valid. On the basis of the concepts of local temporal satisfiability and validity, we define the notions of *satisfiability* and *validity* of formulae. A formula $\mathcal{F}$ is said *satisfiable* if and only if there exists a temporal structure with respect to which it is locally temporally satisfiable. A formula $\mathcal{F}$ is said *valid* if and only if it is locally temporally valid with respect to each temporal domain of every temporal structure.

2.3 The axiomatization

A set of axioms and inference rules expressing the properties of temporal operators are added to axioms and inference rules of first order predicate calculus with equality.

The basic properties of the displacement operator are expressed by the following axiom schemata (hereinafter axioms) :

(Ax1) $\nabla_\alpha(\mathcal{F} \supset \mathcal{G}) \supset (\nabla_\alpha \mathcal{F} \supset \nabla_\alpha \mathcal{G})$
(normality of ∇_α)

(Ax2.1) $\forall x \nabla_\alpha \mathcal{F} \supset \nabla_\alpha \forall x \mathcal{F}$ if x is not in α
(Barcan's formula for ∇_α)

(Ax2.2) $\forall x \Delta_\alpha \mathcal{F} \supset \Delta_\alpha \forall x \mathcal{F}$ if x is not in α
(Barcan's formula for Δ_α)

(Ax3) $\Delta_\alpha \mathcal{F} \supset \nabla_\alpha \mathcal{F}$
(quasi-functionality)

(Ax4) $\nabla_0 \mathcal{F} \supset \mathcal{F}$
(reflexivity of ∇_0)

(Ax5.1) $\nabla_{\alpha \oplus \beta} \mathcal{F} \supset \nabla_\alpha \nabla_\beta \mathcal{F}$
(transitivity)

(Ax5.2) $\Delta_\alpha \nabla_\beta \mathcal{F} \supset \nabla_{\alpha \oplus \beta} \mathcal{F}$
(euclidity)

(Ax6) $\alpha \oplus \beta = \beta \oplus \alpha$
(commutativity)

(Ax7) $\alpha \oplus (\beta \oplus \gamma) = (\alpha \oplus \beta) \oplus \gamma$
(associativity)

(Ax8) $\alpha \oplus 0 = \alpha$
(zero displacement)

(Ax9) $\alpha \oplus -\alpha = 0$
(inverse)

(Ax10) $\alpha \oplus \beta = \alpha \oplus \gamma \supset \beta = \gamma$
($\oplus$ is injective)

(Ax11) $\exists \gamma(\alpha = \beta \oplus \gamma)$
(existence of difference)

where x is a variable, 0 is a time-independent constant denoting the 0 element of the metric relation+, α, β and γ are time-independent terms of sort s_T, and $\mathcal{F}$ and $\mathcal{G}$ are formulae, and by the inference rule:

(IR1) $\vdash \mathcal{F} \longrightarrow \vdash \nabla_\alpha \mathcal{F}$
(necessitation rule for ∇_α)

Axioms $Ax2.1$ and $Ax2.2$ state that the interpretation domain does not change under temporal displacement. These axioms should be weakened to deal with the creation and deletion of objects. Axiom $Ax3$ states that if a time instant t at distance $\Im_i(\alpha)$ from the current instant i exists, then such an instant is unique. Axiom $Ax4$ states that the application of a zero displacement to a formula does not change it. Axioms $Ax5.1$ and $Ax5.2$ state that the metric relation is both transitive and euclidean[2]. Finally, axioms $Ax6 - Ax11$ specify the properties of the function $\oplus$.

The basic properties of the contextual and the projection operators are given by the following axioms:

(Ax12) $\nabla^A(\mathcal{F} \supset \mathcal{G}) \supset (\nabla^A \mathcal{F} \supset \nabla^A \mathcal{G})$
(normality of ∇^A)

(Ax13.1) $\forall x \nabla^A \mathcal{F} \supset \nabla^A \forall x \mathcal{F}$ if x is not in A
(Barcan's formula for ∇^A)

(Ax13.2) $\forall x \Delta^A \mathcal{F} \supset \Delta^A \forall x \mathcal{F}$ if x is not in A
(Barcan's formula for Δ^A)

(Ax14) $\Delta^A \mathcal{F} \supset \mathcal{F}$
("necessity" for Δ^A)

(Ax15) $\nabla^A \nabla^A \mathcal{F} \equiv \nabla^A \mathcal{F}$
(idempotency of ∇^A)

(Ax16) $\nabla^A \nabla_\alpha \mathcal{F} \equiv \nabla_\alpha \nabla^A \mathcal{F}$
(commutativity of ∇^A and ∇_α)

(Ax17) $\Box(\mathcal{F} \supset \mathcal{G}) \supset (\Box \mathcal{F} \supset \Box \mathcal{G})$
(normality of $\Box$)

where A is a temporal sort, and by the inference rules:

(IR2) $\vdash \mathcal{F} \longrightarrow \vdash \nabla^A \mathcal{F}$
(necessitation rule for ∇_A)

(IR3) $\vdash \mathcal{F} \longrightarrow \vdash \Box \mathcal{F}$
(necessitation rule for $\Box$)

Axioms $Ax13.1$ and $Ax13.2$ state that the interpretation domain does not change under temporal contextualization, i.e. it is context independent. Again, to deal with visibility and invisibility of objects in the different contexts, these axioms should be weakened. Such an ability would allow us to prevent the visibility of both a compound object and its components with respect to some

[2]Together with axiom $Ax3$ (quasi-functionality) they allow us to conclude that temporal displacements are compositional, provided that there exists a time instant α units from the current one, that is, $\Delta_\alpha \top \supset \nabla_\alpha \nabla_\beta \mathcal{F} \equiv \nabla_{\alpha \oplus \beta} \mathcal{F}$ (vector addition for ∇_α).

contexts. Axiom $Ax15$ provides us with a *reduction rule* for contextual operators.

Besides the basic logical properties of contextual and projection operators, we can axiomatize the properties of the temporal universe (temporal universe partition, properties of conversion factors, properties of the projection relation). Let us report here the axioms for the temporal universe partition and for the properties of the projection relation:

(Ax18) $\mathcal{F} \supset \exists A \Delta^A \mathcal{F}$
(domain covering)

(Ax19) $\forall A, B(\Delta^A \Delta^B \top \supset A = B)$
(domain disjointedness)

(Ax20) $\Box \mathcal{F} \supset \mathcal{F}$
(reflexivity of $\Box$)

(Ax21) $\mathcal{F} \supset \Box \diamond \mathcal{F}$
(symmetry)

(Ax22) $\forall A, B, C((C \sqsupseteq B \sqsupseteq A \wedge \nabla^A \Box \nabla^C \mathcal{F}) \supset \nabla^A \Box \nabla^B \Box \nabla^C \mathcal{F})$
(downward transitivity)

(Ax23) $\forall A, B, C((A \sqsupseteq C \sqsupseteq B \wedge \nabla^A \Box \nabla^C \mathcal{F}) \supset \nabla^A \Box \nabla^B \Box \nabla^C \mathcal{F})$
(downward/upward transitivity)

(Ax24) $\forall A, B(\exists x(x > 0 \wedge \Delta^A_x \diamond \Delta^B \mathcal{F}) \supset \nabla^A \Box \nabla^B \exists y(y \geq 0 \wedge \nabla_y \mathcal{F}))$
(weak order preservation)

(Ax24') $\forall A, B((A \sqsupseteq B \wedge \exists x(x > 0 \wedge \Delta^A_x \diamond \Delta^B \mathcal{F})) \supset \nabla^A \Box \nabla^B \exists y(y > 0 \wedge \nabla_y \mathcal{F}))$
(strong order preservation)

(Ax25) $\forall A, B \nabla^A \exists z(z = \lceil C_F(B) \rceil \wedge (\diamond \Delta^B \forall x(0 \leq x < z \supset \nabla_x \mathcal{F}) \equiv \exists y(0 \leq y < z \wedge \Box \nabla^B \nabla_y \mathcal{F})))$
(contiguity)

(Ax26) $\forall A, B \exists z \forall x \nabla^A_x \ (C_F(B) = z)$
(homogeneity)

where x, y and z are variables, A, B, and C are contexts, $C_F(B)$ is a time-dependent function denoting the conversion factors, and $\mathcal{F}$ and $\mathcal{G}$ are formulae.

Finally, we introduce upward and downward *projection rules*. For each pair of domains T_i, T_j, with T_i coarser than T_j, the *downward projection rule* states that if a property P holds at a time instant i of T_i, then there exists at least one time instant j of T_j, belonging to its decomposition, such that P holds at j. For each pair of domains T_i, T_j, with T_i finer than T_j, the *upward projection rule* states that if a property P holds at each time i of T_i such that $j \rightarrow i$, then P holds at time j. Formally, *downward projection* is defined by the following axiom:

(Ax27) $\forall A, B(A \sqsupseteq B \supset \nabla^A(\mathcal{F} \supset \diamond \Delta^B \mathcal{F}))$

where $\mathcal{F}$ is a first order formula, i.e. a formula devoid of displacement, contextual and projection operators. It is easy to show the equivalence between this axiom and the definition of *upward projection*:

$$\forall A, B(A \sqsupseteq B \supset \nabla^A(\Box \nabla^B \mathcal{F} \supset \mathcal{F}))$$

This allows us to conclude that the axioms defining downward and upward projection are *interdeducible*. Downward projection rule provides the *weakest semantics* that can be attached to an assertion in a domain finer than the original one, provided that such an assertion is not wholistic[3]. Most often it is too weak so that user qualifications are needed. In the next section, we will see that asynchronous systems are characterized by the stronger semantics they assign to the projected formulae. Furthermore, the approach to the representation of asynchronous systems we are going to present will suggest how to support domain-specific categorizations of assertions according to their behaviour under downward temporal projection.

In [23] we have proved the soundness of the axiomatization by showing that each axiom is S-valid and that inference rules preserve S-validity. In the case of axioms characterizing the structure of the temporal universe we have actually shown that they are temporally valid in every temporal structure $S_F(\mathcal{D}, \mathcal{T}, \Im)$ based on the frame $F = (\mathcal{D}, \mathcal{T})$ if and only if F satisfies the corresponding property (characterization theorems [20]).

3 Dealing with the alignment problem

A major problem in granular system specifications is the *alignment problem* of temporal domains [9]. Consider the case of a system consisting of a remote system R and a monitor M which is in charge of monitoring the state of R [25]. The monitoring procedure requires R to periodically send a state message *msg* to M. As long as R is in a correct state, M must receive a message from R every hour. However, different interpretations of this temporal constraint with respect to domains finer than the domain of hours result into different monitoring policies. Let us consider, for instance, the domain of minutes. A first possible interpretation of the constraint is that there will be a minute, starting from the first minute of the next hour until the last one, when it will be true that M receives the message from R, *no matter which minute of the present hour the constraint is associated with.* Thus, if the constraint is associated with the first minute of the current hour, it will be true that M receives the message from R at time t whose distance d from the current instant is such that $60 \leq d < 119$, if it is evaluated at the second minute the range of values for d becomes $59 \leq d < 118$, and so on. A second possible interpretation of the constraint is that M receives the message from R 60 minutes after the minute the constraint is associated with. Thus, if the sentence is claimed at minute, say, 10, or 55, of a given hour, always it will be true that M receives the message from R at time t whose distance d from such a minute is *exactly* 60 minutes. We call interpretations of the first and second type *synchronous* and *asynchronous*, respectively. They are indistinguishable with respect to the domain the formula explicitly refers to (and to all coarser domain), but differ from each other when the formula is projected on any finer domain. In the synchronous case, if the projected formula is true with respect to a given

[3] Wholistic assertions relate to the structure of the interval over which they hold as a whole, and they do not hold over any proper subinterval of it [30]. Such assertions model phenomena that consume a certain time interval and cannot possibly transpire during any subinterval thereof. They cannot be projected across domains using axiom $Ax27$.

instant belonging to the projection interval of the current instant, then it is true with respect to any instant of such an interval, while, in the asynchronous case, the projected formula can be true with respect to a given instant of the projection interval and false with respect to all the others[4]. Formally, this means that asynchronous models are just a proper subset of synchronous ones.

The basic specification formalism deals with synchronous interpretation only. Supporting asynchronous interpretations mainly requires two extensions: (i) replacing the notion of current instant with the notion of *vector of current instants*; (ii) defining an *asynchronous projection operator* that for each domain T_i, with $i = 1,..,n$, maps the current instant of T_i into the current instant of each T_j, with $j = 1,..,n$. The notion of vector of current instants deserves a detailed analysis. Given a temporal universe $\mathcal{T} = \bigcup_{i=1,..,n} T_i$, any element $v[i]$ of a vector of current instants $v[1,..,n]$ is a time instant of T_i belonging to the projection interval of $v[j]$ on T_i, for each $j = 1,..,i-1$. On the basis of the notion of vector of current instants, for any i, j, with $1 \leq i, j \leq n$, it is possible to define the *phase displacement* between two domains T_i and T_j with respect to the current instant $v[i]$ of T_i as the temporal distance between the first element of the projection interval of $v[i]$ on T_j and $v[j]$. Such a notion allows us to *replace* the vector of current instants by *n vectors of phase displacements*, one for each possible choice of a specific current instant from the vector of current instants. More formally, for each $i = 1,..,n$, we define a vector of phase displacements $\Phi_i[1,..,n]$ such that, for each $j = 1,..,n$, $\Phi_i[j]$ is equal to the phase displacement between T_i and T_j with respect to $v[i]$. Moreover, for each i, j, with $1 \leq i, j \leq n$, we define a *phase displacement* function $P_D : T_i \times \{T_1,..,T_n\} \times \{T_1,..,T_n\} \rightarrow D_{T_j}$ that for each current instant $v[i]$ belonging to T_i and each domain T_j returns the phase displacement between T_i and T_j with respect to $v[i]$. From the homogeneity of $\rightarrow$, it follows that $P_D(v[i], T_i, T_j)$ does not depend on $v[i]$, i.e. there exists a unique phase displacement for any ordered pair of domains. The n vectors of phase displacements are clearly not independent. In the case of a temporal universe totally ordered with respect to the disjointedness relationship $\sqsupseteq$, for each pair of distinct domains T_i, T_j, with $T_i \sqsupseteq T_j$, and each domain T_k, their relationships are expressed by the following conditions:

a. if $T_k(\sqsupseteq \text{ or } =)T_i$ then $\Phi_i[k] = \Phi_j[k] = 0$

b. if $T_i \sqsupseteq T_k$ and $T_k(\sqsupseteq \text{ or } =)T_j$ then $\Phi_i[k] \geq 0$ and $\Phi_j[k] = 0$

c. if $T_j \sqsupseteq T_k$ then $\Phi_i[k] = \Phi_i[j] \cdot C_F(v[j], T_j, T_k) + \Phi_j[k]$

[4]To prevent misinterpretations, it is worth noting that the distinction between synchronous and asynchronous interpretations is not related at all to the ordering according to which displacement and projection operators are applied. This means that formulae can be first translated of the specified displacement over the original domain and then projected or they can be first projected and then translated of a distance equal to the fine grain equivalent of the specified displacement. Formally, this property can be stated by saying that, for both synchronous and asynchronous interpretations. *projection and displacement operators*, modulo the conversion factor. *commute*.

Dealing with n vectors of phase displacements instead of with a vector of current instants makes it possible to keep the semantics of the extended language simpler and closer to the original one. Instead of interpreting formulae with respect to a vector of current instants, it allows us to continue to interpret them with respect to a single current instant taken from such a vector, provided that the corresponding n vectors of phase displacements are given.

Finally, let $v[1,..,n]$ the vector of current instants, $v[i]$ belonging to T_i the current instant with respect to which a given formula is interpreted, and $\Phi_i[1,..,n]$ the vector of phase displacements associated with $v[i]$. The execution of a *(non-zero) displacement* within the current domain modifies all the current instants of the finer domains and possibly the current instants of the coarser ones, but leaves the n vectors of phase displacements unchanged. Then, the new vector of current instants can be easily determined on the basis of the new current instant and the given phase displacement function. It is worth noting that if the hypothesis of $\rightarrow$ homogeneity is removed, this invariance under displacement of the vectors of phase displacements is no more guaranteed. The execution of an *asynchronous projection* from T_i to T_j changes the temporal context and forces $v[j]$ to become the new current instant with respect to which the projected formula is to be interpreted and to replace $\Phi_i[1,..,n]$ by $\Phi_j[1,..,n]$.

4 A unifying formalism

In this section, we extend the basic formalism to deal with synchronous and asynchronous granular systems in a uniform specification framework. We provide two alternative definitions of the asynchronous projection operator; furthermore, we show how both synchronous and asynchronous projection operators can be defined in terms of a simpler projection operator, namely, the alignment projection operator.

4.1 The extended syntax

First of all, the alphabet of the basic language is extended with a *mode* sort s_M denoting the two possible interpretations of assertions, namely synchronous and asynchronous. To explicitly refer to synchronous and asynchronous modes, we introduce the time-independent constants s and a of sort s_M, respectively. Mode terms come into play as a second parameter of the contextual operator. The first parameter C of the contextual operator $\nabla^{C,M}\mathcal{F}$ still denotes the temporal domain the formula $\mathcal{F}$ refers to; the second parameter M of sort s_M constrains the projection of the formula $\mathcal{F}$ across domains. In case this second parameter is missing, the synchronous interpretation is taken as default. Furthermore, we add an interpreted function symbol P_D to denote the phase displacement function. Finally, a new operator $\square_{\phi}$, called the *asynchronous projection operator*, is added that allows us to evaluate a formula $\mathcal{F}$ with respect to a vector of current instants $v[1,...n]$. For any vector of current instants $v[1,...n]$ and any i, with $1 \leq i \leq n$, it projects the current instant $v[i]$ on each current instant $v[j]$ belonging to $v[1...n]$. Formally, the

formula $\Box_\phi \mathcal{F}$ evaluates to true at the current instant $v[i]$ of T_i if and only if $\mathcal{F}$ evaluates to true at the current instant $v[j]$ of T_j, for $j = 1,..,n$. The dual operator $\Diamond_\phi$ is defined as usual. The *asynchronous projection operator* $\Box_\phi$ can be defined in terms of the displacement, contextual, and projection operators as follows:

Definition 4.1

$$\Box_\phi \mathcal{F} =_{def} \forall B \exists y, z (y = \lceil C_F(B) \rceil \wedge z = P_D(B) \wedge 0 \leq z < y \wedge y \wedge \Diamond \Delta^B (\Delta_z \mathcal{F} \wedge \forall x (O \leq x < y \supset \Delta_x \mathcal{G})) \wedge \nabla_1 \Box \nabla^B \neg \mathcal{G})$$

where P_D is a time-dependent function denoting phase displacements and $\mathcal{G}$ is a syntactically univocal atomic formula.

Let us consider now the particular case in which $P_D(v[i], T_i, T_j)$ $(= \Phi_i[j])$ is equal to 0, for each i, j (*synchronization of current instants*). In such a case, for any $i = 1,..,n$, the asynchronous projection operator acts as an alignment projection operator mapping the current instant $v[i]$ on the first instant of its projection intervals on $T_1,..,T_n$. Obviously, as soon as the vector of current instants changes and the elements of the new vector are no more synchronized, the behaviors of the asynchronous and alignment projection operators become different. However, we can introduce a new projection operator $\Box_0$, called the *alignment projection operator*, that maps the current instant i on the first instant of its projection interval on T_j, for $j = 1,..,n$, and redefine the *asynchronous projection operator* by means of this new operator, the displacement operator and the contextual one. Furthermore, it is possible to show that also the synchronous projection operator $\Box$ can be redefined in terms of displacement, contextual, and alignment projection operators. This means that we could take the displacement, contextual, and aligned projection operators as primitive and deriving both the synchronous and asynchronous projection operators from them. We are currently studying and comparing these alternative definitions of our logic.

The alignment projection operator is defined as follows:

Definition 4.2

$$\Box_0 \mathcal{F} =_{def} \forall B \exists y (y = \lceil C_F(B) \rceil \wedge \Diamond \Delta^B (\mathcal{F} \wedge \forall x (0 \leq x < y \supset \Delta_x \mathcal{G})) \wedge \nabla_1 \Box \nabla^B \neg \mathcal{G})$$

As $\Box_\phi$, $\Box_0$ is defined in terms of the displacement, contextual, and projection operators, but differently from the definition of $\Box_\phi$, its definition does not involve the time-dependent function P_D. Formally, the formula $\Box_0 \mathcal{F}$ evaluates to true at the current instant i of T_i if and only if $\mathcal{F}$ evaluates to true at the first instant of the projection interval of i on T_j, for $j = 1,..,n$. The dual operator $\Diamond_0 \mathcal{F}$ is defined as usual. The asynchronous projection operator can then be defined in terms of the alignment as follows:

Definition 4.1'

$$\Box_\phi \mathcal{F} =_{def} \forall B \exists y, z (y = \lceil C_F(B) \rceil \wedge z = P_D(B) \wedge 0 \leq z < y \wedge \Box_0 \nabla^B \nabla_z \mathcal{F})$$

Furthermore, if we take the alignment projection operator $\Box_0$ as primitive and redefine the conversion factor function C_F in terms of it, we can define the synchronous projection operator as follows:

Definition 4.3

$$\Box \mathcal{F} =_{def} \forall B \exists y (y = \lceil C_F(B) \rceil \wedge \forall x (0 \leq x < y \supset \Box_0 \nabla^B \nabla_x \mathcal{F}))$$

Both the proposed definitions of $\Box_\phi$ make use of the time-dependent function P_D, which is needed to determine the value of the phase displacement between the domain the current instant belong to and the projection domain. Even if it denotes a displacement over the projection domain, it must be evaluated with respect to the current instant and, then, before that the projection takes place. This context-dependency of phase displacement evaluation makes definitions difficult to read, because it requests us to introduce variables to "carry" the values of phase displacements from the current domain to the projection one. The use of the P_D function can be avoided in the case of contextualized projections only. When both the original and the projection domain are univocally determined, we can introduce a suitable variable (or, equivalently, a time-independent constant) to denote their phase displacement. In general, we need a distinct variable $\phi_{i,j}$ for each phase displacement $\Phi_i[j]$, with $1 \leq i, j \leq n$. For each pair of contexts A, B, we can define a contextualized asynchronous projection operator $\Box_\phi^{A,B}$ in the following way:

Definition 4.4

$$\Box_\phi^{A,B} \mathcal{F} =_{def} \nabla^A (\phi_{A,B} < \lceil C_F(B) \rceil \wedge \Box_0 \nabla^B \nabla_{\phi_{A,B}} \mathcal{F})$$

where $\phi_{A,B}$ is a variable denoting $\Phi_i[j]$ (with $\Im(A) = T_i$ and $\Im(B) = T_j$). The dual operator $\Diamond_\phi^{A,B}$ is defined as usual.

Finally, the idea of contextualized projections can be immediately generalized to support the projection on a specific instant of the projection interval different from the first one. For instance, to map the current hour into its sixteenth minute, we can use the projection operator $\Box_{15}^{A,B}$ defined as follows[5]:

Definition 4.5

$$\Box_{15}^{hour,minute} \mathcal{F} =_{def} \nabla^{hour} \Box_0 \nabla^{minute} \nabla_{15} \mathcal{F}$$

In [25], we present in detail the application of the extended language to the specification of a granular monitoring system.

4.2 The extended semantics

The semantics of the extended language is still based on the concept of temporal structure. The only difference with respect to the basic language is that state and valuation functions are evaluated with respect to a vector of time instants rather than at a time instant. Nevertheless, given the correspondence between vectors of time instants and vectors of phase displacements, the

[5] It is worth noting that specifiying a numerical value for the displacement from the beginning of the projection interval makes sense only if the projection domain is univocally determined, given that the same numerical value denotes a different displacement with respect to different domains [25].

interpretation $\Im$ of a formula $\mathcal{F}$ with respect to a vector of current instants $v = [1,..,n]$ is defined in terms of the interpretation $\Im$ of $\mathcal{F}$ at a single current instant of such a vector, provided that, for any instant i belonging to T_i and any context B such that $\Im(B) = T_j$, $\Im_i(P_D(B)) = \Phi_i[j]$. In such a way, we provide the extended language with a model-theoretic semantics which is *parametric* with respect to the *vectors of phase displacements*. In particular, this allows us to leave the notions of satisfiability and validity unchanged.

Let us now formally assign a meaning to the new terms and formulae of the language. First of all, for any i belonging to T_i and any context B such that $\Im(B) = T_j$, $\Im_i(P_D(B)) = P_D(i,T_i,T_j) = \Phi_i[j]$. Under the hypothesis of $\rightarrow$ homogeneity, P_D is domain-dependent, rather than time-dependent, and then it does not depends on the instant at which it is interpreted, but only on the domain such an instant belongs to. Furthermore, the semantics of the variables involved in the definitions of contextualized projection operators is given by the clause $\Im_i(\phi_{A,B}) = P_D(j,T_j,T_k) = \Phi_j[k]$, provided that $\Im(A) = T_j$ and $\Im(B) = T_k$. With regard to the operators, the mode characterization of contextual operators does not affect their semantics, but only constrains the applicability of projection rules. The semantics of the alignment and asynchronous projection operators is based on the functions *first* and *last*. For each ordered pair of disjoint domains T_i and T_j and each i in T_i, the functions $first : T_i \times \mathcal{T} \times \mathcal{T} \rightarrow T_j$ and $last : T_i \times \mathcal{T} \times \mathcal{T} \rightarrow T_j$ return the minimum and the maximum of the set of elements j in T_j such that i projects on j, respectively. Since $\rightarrow$ is *contiguous*, such functions can be formally defined as follows:

Definition 4.6

$first(i,T_i,T_j) = j$ iff
$i \in T_i \wedge j \in T_j \wedge i \rightarrow j \wedge \forall d_\alpha (0 \leq d_\alpha < \lceil C_F(i,T_i,T_j) \rceil \supset$
$\exists j' (j' \in T_j \wedge +(j, d_\alpha, j') \wedge i \rightarrow j')) \wedge \forall j' ((j' \in T_j \wedge$
$i \rightarrow j') \supset \exists d_\alpha (0 \leq d_\alpha < \lceil C_F(i,T_i,T_j) \rceil \wedge +(j, d_\alpha, j')))$

Definition 4.7

$last(i,T_i,T_j) = j$ iff
$i \in T_i \wedge j \in T_j \wedge i \rightarrow j \wedge \forall d_\alpha (0 \leq d_\alpha < \lceil C_F(i,T_i,T_j) \rceil \supset$
$\exists j' (j' \in T_j \wedge +(j, -d_\alpha, j') \wedge i \rightarrow j')) \wedge \forall j' ((j' \in T_j \wedge$
$i \rightarrow j') \supset \exists d_\alpha (0 \leq d_\alpha < \lceil C_F(i,T_i,T_j) \rceil \wedge +(j, -d_\alpha, j')))$

where $C_F(i,T_i,T_j)$ is the conversion factor between T_i and T_j with respect to i, respectively. Let us consider now *Definition 4.1'* of $\Box_\phi$. To prove that it actually captures the intended meaning of the asynchronous projection operator, we need to state some preliminary results. We first prove two lemmas about the relationships between *first* and *last* functions; then, we use such lemmas to show that *Definition 4.2* properly expresses the intended meaning of $\Box_0$. The proof for $\Box_\phi$ easily follows from this last result. First of all, from *Definitions 4.6* and *4.7* it is easy to derive the following relationship between $first(i,T_i,T_j)$ and $last(i,T_i,T_j)$.

Lemma 4.8
For any pair of discrete domains T_i and T_j ordered by disjointedness ($T_i \sqsupseteq T_j$), the values $first(i,T_i,T_j)$ and $last(i,T_i,T_j)$ are related as follows:

$$+(first(i,T_i,T_j), \lceil C_F(i,T_i,T_j) \rceil \dot{\oplus} - 1, last(i,T_i,T_j))$$

The proof directly follows from the definitions as shown in [25]. Such an equality can be taken as a definition of the function *first* in terms of the function *last*, or vice versa. In particular, it is immediate to show that it holds for any pair of domains T_i and T_j such that $T_j \sqsupseteq T_i$ too. In such a case, $first(i,T_i,T_j)$ is equal to $last(i,T_i,T_j)$ for all i. Now, let $succ(i)$ be the successor of i, that is, a time instant i' such that $+(i,1,i')$. From quasi-functionality, it follows that for each i if i has a successor $succ(i)$ then such a successor is unique. On the basis of the properties of the projection relation, it is possible to determine the following relationship between $last(i,T_i,T_j)$ and $first(succ(i),T_i,T_j)$.

Lemma 4.9
For any pair of discrete domains T_i and T_j ordered by disjointedness ($T_i \sqsupseteq T_j$), the values $first(i,T_i,T_j)$ and $last(succ(i),T_i,T_j)$ are related as follows:

$$+(last(i,T_i,T_j), 1, first(succ(i),T_i,T_j))$$

provided that i has a successor.
The proof involves the properties of symmetry, strong order preservation and coverage of the projection relation, together with the definitions of the functions *first* and *last* [25]. On the basis of *Lemma 4.8* and *Lemma 4.9*, it is now possible to prove that *Definition 4.2* properly expresses the intended meaning of $\Box_0$.

Lemma 4.10
For any i in $\mathcal{T}$ and $\mathcal{F}$, the formula defining $\Box_0 \mathcal{F}$ evaluates to true at i if and only if $\mathcal{F}$ evaluates to true at the initial point of each interval on which i is projected.
The proof is reported in [25]. According to *Lemma 4.10*, the semantics of $\Box_0$ is given by the semantic clause:

$\Im_i(\Box_0 \mathcal{F}) = true \Longleftrightarrow \Im_j(\mathcal{F}) = true$ for each j
such that $i \in T_i$ and $j \in T_j$ and $j = first(i,T_i,T_j)$

The main theorem stating that *Definition 4.1'* actually captures the intended meaning of the asynchronous projection operator easily follows from *Lemma 4.10*.

Theorem 4.11
For any T_i, $i \in T_i$ and $\mathcal{F}$, the formula defining the $\Box_\phi \mathcal{F}$ operator in terms of $\Box_0$ operator evaluates to true at i if and only if, for each T_j, $\mathcal{F}$ evaluates to true at each j such that $+(first(i,T_i,T_j), P_D(i,T_i,T_j), j)$.
The proof easily follows from *Lemma 4.10* as shown in [25]. According to *Theorem 4.11*, the semantics of $\Box_\phi$ is given by the semantic clause:

$\Im_i(\Box_\phi \mathcal{F}) = true \Longleftrightarrow \Im_j(\mathcal{F}) = true$
for each j such that $i \in T_i$ and $j \in T_j$
and $+(first(i,T_i,T_j), P_D(i,T_i,T_j), j)$

In a very similar way, it is possible to prove that *Definition 4.4* properly expresses the intended meaning of the contextualized asynchronous projection operator $\Box_\phi^{A,B}$ [25].

4.3 The extended axiomatization

The basic properties of the alignment and asynchronous projection operator can be derived from the basic axioms for displacement, contextual and (synchronous) projection operators. The only addition to the axiomatic system for the basic language concerns the projection rules. In the extended language we must indeed distinguish between synchronous and asynchronous downward (and upward) projections. The intended meaning of synchronous projection remains unchanged; the axiom has to be slightly modified by adding the mode qualification:

(**Ax27'**) $\forall A, B(A \sqsupseteq B \supset \nabla^{A,s}(\mathcal{F} \supset \diamond\Delta^{B}\mathcal{F}))$

where s is a predefined time-independent constant of sort mode denoting the synchronous mode.

For each pair of domains T_i, T_j, with T_i coarser than T_j, the *asynchronous* downward projection rule states that if a property P holds at the (current) time instant i of T_i, then P holds at the current time instant j of T_j. Moreover, for each pair of domains T_i, T_j, with T_i finer than T_j, the *asynchronous* upward projection rule states that if a property P holds at the current time instant i of T_i, then P holds at the current time instant j of T_j. Notice that in the asynchronous case the two projection rules become perfectly symmetric.

Formally, *asynchronous downward projection* is defined by the following axiom:

(**Ax28**) $\forall A, B(A \sqsupseteq B \supset \nabla^{A,a}(\mathcal{F} \supset \diamond_{\phi}\Delta^{B}\mathcal{F}))$

where $\mathcal{F}$ is a first order formula, i.e. a formula devoid of displacement, contextual and projection operators, $\diamond_{\phi}$ is asynchronous projection operator, and a denotes the asynchronous mode. As in the synchronous case, it is easy to show the equivalence between $Ax28$ and the definition of asynchronous upward projection, and then to conclude that the axioms defining asynchronous downward and upward projection are interdeducible. On the basis of the projection rules it is straightforward to show that each asynchronous model is also a synchronous one, but not vice versa. In [25], we also prove that synchronous and asynchronous projection rules for first order formulae embedded in a contextual displacement can be derived from the basic ones. Furthermore, under the assumption that $\rightarrow$ is homogeneous, we prove the commutativity of displacement and projection operators, modulo the conversion factor.

Even if the alignment and asynchronous projection operators have been introduced with the specific goal of constraining projected formulae to be evaluated at the current time instant of the projection domain, it is immediate to consider possible extensions of this operator supporting ontological characterizations of assertions. As shown in [30], primitive ontological concepts as event, property, fact and process, can be defined in terms of the behaviour under temporal projection of the corresponding assertions by replacing the point domain over which phase displacement are interpreted with an interval domain. Moreover, introducing different types of truth of assertions over intervals would allow us to cope with wholistic assertions too. These extensions are currently under investigation.

Conclusions and related work

A good specification language must allow one to specify the components of a real-time system and their interactions in a simple and intuitively clear way. Thus, the ability of dealing with different time granularities is a desirable feature of a specification language for granular systems. In the paper, we dealt with the problem of extending the metric and layered temporal logic (MLTL) for synchronous granular systems specification proposed in [6, 23] to asynchronous ones. As a result, we provided an extended specification framework that allows us to model synchronous and asynchronous systems and their interactions in a uniform and flexible way.

Decidability and executability as well as expressiveness refinements of MLTL are currently under investigation. With respect to expressiveness, we are studying the relationships between MLTL and polymodal and dynamic logics [32] as well as the possibility of embedding it into a one sorted standard temporal logic. Parallely, we are exploring a new formulation of the language where all projection operators are defined in terms of the alignment one and absolute contexts are replaced by relative ones. It should result in a uniform logic of granular and metric displacement dealing with 'transalation' within and across domains in a uniform way. With respect to decidability, we are working at the generalization of Alur and Henzinger's decidability results for non-layered, metric temporal logics [2] to suitable propositional fragments of MLTL. The method underlying Alur and Henzinger's approach relies on the finite state character of time which can be expressed as follows: each temporal property that partitions an infinite set of states (instants) into a finite set of classes can be finitely modeled and is then decidable. In [26] we prove the finite state character of contextualization and projection over temporal universes. With respect to executability, we are studying the way in which model checking techniques developed for the flat fragment of MLTL [16] can be extended to the whole logic. Moreover, we are analyzing temporal logic proof systems originally developed for simpler temporal logics [1, 14, 15, 27, 28] in order to evaluate their applicability to MLTL. A preliminary result is given in [10].

Even if the proposed logic has been developed within the temporal framework, it outlines the basic features of a *general logic of granularity*. In this respect it can be seen as a generalization of Rescher and Garson's topological logic [30] to layered structures. It also presents interesting connections with the logics of contexts, where modalities are used to shift variables, domains, and interpretation functions [4, 5].

Acknowledgements

I would like to thank Alberto Policriti for many useful discussions, and one of the anonymous referees for his/her valuable comments.

References

[1] *Nonclausal Deduction in First-Order Temporal Logic.* M. Abadi. Z. Manna: Journal of the ACM.

Vol. 37, n. 2, 1990.

[2] *Real-Time Logics: Complexity and Expressiveness*, R. Alur, T. A. Henzinger, Information and Computation, Vol. 104, 1993.

[3] Temporal Logic, J. van Benthem, in *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. Gabbay, C. Hogger, and J. Robinson, eds., Oxford University Press, 1991.

[4] *General Dynamics*, J. van Benthem; Theoretical Linguistics, Vol. 17. n. 1-2-3, Walter De Gruyter, Berlin - New York, 1991.

[5] *Propositional Logic of Context*, S. Buvac, I.A. Mason, Proc. AAAI-93, Washington, MIT-Press, 1993.

[6] *Embedding Time Granularity in a Logical Specification Language for Synchronous Real-Time Systems*, E. Ciapessoni, E. Corsetti, A. Montanari, P. San Pietro; Science of Computer Programming, Vol. 20, nn. 1-2, Elsevier Science Publishers B.V., 1993.

[7] *A Simple, General Structure for Temporal Domains*, J. Clifford, A. Rao; in *Temporal Aspects of Information Systems*, C. Rolland, M. Leonard, eds., Elsevier Science Publishers B.V., IFIP 1988.

[8] *Dealing with Different Time Granularities in Formal Specifications of Real-Time Systems*, E. Corsetti, A. Montanari, E. Ratto; The Journal of Real-Time Systems, Vol. III, Issue 2, Kluwer Academic Publishers, June 1991.

[9] *Dealing with Different Time Scales in Formal Specifications*, E. Corsetti, E. Crivelli, D. Mandrioli, A. Montanari, A. Morzenti, P. San Pietro, E. Ratto; Proc. 6th International Workshop on Software Specification and Design, Como, Italy, October 1991.

[10] *Translating modal formulae as set-theoretic terms*, G. D'Agostino, A. Montanari, A. Policriti; Research Report 10/94, Dipartimento di Matematica e Informatica, Università di Udine, May 1994 (also in Logic Colloquium '94, Clermont-Ferrand, France, July 1994).

[11] *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354, J. de Bakker, W.P. de Roever, and G. Rozenberg, eds., Springer Verlag, 1989.

[12] *Real-Time: Theory in Practice*, LNCS 600, J.W. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg, eds., Springer Verlag, 1992.

[13] *The Macro-Event Calculus: Representing Temporal Granularity*, C. Evans; Proc. PRICAI, Japan 1990.

[14] *A Resolution Method for Temporal Logic*, M. Fisher; Proc. 12th IJCAI, Sydney, Australia, 1991.

[15] *Modal and Temporal Logic Programming*, D. Gabbay; in *Temporal Logics and their Applications*, A. Galton, ed., Academic Press, 1987.

[16] *TRIO, a logic language for executable specifications of real-time systems*, C. Ghezzi, D. Mandrioli, A.Morzenti; Journal of Systems and Software, Vol.12, no.2, May 1990.

[17] *Fundamentals of Software Engineering*, C. Ghezzi, M. Jazayeri, D. Mandrioli; Prentice Hall, 1991.

[18] *Logics of Time and Computations*, R. Goldblatt; CSLI Lecture Notes 7, The Chicago University Press, Chicago, 1987.

[19] *Granularity*, J.R. Hobbs; Proc. 9th IJCAI, Los Angeles, 1985.

[20] *A Companion to Modal Logic*, G.F. Hughes, M.J. Cresswell; Methuen, London, 1984.

[21] *Specifying Real-Time Properties with Metric Temporal Logic*, R. Koymans; The Journal of Real-Time Systems, Vol. II, 1990.

[22] *The Temporal Logic of Reactive and Concurrent Systems. Specification.*, Z. Manna, A. Pnueli, Springer Verlag, 1992.

[23] *Dealing with Time Granularity in Logical Specifications of Real-Time Systems. The Synchronous Case*, A. Montanari, E. Ciapessoni, E. Corsetti, P. San Pietro; Research Report 07/92, Dipartimento di Matematica e Informatica, Università di Udine, May 1992.

[24] *Dealing with Time Granularity in the Event Calculus*, A. Montanari, E. Maim, E. Ciapessoni, E. Ratto; Proc. International Conference on Fifth Generation Computer Systems '92, Tokyo, Japan, June 1992.

[25] *Dealing with Time Granularity in Logical Specifications of Real-Time Systems. The Asynchronous Case*, A. Montanari; Research Report 12/93, Dipartimento di Matematica e Informatica, Università di Udine, September 1993.

[26] *Decidability Results on Metric and Layered Temporal Logics*, A. Montanari, A. Policriti; Research Report, Dipartimento di Matematica e Informatica, Università di Udine, 1994 (in preparation).

[27] *First-Order Modal Logic Theorem Proving and Functional Simulation*, A. Nonnengart; Proc. IJCAI-93, Chambery, France, 1993.

[28] *Transalation Methods for Non-Classical Logics: An Overview*, H.J. Ohlbach; Bull. of the IGLP, Vol.1, No.1, 1993.

[29] *Temporal Logic for Real-Time Systems*, J.S. Ostroff; Advanced Software Development Series, Research Studies Press, John Wiley & sons, 1989.

[30] *Temporal Logic*, N. Rescher, A. Urquhart; Library of Exact Philosophy, Springer-Verlag, 1971.

[31] *Software Engineering*, I. Sommerville; Third Edition, Addison Wesley, 1989.

[32] *Many-dimensional modal logic*, Y. Venema; PhD Dissertation, Universiteit van Amsterdam, 1991.

[33] *Dealing with Granularity of Time in Temporal Databases*, G. Wiederhold, S. Jajodia, W. Litwin; in *Advanced Information Systems Engineering*, R. Andersen, J.A. Bubenko jr., A. Solvberg, eds., Springer-Verlag, 1991.

On Continuous Extensions of Temporal Logic Programming (Extended Abstract)

Mehmet A. Orgun
Department of Computing, Macquarie University
Sydney, NSW 2109, Australia
Tel: +61 2 850-9570, Fax: +61 2 850-9551
E-mail: mehmet@mpce.mq.edu.au

Abstract

Chronolog is a temporal logic programming language based on a temporal logic with two temporal operators: **first**, and **next**. We first extend the temporal logic of Chronolog with countable conjunctions and disjunctions and call the extension *TL*. We then study continuous extensions of temporal logic programming based on *TL*. In particular, we show that

- when *TL* is enriched with *nominal* symbols (which are used as names for moments in time), it is as expressive as any of its continuous extensions;
- temporal logic programming based on *TL* is as expressive as temporal logic programming based on any continuous extension of *TL*;
- with respect to the minimum model semantics, temporal logic programming based on continuous extensions of *TL* is as expressive as temporal logic programming based on *TL* without nominal symbols.

These results suggest that Chronolog is as expressive as any other temporal language based on continuous extensions of *TL*.

Categories: Temporal Logic: algebraic properties of temporal operators; temporal logic programming.

1 Introduction

Temporal logic has been widely used as a formalism in many areas in computer science, including program specification and verification [Kroger, 1987, Manna and Pnueli, 1981], modelling temporal databases [Baudinet *et al.*, 1993, Gabbay and McBrien, 1991, Tuzhilin and Clifford, 1990], and various forms of temporal reasoning [Sadri, 1987]. In these application areas, temporal logic has been used to model time-dependent and dynamic properties of certain problems in a natural and problem-oriented way. More recently, several researchers have suggested that temporal logic can be directly used as a programming language in applications such as those mentioned above; hence the term "temporal logic programming (TLP)". Tempura [Hale, 1987, Moszkowski, 1986] and Tokio [Aoyagi *et al.*, 1986] are based on interval logic; Templog [Abadi and Manna, 1989] and Chronolog [Orgun and Wadge, 1992a] are based on linear-time temporal logic and Temporal Prolog [Gabbay, 1987, Gabbay, 1989] is based on linear- and branching-time temporal logics. A recent overview of the status of research on temporal (and modal) extensions of logic programming is given by Orgun and Ma [Orgun and Ma, 1994].

Chronolog is one of the simplest among these temporal languages. The underlying logic of Chronolog has only two temporal operators, namely, **first** and **next**. The temporal operators **first** and **next** refer to the initial moment in time and the next moment in time, respectively. The set of natural numbers models the collection of moments in time. The design of Chronolog is very much influenced by the dataflow language Lucid [Wadge and Ashcroft, 1985]. Its target application, right from the start, has been modelling dataflow computations, non-terminating computations and modelling infinite objects such as streams [Rolston, 1986, Orgun and Wadge, 1992a, Wadge, 1988]. The simplicity of Chronolog allows a smooth extension of the standard declarative and operational semantics for ordinary logic programs to temporal logic programs [Orgun and Wadge, 1992a]. In particular, it is shown in [Orgun and Wadge, 1993] that Chronolog admits a complete proof procedure called TiSLD-resolution which is a temporal extension of standard SLD-resolution. TiSLD-resolution forms the basis for implementations of Chronolog.

There are other temporal languages based on more standard temporal logics than that of Chronolog. For instance, Templog [Abadi and Manna, 1987, Abadi and Manna, 1989], which is also based on a linear-time temporal logic, offers the traditional temporal operators $\Box$ ("from now on") and $\Diamond$ ("sometime in the future") as well as a next time operator $\bigcirc$. Compared to Templog, Chronolog seems to have a limited expressive power. However, it is always possible to extend Chronolog with the traditional temporal operators, with **since** and **until**, or even with Lucid-like operators such as **fby** ("followed-by"), **asa** ("as soon as") and **wvr** ("whenever").

Before we can proceed with any of these extensions, we need to keep in mind that (Horn) logic programming is based on *monotonic* logics. A more practical restriction is that, given that we want to compute with these temporal logic languages, we should consider *continuous* extensions only. As is usual in programming language semantics [Stoy, 1977], continuity is associated with computability in (temporal and modal) logic programming [Lloyd, 1984, Orgun and Wadge, 1992b]. In [Orgun and Wadge, 1992b], it is shown that intensional logic languages enjoy an extended fixed-point semantics provided that the denotations of all the intensional operators appearing in the bodies of program clauses are continuous. The fixed-point semantics establish the link between the declarative (model-theoretic) and operational (proof-tehoretic) semantics of logic programming.

When Chronolog is extended with additional continuous operators such as $\Diamond$ and `fby`, the declarative and operational semantics of the language need to be reworked; in turn implementations of the language need to be modified to accommodate the additional operators. An important question is then whether adding continuous operators to a given temporal logic language such as Chronolog improves the expressive power of the language or not. In this paper, we settle this question in the negative. A theoretical implication of this result is that we do not need any extra semantic apparatus to deal with continuous extensions of TLP. In practice, however, we do not rule out extending Chronolog for ease of programming.

In the sequel, we develop a semantic framework, starting from an infinitary temporal logic (called *TL* from here on) which is more expressive than the underlying temporal logic of Chronolog. *TL* has two temporal operators, namely **first** and **next**, and countable conjunctions and disjunctions. It also has an infinite number of *nominal symbols*, one for each moment in time. Nominal symbols are instantaneous propositions, and the instant at which a nominal symbol is true is the instant it names. Blackburn [1993] shows that nominal symbols lead to richer and more expressive tense logics; many classes of frames not standardly definable such as the partial orders, the strict total orders, and the integers become definable in nominal tense logics.

We first establish a restricted form of a functional completeness result for *TL*: it is shown that any continuous extension of *TL* can be embedded in it. The embedding result is based on the fact that for any given continuous temporal operator there is a monotonic formula of *TL*, made up of countable conjunctions and disjunctions, temporal operators **first** and **next**, and nominal symbols, which can be used as its defining formula. A consequence of the embedding result is that the expressive power of TLP based on any continuous extension of *TL* is the same as TLP based on *TL*.

We next show that, with respect to the minimum model semantics, the expressive power of TLP based on *TL* without the nominal symbols is the same as the expressive power of TLP based on any continuous extension of *TL*. In *TL*, we are able to give the defining formulas of continuous operators with the help of nominal symbols. As for TLP, whether the underlying language has nominal symbols or not does not make any difference, because, through the use of the minimum model semantics, nominal symbols can be made available in temporal logic programs using program clauses as their definitions. As far as logic programming is concerned, the embedding result mentioned above is all that we need: the minimum model of a given temporal logic program is its canonical meaning (as in standard logic programming semantics [Lloyd, 1984]).

In Templog [Abadi and Manna, 1987, Abadi and Manna, 1987], certain restrictions on the use of $\Box$ and $\Diamond$ are imposed: $\Diamond$ is allowed to appear only in a body of a program clause and $\Box$ is always applied to a whole program clause. In Chronolog, the application of $\Box$ to whole program clauses is implicit. It follows that, Templog, which is based on a linear-time temporal logic with the same time-line as Chronolog, does not offer any extra expressive power. In fact, Templog is equivalent to a fragment of itself, called *TL1*, with the next time operator $\bigcirc$ as the sole temporal operator [Baudinet, 1992]. However, the results of this paper do not extend to more expressive TLP languages such as Temporal Prolog proposed by Gabbay [1987, 1989]. It should be noted that Templog also admits a complete resolution-type proof procedure [Baudinet, 1989, Baudinet, 1992].

In the sequel, we assume some familiarity with logic programming, model-theoretic semantics, and temporal logic programming; we refer the reader to [Abadi and Manna, 1987, Abadi and Manna, 1989, Baudinet, 1989, Gabbay, 1987, Gabbay, 1989, Lloyd, 1984, Orgun and Wadge, 1992a] for more details. We approach the underlying temporal logic from a purely semantical point of view. We give an example of a temporal logic program later in the paper.

2 Preliminaries

The temporal logic *TL* is defined as follows: We begin with a standard first-order language with countable conjunctions and disjunctions, such as the logic $L^{\omega}_{\infty\omega}$ of Kolaitis and Vardi [1992], and extend it with two new formation rules: if A is a formula, so are **first** A and **next** A. We take the set of natural numbers ω to be the collection of moments in time in *TL*. In addition we enrich the logic with an infinite number of nominal symbols, one for each moment in time: $\{\odot_t \mid t \in \omega\}$. Each nominal symbol is also a formula. Note that the temporal operators are applied to formulas, not to terms of the language. Formulas of *TL* can only have a finite number of distinct variables.

A temporal interpretation assigns meanings to all elements of the language at all moments in time. A standard satisfaction relation $\models$ completes the picture by extending temporal interpretations upwards to all formulas of *TL*. The notation $\models_{I,t} A$ means that the formula A is true in I at time t.

The denotation of temporal operators **first** and **next** are formalized as functions over the power set of ω, that is, an element of the set of functions from $\mathcal{P}(\omega)$ to $\mathcal{P}(\omega)$. In other words, these functions do subset-

transformations over ω. We refer the reader to [Bull and Segerberg, 1984, Scott, 1970] for more details on algebraic semantics. We refer to the denotations of **first** and **next** as $[\![\mathbf{first}]\!]$ and $[\![\mathbf{next}]\!]$ respectively.

The functions $[\![\mathbf{first}]\!]$ and $[\![\mathbf{next}]\!]$ are given as follows:

$$[\![\mathbf{first}]\!] = \lambda X.\begin{cases} \omega & \text{if } 0 \in X \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![\mathbf{next}]\!] = \lambda X.\{t \in \omega \mid t+1 \in X\}$$

Using these functions, the semantics of formulas of the form **first** A and **next** A are given as follows:

$\models_{I,t}$ **first** A iff $t \in [\![\mathbf{first}]\!]([\![A]\!]^I)$

$\models_{I,t}$ **next** A iff $t \in [\![\mathbf{next}]\!]([\![A]\!]^I)$

where $[\![A]\!]^I = \{x \in \omega \mid \models_{I,x} A\}$.

The denotations of nominal symbols are singleton sets. The semantics of formulas of the form $\odot_t$ are given as follows:

$\models_{I,z} \odot_t$ iff $z \in [\![\odot_t]\!]^I$

where $[\![\odot_t]\!]^I = \{t\}$. In other words, the nominal symbol $\odot_t$ names the instant t in *any* temporal interpretation.

Let Θ be an element of $[\mathcal{P}(\omega)^n \to \mathcal{P}(\omega)]$, that is, an element of the set of functions from $\mathcal{P}(\omega)^n$ to $\mathcal{P}(\omega)$. Then the underlying temporal language can be enriched with an extra n-ary operator, say $\bigtriangledown$, by setting $[\![\bigtriangledown]\!] = \Theta$ and by extending the definition of the satisfaction relation $\models$ by the following:

$$\models_{I,t} \bigtriangledown(A_0, \ldots, A_{n-1}) \text{ iff } t \in \Theta([\![A_0]\!]^I, \ldots, [\![A_{n-1}]\!]^I)$$

where each A_i is a formula, I is a temporal interpretation, and $t \in \omega$.

Let $\vec{X} =_{def} \langle X_0, X_1, \ldots, X_{n-1}\rangle$ where X_i's are subsets of ω. Let $\leq$ be an ordering relation over $\mathcal{P}(\omega)^n$, i.e. for all $\vec{X}$ and $\vec{Y} \in \mathcal{P}(\omega)^n$, $\vec{X} \leq \vec{Y}$ iff for all $i \in n$, $X_i \subseteq Y_i$. Let $\vec{X} \cap \vec{Y} =_{def} \langle X_0 \cap Y_0, \ldots, X_{n-1} \cap Y_{n-1}\rangle$ and $\vec{X} \cup \vec{Y} =_{def} \langle X_0 \cup Y_0, \ldots, X_{n-1} \cup Y_{n-1}\rangle$. We assume that $\cap$ and $\cup$ also apply to arbitrary subsets of $\mathcal{P}(\omega)^n$.

We now define a few important properties of functions:

- We say that Θ is *monotonic* iff for all $\vec{X}$ and $\vec{Y} \in \mathcal{P}(\omega)^n$, $\vec{X} \leq \vec{Y}$ implies that $\Theta(\vec{X}) \subseteq \Theta(\vec{Y})$.
- We say that Θ is *finitary* iff for all $\vec{X} \in \mathcal{P}(\omega)^n$ and for all $t \in \omega$, $t \in \Theta(\vec{X})$ implies $t \in \Theta(\vec{S})$ for some $\vec{S} \leq \vec{X}$ where S_0, S_1 , ..., and S_{n-1} are all finite.
- Let $\{\vec{X}_\alpha\}_{\alpha \in \mathcal{I}}$ be any subset of $\mathcal{P}(\omega)^n$ where $\mathcal{I}$ is the index set of S. Then we say that Θ is *conjunctive* iff $\Theta(\cap_{\alpha \in \mathcal{I}} \vec{X}_\alpha) = \cap_{\alpha \in \mathcal{I}} \Theta(\vec{X}_\alpha)$.
- We say that Θ is *continuous* iff for all chains $\langle \vec{X}_\alpha \rangle_{\alpha \in \omega}$ over $\mathcal{P}(\omega)^n$, $\Theta(\cup_{\alpha \in \omega} \vec{X}_\alpha) = \cup_{\alpha \in \omega} \Theta(\vec{X}_\alpha)$.

In Orgun and Wadge [1992b], these properties (among others) are used as constraints on logic programming languages based on temporal and modal logics. The properties are interrelated; in particular it is shown there that a given function Θ is both monotonic and finitary if and only if it is continuous. In this paper, we do not directly make use of the property of continuity, but we do use the properties of monotonicity and finitariness.

Note that both $[\![\mathbf{first}]\!]$ and $[\![\mathbf{next}]\!]$ are monotonic, conjunctive and finitary; and hence continuous.

The following two operations, which we call $\sqcup$-union and $\sqcap$-intersection respectively, will be used to form new functions from given ones. For any given functions Φ and Ψ with the same arity,

$$\Psi \sqcup \Phi =_{def} \lambda\vec{X}.\, \Psi(\vec{X}) \cup \Phi(\vec{X}),$$

$$\Psi \sqcap \Phi =_{def} \lambda\vec{X}.\, \Psi(\vec{X}) \cap \Phi(\vec{X}).$$

These two operations naturally extend to a family of functions.

Let $\Theta|_t = \{\vec{X} \in \mathcal{P}(\omega)^n \mid t \in \Theta(\vec{X})\}$. In the terminology of Scott [1970], each $\Theta|_t$ is the set of neighborhoods of t with respect to the function Θ. Then an alternative definition for $\sqcup$-union and $\sqcap$-intersection can be given by the following. For any given functions Ψ and Φ, and for all $t \in \omega$,

$$(\Psi \sqcup \Phi)|_t =_{def} \Psi|_t \cup \Phi|_t,$$

$$(\Psi \sqcap \Phi)|_t =_{def} \Psi|_t \cap \Phi|_t.$$

It can be easily established that these two alternative definitions are semantically equivalent to those given previously. We leave out the details.

The composition operation denoted by $\circ$ is defined as follows: for any given unary function Ψ and n-ary function Φ $(n > 0)$, for all $\vec{X} \in \mathcal{P}(\omega)^n$,

$$(\Psi \circ \Phi)(\vec{X}) =_{def} \Psi(\Phi(\vec{X})).$$

For any unary function Ψ, let Ψ^z denote the z-fold composition of Ψ. Note that $\Psi^0(X) = X$. It can be shown that compositions of functions having the same properties yield a function with the same properties. Of course, the same goes for operations $\sqcap$ and $\sqcup$, except that $\sqcup$ does not preserve conjunctivity.

The syntactic counterparts of the operations $\sqcap$-intersection and $\sqcup$-union are conjunction and disjunction respectively. The composition corresponds to successive applications of temporal operators.

These three properties of functions have semantic implications as to the structure of each neighborhood set $\Theta|_t$.

When Θ is monotonic, then, for all $t \in \omega$, $\Theta|_t$ is closed under the less-than relation $\leq$, that is, $\Theta|_t$ is supplemented in the terminology of Chellas [1980]. And it is clear that $\Theta|_t \neq \emptyset$ implies $\omega \in \Theta|_t$.

If Θ is finitary, then for all $t \in \omega$, all the minimal elements of $\Theta|_t$ are finite.

When Θ is conjunctive, then Θ is also monotonic and for all $t \in \omega$, $\Theta|_t$ is closed under (arbitrary) intersections (and vice versa).

We refer the reader to [Orgun, 1991, Orgun and Wadge, 1992b] for more details.

3 Unary Continuous Operators

Let Θ be an element of $\mathcal{P}(\omega) \to \mathcal{P}(\omega)$. We claim that if Θ is continuous, then it can be defined as a *countable* $\sqcup$-union of a family of continuous functions. By a countable

$\sqcup$-union we mean that it is either finite or countably infinite.

We first introduce some notation. Let Ψ and Φ be two functions with the same arity. Then we write $\Psi \equiv \Phi$ if we have that $\Psi(\vec{X}) = \Phi(\vec{X})$ for all $\vec{X} \in \mathcal{P}(\omega)^n$. In other words, Ψ and Φ agree on every input value (they are the same function). We also make use of the following result:

$$\Psi \equiv \Phi \text{ iff } \Psi|_t = \Phi|_t \text{ for all } t \in \omega.$$

Let $\mathcal{V} = \{t \in \omega \mid \Theta|_t \neq \emptyset\}$ and $\mathcal{M}(\Theta|_t)$ be the set of minimal elements of $\Theta|_t$. In other words, any element of $\mathcal{M}(\Theta|_t)$ as input is sufficient to make the operation true at time t and the operation is not true at time t if anything is removed from it. For continuous (and hence finitary) functions, $\mathcal{M}(\Theta|_t)$ is a countable set of minimal elements of $\Theta|_t$, each of which is finite. Then we claim that $\Theta \equiv \sqcup_{t \in \mathcal{V}} \Psi_t$ where each Ψ_t is defined as

$$\Psi_t =_{def} \sqcup_{X \in \mathcal{M}(\Theta|_t)} \sqcap_{z \in X} (\llbracket \mathbf{first} \rrbracket \circ \llbracket \mathbf{next} \rrbracket^z) \sqcap \llbracket \odot_t \rrbracket.$$

The intuition behind the definition of Θ is as follows. Let Θ^* represent the expression on the right-hand side of the definition of Θ. We want to establish that for all $t \in \omega$, $\Theta|_t = \Theta^*|_t$; in other words, $\Theta \equiv \Theta^*$. Let us define each Ψ_t as the $\sqcap$-intersection of two functions: $\Psi_t =_{def} \mathcal{F}_t \sqcap \llbracket \odot_t \rrbracket$ where

$$\mathcal{F}_t =_{def} \sqcup_{X \in \mathcal{M}(\Theta|_t)} \sqcap_{z \in X} (\llbracket \mathbf{first} \rrbracket \circ \llbracket \mathbf{next} \rrbracket^z).$$

Note that for any given $v \in \omega$, $\llbracket \odot_t \rrbracket|_v = \mathcal{P}(\omega)$ if $v = t$; $\emptyset$ otherwise. It can be verified that $\mathcal{F}_t$ is just like Θ but restricted to t and for any $v \in \omega$, $\mathcal{F}_t|_v = \Theta|_t$. When the nominal function $\llbracket \odot_t \rrbracket$ is $\sqcap$-intersected with $\mathcal{F}_t$, then the resultant function Ψ_t acts like the original function Θ only for t, i.e. $\Theta|_t = \Psi_t|_t$ and for any $v \neq t$, $\Psi_t|_v = \emptyset$. Therefore the $\sqcup$-union of these functions, Θ^*, behaves just like Θ.

We have the following result:

Lemma 1 *Let $\Theta \in [\mathcal{P}(\omega) \rightarrow \mathcal{P}(\omega)]$. If Θ is continuous, then it can be defined in terms of compositions of $\sqcap$, $\sqcup$, $\llbracket \mathbf{first} \rrbracket$, $\llbracket \mathbf{next} \rrbracket$, and (denotations of) nominal symbols.*

The techniques given above can be applied to conjunctive functions as well as to continuous functions. Recall that the neighborhood sets of any conjunctive function Θ are closed under intersection, and hence each has a single countable minimum element, which is the intersection of all the elements in the neighborhood set. In other words, we have that for all $t \in \omega$, $\cap\Theta|_t \in \Theta|_t$. In this case the definition of Θ can be simplified as: $\Theta \equiv \sqcup_{t \in \mathcal{V}} \Psi_t$ where

$$\Psi_t =_{def} \sqcap_{z \in \cap\Theta|_t} (\llbracket \mathbf{first} \rrbracket \circ \llbracket \mathbf{next} \rrbracket^z) \sqcap \llbracket \odot_t \rrbracket.$$

The monotonicity of Θ is essential for this approach (which is implied by the conjunctivity (or the continuity) of Θ), because we are assuming that for any $t \in \omega$, $\Theta|_t$ is supplemented.

A standard example of a conjunctive function is the denotation of $\Box$ (here we take $\Box A$ to mean that A is true at *all* moments in time.) Note that $\llbracket \Box \rrbracket$ is monotonic, but not finitary; hence not continuous. Below is the straightforward definition of $\llbracket \Box \rrbracket$.

$$\llbracket \Box \rrbracket = \lambda X. \begin{cases} \omega & \text{if } X = \omega \\ \emptyset & \text{otherwise} \end{cases}$$

Let $\mathcal{V} = \{t \in \omega \mid \Theta|_t \neq \emptyset\}$. Here is the definition of $\llbracket \Box \rrbracket$:

$$\llbracket \Box \rrbracket \equiv \sqcup_{t \in \mathcal{V}} (\sqcap_{z \in \cap \llbracket \Box \rrbracket|_t} \llbracket \mathbf{first} \rrbracket \circ \llbracket \mathbf{next} \rrbracket^z) \sqcap \llbracket \odot_t \rrbracket)$$

For any given $x \in \omega$, we have that $\llbracket \Box \rrbracket|_x = \{\omega\}$. Then the definition can be simplified further by replacing both $\cap \llbracket \Box \rrbracket|_t$ and $\mathcal{V}$ by ω. At this stage, the nominal function $\llbracket \odot_t \rrbracket$ serves no purpose, because the left operand of the inner $\sqcap$-intersection operation is independent of t. Thus we obtain the final definition of $\llbracket \Box \rrbracket$ as:

$$\llbracket \Box \rrbracket \equiv \sqcap_{z \in \omega} (\llbracket \mathbf{first} \rrbracket \circ \llbracket \mathbf{next} \rrbracket^z)$$

This definition conforms to the definition of $\llbracket \Box \rrbracket$ given above. Using the syntactic counterparts of $\sqcap$, $\sqcup$ and $\circ$, we obtain a defining formula for $\Box$ in *TL* as

$$\Box(A) =_{def} \wedge_{z \in \omega} \mathbf{first}\ \mathbf{next}^z\ A.$$

It should be noted that, since $\llbracket \Box \rrbracket$ is not continuous, we rule out the use of $\Box$ in temporal logic programs; if we do not, the minimum model semantics and the fixed-point semantics for temporal logic programs do not coincide [Orgun and Wadge, 1992b].

4 N-ary Continuous Operators

In this section, we show that any continuous function can be defined in terms of more primitive ones. From here on, we assume that Θ is an arbitrary n-ary continuous function, i.e. $\Theta \in [\mathcal{P}(\omega)^n \rightarrow \mathcal{P}(\omega)]$.

If Θ is also conjunctive, then it can be defined as: $\Theta \equiv \sqcap_{0 \leq i \leq n-1} (\Psi_i \circ \Pi_i)$ where each Π_i is the i-th projection function, i.e., $\Pi_i(\vec{X}) = X_i$. Each Ψ_i is a unary continuous and conjunctive function determined as follows:

$$\Psi_i =_{def} \lambda X. \{t \in \omega \mid X \in \Psi_i|_t\}$$

where $\Psi_i|_t = \{X_i \mid \vec{X} \in \Theta|_t\}$. By this construction, it can be shown that for all $t \in \omega$ and for all $\vec{X} \in \mathcal{P}(\omega)^n$,

$$\vec{X} \in \Theta|_t \text{ iff } X_0 \in \Psi_0|_t \text{ and } \ldots \text{ and } X_{n-1} \in \Psi_{n-1}|_t.$$

Then in defining each Ψ_i, we can simply use unary continuous functions as described in the previous section.

The above discussion leads to the following result:

Lemma 2 *Let $\Theta \in [\mathcal{P}(\omega)^n \rightarrow \mathcal{P}(\omega)]$. If Θ is conjunctive, then it can be defined in terms of compositions of $\sqcap$, $\sqcup$, $\llbracket \mathbf{first} \rrbracket$, $\llbracket \mathbf{next} \rrbracket$, projection functions, and (denotations of) nominal symbols.*

Projection functions are not really necessary in the definition of Θ. Indeed, by adopting the λ-notation, Θ could be defined as follows.

$$\Theta \equiv \lambda \vec{X}. \cap_{0 \leq i \leq n-1} \Psi_i(X_i)$$

However, we keep the projection functions for notational convenience.

We now return to the case that Θ is not conjunctive. The basic idea is again to define Θ as a countable $\sqcup$-union of n-ary conjunctive functions since we know how to obtain the definition of an arbitrary continuous and conjunctive function in terms of unary ones.

Let $max \in \omega$ such that for $z < max$, $\mathbf{card}(\mathcal{M}(\Theta|_{max})) > \mathbf{card}(\mathcal{M}(\Theta|_z))$, and for $z \geq max$,

$\mathbf{card}(\mathcal{M}(\Theta|_{max})) \geq \mathbf{card}(\mathcal{M}(\Theta|_z))$. Let $\mathcal{I}$ be the index set of $\mathcal{M}(\Theta|_{max})$; it is clear that $\mathcal{I}$ is countable. Then we can define Θ as the $\sqcup$-union of a countable family of conjunctive functions:

$$\Theta \equiv \sqcup_{\alpha \in \mathcal{I}} \Psi_\alpha.$$

Here the definition of each Ψ_α is more elaborate. The intuitive idea is that for any given $t \in \omega$, each $\Psi_\alpha|_t$ must cover exactly one minimal element from $\Theta|_t$ and all minimal elements of $\Theta|_t$ must be covered so that each Ψ_α is conjunctive and $\sqcup_{\alpha \mathcal{I}} \Psi_\alpha$ behaves just like Θ at time t. Thus the number of Ψ_α's required is equal to the cardinality of the largest set of minimal elements of Θ's neighborhood sets.

For all $z \in \omega$, let $\mathcal{F}_z$ be a (surjective) function from $\mathcal{I}$ onto $\mathcal{M}(\Theta|_z)$; in other words, $\mathcal{F}_z \in \mathcal{I} \twoheadrightarrow \mathcal{M}(\Theta|_z)$. Clearly, $\mathcal{F}_t$ is a one-to-one (bijective) function. Then for all $z \in \omega$, if $\Theta|_z = \emptyset$, then $\Psi_\alpha|_z = \emptyset$; otherwise

$$\Psi_\alpha|_z = \{\vec{X} \in \mathcal{P}(\omega)^n \mid \vec{X} \geq \mathcal{F}_z(\alpha)\}.$$

It can be shown that for all $z \in \omega$,

$$\Theta|_z = (\sqcup_{\alpha \in \mathcal{I}} \Psi_\alpha)|_z$$

which is the desired result in terms of neighborhood sets. The definition of each continuous and conjunctive Ψ_α can be obtained as described above.

Given lemmas 1 and 2, we then have the following theorem:

Theorem 3 *Let $\Theta \in [\mathcal{P}(\omega)^n \to \mathcal{P}(\omega)]$. If Θ is continuous, then it can be defined in terms of compositions of $\sqcap$, $\sqcup$, $[\![\mathbf{first}]\!]$, $[\![\mathbf{next}]\!]$, and (denotations of) nominal symbols.*

This approach can also be applied to unary continuous functions. Below is the definition of the denotation of the temporal operator 'sometime' ($\Diamond$).

$$[\![\Diamond]\!] = \lambda X. \begin{cases} \omega & \text{if } X \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

It can be shown that $[\![\Diamond]\!]$ is continuous. Let $\mathcal{I}$ be the index set of $\mathcal{M}([\![\Diamond]\!]|_{max})$. We want to define $[\![\Diamond]\!]$ as follows:

$$[\![\Diamond]\!] \equiv \sqcup_{\alpha \in \mathcal{I}} \Psi_\alpha$$

Let $\mathcal{S}$ be the set of singleton sets in $\mathcal{P}(\omega)$, that is $\mathcal{S} = \{\{t\} \mid t \in \omega\}$. Observe that for any $z \in \omega$, $\mathcal{M}([\![\Diamond]\!]|_z) = \{\{t\} \mid t \in \omega\} = \mathcal{S}$. Hence one 1-1 function $\mathcal{F}$ suffices in defining each $\Psi_\alpha|_z$. Since $\mathcal{S}$ can be mapped onto ω, ω can be used as the index set of $\mathcal{S}$; here indeed $\mathcal{I} = \omega$ and $\mathcal{F}$: $\alpha \longmapsto \{\alpha\}$. Then for any $\alpha \in \mathcal{I}$ and any $z \in \omega$,

$$\begin{aligned} \Psi_\alpha|_z &= \{X \in \mathcal{P}(\omega) \mid X \supseteq \mathcal{F}(\alpha)\} \\ &= \{X \in \mathcal{P}(\omega) \mid X \supseteq \{\alpha\}\} \end{aligned}$$

which implies that for all $\alpha \in \mathcal{I}$,

$$\Psi_\alpha =_{def} [\![\mathbf{first}]\!] \circ [\![\mathbf{next}]\!]^\alpha$$

Here we have given the simplified versions of conjunctive Ψ_α's. Now the definition of $[\![\Diamond]\!]$ is straightforward.

$$[\![\Diamond]\!] \equiv \sqcup_{\alpha \in \mathcal{I}} ([\![\mathbf{first}]\!] \circ [\![\mathbf{next}]\!]^\alpha).$$

We conclude that any arbitrary continuous function can be defined in terms of primitive functions of *TL* and their countable $\sqcup$-unions, $\sqcap$-intersections and compositions. Using the syntactic counterparts of $\sqcap$, $\sqcup$ and $\circ$, we obtain a defining formula for $\Diamond$ in *TL* as

$$\Diamond(A) =_{def} \vee_{\alpha \in \mathcal{I}} \mathbf{first}\ \mathbf{next}^\alpha\ A.$$

It follows that any continuous extension of *TL* can be embedded in it. However, in general, we may or may not eliminate all the occurrences of the nominal symbols from the defining formulas.

5 Temporal Logic Programming

In this section, we show that any arbitrary continuous function can be dealt with in the framework of TLP. The idea is that we can replace applications of temporal operators whose denotations are continuous with their defining formulas obtained as above. We also show that we can also dispense with the nominal symbols which may exist in defining formulas by making use of the minimum model semantics of temporal logic programs.

Temporal logic programs of Chronolog consist of program clauses with a single temporal atom in the head, and a list of temporal atoms in the body. A temporal atom is an atomic formula with a number of applications of the temporal operators **first** and **next**. For instance, the following is a Chronolog program to produce the increasing sequence of Hamming numbers (multiples of 2, 3 and 5 of the form $2^i 3^j 5^k$ for some i, j and k):

```
hamming(X) <-
     residue([X|L]).

first residue([1]).
next residue(L1) <-
     residue([X|L]), times(2,X,X2),
     times(3,X,X3), times(5,X,X5),
     merge([X2,X3,X5],L,L1).

merge(L1,L2,L3) <- ...
times(X,Y,Z) <- ...
```

All program clauses are read as assertions true at all moments in time. We assume the standard definitions for **times** and **merge**. At each moment in time, **hamming** is true of the number from the sequence of Hamming numbers indexed by the moment. The **residue** predicate holds all those Hamming numbers produced but not yet consumed by the **hamming** predicate. For more examples on TLP, we refer the reader to the collections [Galton, 1987, Fariñas del Cerro and Penttonen, 1992].

From here on, we assume that temporal logic programs may include countably many clauses and each clause may include countable disjunctions and finite conjunctions. In fact, the continuity restriction will guarantee that we only have finite conjunctions in the defining formulas of additional temporal operators. Again, the only (primitive) temporal operators which we assume are available to us are **first** and **next**.

Recall the definition of an additional unary continuous function in terms of primitive functions. Let $\Theta \in [\mathcal{P}(\omega) \to \mathcal{P}(\omega)]$ and $\mathcal{V} = \{t \in \omega \mid \Theta|_t \neq \emptyset\}$. Then

$\Theta \equiv \sqcup_{t \in \mathcal{V}} (\sqcup_{X \in \mathcal{M}(\Theta|_t)} \sqcap_{z \in X} ([\![\mathbf{first}]\!] \circ [\![\mathbf{next}]\!]^z) \sqcap [\![\ldots_t]\!])$.

We know that for any given continuous Θ, each $X \in \mathcal{M}(\Theta|_t)$ is finite. Then using the equivalent set-theoretic notation, for all $Z \subseteq \omega$,

$$\Theta(Z) = \cup_{t\in\mathcal{V}}(\cup_{X\in\mathcal{M}(\Theta|_t)} \cap_{z\in X} ([\![\mathbf{first}]\!] \circ [\![\mathbf{next}]\!]^z(Z)) \cap [\![\odot_t]\!]).$$

Let $\bigtriangledown$ be the temporal operator whose denotation is given as $\Theta = [\![\bigtriangledown]\!]$. We can either directly extend the language with $\bigtriangledown$ or select an appropriate defining formula in the given language which is some combination of primitive temporal operators applied to some formula A. The latter approach will be followed here.

The definition of Θ given above can be used to obtain the defining formula of $\bigtriangledown$. Then we have that

$$\bigtriangledown(A) =_{def} \vee_{t\in\mathcal{V}}(\vee_{X\in\mathcal{M}(\Theta|_t)}(\wedge_{z\in X}\mathbf{first\ next}^z A)\wedge\odot_t)$$

Now any formula of the form $\bigtriangledown B$ in fact refers to the defining formula of $\bigtriangledown$ after substituting A by B.

Let $\mathcal{P}$ be a temporal logic program in which some temporal operators other than primitive ones appear. We may replace all right-hand side occurrences of those operators together with atomic formulas to which they are applied by their definitions as constructed above. Suppose that $\mathcal{P}$ includes just one program clause with non-primitive temporal operator $\bigtriangledown$ applied to a temporal atom.

$$(A \leftarrow B_0, \ldots, \bigtriangledown B, \ldots, B_{n-1}) \in \mathcal{P}$$

After replacing $\bigtriangledown B$ by the defining formula of $\bigtriangledown$ with B substituted for A. the new clause becomes

$$A \leftarrow B_0, \ldots, \vee_{t\in\mathcal{V}}(\vee_{X\in\mathcal{M}(\Theta|_t)}(\wedge_{z\in X}\mathbf{first\ next}^z B)\wedge\odot_t), \ldots, B_{n-1}$$

Supposing conjunctions and disjunctions cannot appear in program clauses, we can obtain an equivalent set of program clauses to replace the original clause given previously.

Disjunctions on the right-hand side do not cause any problems, because for each occurrence of a disjunction, we can split the clause into a set of alternative ones. Nor do finite conjunctions cause any problems.

We now obtain step-by-step a set of clauses which is equivalent to the original clause with $\bigtriangledown$. Suppose that $\mathcal{V} = \{0, 2, 3, \ldots\}$. Firstly, the outermost disjunction is eliminated by splitting the clause into:

$$A \leftarrow B_0, \ldots, \vee_{X\in\mathcal{M}(\Theta|_0)}(\wedge_{z\in X}\mathbf{first\ next}^z B)\wedge\odot_0), \ldots, B_{n-1}$$

$$A \leftarrow B_0, \ldots, \vee_{X\in\mathcal{M}(\Theta|_2)}(\wedge_{z\in X}\mathbf{first\ next}^z B)\wedge\odot_2), \ldots, B_{n-1}$$

$$A \leftarrow B_0, \ldots, \vee_{X\in\mathcal{M}(\Theta|_3)}(\wedge_{z\in X}\mathbf{first\ next}^z B)\wedge\odot_3), \ldots, B_{n-1}$$

$\ldots$

and so on.

Secondly. each clause given above is replaced by a set of clauses to eliminate the next disjunction in it. Below is the set of clauses for the first clause. Let $I_0 = \{\alpha_0, \alpha_1, \alpha_2, \ldots\}$ be the index set of $\mathcal{M}(\Theta|_0)$.

$$A \leftarrow B_0, \ldots, \wedge_{z\in X_{\alpha_0}}(\mathbf{first\ next}^z B)\wedge\odot_0), \ldots, B_{n-1}$$

$$A \leftarrow B_0, \ldots, \wedge_{z\in X_{\alpha_1}}(\mathbf{first\ next}^z B)\wedge\odot_0), \ldots, B_{n-1}$$

$$A \leftarrow B_0, \ldots, \wedge_{z\in X_{\alpha_2}}(\mathbf{first\ next}^z B)\wedge\odot_0), \ldots, B_{n-1}$$

$\ldots$

and so on.

Finally, all conjunctions are eliminated. The resultant set of clauses is countable as well.

We now outline an approach to defining nominal symbols using program clauses. As the meaning of a temporal logic program is characterised by its minimum model [Orgun and Wadge, 1992a], each nominal symbol that appears in a program can be defined as a predicate which receives a singleton set as its meaning with respect to the minimum model of the program.

First of all, the underlying first-order language is enriched with a set of predicate symbols $\{\mathbf{nom}_0, \mathbf{nom}_1, \ldots\}$ for nominal symbols $\odot_0, \odot_1, \ldots$ and we replace each occurrence of a formula of the form $\odot_t$ by $\mathbf{nom}_t$. Temporal logic programs are at the same time extended by a set of clauses given below.

$$\mathcal{C} = \{\mathbf{first\ next}^t\ \mathbf{nom}_t \leftarrow \mid t \in \omega\}$$

Note that users should not define $\mathbf{nom}_t$'s in a program for a correct interpretation of these nominal symbols. It can be shown that the denotation of each $\mathbf{nom}_t$ with respect to the minimum model of any temporal logic program is exactly the singleton set $\{t\}$. In any other model of the program, the denotation of nominal symbols are not necessarily singleton sets.

Let $\mathcal{P}^\star$ be the modified version of $\mathcal{P}$ through the procedure described above, that is $\mathcal{P}^\star$ consists of $\mathcal{C}$ and all clauses in $\mathcal{P}$ with $(A \leftarrow B_0, \ldots, \bigtriangledown B, \ldots, B_{n-1})$ removed and replaced by its equivalent set of clauses. It can be shown that the minimum model of $\mathcal{P}^\star$ can be constructed from that of $\mathcal{P}$ and vice versa, which in turn means that they are in a sense equivalent. The minimum model of any given temporal logic program is regarded as the canonical meaning of the program [Baudinet, 1989, Orgun and Wadge, 1992a].

Note that $\mathcal{P}^\star$ does not have any occurrences of nominal symbols, and $\mathcal{P}$ does not have the use of any extra predicate symbols $\mathbf{nom}_t$'s in it. In the minimum model of $\mathcal{P}$, the denotation of any $\mathbf{nom}_t$ would be the empty set. The denotation of any other predicate symbol in the minimum model of $\mathcal{P}$ would be the same as the denotation of the symbol in the minimum model of $\mathcal{P}^\star$. Hence there is a straightforward mapping from the minimum model of $\mathcal{P}$ to that of $\mathcal{P}^\star$ and vice versa. Other models of $\mathcal{P}^\star$ do not necessarily correspond to any model of $\mathcal{P}$.

The results of this section can be extended smoothly to arbitrary n-ary continuous functions. Let $\vec{A}$ denote $\langle A_0, \ldots, A_{n-1} \rangle$ where each A_i stands for an atom. The syntactic counterparts of projection functions that appear in the definition of a continuous and conjunctive n-ary function are defined as follows.

$$\mathtt{pi}_i(\vec{A}) =_{def} A_i$$

Here $\Pi_i(\vec{X}) = X_i$ and $[\![\mathtt{pi}_i]\!] = \Pi_i$. We can in fact add the definitions of projection operators to temporal logic programs as program clauses (see below). With these extra definitions, any arbitrary n-ary continuous function can be dealt with in TLP.

In short, we conclude that TLP is "functionally complete" with respect to the minimum models of temporal logic programs when those temporal operators whose denotations are monotonic and finitary (that is, continuous) are considered. We can obtain the defining formulas of continuous operators in TL and then "compile out" the nominal symbols so that we can still stay in the (infinitary) temporal logic with **first** and **next**. Therefore the results for Chronolog such as the minimum model semantics and the fixpoint semantics [Orgun and Wadge, 1992a] are still valid for continuous extensions. In short, the expressive power of TLP with **first** and **next** is the same as the expressive power of TLP based on any continuous extension of TL.

In Templog [Abadi and Manna, 1987, Abadi and Manna, 1989], the use of $\Diamond$ ("sometime in the future") is restricted to the bodies of program clauses, and $\Box$ ("from now on") is applied to whole clauses to turn them into assertions true at all moments in time. In Chronolog, the application of $\Box$ to whole program clauses is implicit: all program clauses are regarded as assertions true at *all* moments in time. The denotation of $\Diamond$ is continuous, and thus the use of $\Diamond$ in the bodies of program clauses does not add any extra expressive power to Templog. In fact, Templog is equivalent to a fragment of itself, called $TL1$, with the next time operator $\bigcirc$ as the sole temporal operator [Baudinet, 1992].

6 Operators Defined by Program Clauses

We now study the issue of additional temporal operators from a different perspective. Suppose that we want to say that a formula, say A, is true "now and during the next n moments." This can be expressed as $\wedge_{0 \le i \le n}\ \mathbf{next}^i\ A$. Then we can move to a *definitional extension* [Segerberg, 1982] of the underlying logic and use this formula as the definition of a new temporal operator, say $[n]$. This is fine, but, in TLP, we can do a better job and define $[n]$ by a program clause on the fly, and invoke this clause whenever a formula of the form $[n]\ B$ needs to be proved.

The definition of, or the axiom schema for, $[n]$ in the form of a program clause can be given as:

$$[n](A) \leftarrow \wedge_{0 \le i \le n}\ \mathbf{next}^i A$$

where A is just a place-holder for formulas. Let $\mathcal{P}$ be a temporal logic program in which $[n]$ is applied to some atom B in the body of some clause. To prove $[n]\ B$, the axiom schema can be invoked after substituting B for A.

$$([n]\ (A) \leftarrow \wedge_{0 \le i \le n}\ \mathbf{next}^i\ A)\{A/B\} \implies$$

$$[n]\ (B) \leftarrow \wedge_{0 \le i \le n}\ \mathbf{next}^i\ B$$

Here the occurrence of $[n]$ is not a direct application of $[n]$ to B; it is there to establish the connection between the use and the definition of $[n]$.

The meaning of the symbol $[n]$ is unknown in the object language. However, we can formulate a straightforward model-theoretic condition for the denotation of $[n]$ by the following: for all $X \in \mathcal{P}(\omega)$,

$$[\![[n]]\!](X) \supseteq \cap_{0 \le i \le n} [\![\mathbf{next}^i]\!](X)$$

Many functions in $\mathcal{P}(\omega) \rightarrow \mathcal{P}(\omega)$ may satisfy this model-theoretic condition. It can be shown that $\lambda X.\ \cap_{0 \le i \le n} [\![\mathbf{next}]\!]^i(X)$ is the least such function. We omit the details.

In general, a new temporal operator can be defined by a set of program clauses whose heads are identical, up to renaming of place-holders for formulas. Then the following set is the general form for the definition of new temporal operators:

$$\{\ (\nabla(A_0, \ldots, A_{n-1}) \leftarrow \wedge_{k \in m_\alpha} B_{\alpha,k})\ |\ \alpha \in S\ \}$$

where all $B_{\alpha,k}$'s are made up of A_i's and temporal operators. We stipulate that all m_α's are finite. For convenience, we assume that new temporal operators are defined by a single clause, since the above set of clauses is equivalent to the clause given below:

$$\nabla(A_0, \ldots, A_{n-1}) \leftarrow \vee_{\alpha \in S} \wedge_{k \in m_\alpha} B_{\alpha,k}.$$

Then the least function for $[\![\nabla]\!]$ is

$$\lambda \vec{X}.\ \cup_{\alpha \in S} \cap_{k \in m_\alpha} [\![\nabla_{\alpha,k}]\!](\vec{X}).$$

We now claim that $[\![\nabla]\!]$ is monotonic. We have that the denotations of all temporal operators that appear in the clause are monotonic, and the $\sqcap$-intersection and $\sqcup$-union of monotonic functions are still monotonic. Thus $[\![\nabla]\!]$ is also monotonic because monotonicity is preserved under function composition.

We want to show that $[\![\nabla]\!]$ is finitary too. The denotations of all temporal operators that appear in the definition of ∇ are monotonic and finitary. As the operations $\sqcap$-intersection, $\sqcup$-union and composition ($\circ$) which are used to form new operators from given ones preserve monotonicity and finitariness, and given that we have only finite conjunctions in a clause, we conclude that $[\![\nabla]\!]$ is also finitary; hence continuous. Now we can add ∇ into the pool of temporal operators available to us and keep defining new temporal operators using program clauses.

We have shown that any continuous temporal operator can be defined in TL, and the denotations of operators defined by program clauses are continuous. The next question is whether *any* operator whose denotation is continuous can be defined by a set of program clauses. In case of direct or mutual recursion in operator definitions, fixpoint techniques [Manna, 1974, chapter 5] can be employed to obtain non-recursive definitions of operators. We refer the reader to [Orgun, 1991,

Orgun and Wadge, 1992b] for a more detailed discussion on recursive temporal operator definitions and their interpretations using fixpoint theory. As an alternative, we can use a temporal fixpoint calculus such as that of μTL of Vardi [1988]. For instance, Baudinet [1989] showed that the temporal expressiveness of Templog in the propositional case corresponds to the positive fragment of μTL that allows only least fixpoint operators.

Recursive definitions lead to succinct representations for additional operators. For instance, the following is the recursive definition of $\Diamond$ ("sometime"):

$$\Diamond(A) \text{ — } \mathtt{first}\ A \vee \Diamond(\mathtt{next}\ A).$$

We can also use recursion to define more complicated temporal operators such as **even** where **even** B reads "the current moment is even and B is true". Below is the definition of **even**:

first even(A) — A.
next next even(A) — **even**(**next next** A).

In the above case, we are relaxing the restriction that new temporal operators are defined by a set of program clauses whose heads are identical. Since the fixpoint theory in its current form does not suffice to explain the meaning of unrestricted definitions such as the one above, we must develop another technique (perhaps based on higher-order logic) as the basis for further investigation. Another research problem here is to establish the expressive power of TLP with unrestricted temporal operator definitions, and to investigate some functional completeness results.

We believe that unrestricted operator definitions promise to be a fruitful avenue of further research.

Acknowledgements

Thanks are due to Bill Wadge for many discussions that have led to the formation of the ideas presented in this paper, and Peter Pleasants for his helpful comments in identifying certain properties of neighborhood sets. Lee Flax and Jan Hext made useful comments and suggestions on an earlier draft. Any mistakes are, of course, all mine.

References

[Abadi and Manna, 1987] M. Abadi and Z. Manna. Temporal logic programming. In *Proc. of the 1987 Symposium on Logic Programming*, pages 4–16, San Fransisco, Calif, 1987. IEEE Computer Society Press.

[Abadi and Manna, 1989] M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8:277–295, 1989.

[Aoyagi *et al.*, 1986] T. Aoyagi, M. Fujita, and T. Motooka. Temporal logic programming language Tokio. In E. Wada, editor, *Logic Programming'85*, volume 221 of *LNCS*, pages 138–147. Springer-Verlag, 1986.

[Baudinet *et al.*, 1993] M. Baudinet, J. Chomicki, and P. Wolper. Temporal deductive databases. In A. Tansel and et al. editors, *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings Publishing Company, Redwood City, CA, 1993.

[Baudinet, 1989] M. Baudinet. Temporal logic programming is complete and expressive. In *Conference Record of the Sixteenth ACM Symposium on Principles of Programming Languages*, pages 267–280, Austin, Texas, January 1989. The Association for Computing Machinery.

[Baudinet, 1992] M. Baudinet. A simple proof of the completeness of temporal logic programming. In L. Fariñas del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*, pages 51–83. Oxford University Press, 1992.

[Blackburn, 1993] Patrick Blackburn. Nominal tense logic. *Notre Dame Journal of Formal Logic*, 34:56–83, Winter 1993.

[Bull and Segerberg, 1984] R. Bull and K. Segerberg. Basic modal logic. In D. M. Gabbay and F. Guethner, editors, *Handbook of Philosophical Logic. Vol. II*. pages 1–88. D. Reidel Publishing Company, 1984.

[Chellas, 1980] B. F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.

[Fariñas del Cerro and Penttonen, 1992] L. Fariñas del Cerro and M. Penttonen, editors. *Intensional Logics for Programming*. Oxford University Press, 1992. ISBN 019-853775-1.

[Gabbay and McBrien, 1991] D. Gabbay and P. McBrien. Temporal logic & historical databases. In *Proc. of the 17th Very Large Data Bases Conference*, pages 423–430, Barcelona, Spain, September 1991. Morgan Kauffman, Los Altos, Calif.

[Gabbay, 1987] D. M. Gabbay. Modal and temporal logic programming. In A. Galton, editor, *Temporal Logics and Their Applications*, pages 197–237. Academic Press, 1987.

[Gabbay, 1989] D. M. Gabbay. A temporal logic programming machine [modal and temporal logic programming, Part 2], Department of Computing, Imperial College, November 1989.

[Galton, 1987] A. Galton, editor. *Temporal Logics and Their Applications*. Academic Press, 1987.

[Hale, 1987] R. Hale. Temporal logic programming. In A. Galton, editor, *Temporal Logics and Their Applications*, pages 91–119. Academic Press, 1987.

[Kolaitis and Vardi, 1992] P. G. Kolaitis and M. Y. Vardi. Fixpoint logic vs. infinitary logic in finite-model theory. In *Proc. of the Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 46–57. IEEE Computer Society Press, 1992.

[Kroger, 1987] Fred Kroger. *Temporal Logic of Programs*. Springer-Verlag, Berlin Heidelberg, 1987.

[Lloyd, 1984] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1984.

[Manna and Pnueli, 1981] Z. Manna and A. Pnueli. Verification of concurrent programs: the temporal framework. In Boyer and Moore, editors. *Correctness Prob-*

lem in Computer Science, pages 215–273. Academic Press, 1981.

[Manna, 1974] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, 1974.

[Moszkowski, 1986] B. Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, 1986.

[Orgun and Ma, 1994] M. A. Orgun and W. Ma. An overview of temporal and modal logic programming. In *Proc. of ICTL'94: First International Conference on Temporal Logic*, Gustav Stresemann Institut, Bonn, Germany, July 11–15 1994. Springer-Verlag.

[Orgun and Wadge, 1992a] M. A. Orgun and W. W. Wadge. Theory and practice of temporal logic programming. In L. Fariñas del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*, pages 23–50. Oxford University Press, 1992.

[Orgun and Wadge, 1992b] M. A. Orgun and W. W. Wadge. Towards a unified theory of intensional logic programming. *Journal of Logic Programming*, 13(4):413–440, August 1992.

[Orgun and Wadge, 1993] M. A. Orgun and W. W. Wadge. Chronolog admits a complete proof procedure. In *Proc. of the Sixth International Symposium on Lucid and Intensional Programming*, pages 120–135. Université Laval, Québec City, Québec, Canada, April 26–27 1993.

[Orgun, 1991] M. A. Orgun. *Intensional Logic Programming*. PhD thesis, Department of Computer Science, University of Victoria, Victoria, B.C., Canada, 1991.

[Rolston, 1986] D. W. Rolston. Chronolog: A pure tense-logic-based infinite-object programming language. Department of Computer Science and Engineering, Arizona State University, Tempe, Arizona, August 1986.

[Sadri, 1987] F. Sadri. Three approaches to temporal reasoning. In A. Galton, editor, *Temporal Logics and Their Applications*, pages 121–168. Academic Press, 1987.

[Scott, 1970] D. Scott. Advice on modal logic. In K. Lambert, editor, *Philosophical Problems in Logic*, pages 143–173. D.Reidel Publishing Company, 1970.

[Segerberg, 1982] K. Segerberg. *Classical Propositional Operators*. Oxford University Press, 1982.

[Stoy, 1977] J. E. Stoy. *Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[Tuzhilin and Clifford, 1990] A. Tuzhilin and J. Clifford. A temporal relational algebra as a basis for temporal relational completeness. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proc. of the 16th International Conference on Very Large Data Bases*, pages 13–23, Brisbane, Australia, August 13–16 1990. Morgan Kaufmann Publishers Inc., Los Altos, Calif.

[Vardi, 1988] M. Y. Vardi. A temporal fixpoint calculus. In *Conference Record of the Sixteenth ACM Symposium on Principles of Programming Languages*, pages 250–259. San Diego, Calif. January 1988. ACM Press.

[Wadge and Ashcroft, 1985] W. W. Wadge and E. A. Ashcroft. *Lucid, the Dataflow Programming Language*. Academic Press, 1985.

[Wadge, 1988] W. W. Wadge. Tense logic programming: a respectable alternative. In *Proc. of the 1988 International Symposium on Lucid and Intensional Programming*, pages 26–32, Sidney, B.C., Canada, April 7-8 1988.

ACTLab: an Action Based Toolset

(Verifying reactive systems by talking to them)

Alessandro Fantechi
Dip. di Ingegneria dell'Informazione, Univ. di Pisa and I.E.I. - C.N.R, fantechi@vm.iei.pi.cnr.it
Stefania Gnesi
I.E.I. - C.N.R, gnesi@vm.iei.pi.cnr.it,
Gioia Ristori
Dip. di Informatica, Univ. di Pisa, gioia@di.unipi.it

Abstract In this paper the verification environment ACTLab is presented. ACTLab is centered around the ACTL Model Checker for the branching time action-based temporal logic ACTL. The AMC checks, in linear time, whether a given LTS satisfies a property expressed by an ACTL formula. The helpful explanation facility is available in AMC; also, an interface with the behavioural verification tool AUTO is provided. ACTLab has been designed having a precise purpose in mind: make the specification and the verification of temporal logic formulae easy for the user. Properties of reactive systems are often expressed by natural language sentences, thus many imprecisions occur in the passage from informal expressions of system properties to temporal logic formulae. We thus developed a prototype translator, NL2ACTL, from Natural Language expressions to Temporal Logic formulae, that would help to generate logic formulae, by working out ambiguities by means of interactions with the user. NL2ACTL has been integrated into ACTLab, providing a way to make the expression of properties in the logic easier for the user.

1 Introduction

Labelled Transition Systems (LTS in short) [11] are structures of states related via action labelled arcs. LTSs are widely used semantic models for both reactive systems, characterized by the occurrence of actions over time, and for formulae of action-based temporal logics. Several verification environments have been developed that make available both behavioural and logic automatic verification of systems modelled via LTSs.

The verification environment ACTLab presented in this paper belongs to this class of verification tools, but, differently from other ones, ACTLab has been designed having a precise purpose in mind: make the specification and the verification of temporal logic formulae easy for the user. In fact, our experience on the specification and verification of properties by means of temporal

This work has been partially funded by the "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo", sottoprogetto 4, obiettivo Lambrusco.

logic has shown that many imprecisions frequently occur in the passage from informal expressions of system properties, expressed by natural language sentences, to formulae of a temporal logic. These imprecisions are mainly due to the inherent ambiguities derived from different interpretations of natural language expressions. On the other hand, the requirements capture phase of software development, although systematic in some sense, is usually intuitive and informal; thus, functional requirements (what a program should do) are often expressed by natural language sentences.

We thus looked for an answer to this problem in Natural Language Understanding [1], developing a prototype translator NL2ACTL [5] from Natural Language expressions to the formulae of a particular Temporal Logic ACTL, based on actions. Indeed, ACTL has been recognized as a suitable formalism to express interesting properties of systems specified by LTSs, at the same time avoiding the use of fixed point operators, that make the understanding of formulae rather difficult.

NL2ACTL would help to generate logic formulae, by working out ambiguities by means of interactions with the user. NL2ACTL has been integrated into ACTLab, providing a way to make the expression of properties in the logic easier for the user.

ACTLab is centered around the ACTL Model Checker (AMC in short) for the branching time action-based temporal logic ACTL [4]. The AMC checks, in linear time, whether a given LTS satisfies a property expressed by an ACTL formula. The helpful explanation facility is available in AMC, which provides information on the states and paths which contribute to satisfy or falsify the formula.

The paper is organized as follows. In Section 2 the logic ACTL is presented. In Section 3 we present the overall architecture of ACTLab, while Section 4 and Section 5 describe the Natural language interface NL2ACTL and the verification part of the environment respectively. Finally, Section 6 concludes the paper.

2 An action based logic: ACTL

In this section we present the logic ACTL, whose interpretation domain is that of LTSs; we need therefore some preliminary definitions.

Definition 2.1 (LTS) A Labelled Transition System is a 4-tuple $A = (Q, A \cup \{\tau\}, \rightarrow, 0_Q)$ where:

- Q is a set of states;
- A is a finite, non-empty set of visible actions; the silent action τ is not in A;
- $\rightarrow \subseteq Q \times (A \cup \{\tau\}) \times Q$ is the transition relation; an element $(r,a,q) \in \rightarrow$ is called a transition, and is written as $r -a\rightarrow q$;
- 0_Q is the initial state.

We let $A_\tau = A \cup \{\tau\}$. Moreover, we let r, q, s ... range over states; a, b, over A; α, β, over A_τ. $\Diamond$

Definition 2.2 (Paths) Let $A = (Q, A \cup \{\tau\}, \rightarrow, 0_Q)$ be an LTS.

- A sequence of transitions $(q_0,a_0,q_1)\ (q_1,a_1,q_2)\ldots$ is called a path from q_0; a path that cannot be extended, i.e. is infinite or ends in a state without outgoing transitions, is called a

fullpath; the empty path consists of a single state $q \in Q$ and is denoted by q;

- if $\pi = (q_0,a_0,q_1)\,(q_1,a_1,q_2)\ldots$ we denote the starting state, q_0, of the sequence by first(π) and the last state in the sequence (if the sequence is finite) by last(π); if π is an empty path (i.e. $\pi = q$), then first(π) = last(π) = q;
- concatenation of paths is denoted by juxtaposition: $\pi = \rho\theta$ it is only defined if ρ is finite and last(ρ)=first(θ). When $\pi = \rho\theta$ we say that θ is a suffix of π and that it is a proper suffix if $\rho \neq \theta$, $\theta \in Q$.

We write *path(q)* for the set of fullpaths from q and let π, ρ, σ, η range over paths. ◊

In order to define the logic ACTL an auxiliary logic of actions is introduced.

Definition 2.3 (Action formulae) Let A be a set of actions. The collection *Afor* of action formulae over A is defined by the following grammar, where χ, χ', range over action formulae, and $a \in A$:

$$\chi ::= a \mid \sim\chi \mid \chi\&\chi'$$

The satisfaction of an action formula χ by an action b, notation $b \models \chi$, is defined inductively by:

- $b \models a$ iff $b = a$;
- $b \models \sim\chi$ iff $b \not\models \chi$;
- $b \models \chi\&\chi'$ iff $b \models \chi$ and $b \models \chi'$. ◊

We write `false` for a_0 `&` $\sim a_0$, where a_0 is some arbitrarily chosen action, and `true` for `~false`. Moreover, we will write $\chi \mid \chi'$ for $\sim(\sim\chi \;\&\; \sim\chi')$. An action formula permits the expression of constraints on the actions that can be observed (along a path or at the next step); for instance, `a|b` says that the only possible observations are `a` and `b`, while `true` and `false` stand for "all actions are allowed" and "no actions can be observed" (that is only silent actions can be performed) respectively.

Definition 2.4 (ACTL) Let $A = (Q, A\cup\{\tau\}, \rightarrow, 0_Q)$ be a LTS. The syntax of ACTL formulae over A is defined by the state formulae generated by the following grammar, where φ, φ', ... range over state-formulae, γ over path formulae and χ and χ' are action formulae over A:

$\varphi ::=$ `true` $\mid \sim\varphi \mid \varphi \& \varphi' \mid E\gamma \mid A\gamma$

$\gamma ::=$ `X{`χ`}`$\varphi \mid$ `T`$\varphi \mid$ `[`φ`{`χ`}U{`χ'`}`φ`'] \mid` `[`φ`{`χ`}U`φ`']`

The satisfaction of a state formula φ (path formula γ) by a state q (path ρ), notation $q \models_A \varphi$ or just $q \models \varphi$ ($\rho \models_A \gamma$, or $\rho \models \gamma$), is given inductively by:

- $q \models$ `true` always;
- $q \models \sim\varphi$ iff $q \not\models \varphi$;
- $q \models \varphi$ `&` φ' iff $q \models \varphi$ and $q \models \varphi'$;
- $q \models E\gamma$ iff there exists a path $\theta \in$ path(q) such that $\theta \models \gamma$;
- $q \models A\gamma$ iff for all paths $\theta \in$ path(q) $\theta \models \gamma$;
- $\rho \models$ `[`φ{`$\chi$`}U{`$\chi'$`}`$\varphi$`']` iff there exists $\theta = (q,a,q')\theta'$, suffix of ρ, s.t. $q' \models \varphi'$, $a \models \chi'$, $q \models \varphi$ and for all $\eta = (r,b,r')\eta'$, suffixes of ρ, of which θ is a proper suffix, we have $r \models \varphi$ and ($b \models \chi$ or $b = \tau$);

- $\rho \models [\varphi\{\chi\}U\varphi']$ iff there exists a suffix θ of ρ s. t. first(θ) $\models \varphi'$ and for all $\eta = (r,b,r')\eta'$ of which θ is a proper suffix we have $r \models \varphi$ and ($b \models \chi$ or $b = \tau$);
- $\rho \models X\{\chi\}\varphi$ iff $\rho = (q,a,q')\theta$ and $q' \models \varphi$ and $a \models \chi$;
- $\rho \models T\varphi$ iff $\rho = (q,\tau,q')\theta$ and $q' \models \varphi$. ◊

A set of logical operators and modalities derived from the basic ones could be introduced: `false`, $\varphi \mid \varphi'$ (or), $\varphi \rightarrow \varphi'$ (implies), $F\varphi$ (eventually), $G\varphi$ (always). Other derived modalities are:

- $\langle\chi\rangle\varphi$ for `E[true {false}U{`χ`} `φ`]`
- $[\chi]\varphi$ for $\sim\langle\chi\rangle\sim\varphi$.

Figure 1 gives the informal meaning of the most used ACTL operators.

ACTL

STATE-FORMULAE

```
p  --> true                          "any behaviour"
   --> false                         "no behaviour"
   --> ~p                            "not p"
   --> p & p                         "p and p"
   --> p | p                         "p or p"
   --> E q                  "there exists a path in which q"
   --> A q                                "for all paths q"
   --> <ACTION> p        "there exists a next state reachable
                                    with ACTION, in which p"
   --> [ACTION] p             "for all next states reachable
                                          with ACTION, p"
```

PATH-FORMULAE

```
q  --> [p {r} U {r'} p']     "there exists a state in the path, reached
                              by an action satisfying r', in which p'
                              is true, and all the previous states in
                              the path satisfy p and are reached by
                              an action satisfying r"
   --> G p             "p is true for all the states in the path"
```

ACTION-FORMULAE

```
r  --> true                          "any action"
   --> false                         "no action"
   --> ACTION
   --> ~p                            "not p"
   --> r | r                         "p or p"
```

Figure 1: Informal semantics of the ACTL operators.

3 The ACTLab toolset

ACTLab provides an environment to perform logic verification on concurrent systems, together with the added functionality of a user friendly, Natural Language, interface.

The input to ACTLab is a description, a LTS, of a reactive system, and a set of ACTL formulae that we intend to check on the system. The LTS can be directly given in input to ACTLab, using a particular description format for automata. The format of LTSs in ACTLab is the so called *format commun* fc2, that has been proposed as standard format for automata. Otherwise it is possible to deal with the description of the system given by a term of a particular process algebra and then input this description to a LTS generator. In ACTL this job is performed by AUTO [9], that given a process algebra term [11], like a CCS, LOTOS, or Meje term, produces the LTS in fc2 format.

When the LTS describing the behaviour of a system has been produced it can be given in input to ACTLab, thus starting a verification session of the system properties. In Figure 2 the architecture of the ACTLab environment is shown. ACTLab is basically composed by two different tools: the AMC model checker for the logic ACTL, and the NL2ACTL translator [5] from natural language expressions into ACTL temporal formulae. A third component of the environment is constituted by the imported tool AUTO, that, given a process algebra term, provides the fc2 format of the associated LTS.

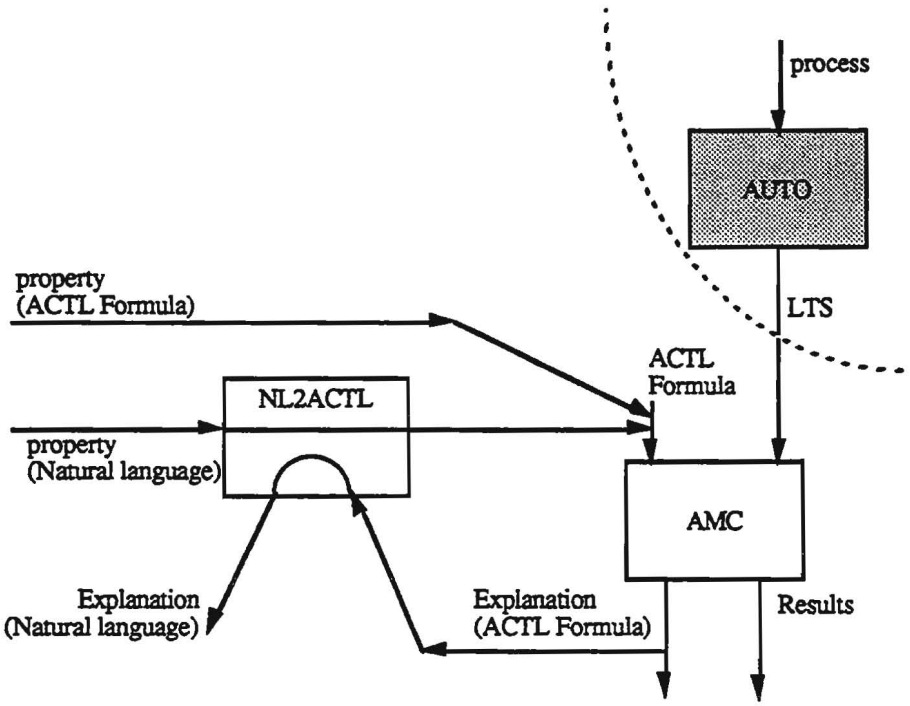


Fig. 2: The ACTLab architecture.

To perform the check of a property, expressed by a natural language expression N (or already formalized by an ACTL formula φ) on a process algebra term T (or on a LTS L), the following steps are needed:

1) Generate the ACTL formula φ expressing the property N using NL2ACTL;
2) Produce the LTS L corresponding to T using AUTO (or give in input the LTS L);
3) Perform the Model Checking of φ on L using AMC.

The output of AMC is the result of the verification test of φ on L. The user may require an explanation; in this case, AMC provides information about the reason of the negative or positive result, in terms of the contribution given to the result by example states and paths.
In the following we will present in more details the components of ACTLab, i. e. NL2ACTL and AMC.

4 NL2ACTL: An automatic translator from Natural Language

The translation tool, NL2ACTL, deals with sentences expressing informal requirements of reactive systems. The behaviour of reactive systems is characterized by the non-deterministic occurrence of actions over time and it can be modelled as a possibly infinite tree, labelled with the actions the system may perform. The requirements express properties of the behaviour of such systems. A translation of informal requirements into ACTL formulae is provided by NL2ACTL.

First step in the construction of NL2ACTL was to collect a set of natural language expressions that contain informal requirements from a specific and restricted application domain, in our case the reactive systems. As a starting point for the development of the tool we considered the translation of the informal requirements into ACTL formulae. This translation entails resolving a twofold problem:

-men communicate with each other informally. When talking within a particular area of interest (such as temporal logics), they often use a kind of jargon (i.e. a particular subset of Natural Language, characterized by its being highly informal and elliptical, hereafter, "abbreviated").

-when communicating with machines, on the other hand, a rigid and formalized language must be used.

For each (abbreviated) sentence in the collected set it is possible to think of one or more extended forms which make all the elements that constitute its meaning explicit. We thus extracted a subset of basic sentences that have in our opinion only one interpretation and hence only one extended form. We then characterized another set of sentences which need to be more detailed to make their meaning unambiguous. For these sentences we have derived more than one extended form, each of which corresponds to a minimal, but significant, different interpretation of the original sentence; for each extended form we have then associated an ACTL formula.

Example.

Sentence:	after action1 has been performed it is never possible that action2 is performed
Extended form:	*for each next state reached by the action "action1" there not exists a computation path in which the action "action2" is performed.*
ACTL formula:	`[action1]AG <action2> false`
Sentence:	if action1 has been performed then action2 is performed.
Extended form 1:	*for each next state reached by the action "action1" the action "action2" is soon performed in all computation paths.*
ACTL formula 1:	`[action1]A[true {false} U {action2} true]`
Extended form 2:	*for each next state reached by the action "action1" the action "action2" is soon performed in a computation path.*
ACTL formula 2:	`[action1]E[true {false} U {action2} true]`
Extended form 3:	*for each next state reached by the action "action1" the action "action2" will be eventually performed in all computation paths.*
ACTL formula 3:	`[action1]A[true {true} U {action2} true]`
Extended form 4:	*for each next state reached by the action "action1" the action "action2" will be eventually performed in a computation path.*
ACTL formula 4:	`[action1]E[true {true} U {action2} true]`

Our translation of the NL sentences of the basic set of informal requirements into ACTL formulae was the basis for building a grammar which identifies the correspondence between natural language and ACTL, and in particular embodies the knowledge of conventions which allow the informative gaps of jargon expressions to be completed without ambiguity.

When this knowledge is not sufficient, some interactions with the user are started to resolve the ambiguities.

The grammar together with a dictionary, which is the source of lexical elements in the sentences, becomes the starting point in the development of the translation tool NL2ACTL.

NL2ACTL was developed by using a general development environment, PGDE, which allows to build, debug and test natural language grammars and dictionaries and which permits, using the given grammar as a basis, an application to be built which recognises input natural language sentences and produces their semantics [10]. With PGDE one can choose the type of semantics, without any restrictions. Furthermore, any grammar formalism can be chosen (and implemented), as the parser is totally independent of the chosen syntactic representation.

NL2ACTL was developed from PGDE as an autonomous application. Each sentence given in input, be it typed separately or stored with other sentences in a file (the requirement document), produces, if recognized (possibly after some dialogue with the user), the corresponding ACTL translation.

5. Model Checking

The AMC model checker is the new version of the ACTL model checker presented in [3]. The latter was obtained exploiting the correspondence [4] between ACTL and CTL [2]. The new model checker, instead, has been based on the algorithm of the EMC model checker for the logic CTL [2]. The algorithm works by recursively labelling each state of the LTS with formulae which are true in that state, and which are subformulae of the formula under checking: the algorithm starts by labelling the states with the simplest subformulae, like `true`, `false`, or $\texttt{EX}_{\texttt{a}}\,\texttt{true}$, whose validity can be simply checked on the state itself and on its transitions. The labelling of states with more complex subformulae is done by looking at the validity of their already checked subformulae on the states. The algorithm terminates when the validity of the original formula has been checked on all the states; if the formula labels the initial state, then it is valid for the LTS. The AMC model checker permits to verify, in linear time, the validity of an ACTL formula on a LTS.

The "counterexample" facility, as implemented in EMC, gives, when an universally quantified formula is not satisfied by the model, an example path which falsifies the formula; this facility could not be used, instead, when an existentially quantified formula is false on the model. AMC extends the above counterexample facility to a "explanation" facility, which can be used to have information on the reason why a formula is false, or why is true, on a given model.

The explanation facility reduces to the counterexample facility when a universally quantified formula is not satisfied by the model; in addition, in the case that an existentially quantified formula is satisfied by the model, this facility shows an example path that verifies the formula. In both cases, each state of the path displays the subformulae which are or are not satisfied by that state, thus showing its contribution to the dissatisfaction or satisfaction of the original formula on the whole model. In the other cases, that is when an existentially quantified formula is falsified by the model or when an universally quantified formula is satisfied by the model, it is not sufficient to show a single path to explain the reason of the model checking result; anyway, the explanation facility still gives the user the possibility, by exploring the computation tree, to follow an example path, choosing among several alternatives. Also in this case, the path displays the subformula satisfied or not satisfied by each state of the path. The realization of the explanation facility is based on the construction of "explicative sequences", which are built by the model checking algorithm to give the reason why that state satisfies or not a given subformula.

6 Conclusions

ACTLab has been integrated into the verification environment JACK [8], which is able to cover a large extent of the formal software development process, such as formalization of requirement [5], rewriting techniques [6], behavioural equivalence proofs [7,9], graph transformations [12], and logic verification [3].

The model checker AMC allows the satisfiabilty of ACTL formulae on the model of a reactive system to be verified. In this respect, requirements can also be maintained and enhanced, due to the benefits obtained from the verification phase: on the basis of the concrete model of the system and the formalization of requirements (a list of temporal logic formulae), the verification of the latter on the former - by means of the model checker - may provide useful information.

NL2ACTL provides a way to make the expression of properties in the logic easier for the user, and it stands between the traditional Natural Language Interface systems and the tools for the automatic processing of informal requirements.

Acknowledgements

We would like to acknowledge the AITech team for their work on the NL2ACTL prototype. We would also like to thank Giovanni Ferro, who developed the AMC model checker.

References

[1] J. Allen: Natural Language Understanding, Addison-Wesley Publishing Company, 1987.

[2] E. M. Clarke, E. A. Emerson, A. P. Sistla: Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications, ACM Toplas 8 (2), 1986, pp.244-263.

[3] R. De Nicola, A. Fantechi, S. Gnesi, G. Ristori: An action-based framework for verifying logical and behavioural properties of concurrent systems, Computer Networks and ISDN Systems, Vol. 25, N. 7, pp. 761-778, North Holland, February 1993.

[4] R. De Nicola, F. W. Vaandrager: Action versus State based Logics for Transition Systems, Proceedings Ecole de Printemps on Semantics of Concurrency, Lecture Notes in Computer Science 469, Springer-Verlag, 1990, pp. 407-419.

[5] A. Fantechi, S. Gnesi, G. Ristori, P. Moreschini: Assisting requirement formalization by means of natural language translation, Formal Methods in System Design, 4, 1994, pp. 243-263.

[6] R. De Nicola, P. Inverardi, M. Nesi: Using Axiomatic Presentation of Behavioural Equivalences for Manipulating CCS Specifications, Lecture Notes in Computer Science 407, Springer-Verlag, 1991, pp. 54-67.

[7] P. Inverardi, D. Yankelevich, C. Priami: Verification of Concurrent Systems in SML, Proceedings SML Workshop, San Francisco, 1992.

[8] JACK: Just Another Concurrency toolKit. Reference Manual, IEI Technical Report, 1993.

[9] E. Madeleine, D. Vergamini: AUTO: A Verification Tool for Distributed Systems Using Reduction of Finite Automata Networks, Proceedings of Formal Description Techniques, II (FORTE '89), North-Holland, 1990, pp. 61-66.

[10] M. Marino: A framework for the Development of Natural Language Grammars, Proceedings of International Workshop on Parsing Technologies, Pittsburgh, 1989, pp. 350-360.

[11] R. Milner: A Calculus of Communicating Systems, Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[12] V. Roy, R. de Simone: Auto and autograph, Proceedings of Workshop on Computer Aided Verification, AMS-DIMACS, June 1990.

Nonmonotonic reasoning about action and change*

John Gooday
Artificial Intelligence Division,
School of Computer Studies,
University of Leeds, Leeds LS2 9JT, UK.

Antony Galton
Department of Computer Science,
University of Exeter, Exeter EX4 4PT, UK.

Abstract

Reasoning about action and change is of fundamental importance to many areas of AI e.g. planning, diagnosis, qualitative modeling etc. In this paper we present a new formalism for representing and solving action and change problems in a straightforward manner. In particular, we introduce a preference strategy that assigns *penalty values* to models containing information that cannot be explained by observations, inertia, motivated actions or ramifications. We demonstrate the simplicity and effectiveness of the system by applying it to a number of representative problems and compare our approach with other formalisms.

1 Introduction

During the last fifteen years numerous formalisms for reasoning about action and change have been developed. Unfortunately, many of these have fallen by the wayside, the victims of newly discovered problem scenarios that refused to be solved easily. As a result of this, nonmonotonic reasoning systems have tended to become more and more complex in an effort to confront the latest, ever more subtle, action and change problems (the system described in (Baker, 1991) is a notable exception to this rule). In response to this trend, we propose a very simple formalism – Transition Calculus – which uses a straightforward penalty counting approach to select minimal models thus avoiding the pitfalls of set-wise minimization.

In this paper we describe a simplified version of Transition Calculus and show how it may be used to solve a variety of representative problems involving inertia and ramification (such as the infamous examples documented in the literature).

*The authors would like to thank Tony Cohn and Nick Gotts for comments on an earlier draft of this paper. John Gooday gratefully acknowledges the support of the CEC under the ESPRIT basic research action. MEDLAR II. 6471 and the SERC under grant GE/H 78955 and studentship 90310659.

2 Transition Calculus

Transition Calculus consists of three parts: a *state language* for describing objects and their properties; a *transition language* that is used to describe events in terms of state changes; and an *action language* that allows the definition of actions and action sequences. Together, these enable us to represent and complete partially specified sequences of actions. As such action sequences often yield many alternative completions, Transition Calculus incorporates a *preference strategy* for selecting the most plausible ones.

2.1 State language

For the purposes of this paper we shall adopt a simplified version of the state language that is essentially propositional in nature. This consists of *atoms, negated atoms, states* and *state sets*. An atom is written $AtomName(Object_1, ..., Object_n)$ where $n \geq 0$. A negated atom is of the form $-s$, where s is an atom. For example, $Colour(Car_1, Red)$ is an an atom and $-Swimming(Rupert)$ is a negated atom. We refer to atoms and negated atoms collectively as *states*. Finally, a *state set* is simply a set of states. The only requirement of a state set is that it must be *compatible* i.e. satisfy the following compatibility conditions

1. $\{\}$ is compatible
2. $\{s\}$ is compatible
3. if $S = \{s_1, ..., s_n\}$ is compatible then every subset of S must be compatible
4. $\{s, \overline{s}\}$ is not compatible
5. if $S \cup \{\overline{s}\}$ is not compatible and $S \cup \{s, \overline{s'}\}$ is not compatible then $S \cup \{\overline{s'}\}$ is not compatible

Here, $\overline{s}$ denotes the complement of s. Sets of incompatible states may be specified according to domain requirements, for example $\{Alive, Dead\}$, $\{-Alive, -Dead\}$.

2.2 Transition language

The transition language is based around the concept of *transition schemas*. These can be thought of as types of event which we characterise in terms of the state changes associated with them. Actual occurrences of transitions belong to the types denoted by these schemas. We will write a transition schema as a tuple

$$\langle\langle S_1 , S_2 \rangle\rangle$$

where

- S_1 is the state set that holds immediately before the transition (the set of *preconditions*)
- S_2 is the state set that holds immediately after the transition (the *the set of postconditions*)

As an example, we can represent an aeroplane's journey from London to Bonn using the following transition schema

$$\langle\langle \{In(London, Aeroplane_1)\}, \{In(Bonn, Aeroplane_1)\}\rangle\rangle$$

A transition schema $\langle\langle S_1, S_2\rangle\rangle$ is said to be a subtype of the schema $\langle\langle S'_1, S'_2\rangle\rangle$, written

$$\langle\langle S_1, S_2\rangle\rangle \sqsubseteq \langle\langle S'_1, S'_2\rangle\rangle,$$

if $S'_2 \subseteq S_2$ and $S'_1 \subseteq S_1$. Consequently, every schema is a subtype of $\langle\langle \{\}, \{\}\rangle\rangle$.

A single transition schema is somewhat limited as a device for knowledge representation. However, we can create more sophisticated structures by combining schemas using *sequential composition*. Two transition schemas, $\langle\langle S_{1,1}, S_{2,1}\rangle\rangle$ and $\langle\langle S_{1,2}, S_{2,2}\rangle\rangle$, may be composed to form a *transition sequence*, in which the first transition occurs immediately prior to the second. We write this as

$$\langle\langle S_{1,1}, S_{2,1}\rangle\rangle \circ \langle\langle S_{1,2}, S_{2,2}\rangle\rangle.$$

Sequential composition is generalised to allow sequences of any length. The above transition sequence is a *subtype* of the transition schema $\langle\langle S_a, S_b\rangle\rangle$ iff $S_a \subseteq S_{1,1}$ and $S_b \subseteq S_{2,2}$. The sequence $\langle\langle S_{1,1}, S_{2,1}\rangle\rangle \circ ... \circ \langle\langle S_{1,n}, S_{2,n}\rangle\rangle$ is said to be Σ-*complete* (complete with respect to the states in some set Σ) iff

for each $s \in \Sigma$
for all $j, 1 \leq j \leq n$, and for $k = 1, 2$
either $s \in S_{k,j}$ or $-s \in S_{k,j}$.

A *disjunctive transition schema* is a schema of the form

$$\langle\langle S_{1,1}, S_{2,1}\rangle\rangle \sqcup \langle\langle S_{1,2}, S_{2,2}\rangle\rangle \sqcup ... \sqcup \langle\langle S_{1,n}, S_{2,n}\rangle\rangle$$

where $n > 1$. Intuitively, we say that a particular occurrence of a transition schema is of the type denoted by the above schema iff it is of the type denoted by one of the disjuncts. Allowing disjunctive schemas in transition sequences leads naturally to the notion of disjunctive normal form (DNF) for transition sequences. A sequence can be put into DNF by expanding it according to

$$A \circ (B \sqcup C) = (A \circ B) \sqcup (A \circ C)$$
where A,B,C are transition sequences.

2.3 Modelling actions

The action language consists of a set, $\mathcal{A}$, of simple actions (e.g. *Shoot*, *Wait*, *Grasp*(*Object*)) from which sequences of actions can be created by sequentially composing one or more actions using the $\odot$ operator: $P_1 \odot ... \odot P_n$. We define a mapping from $\mathcal{A}$ onto the set of all transitions, $\mathcal{T}$, $\mu: \mathcal{A} \mapsto \mathcal{T}$. We can define individual actions by providing the appropriate mapping instance. For example, the action of moving a box from the laboratory to the office could be defined by the following mapping

$$\mu[\![Move(Box_1, Lab_1, Office_1)]\!] = \langle\langle \{In(Box_1, Lab_1)\}, \{In(Box_1, Office_1)\}\rangle\rangle$$

We can extend μ to handle action sequences by using the following rewrite rule

$$\mu[\![P \odot Q]\!] = \mu[\![P]\!] \circ \mu[\![Q]\!]$$

For an action sequence A, where

$$\mu[\![A]\!] = \langle\langle S'_{1,1}, S'_{2,1}\rangle\rangle \circ ... \circ \langle\langle S'_{1,n}, S'_{2,n}\rangle\rangle,$$

we define an *annotation* of A to be the above sequence supplemented by the addition of states to represent any *domain observations*. Domain observations are limited to the initial and final state sets of a particular action sequence. These are expressed as a single transition schema $\langle\langle I, F\rangle\rangle$ where I is a state set containing observations about the initial conditions of the action sequence and F is a state set representing the observed final conditions. This transition is referred to as the *start set* of A. Either, or both of I and F may be empty. The annotation of A by $\langle\langle I, F\rangle\rangle$ is written

$$\mu[\![A]\!]^{I,F} = \langle\langle I \cup S'_{1,1}, S'_{2,1}\rangle\rangle \circ ... \circ \langle\langle S'_{1,n}, F \cup S'_{2,n}\rangle\rangle$$

We can extend the above process to cover disjunctive transition sequences. For example, if

$$\mu[\![A]\!] = \langle\langle S'_{1,1}, S'_{2,1}\rangle\rangle \circ ... \circ \langle\langle S'_{1,n}, S'_{2,n}\rangle\rangle \sqcup \langle\langle S'_{1,n+1}, S'_{2,n+1}\rangle\rangle \circ ... \circ \langle\langle S'_{1,n+m}, S'_{2,n+m}\rangle\rangle \sqcup ...$$

then

$$\mu[\![A]\!]^{I,F} = \langle\langle I \cup S'_{1,1}, S'_{2,1}\rangle\rangle \circ ... \circ \langle\langle S'_{1,n}, F \cup S'_{2,n}\rangle\rangle \sqcup \langle\langle I \cup S'_{1,n+1}, S'_{2,n+1}\rangle\rangle \circ ... \circ \langle\langle S'_{1,n+m}, F \cup S'_{2,n+m}\rangle\rangle \sqcup ...$$

Note that annotations in which observations are inconsistent with A are disallowed. A transition sequence $T = \langle\langle S_{1,1}, S_{2,1}\rangle\rangle \circ ... \circ \langle\langle S_{1,n}, S_{2,n}\rangle\rangle$ is said to be a Σ-*model* for A iff

1. T is a Σ-completion of $\mu[\![A]\!]^{I,F}$
2. For all $1 \leq i \leq n$ and $j = 1, 2$, every $S_{i,j}$ is compatible
3. T obeys the composition rule: for all $1 \leq i \leq n-1, S_{2,i} \cup S_{1,i+1}$ must be compatible.

Now, if we define Σ to be the set of all states that occur in $\mu[\![A]\!]^{I,F}$ then the Σ-models will correspond to classical models (unless otherwise stated, Σ will refer to just this state for the purposes of this paper). We will illustrate this with an example. Consider an action language containing two actions: *Leave* and *Enter*:

$$\mu[\![Leave]\!] = \langle\langle \{InRoom\}, \{-InRoom\}\rangle\rangle$$
$$\mu[\![Enter]\!] = \langle\langle \{-InRoom\}, \{InRoom\}\rangle\rangle$$

The action sequence $Leave \odot Enter$ is therefore defined by

$$\mu[\![Leave \odot Enter]\!] = \langle\langle \{InRoom\}, \{-InRoom\}\rangle\rangle \circ \langle\langle \{-InRoom\}, \{InRoom\}\rangle\rangle$$

As there are no domain observations the start set is

$$\langle\langle \{\}, \{\}\rangle\rangle$$

and

$$\Sigma = \{InRoom\}$$

and so the annotation is identical to the original transition sequence. This sequence is, in fact, already complete so it is also the (only) Σ-model of $Leave \odot Enter$.

Now, let us suppose that in addition to the two actions mentioned we also have a domain observation: a light is on in the room at the beginning of the action sequence. The start set is now

$$\langle\langle \{LightOn\}, \{\} \rangle\rangle$$

This time the annotation is

$$\langle\langle \{InRoom, LightOn\}, \{-InRoom\} \rangle\rangle \circ \langle\langle \{-InRoom\}, \{InRoom\} \rangle\rangle$$

and, by our convention,

$$\Sigma = \{InRoom, LightOn\}$$

The difference between this example and the previous one is that the annotation contains *LightOn* in the precondition state of *Leave*. As we are given no information about whether the light is on or off at any other point during the sequence we are forced to add either *LightOn* or $-LightOn$ to each of the other state sets during the completion process. Doing this we obtain four alternative Σ-models, each of which contains different information about when the light is on or off.

M_1:
$\langle\langle \{InRoom, LightOn\}, \{-InRoom, LightOn\} \rangle\rangle$
$\circ \langle\langle \{-InRoom, LightOn\}, \{InRoom, LightOn\} \rangle\rangle$
M_2:
$\langle\langle \{InRoom, LightOn\}, \{-InRoom, -LightOn\} \rangle\rangle$
$\circ \langle\langle \{-InRoom, -LightOn\}, \{InRoom, -LightOn\} \rangle\rangle$
M_3:
$\langle\langle \{InRoom, LightOn\}, \{-InRoom, LightOn\} \rangle\rangle$
$\circ \langle\langle \{-InRoom, LightOn\}, \{InRoom, -LightOn\} \rangle\rangle$
M_4:
$\langle\langle \{InRoom, LightOn\}, \{-InRoom, -LightOn\} \rangle\rangle$
$\circ \langle\langle \{-InRoom, -LightOn\}, \{InRoom, LightOn\} \rangle\rangle$

2.4 Preferred models

The last example illustrates a problem common in nonmonotonic reasoning: given a set of alternative, incompatible, models for a single scenario how can we select the most appropriate one? Shoham answered this question for logics of nonmonotonicity (Shoham, 1986b) by proposing *model preference semantics* in which a relation, $<$, could be used to place a partial order on models. Intuitively, $<$ corresponds to some *preference criterion* which must be defined with applications in mind. For Transition Calculus we follow a similar approach but define a novel preference criterion somewhat different to Shoham's.

Let

$$M = \langle\langle S_{1,1}, S_{2,1} \rangle\rangle \circ \ldots \circ \langle\langle S_{1,n}, S_{2,n} \rangle\rangle$$

be a Σ-model for the action sequence A which has an annotation

$$\langle\langle S'_{1,1}, S'_{2,1} \rangle\rangle \circ \ldots \circ \langle\langle S'_{1,n}, S'_{2,n} \rangle\rangle$$

We define a penalty assignment function, η, that returns the *penalty* associated with a Σ-model. This is an integer value calculated according to the following scheme:

For each $s \in \Sigma$
for each $k = 1 \ldots n$,
(1) *If* $s \in S_{1,k}$ *and* $s \in S_{2,k}$ *then no penalty.*
(2)(*a*) *If* $s \in S'_{1,k}$ *and* $\overline{s} \in S'_{2,k}$ *then no penalty.*
(3)(*a*) *If* $s \in S_{1,k}$ *and* $\{\overline{s}\} \cup S'_{1,k}$ *is incompatible then no penalty.*
(*b*) *If* $s \in S_{2,k}$ *and* $\{\overline{s}\} \cup S'_{2,k}$ *is incompatible then no penalty.*
(4) *A penalty of 1 is awarded for each instance of every other case.*

The intuitive meaning of the penalty assignment scheme is as follows. Given a model, the penalty assignment function attempts to find explanations for the presence of all the states in each of the transition schemas. If the principles of inertia, ramification or motivated action/observation justify the presence of a state then it is explained and no penalty is awarded. Each of the rules (1) - (3) in the above scheme takes a state and checks for a particular type of explanation.

Rule (1) Inertia. An action will, normally, only result in a few state changes. The majority of states that hold prior to the action will continue to hold, unchanged, after the action has been performed. We can say that these states are subject to inertia. Rule (1) checks for inertia. If a state is present in the preconditions of a schema then inertia explains its presence in the postconditions. (An alternative way of looking at this is that a state present in the postcondition of a schema explains its presence in the precondition – inertia also operates backwards in time.)

Rule (2) Motivated action and observation. In this case a state change from s to $-s$ (or vice versa) between the pre and post conditions of a transition schema can be explained as motivated action or observation if it is in the annotation (i.e. it is a result of an action from the original action sequence or it is an observation).

Rule (3) Ramification. Quite often a state change that may be explained by an action or observation will force additional state changes. For example, suppose the action *Murder* results in a state change from *Alive* to $-Alive$. This, in turn, can force a state change from $-Dead$ to *Dead* as *Alive* and *Dead* are incompatible states. Rule (3) checks for such situations.

If the above rules fail to find an explanation for presence of a state then rule (4) awards a penalty point. This process is repeated for every state in each of the model's transition schemas and the total number of penalty points awarded is the penalty associated with

the model. In effect, the penalty associated with a Σ-model reflects the number of unnecessary state changes it contains.

Let M' be a second Σ-model for A. We say that M is preferred to M' if

$$\eta(M) < \eta(M')$$

M is a maximally preferred Σ-model of A if there is no other Σ-model, M'', such that M'' is preferred to M. Note that there may be more than one maximally preferred Σ-model.

If we apply our preference strategy to the example of section 2.3 we find

$$M_1 < (M_2, M_3) < M_4$$

M_1 has no penalty points ($\eta(M_1) = 0$) as *LightOn* is present in every pre and postcondition. In M_2 *LightOn* is present in $S_{1,1}$ but $-$*LightOn* is a member of $S_{2,1}$. This state change cannot be explained and results in one penalty point. M_3 also earns one penalty point because *LightOn* $\in S_{1,2}$ but $-$*LightOn* $\in S_{2,2}$. Finally, M_4 has a penalty score of 2 because the light's status changes to $-$*LightOn* in $S_{2,1}$ and then back to *LightOn* in $S_{2,2}$.

Lin and Shoham have defined the notion of epistemological completeness for nonmonotonic systems based on the situation calculus (Lin and Shoham, 1991). A theory is epistemologically complete with respect to P, a set of fluents, if each model of the theory is consistent, every model includes the set of domain observations, and for each fluent $p \in P$ either p or $\neg p$ holds in every situation. In Transition Calculus, the simple states play the roles of fluents and action pre and postconditions define situations. A theory corresponds to the action descriptions, the set of domain observations and Transition Calculus itself. By definition, any Transition Calculus theory in which the action descriptions are complete will also be epistemologically complete.

3 Example applications

In this section we demonstrate the power of Transition Calculus by applying it to a number of selected problems involving action and change. Almost all nonmonotonic formalisms to date produce incorrect answers for one or more of these problems. Those that can provide adequate solutions are generally less straightforward and intuitive than Transition Calculus.

3.1 The Yale Shooting Problem

The Yale Shooting Problem (YSP), originally presented in (Hanks and McDermott, 1986), illustrates the need to handle temporal projection correctly. To solve this problem a system must not only use current information to make appropriate inferences about the future state of the world, but it must also avoid spurious conclusions. The YSP can be paraphrased as follows[1]. A loaded gun is pointed at a healthy graduate student by his thesis supervisor. There is a short period of waiting about which no details are given. Then the supervisor fires the gun. The aim is to determine what follows from this sequence of events. In addition, it is known that shooting somebody with a loaded gun will have a detrimental effect on their health. Hanks and McDermott showed that nonmonotonic reasoning formalisms of the time were unable to solve the YSP adequately. As well as the intended solution — the graduate student is injured (presumably leaving the supervisor free to continue his own research uninterrupted!) — these formalisms produce a second, unintended solution — the gun becomes unloaded during the period of waiting and so the student survives the encounter unscathed. Using Transition Calculus we can represent the sequence of events as

$$\mathit{Wait} \odot \mathit{Shoot}$$

The *Shoot* action can be represented by the mapping

$$\begin{aligned}&\mu[\![\mathit{Shoot}]\!] = \\ &\quad \langle\!\langle \{\mathit{Loaded}, -\mathit{Harmed}\}, \{-\mathit{Loaded}, \mathit{Harmed}\} \rangle\!\rangle \sqcup \\ &\quad \langle\!\langle \{-\mathit{Loaded}\}, \{-\mathit{Loaded}\} \rangle\!\rangle \sqcup \\ &\quad \langle\!\langle \{\mathit{Loaded}, \mathit{Harmed}\}, \{-\mathit{Loaded}\} \rangle\!\rangle\end{aligned}$$

and the *Wait* action by a mapping to the empty schema

$$\mu[\![\mathit{Wait}]\!] = \langle\!\langle \{\}, \{\} \rangle\!\rangle$$

Finally, the start set is $\langle\!\langle \{-\mathit{Loaded}, -\mathit{Harmed}\}, \{\} \rangle\!\rangle$ (i.e. all annotations will contain the simple states $-$*Loaded* and $-$*Harmed* as part of the preconditions of the first schema) and we have.

$$\Sigma = \{\mathit{Loaded}, \mathit{Harmed}\}$$

Clearly there are three distinct groups of potential models — those that use the first, second and third versions of *Shoot*. The *composition rule* disallows models that satisfy the schema subtypes

$$\begin{aligned}&\langle\!\langle \Phi, \{-\mathit{Loaded}\} \rangle\!\rangle \circ \langle\!\langle \{\mathit{Loaded}\}, \Psi \rangle\!\rangle \\ &\langle\!\langle \Phi, \{\mathit{Harmed}\} \rangle\!\rangle \circ \langle\!\langle \{-\mathit{Harmed}\}, \Psi \rangle\!\rangle \\ &\langle\!\langle \Phi, \{-\mathit{Harmed}\} \rangle\!\rangle \circ \langle\!\langle \{\mathit{Harmed}\}, \Psi \rangle\!\rangle \\ &\langle\!\langle \Phi, \{\mathit{Loaded}\} \rangle\!\rangle \circ \langle\!\langle \{-\mathit{Loaded}\}, \Psi \rangle\!\rangle\end{aligned}$$

where Φ, Ψ are any states. This leaves the following models (penalties are shown in brackets)

$\eta(M_1) = 0$
$\langle\!\langle \{\mathit{Loaded}, -\mathit{Harmed}\}, \{\mathit{Loaded}, -\mathit{Harmed}\} \rangle\!\rangle$
$\quad \circ \langle\!\langle \{\mathit{Loaded}, -\mathit{Harmed}\}, \{-\mathit{Loaded}, \mathit{Harmed}\} \rangle\!\rangle$

$\eta(M_2) = 1$
$\langle\!\langle \{\mathit{Loaded}, -\mathit{Harmed}\}, \{\mathit{Loaded}, \mathit{Harmed}\}$
$\quad \circ \langle\!\langle \{\mathit{Loaded}, \mathit{Harmed}\}, \{-\mathit{Loaded}, \mathit{Harmed}\} \rangle\!\rangle$

$\eta(M_3) = 1$
$\langle\!\langle \{\mathit{Loaded}, -\mathit{Harmed}\}, \{-\mathit{Loaded}, -\mathit{Harmed}\} \rangle\!\rangle$
$\quad \circ \langle\!\langle \{-\mathit{Loaded}, -\mathit{Harmed}\}, \{-\mathit{Loaded}, -\mathit{Harmed}\} \rangle\!\rangle$

$\eta(M_4) = 2$
$\langle\!\langle \{\mathit{Loaded}, -\mathit{Harmed}\}, \{-\mathit{Loaded}, -\mathit{Harmed}\} \rangle\!\rangle$
$\quad \circ \langle\!\langle \{-\mathit{Loaded}, -\mathit{Harmed}\}, \{-\mathit{Loaded}, \mathit{Harmed}\} \rangle\!\rangle$

$\eta(M_5) = 2$
$\langle\!\langle \{\mathit{Loaded}, -\mathit{Harmed}\}, \{-\mathit{Loaded}, \mathit{Harmed}\} \rangle\!\rangle$
$\quad \circ \langle\!\langle \{-\mathit{Loaded}, \mathit{Harmed}\}, \{-\mathit{Loaded}, \mathit{Harmed}\} \rangle\!\rangle$

[1] This version is slightly simplified in order to make the finer points of our solution as clear as possible (we omit a separate gun loading action). The full problem can be solved in exactly the same manner. it just generates additional penalised models.

$\eta(M_6) = 2$
$\langle\langle\{Loaded, -Harmed\}, \{-Loaded, Harmed\}\rangle\rangle$
$\circ\langle\langle\{-Loaded, Harmed\}, \{-Loaded, -Harmed\}\rangle\rangle$

$\eta(M_7) = 3$
$\langle\langle\{Loaded, -Harmed\}, \{-Loaded, Harmed\}\rangle\rangle$
$\circ\langle\langle\{-Loaded, Harmed\}, \{-Loaded, -Harmed\}\rangle\rangle$

M_1 is the single maximally preferred model. It incurs no penalties. Nothing changes during the wait and the student is injured by the shooting. In the second model, a penalty is incurred because an additional, unnecessary state change occurs: the student becomes harmed during the *Wait* transition. In the third model, the gun's state changes to unloaded and so a penalty is awarded. The fourth and fifth models both contain spontaneous unloading of the gun and unexplained injury to the student so these receive two penalties each. The sixth model contains unexplained injury together with a miraculous, unexplained cure. In the final model not only does the gun become unloaded and the student become injured but he is also cured — these state changes give rise to three penalties. So Transition Calculus prefers the intended solution (the first model) over the other possibilities.

3.2 The Hiding Turkey Problem

This problem is due to (Sandewall, 1992). Like the YSP it is a temporal projection problem but this time ambiguous data are included that force us to favour two solutions equally. In our version the world is the same as for the YSP but with the following addition. Before the supervisor can shoot the student he must load the (initially unloaded) gun. If the student sees the supervisor doing this he will hide and thereby avoid being harmed by the shooting. It is not known whether the student actually sees the gun being loaded. We first rewrite the *Shoot* rule to include the necessary information about hiding and introduce a transition schema representation of the *Load* action

$$\begin{aligned}
\mu[\![Shoot]\!] &= \langle\langle\{Loaded, -Harmed, -Hiding\}, \\
&\qquad \{-Loaded, Harmed, -Hiding\}\rangle\rangle \sqcup \\
&\quad \langle\langle\{Loaded, Hiding\}, \{-Loaded\}\rangle\rangle \sqcup \\
&\quad \langle\langle\{-Loaded\}, \{-Loaded\}\rangle\rangle \\
\mu[\![Load]\!] &= \langle\langle\{\}, \{Loaded, Hiding\}\rangle\rangle \sqcup \\
&\quad \langle\langle\{\}, \{Loaded, -Hiding\}\rangle\rangle \\
\mu[\![Wait]\!] &= \langle\langle\{\}, \{\}\rangle\rangle
\end{aligned}$$

We can now state the problem: find all models of the action sequence

$$Load \odot Wait \odot Shoot$$

with start set

$$\langle\langle\{-Loaded, -Harmed, -Hiding\}, \{\}\rangle\rangle$$

There are 100 models for this problem. However, there are only two maximally preferred models (with penalties of 0):

$\langle\langle\{-Loaded, -Harmed, -Hiding\},$
$\{-Loaded, -Harmed, -Hiding\}\rangle\rangle$
$\circ\langle\langle\{Loaded, -Harmed, -Hiding\},$
$\{-Loaded, -Harmed, -Hiding\}\rangle\rangle$
$\circ\langle\langle\{Loaded, -Harmed, -Hiding\},$
$\{-Loaded, Harmed, -Hiding\}\rangle\rangle$
and
$\langle\langle\{-Loaded, -Harmed, -Hiding\},$
$\{Loaded, -Harmed, Hiding\}\rangle\rangle$
$\circ\langle\langle\{Loaded, -Harmed, Hiding\},$
$\{Loaded, -Harmed, Hiding\}\rangle\rangle$
$\circ\langle\langle\{Loaded, -Harmed, Hiding\},$
$\{-Loaded, -Harmed, Hiding\}\rangle\rangle$

In the first model the student does not hide and is injured by the shooting. In the second model the student hides during the loading action and remains hidden throughout the waiting period and the shooting, thus escaping injury. These correspond to the two intended solutions.

3.3 The Stanford Murder Mystery

The Stanford Murder Mystery (SMM) (Baker and Ginsberg, 1989) illustrates a different kind of problem: postdiction – reasoning backwards in time to explain an observation. The supervisor fires a gun at his student and then there is a period of waiting. It is known that initially the student is unharmed and that after the wait he is injured. No information about the loaded state of the gun is given. The world is similar to that of the YSP (i.e. the student does not have the option to hide). We are required to reason backwards in time in order to explain the student's injury. In Transition Calculus the problem is to find the maximally preferred models of the action sequence *Shoot* $\odot$ *Wait* with start set:

$$\langle\langle\{-Harmed\}, \{Harmed\}\rangle\rangle$$

The single maximally preferred model (with a penalty score of 0) is the intended one in which the gun is initially loaded and the student is injured by the shooting:

$\langle\langle\{Loaded, -Harmed\}, \{-Loaded, Harmed\}\rangle\rangle$
$\circ\langle\langle\{-Loaded, Harmed\}, \{-Loaded, Harmed\}\rangle\rangle$

Using the alternative version of shoot results in the gun being initially unloaded and so the student becomes spontaneously injured rather than harmed by the shooting. All such models incur penalties as does the model using the correct version of shoot but with the gun spontaneously reloading after the shooting action.

3.4 The Stolen Car Problem

This problem, due to (Kautz, 1986) has proved to be particularly troublesome to many nonmonotonic systems because of its multiple solutions, surprise factor and postdiction requirements. A car is parked at time $t = 1$ and its owner leaves it for a while. When he returns (say, at $t = 4$) he discovers that the car has been stolen (this is the surprise). The problem is to determine when the car disappeared. We will use a separate *Wait* for each time period $t = 1.2.3$. The problem in Transition Calculus

is to find the maximally preferred models of the action sequence $Wait \odot Wait \odot Wait$ with the start set

$$\langle\langle\{Parked\},\ \{\neg Parked\}\rangle\rangle$$

All of the possible models have penalties associated with them. There are three maximally preferred models with a score of one point each

$$\begin{aligned} M_1 &= \langle\langle\{Parked\},\{\neg Parked\}\rangle\rangle \\ &\quad \circ\langle\langle\{\neg Parked\},\{\neg Parked\}\rangle\rangle \\ &\quad\quad \circ\langle\langle\{\neg Parked\},\{\neg Parked\}\rangle\rangle \\ M_2 &= \langle\langle\{Parked\},\{Parked\}\rangle\rangle \\ &\quad \circ\langle\langle\{Parked\},\{\neg Parked\}\rangle\rangle \\ &\quad\quad \circ\langle\langle\{\neg Parked\},\{\neg Parked\}\rangle\rangle \\ M_3 &= \langle\langle\{Parked\},\{Parked\}\rangle\rangle \\ &\quad \circ\langle\langle\{Parked\},\{Parked\}\rangle\rangle \\ &\quad\quad \circ\langle\langle\{Parked\},\{\neg Parked\}\rangle\rangle \end{aligned}$$

In other words, although Transition Calculus shows equal preference for models in which the car was taken once only (i.e. not taken, returned and taken again) it is impossible to decide at which particular time point the theft occurred. Again, this is the intended conclusion.

3.5 Russian Turkey Shoot

In the Russian Turkey Shoot, the gun is initially unloaded. The supervisor loads the gun, spins the chamber and fires. Spinning the chamber has the (random) result that the gun either becomes unloaded, or remains loaded. Initially the gun is unloaded and the student unharmed. We represent the actions as

$$\begin{aligned} \mu[\![Load]\!] &= \langle\langle\{\},\{Loaded\}\rangle\rangle \\ \mu[\![Spin]\!] &= \langle\langle\{Loaded\},\{\neg Loaded\}\rangle\rangle \sqcup \\ &\quad \langle\langle\{Loaded\},\{Loaded\}\rangle\rangle \sqcup \\ &\quad \langle\langle\{\neg Loaded\},\{\}\rangle\rangle \\ \mu[\![Shoot]\!] &= \langle\langle\{Loaded,\neg Harmed\}, \\ &\quad\quad \{\neg Loaded, Harmed\}\rangle\rangle \sqcup \\ &\quad \langle\langle\{Loaded, Harmed\},\{\neg Loaded\}\rangle\rangle \sqcup \\ &\quad \langle\langle\{Loaded\},\{\neg Loaded\}\rangle\rangle \end{aligned}$$

Our task is to find the maximally preferred models of

$$Load \odot Spin \odot Shoot$$

with start set

$$\langle\langle\{\neg Loaded, \neg Harmed\},\{\}\rangle\rangle$$

Of the fourteen models two are maximally preferred (both have penalties of 0)

$$\begin{aligned} &\langle\langle\{\neg Loaded, \neg Harmed\},\{Loaded, \neg Harmed\}\rangle\rangle \\ &\quad \circ\langle\langle\{Loaded, \neg Harmed\},\{\neg Loaded, \neg Harmed\}\rangle\rangle \\ &\quad\quad \circ\langle\langle\{\neg Loaded, \neg Harmed\},\{\neg Loaded, \neg Harmed\} \\ &\textit{and} \\ &\langle\langle\{\neg Loaded, \neg Harmed\},\{Loaded, \neg Harmed\}\rangle\rangle \\ &\quad \circ\langle\langle\{Loaded, \neg Harmed\},\{Loaded, \neg Harmed\}\rangle\rangle \\ &\quad\quad \circ\langle\langle\{Loaded, \neg Harmed\},\{\neg Loaded, Harmed\} \end{aligned}$$

In the first model, spinning the chamber unloads the gun and the student is unharmed by the shooting. In the second model, the spinning action fails to unload the gun and the student is harmed as a result of the shooting action. These correspond to the two intended models.

3.6 The Talking Student Problem

This problem (normally referred to as the Walking Turkey Problem) is an example of how ramification plays an important part in reasoning about action and change. The problem is the same as the Yale Shooting but with the additional observation that the student is talking prior to the shooting. It is also known that, in general, students who have been harmed by firearms do not continue to talk. We use the same *Wait* and *Shoot* actions as before. However, this time we need to identify an incompatible state set: $\{Harmed, Talking\}$. We can now turn our attention to solving the problem of what happens when the sequence $Shoot \circ Wait$ is performed. We must find the maximally preferred models of this sequence with start set

$$\langle\langle\{Loaded, \neg Harmed, Talking\},\{\}\rangle\rangle$$

The single maximally preferred model is

$$\begin{aligned} &\langle\langle\{Loaded, \neg Harmed, Talking\}, \\ &\quad\quad \{Loaded, \neg Harmed, Talking\}\rangle\rangle \\ &\quad \circ\langle\langle\{Loaded, \neg Harmed, Talking\}, \\ &\quad\quad\quad \{\neg Loaded, Harmed, \neg Talking\}\rangle\rangle \end{aligned}$$

with a penalty of zero. The important point here is that $\neg Talking$ has been inferred as a ramification of the action sequence – it was not explicitly specified as a postcondition of either *Wait* or *Shoot*. No penalty was incurred thanks to rule (3) of the penalty assignment function.

3.7 Ticketed Car Problem

This is a variant of the Stolen Car Problem due to (Sandewall, 1992). In a certain street parking is allowed only during the day. If a car is left in the street overnight it may, randomly, receive a parking ticket. No parking tickets are issued during the day. If a car receives a ticket it cannot become unticketed on subsequent nights of illegal parking. In the problem, an initially unticketed car is left in the street for two consecutive nights. At the end of this time it is found to be ticketed. We must find out what happened. The problem differs from the Stolen Car scenario in that getting a ticket is not an exception, whereas the theft of the car was.

First, we formulate descriptions of night and daytime parking:

$$\begin{aligned} Night &= \langle\langle\{\neg Ticketed\},\{Ticketed\}\rangle\rangle \sqcup \\ &\quad \langle\langle\{\},\{\}\rangle\rangle \\ Day &= \langle\langle\{\},\{\}\rangle\rangle \end{aligned}$$

To solve the problem we need to find the maximally preferred models of the sequence $Night \odot Day \odot Night$ with start set

$$\langle\langle\{\neg Ticketed\},\{Ticketed\}\rangle\rangle.$$

There are two:

$$\begin{aligned} M_1 &= \langle\langle\{\neg Ticketed\},\{Ticketed\}\rangle\rangle \\ &\quad \circ\langle\langle\{Ticketed\},\{Ticketed\}\rangle\rangle \\ &\quad\quad \circ\langle\langle\{Ticketed\},\{Ticketed\}\rangle\rangle \\ M_2 &= \langle\langle\{\neg Ticketed\},\{\neg Ticketed\}\rangle\rangle \\ &\quad \circ\langle\langle\{\neg Ticketed\},\{\neg Ticketed\}\rangle\rangle \\ &\quad\quad \circ\langle\langle\{\neg Ticketed\},\{Ticketed\}\rangle\rangle \end{aligned}$$

In the first model, the car becomes ticketed during the first night and remains ticketed. In the second model the car remains unticketed until the second night and is then ticketed. These are Sandewall's intended solutions.

4 Comparisons

Transition Calculus has proved surprisingly effective at solving problems which require effective reasoning about *inertia* and *actions with alternative possible results*, such as those featured in section 3. The ability to handle such problems is a requirement of any realistic intelligent agent.

Transition Calculus is not based on chronological minimisation[2] – the idea that as little as possible be known for as long as possible – so it does not suffer from the well documented drawbacks of formalisms that use this approach (Kautz, 1986; Lifschitz, 1987b; Shoham, 1986a). Unlike Haugh's causal minimisation system (Haugh, 1987), Transition Calculus is able to handle postdiction and ramification, areas which the situation calculus based system of (Lifschitz, 1987a) also has difficulty with. Motivated Action Theory (Morgenstern and Stein, 1988) copes well with most of the problems in section 3 and is, perhaps, the system closest to Transition Calculus. MAT consists of a set of causal rules which can be used to describe the effects of actions, and a set of persistence rules, used to model inertia. The model preference criteria works by minimising *unmotivated change* (change that doesn't result from causal rules). The main drawback with this system is that a comprehensive set of persistence rules must be specified for each theory whereas Transition Calculus is able to do away with persistence rules altogether.

Baker has developed a novel approach to circumscriptive reasoning within situation calculus (Baker, 1991). When minimising abnormality he allows the situation calculus' *Result* function to vary, rather than *Holds*. This entails generating all possible situations that could conceivably result from an action and a second order axiom is provided for this purpose. Using circumscription in this manner has the effect of minimising change with respect to abnormality. Baker has demonstrated that this approach works for prediction and postdiction problems (Yale Shooting Problem, Stanford Murder Mystery, Stolen Car Problem). Like Transition Calculus, the system is concerned with inertia and ramification – qualification problems are not treated[3]. One of the chief virtues of Baker's system is its simplicity – it makes use of straightforward circumscription. However, it is debatable whether this is approach is as conceptually simple as Transition Calculus which does not use higher order axioms. Another promising formalism is that of (Sandewall, 1992). This is a first order logic with a CAMOC preference relation on models. CAMOC is a chronological minimisation system based on occlusion and filtering. In filtering (Sandewall, 1989) observations are used to rule out models that have already undergone chronological minimisation, rather than minimising the models augmented by observations. In occlusion, changes that come about as a result of action laws are not included in the minimisation process during the temporal scope of the action. Again, it is not clear that CAMOC is as conceptually straightforward as Transition Calculus or, in the light of syntactic constraints Sandewall places on the action language, as expressive.

[2]See (Gooday and Galton, 1992) for an alternative, chronological minimisation based version of Transition Calculus

[3]See (Gooday, 1994) for a partial treatment of qualification within Transition Calculus

5 Conclusions and further work

In this paper we have proposed a nonmonotonic reasoning formalism based on the notion of transitions – event-based action descriptions. We have shown that a simple preference criterion, in which models are awarded penalty points for unexplained state changes, is sufficient to provide solutions to a wide variety of representative problems which other, more complex, approaches have failed to do. We believe that Transition Calculus is the most conceptually simple system for nonmonotonic reasoning about action and change to date.

Problem	Time (CPU secs)
Yale Shooting (Simplified)	0.0
Yale Shooting (Original)	0.1
Hiding Turkey	2.0
Murder Mystery	0.0
Stolen Car	0.1
Russian Turkey	0.2
Walking Turkey	0.1
Ticketed Car	0.1

Table 1: Performance of Sicstus Prolog 2.1(#6) implementation on SPARCstation 10

Our system is readily implementable – a simple Prolog program has been written to demonstrate this. The examples of section three can all be solved relatively quickly using this software. For example, all models of the Yale Shooting Problem can be found and the maximally preferred model selected in less than 0.1 seconds of CPU time on a SPARCstation 10 running Sicstus Prolog version 2.1 (see table 1). It is expected that significant efficiency increases can be obtained by further combining model generation with testing and by the use of heuristics to guide model generation.

In this paper we have focussed on the nonmonotonic aspects of Transition Calculus. In fact, Transition Calculus can be incorporated into a full event-based temporal logic in which temporal incidence predicates are used to map transitions and states onto time points and intervals in the manner of (Gooday and Galton, 1992). The resulting system may be used to define a nonmonotonic language for planning applications.

References

Baker, A. B. (1991). Nonmonotonic reasoning in the framework of the situation calculus. *Artificial Intelligence*, 49:5–23. Special issue on knowledge representation.

Baker, A. B. and Ginsberg, M. L. (1989). Temporal projection and explanation. In *Proceedings IJCAI-89*, pages 906–911.

Gooday, J. M. (1994). *A transition-based approach to reasoning about action and change*. PhD thesis, Department of Computer Science, University of Exeter, Exeter EX4 4PT, England. To appear.

Gooday, J. M. and Galton, A. P. (1992). A calculus of transitions. In *Logic and Change, GWAI-92*.

Hanks, S. and McDermott, D. (1986). Default reasoning, nonmonotonic logics, and the frame problem. In *Proceeding AAAI-86*, pages 328–333.

Haugh, B. A. (1987). Simple causal minimizations for temporal persistence and projection. In *Proceedings AAAI-87*, pages 218–223.

Kautz, H. A. (1986). The logic of persistence. In *Proceedings AAAI-86*, pages 401–405.

Lifschitz, V. (1987a). Formal theories of action. In Brown, F., editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, pages 35–58. Morgan-Kaufman, Los Altos, CA.

Lifschitz, V. (1987b). Pointwise circumscription. In Ginsberg, M. A., editor, *Readings in Nonmonotonic Reasoning*. Morgan-Kaufman, Los Altos, CA.

Lin, F. and Shoham, Y. (1991). Provably correct theories of action. In *Proceedings AAAI-91*, pages 349–709.

Morgenstern, L. and Stein, L. (1988). Why things go wrong: A formal theory of causal reasoning. In *Proceedings AAAI-88*, pages 518–523.

Sandewall, E. (1989). Filter preferential entailment for the logic of action in almost continuous worlds. In *Proceedings IJCAI-89*, pages 894–899.

Sandewall, E. (1992). Features and fluents (a systematic approach to the representation of knowledge about dynamical systems). Technical Report LiTH-IDA-R-92-30, Department of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden.

Shoham, Y. (1986a). Chronological ignorance: Time, nonmonotonicity, necessity and causal theories. In *Proceedings AAAI-86*, pages 389–393.

Shoham, Y. (1986b). Reified temporal logics: Semantical and ontological considerations. In *Proceedings 7th ECAI*, pages 390–397, Brighton, UK.

Mapping an LPTL formula into a Büchi alternating automaton accepting its models

Amar ISLI
LIPN - CNRS URA 1507
Institut Galilée, Université Paris XIII
Av. J.B. Clément, 93430 Villetaneuse, France
E-mail: isli@ura1507.univ-paris13.fr

Abstract

We propose an effective construction mapping a formula of Linear Propositional Temporal Logic (LPTL) into a Büchi alternating automaton accepting its models . This transforms the satisfiability problem of an LPTL formula to the emptiness problem of the associated alternating automaton. We then give the main steps of how to simulate, in the case of Büchi, an alternating automaton by a usual nondeterministic one. This makes arrive to a new approach to the problem of associating a Büchi nondeterministic automaton to an LPTL formula. This new approach is advantageousely compared to the well known one encountered in the literature, which is of Wolper.

Keywords: alternating automaton, temporal logic, interpretation, run, elementary formula, and closure.

1 Introduction

We investigate in this paper the problem of mapping an LPTL formula into a Büchi alternating automaton accepting its models. We also include the main steps of a method simulating, in the case of Büchi, an alternating automaton by a usual nondeterministic one. The global idea is clearly a new approach to the satisfiability problem of LPTL: for a formula f of LPTL, first trasform its satisfiability problem to the emptiness problem of a Büchi alternating automaton, and then the emptiness problem of the alternating automaton to the well-known emptiness problem of a usual Büchi nondeterministic automaton.

The well known method in the literature associating to an LPTL formula a Büchi nondeterministic automaton accepting its models is the method of Wolper ([Wol 88]). This method starts by constructing a first automaton called "local automaton", and a second automaton called "eventuality automaton". The Büchi nondeterministic automaton is obtained by the cross product of the local automaton and the eventuality automaton. It will be given a comparison of the method of Wolper and ours.

2 Background: the logic LPTL

LPTL is an extension of classical propositional logic. This extension is obtained by the adding of the temporal operators $\bigcirc$ (the "next"), $\Diamond$ (the "eventually") and U (the "until").

Syntax

LPTL formulas are built from the following alphabet:

- a finite or enumerable set $\mathcal{P}$ of atomic propositions $p, q, r, \ldots$,
- the boolean constructors $\wedge$ and $\neg$, and
- the temporal operators $\bigcirc$, $\Diamond$ and U.

LPTL (well-formed) formulas are defined recursively in the following manner:

(a) all atomic proposition $p \in \mathcal{P}$ is a formula,

(b) if f and g are formulas, then so are $f \wedge g$, $\neg f$, $\bigcirc f$, $\Diamond f$ and fUg, and

(c) all LPTL formula is generated by rules (a) and (b) above.

Remark 2.1 *The temporal operator G (the "always") is used as an abbreviation of $\neg\Diamond\neg$: $Gf \equiv \neg\Diamond\neg f$.*

Semantics

LPTL is complete for the class $\mathcal{K}$ of structures $\xi = (S, N, \pi)$ defined as follows ([Man et al. 84, Wol 83, Wol 85]):

(a) S is a finite or enumerable state set,

(b) N: $S \longrightarrow S$ is a successor function mapping each state s into a unique successor state $N(s)$, and

(c) π: $S \longrightarrow 2^{\mathcal{P}}$ is a function mapping each state s into a set of atomic propositions.

Remark 2.2 *In a structure $\xi = (S, N, \pi)$, the function π partitions in each state $s \in S$ the set $\mathcal{P}$ of atomic propositions into the set $\pi(s)$ of atomic propositions true in s, and the set $\mathcal{P} \setminus \pi(s)$ of atomic propositions false in s. Hence, π is a function assigning truth values to atomic propositions in each state.*

Satisfiability

Let $\xi = (S, N, \pi)$ be a structure of the class $\mathcal{K}$, and s a state from S. The satisfiability of an LPTL formula f by the state s of the stucture, denoted by $< \xi, s > \models f$, is defined recursively as follows:

(a) if f is an atomic proposition then: $< \xi, s > \models f$ if and only if $f \in \pi(s)$,

(b) $< \xi, s > \models f_1 \wedge f_2$ if and only if $< \xi, s > \models f_1$ and $< \xi, s > \models f_2$,

(c) $< \xi, s > \models \neg f_1$ if and only if $not(< \xi, s > \models f_1)$,

(d) $< \xi, s > \models \bigcirc f_1$ if and only if $< \xi, N(s) > \models f_1$,

(e) $< \xi, s > \models \Diamond f_1$ if and only if $(\exists i \geq 0)(< \xi, N^i(s) > \models f_1)$, and

(f) $< \xi, s > \models f_1 U f_2$ if and only if

- $(\forall i \geq 0)(< \xi, N^i(s) > \models f_1)$, or
- $(\exists i \geq 0)(< \xi, N^i(s) > \models f_2$ and $\forall j (0 \leq j < i \Rightarrow < \xi, N^j(s) > \models f_1))$.

Remark 2.3 *In the definition above of satisfiability:*

- $N^0(s)$ *stands for* s, *and*
- $N^{i+1}(s)$, $i \geq 0$, *stands for* $N(N^i(s))$.

An interpretation $\imath$ consists of a structure $\xi = (S, N, \pi)$ and an initial state s_0: it is denoted by $\imath =< \xi, s_0 >$.
An interpretation $\imath =< \xi, s_0 >$ satisfies a formula f if and only if $< \xi, s_0 > \models f$.
An interpretation satisfying a formula f is a model of f.
The satisfiability problem for LPTL consists of answering the question of whether a given formula of this logic is satisfiable. That is, whether it has a model.

Remark 2.4 *An interpretation over a set* $\mathcal{P}$ *of propositions can be viewed (and will be viewed so below) as an infinite word on the alphabet* $2^{\mathcal{P}}$ *consisting of the set of subsets of* $\mathcal{P}$.

3 Büchi automata on interpretations

The automata we are concerned with is this paper are automata on interpretations. That is, automata of infinite sequences of sets of atomic propositions.

Given a set A of literals built from a set $\mathcal{P}$ of atomic propositions, we define $\neg A$ as being the following set of literals:

$$\neg A = \{p : p \in \mathcal{P} \text{ and } \neg p \in A\} \cup \{\neg p : p \in \mathcal{P} \cap A\}.$$

Definition 1 *A Büchi nondeterministic automaton on interpretations is a 5-tuple* $M =< Q, \Sigma, \delta, q_0, F >$ *defined as follows:*

- Σ *is the input alphabet:* $\Sigma = \mathcal{P} \cup \neg\mathcal{P}$, $\mathcal{P}$ *being a finite or enumerable set of atomic propositions,*
- Q *is a finite state set,*
- $\delta : Q \longrightarrow 2^{2^{\Sigma} \times Q}$ *is a transition function,*
- $q_0 \in Q$ *is the initial state of* M, *and*
- $F \subseteq Q$ *is a set of distinguished states.*

Definition 2 *A run of a Büchi nondeterministic automaton* $M =< Q, \Sigma, \delta, q_0, F >$ *on an interpretation* $\imath = \epsilon_0 \epsilon_1 \epsilon_2 \ldots \epsilon_n \ldots$ *is an infinite sequence* $c = q_{i_0} q_{i_1} q_{i_2} \ldots q_{i_n} \ldots$ *of states of* M *verifying the following:*

q	$\delta(q)$
q_0	$\{(\{\}, q_1)\}$
q_1	$\{(\{\}, q_2), (\{\}, q_3)\}$
q_2	$\{(\{j\}, q_2), (\{\neg c, j\}, q_4), (\{\neg e, j\}, q_4), (\{\neg c, j\}, q_5), (\{\neg e, j\}, q_5)\}$
q_3	$\{(\{j\}, q_2), (\{j\}, q_3), (\{j\}, q_6)\}$
q_4	$\{(\{j\}, q_2), (\{\neg c, j\}, q_3), (\{\neg e, j\}, q_3)\}$
q_5	$\{(\{j\}, q_2), (\{j\}, q_3)\}$
q_6	$\{(\{j\}, q_2), (\{\neg c, j\}, q_3), (\{\neg e, j\}, q_3), (\{j\}, q_6)\}$

Figure 1: the transition function δ of the example.

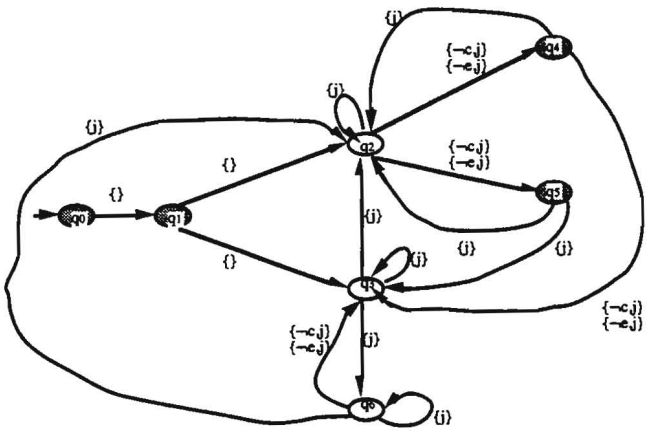


Figure 2: the graphical representation of the automaton M of the example.

- $q_{i_0} = q_0$,
- $\forall n \geq 0\ \exists (A, q) \in \delta(q_{i_n})$ *such that*:
  - $q = q_{i_{n+1}}$, *and*
  - *for all atomic proposition* p:
    - $(p \in A) \Rightarrow (p \in e_n)$,
    - $(\neg p \in A) \Rightarrow (p \notin e_n)$.

Example

Let us consider the following Büchi nondeterministic automaton $M =< Q, \Sigma, \delta, q_0, F >$:

- $\Sigma = \mathcal{P} \cup \neg\mathcal{P}$, with $\mathcal{P} = \{c, e, j\}$,
- $Q = \{q_0, q_1, \ldots, q_6\}$,
- the initial state is q_0,
- the transition function is given by the table of Figure 1, and
- $F = \{q_0, q_1, q_4, q_5\}$.

The graphical representation of the automaton M is given by Figure 2. The dashed states are the distinguished states of M. The ingoing arrow shows the initial state.
The infinite sequence $q_0 q_1 (q_2 q_4)^{\omega}$ is a run of the automaton M on each of the interpretations $\{c, e, j\}^2(\{c, j\}\{e, j\})^{\omega}$, $\{c, e\}\{c, j\}(\{j\}\{c, j\})^{\omega}$ and $\{j\}\{c\}(\{j\}\{c, e, j\})^{\omega}$. The run $q_0 q_1 (q_2 q_4)^{\omega}$ is accepting, for it repeats infinitely often the distinguished state q_4.

Definition 3 *A Büchi alternating automaton on interpretations is a 5-tuple* $M =< \mathcal{L}(\Sigma \cup Q), \delta, q_0, q_v, F >$ *defined in the following manner:*

- Σ *is the input alphabet:* $\Sigma = \mathcal{P} \cup \neg\mathcal{P}$, $\mathcal{P}$ *being a finite or enumerable set of atomic propositions,*
- $Q \cup \{q_v\}$ *is a finite state set,* $q_v \notin Q$ *being a special state called "the valid state" of* M,
- $\mathcal{L}(\Sigma \cup Q)$ *is the free distributive lattice*[1] *generated by* $\Sigma \cup Q$,
- $\delta : Q \cup \{q_v\} \longrightarrow \mathcal{L}(\Sigma \cup Q) \cup \{q_v\}$ *is a transition function verifying* $\delta(q_v) = q_v$,
- $q_0 \in Q$ *is the initial state of* M, *and*
- F *is a set of distinguished states including* q_v.

Definition 4 *A run of a Büchi alternating automaton* $M =< \mathcal{L}(\Sigma \cup Q), \delta, q_0, q_v, F >$ *on an interpretation* $\imath = e_0 e_1 \ldots e_n \ldots \in (2^{\mathcal{P}})^\omega$ *is a tree defined as follows:*

- *all node of the tree is labeled by a state of* M,
- *the label of the root is* q_0, *the initial state of* M, *and*
- *if*
  - v *is a node of level* n *labeled by* q,
  - e_n *is the* n^{th} *element of the interpretation* $\imath$, *i.e. the* n^{th} *element of the infinite sequence* $e_0 e_1 \ldots e_n \ldots$,
  - $\{q_1, \ldots, q_m\}$ *is the set of labels of of the immediate successors of* v, *and*
  - $t_1 \vee t_2 \vee \ldots \vee t_r$ *is the disjunctive normal form of* $\delta(q)$,

  then there exists $j \in \{1, \ldots, r\}$ *such that:*
  - $\{q_1, \ldots, q_m\} =$
    $\begin{cases} \{p \in Q : p \text{ occurs in } t_j\} \\ \qquad \text{if } \{p \in Q : p \text{ occurs in } t_j\} \neq \emptyset, \\ \{q_v\} \text{ otherwise,} \end{cases}$
  - $\{p \in \mathcal{P} : p \text{ occurs in } t_j\} \subseteq e_n$, *and*
  - $e_n \cap \{p \in \mathcal{P} : \neg p \text{ occurs in } t_j\} = \emptyset$.

We need some further definitions before giving our construction mapping an LPTL formula into a Büchi alternating automaton accepting its models.

Definition 5 *The set* $Subf(f)$ *of subformulas of an LPTL formula* f *is defined recursively as follows:*

- *if* f *is an atomic proposition then* $Subf(f) = \{f\}$,
- $Subf(\theta f) = \{\theta f\} \cup Subf(f)$, $\theta \in \{\neg, \bigcirc, \Diamond\}$,
- $Subf(f \theta g) = \{f \theta g\} \cup Subf(f) \cup Subf(g)$, $\theta \in \{\wedge, U\}$.

Definition 6 *An LPTL formula is said to be elementary if it has one of the following forms:*

- f *is a literal, or*
- f *is prefixed by the temporal operator* $\bigcirc$, *i.e.* $\bigcirc$ *is the main operator of* f *(* f *of the form* $\bigcirc g$*).*

Definition 7 *An eventuality is a formula of the form* $\Diamond f$ *or* $\neg(f U g)$.

[1] If E is a finite or enumerable set then the free distributive lattice. $\mathcal{L}(E)$, generated by E is defined in the following manner: $E \subset \mathcal{L}(E)$, and for all ϵ_1 and ϵ_2 in $\mathcal{L}(E)$, both of $\epsilon_1 \wedge \epsilon_2$ and $\epsilon_1 \vee \epsilon_2$ are in $\mathcal{L}(E)$.

The following two definitions are the key points in the construction of a Büchi alternating automaton accepting the models of an LPTL formula. The first concerns the decomposition of a formula into elementary ones, and will be used for finding the transition function of the alternating automaton to associate to a formula. While the second definition, it consists of the closure of a formula, and will be used for determining the state set of the alternating automaton. These definitions are based on the following equivalences which are straightforward from the definition of satisfiability:

(equiv 1)	$\Diamond f$	$\equiv$	$f \vee \bigcirc\Diamond f$
(equiv 2)	$f_1 U f_2$	$\equiv$	$f_2 \vee f_1 \wedge \bigcirc(f_1 U f_2)$
(equiv 3)	$\neg\Diamond f$	$\equiv$	$\neg f \wedge \bigcirc\neg\Diamond f$
(equiv 4)	$\neg(f_1 U f_2)$	$\equiv$	$\neg f_2 \wedge (\neg f_1 \vee \bigcirc\neg(f_1 U f_2))$
(equiv 5)	$\neg \bigcirc f$	$\equiv$	$\bigcirc\neg f$

Definition 8 *The decomposition,* $\mathrm{elem}(f)$, *of an LPTL formula* f *into elementary formulas is defined recursively as follows:*

- *if* f *is a literal:* $\mathrm{elem}(f) = f$,
- $\mathrm{elem}(\neg\neg f_1) = \mathrm{elem}(f_1)$,
- $\mathrm{elem}(f_1 \wedge f_2) = \mathrm{elem}(f_1) \wedge \mathrm{elem}(f_2)$,
- $\mathrm{elem}(\neg(f_1 \wedge f_2)) = \mathrm{elem}(\neg f_1) \vee \mathrm{elem}(\neg f_2)$,
- $\mathrm{elem}(\Diamond f_1) = \mathrm{elem}(f_1) \vee \bigcirc\Diamond f_1$.
- $\mathrm{elem}(f_1 U f_2) = \mathrm{elem}(f_2) \vee \mathrm{elem}(f_1) \wedge \bigcirc(f_1 U f_2)$,
- $\mathrm{elem}(\bigcirc f_1) = \bigcirc f_1$,
- $\mathrm{elem}(\neg\Diamond f_1) = \mathrm{elem}(\neg f_1) \wedge \bigcirc\neg\Diamond f_1$,
- $\mathrm{elem}(\neg(f_1 U f_2)) = \mathrm{elem}(\neg f_2) \wedge (\mathrm{elem}(\neg f_1) \vee \bigcirc\neg(f_1 U f_2))$,
- $\mathrm{elem}(\neg \bigcirc f_1) = \bigcirc\neg f_1$.

Definition 9 *The closure,* $\mathrm{cl}(f)$, *of an LPTL formula* f *is defined recursively as follows:*

- *if* f *is a literal:* $\mathrm{cl}(f) = \emptyset$,
- $\mathrm{cl}(\neg\neg f_1) = \mathrm{cl}(f_1)$,
- $\mathrm{cl}(f_1 \wedge f_2) = \mathrm{cl}(f_1) \cup \mathrm{cl}(f_2)$,
- $\mathrm{cl}(\neg(f_1 \wedge f_2)) = \mathrm{cl}(\neg f_1) \cup \mathrm{cl}(\neg f_2)$,
- $\mathrm{cl}(\Diamond f_1) = \mathrm{cl}(f_1) \cup \{\Diamond f_1\}$,
- $\mathrm{cl}(f_1 U f_2) = \mathrm{cl}(f_2) \cup \mathrm{cl}(f_1) \cup \{f_1 U f_2\}$,
- $\mathrm{cl}(\bigcirc f_1) = \mathrm{cl}(f_1) \cup \{\bigcirc f_1\}$,
- $\mathrm{cl}(\neg\Diamond f_1) = \mathrm{cl}(\neg f_1) \cup \{\neg\Diamond f_1\}$,
- $\mathrm{cl}(\neg(f_1 U f_2)) = \mathrm{cl}(\neg f_2) \cup \mathrm{cl}(\neg f_1) \cup \{\neg(f_1 U f_2)\}$,
- $\mathrm{cl}(\neg \bigcirc f_1) = \mathrm{cl}(\neg f_1) \cup \{\bigcirc\neg f_1\}$.

4 Büchi alternating automaton of an LPTL formula

The following theorem gives an effective construction of a Büchi alternating automaton accepting the models of an LPTL formula.

Theorem 1 *Let* f *be an LPTL formula. The set of models of* f *is the language accepted by the Büchi alternating automaton* $B_f =< \mathcal{L}(\Sigma \cup Q), \delta, q_0, q_v, F >$, *called Büchi alternating automaton of* f, *defined in the following manner:*

- $\Sigma = \mathcal{P} \cup \neg\mathcal{P}$, *$\mathcal{P}$ being the set of atomic propositions occurring in f,*
- $Q = \{< f >\} \cup \{< g >:\ g \in \mathrm{cl}(f)\}$,
- $\delta(< g >)$ *is the result obtained by substituting, for each non-nested occurrence in* elem(g) *having the form* $\bigcirc h$, *of* $< h >$ *to* $\bigcirc h$,
- $q_0 =< f >$, *and*
- *the set F of distinguished states is:*

$$F = \{< g >\in Q :\ g \text{ is not an eventuality formula}\} \cup \{q_v\}.$$

Proof The only point we clarify is the choice of the set F of distinguished states. For the initial formula to be satisfied, there must exist an interpretation satisfying it, that is a model of it, on which one can construct a run of the automaton verifying the following: every time we meet on a branch a node labeled by a state $< g >$ with g being an eventuality formula, we can find, while continuing the traversing of that branch, a state whose label $< h >$ is such that h is not an eventuality formula. Clearly, the run must repeat infinitely often, on each of its branches, states $< g >$ such that g is not an eventuality. Hence the result.

□

5 Büchi automata: simulating an alternating by a usual nondeterministic

We give in this section the main steps of a method simulating, in the case of Büchi, an alternating automaton by a usual nondeterministic one. Some further definitions are needed.

A run of an alternating automaton is said to be uniform if for all nodes v_1 and v_2 of a same level and a same label, and for all state q of the automaton we have the following: v_1 has an immediate successor labeled by q if and only if v_2 has an immediate successor labeled by q. It is a simple matter of showing that for all interpretation accepted by an alternating automaton, one can construct a uniform run of the automaton on the interpretaion. This clearly implies that for alternating automata one can restrict oneself to uniform runs. We now define a run DAG (Directed Acyclic Graph) to be the quotient of a uniform run by the equivalence relation defined on the set of nodes of that uniform run in the following manner: two nodes are equivalent if and only if they are of a same level and of a same label. The set of distinguished levels of a run DAG is defined as follows. The lowest distinguished level, n_0, is the least level such that all path joining the root (level 0) to a node of level n_0 meets at least once an element of F. The $(i+1)^{st}$ $(i \geq 0)$ distinguished level, n_{i+1}, consists of the least level greater than n_i (the i^{th} distinguished level) such that all path joining a node of level $n_i + 1$ to a node of level n_{i+1} meets at least once an element of F. To characterize the distinguished levels of a run DAG G, we define a mapping u_G associating to all node v of G an element from the cross product $Q \times \{0,1\}$ in the following manner. A node of label q will be such that $u_G(v) = (q,0)$ or $u_G(v) = (q,1)$. Let r_G be the root of G. Then:

$$u_G(r_G) = \begin{cases} (q_0,1) \text{ if } q_0 \in F, \\ (q_0,0) \text{ otherwise.} \end{cases}$$

For a node v of level $n+1$, $n \geq 0$, $u_G(v)$ is defined in the case-by-case following manner:

- if all node v' of level n verifies $u_G(v') \in Q \times \{1\}$, then:

$$u_G(v) = \begin{cases} (q,1) \text{ if } q \in F, \\ (q,0) \text{ otherwise.} \end{cases}$$

- otherwise:

$$u_G(v) = \begin{cases} (q,1) \text{ if } q \in F, \text{ or all immediate predecessor} \\ \qquad v' \text{ of } v \text{ verifies } u_G(v') \in Q \times \{1\}, \\ (q,0) \text{ otherwise.} \end{cases}$$

Let us associate to G a second mapping E_G defined in the following manner[2]:

$$E_G : \begin{cases} \mathbb{N} \longrightarrow 2^{Q \times \{0,1\}} \\ E_G(n) = \{(q,i) \in Q \times \{0,1\} \mid \exists \text{ a node } v \text{ of } G \text{ of} \\ \qquad \text{level } n \text{ verifying } u_G(v) = (q,i)\}. \end{cases}$$

It is easily seen that the distinguished levels of G are the levels n verifying the following:

for all node v of level n, $u_G(v) \in Q \times \{1\}$.

Stated otherwise, a level n $(n \geq 0)$ is a distinguished level if and only if $E_G(n) \in 2^{Q \times \{1\}}$. It is now easy to guess from what precedes the construction of a usual Büchi nondeterministic automaton simulating a Büchi alternating one $M =< \mathcal{L}(\Sigma \cup Q), \delta, q_0, q_v, F >$. Let us consider the set $\mathcal{G}$ of all run DAGs of M, and the equivalence relation R_M on the cross product $\mathbb{N} \times \mathcal{G}$ defined in the following manner:

$$((n_1, G_1), (n_2, G_2)) \in R_M$$

if and only if

$$E_{G_1}(n_1) = E_{G_2}(n_2).$$

The equivalence class, $R_M(n, G)$, of an element (n, G) of the cross product $\mathbb{N} \times \mathcal{G}$, is represented by the part of $Q \times \{0,1\}$ consisting of the set $E_G(n)$. The Büchi nondeterministic automaton we use to simulate M is given by the quotient of $\mathbb{N} \times \mathcal{G}$ by the equivalence relation R_M. The distinguished states consist of the set $2^{Q \times \{1\}}$. The initial state is $\{(q_0,1)\}$ or $\{(q_0,0)\}$, depending on whether q_0 belongs or not to F.

5.1 Size of the simulating automaton

Let $M =< \mathcal{L}(\Sigma \cup Q), \delta, q_0, q_v, F >$ be a Büchi alternating automaton, and n the cardinal of Q. Let also $\mathcal{Q}$ be the state set of the Büchi nondeterministic automaton simulating M, constructed by the method described above. Recall that $\mathcal{Q} \subseteq 2^{Q \times \{0,1\}}$. The set $\mathcal{Q}$ verifies what follows:

[2] $\mathbb{N}$ stands for the set of positive integers.

for all $q \in Q$, and for all $Q_1 \in \mathcal{Q}$,

$not[(q,0) \in Q_1 \text{ and } (q,1) \in Q_1]$.

In other words, $(q,0)$ and $(q,1)$ cannot be simultaneously in Q_1. It follows that the cardinal of the state set $\mathcal{Q}$ of B is bounded by the number of mappings from Q into $\{0,1,2\}$. This is so because each element Q_1 of $\mathcal{Q}$ can be characterized by the mapping $f : Q \longrightarrow \{0,1,2\}$ defined as follows:

- $f(q) = 0 \Leftrightarrow (q,0) \in Q_1$ (and $(q,1) \notin Q_1$),
- $f(q) = 1 \Leftrightarrow (q,1) \in Q_1$ (and $(q,0) \notin Q_1$), and
- $f(q) = 2 \Leftrightarrow (q,0) \notin Q_1$ and $(q,1) \notin Q_1$.

The number of such mappings is 3^n.
The following two points imply that this upper bound is better than 3^n:

for all $q \in F$, and for all $Q_1 \in \mathcal{Q}$: $(q,0) \notin Q_1$.

In summary, given a Büchi alternating automaton M of size n, whose set of distinguished states is of size d, the size of the Büchi nondeterministic automaton simulating M constructed by method described above is bounded by $2^d 3^{n-d}$, i.e. $(\frac{2}{3})^d 3^n$.

6 Comparison with the method of Wolper

Wolper ([Wol 88]) proposed a method for constructing a Büchi nondeterministic automaton accepting the models of an LPTL formula. We do not give the complete description of this method. We restrict ourselves to its main lines which consist essentially of performing the cross product of the following two automata:

- a first automaton called "local automaton", which is nothing else than the satisfiability graph of the formula ([Man et al. 84, Wol 83, Wol 85, Wol 88]), and
- a second automaton called "eventuality automaton" ([Wol 88]).

The result given by the cross product is a Büchi nondeterministic automaton accepting the models of the input formula.

We now give a comparison of this method and ours.

In the preceding section, we gave a upper bound of the size of the Büchi nondeterministic automaton simulating a Büchi alternating one, constructed by the method described in this same section. This upper bound is $(\frac{2}{3})^d 3^n$, n and d being, respectively, the size and the number of distinguished states of the alternating automaton.

The size of what we called the Büchi alternating automaton of an LPTL formula f is bounded by $2n$, n being the length of f. If e is the number of eventualities contained in the state set of the alternating automaton of f, then the size of the set of distinguished states of this same alternating automaton is bounded by $(2n-e)$.

It follows from what precedes that the size of the Büchi nondeterministic automaton accepting the models of an LPTL formula of size n, automaton constructed by our method which uses alternating automata, is bounded by $(\frac{2}{3})^{2n-e} 3^{2n} = (\frac{3}{2})^e 4^n$.

In the other hand, the size of the Büchi nondeterministic automaton of an LPTL formula constructed by the method of Wolper is the product of the size of the local automaton and the size of the eventuality automaton. This is clearly bounded by $2^{3n} = 8^n$.

Of course, the bound $(\frac{3}{2})^e 4^n$ is better than 8^n. This result is confirmed by an experimental comparison which is done in the following manner.

We wrote in C on Unix System a verificator system for LPTL formulas. This system performs the following two tasks. It constructs a Büchi nondeterministic automaton accepting the models of an input LPTL formula for each of the following methods:

- our method using alternating automata, which first constructs a Büchi alternating automaton and then simulates this alternating automaton by a nondeterministic one,
- Wolper's method ([Wol 88]), which first constructs the local automaton and the eventuality automaton before performing their cross product to finally obtain a nondeterministic automaton accepting the models of the input formula.

The answer of our system to LPTL formulas randomly generated is presented by the table of Figure 3. The table contains four columns. Column i gives the number of the randomly generated formula, Column t_1 gives its size (the cardinal of the closure), and column t_2 (respectively t_3) gives the size (the number of states) of the corresponding Büchi nondeterministic automaton constructed by the method of alternating automata (respectively by Wolper's method).

Conclusion

Our investigation in this paper was a construction of a Büchi alternating automaton accepting the models of an LPTL formula. We also described the main steps of a method simulating, in the case of Büchi, an alternating automaton by a usual nondeterministic one. Combining these two results gives a new approach for the construction of a Büchi nondeterministic automaton accepting the models of an LPTL formula. This new approach has been advantageously compared to the one of Wolper ([Wol 88]).

Our current concern can be summarized by the following two points:

- the emptiness problem of Muller alternating automata, and
- the minimal model property for LPTL: use the decreasing property of very weak alternating automata[3] (see [Isl 93] for the class of very weak alternating automata, and [Mul et al. 88, Mul et al. 92] for the class of weak alternating automata which has at least the same expressive power) to see whether we can improve the minimal model property for LPTL.

[3] The Büchi alternating automaton associated to an LPTL formula is very weak.

i	t_1	t_2	t_3	i	t_1	t_2	t_3
0	8	25	46	37	10	9	17
1	5	4	12	38	3	5	9
2	5	4	4	39	12	11	11
3	5	6	21	40	5	8	15
4	3	2	2	41	5	2	4
5	5	4	7	42	4	7	37
6	5	3	6	43	6	9	15
7	11	7	7	44	8	7	6
8	11	44	176	45	3	3	3
9	9	25	49	46	5	7	16
10	11	11	21	47	3	3	3
11	8	2	2	48	1	2	4
12	1	2	2	49	5	4	4
13	3	5	9	50	6	6	11
14	3	3	3	51	4	3	3
15	6	4	7	52	3	3	6
16	5	7	7	53	2	3	3
17	7	5	5	54	4	4	7
18	3	4	10	55	4	6	11
19	3	6	6	56	2	4	8
20	9	11	21	57	8	13	25
21	5	3	3	58	2	3	9
22	9	31	61	59	6	8	8
23	2	3	3	60	5	7	12
24	6	7	13	61	4	3	3
25	5	4	4	62	11	11	21
26	1	1	1	63	3	4	7
27	9	3	3	64	3	4	7
28	5	4	4	65	2	3	9
29	11	2	2	66	10	9	17
30	4	5	5	67	5	18	37
31	11	10	28	68	6	7	7
32	1	2	2	69	5	4	7
33	6	7	7	70	3	4	4
34	4	3	3	71	1	2	2
35	7	6	24	72	9	9	17
36	3	5	9	73	5	7	14

Figure 3: experimental comparison: i number of the randomly generated formula, t_1 its size and t_2 (respectively t_3) size of the Büchi nondeterministic automaton obtained by the method of alternating automata (respectively Wolper's method).

Acknowledgement

I am indebted to Professor Ahmed SAOUDI who was my PhD thesis advisor, and has been a major contributor to this work. Professor Ahmed SAOUDI passed away on August 11, 1993.

References

[Isl 93] **Isli, A.**, *Automates Alternants et Logiques Temporelles, Satisfaction de Contraintes Temporelles*, PhD thesis, L.I.P.N., Université Paris 13, France, 1993.

[Man et al. 84] **Manna, Z.** and **Wolper, P.**, *Synthesis of communicating processes from temporal logic specifications*, ACM Transactions on programming languages and systems 6 (1984) 68 – 93.

[Mul et al. 88] **Muller, D.E., Saoudi, A.** and **Schupp, P.E.**, *Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time*, in: proceedings 3rd IEEE Ann. Symp. on Logic in Computer Science (1988) 422 – 427.

[Mul et al. 92] **Muller, D.E., Saoudi, A.** and **Schupp, P.E.**, *Alternating Automata, The Weak Monadic Theory of Trees and its Complexity*, Theoretical Computer Science 97 (1992) 233 – 244.

[Wol 83] **Wolper, P.**, *Temporal logic can be more expressive*, Information and control 56 (1983) 72 – 99.

[Wol 85] **Wolper, P.**, *The Tableau Method for Temporal Logic: An Overview*, Logique et Analyse (1985) 119 – 136.

[Wol 88] **Wolper, P.**, *On the Relation of Programs and Computations to Models of Temporal Logic*, 1 – 11.

Combining Temporal and Hierarchical Constraints for Temporal Reasoning

Fei Song
Dept. of Computing and Information Science
University of Guelph
Guelph, Ontario, Canada N1G 2W1
fsong@uoguelph.ca

Abstract

Existing reasoning algorithms for Allen's interval algebra may produce weak results when applied to temporal networks that involve decompositions of intervals. We present a strengthened procedure for reasoning about such hierarchical constraints, which works interactively with an existing algorithm for temporal reasoning, to produce the desired stronger results. We further apply our algorithm to the process of plan recognition and show that such an application can both reduce the number of candidate plans and make the constraints in the remaining plans more specific.

1. Introduction

Allen's (1983a) interval algebra has shown to be useful for such applications as knowledge-based systems, natural language processing, and planning (as described in van Beek, 1990). For example, a simplified plan for making a pasta dish can be represented as the temporal network in figure 1, where a node corresponds to the time interval over which a state holds or an event occurs, and a link label represents the temporal constraint between two intervals[1].

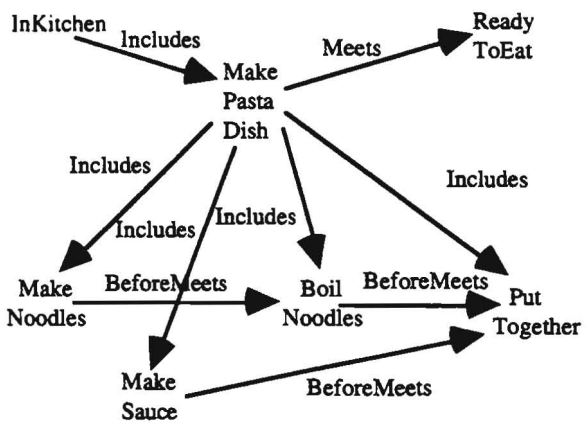


Figure 1: Temporal network of a cooking plan

Given a temporal network, an important reasoning task is to compute the so-called minimal labeling, that is, to find the set of minimal constraints if the network is consistent (Vilain and Kautz, 1986). A constraint (or a link label) is minimal if each of its basic relations is part of a consistent singleton labeling, for which each link is labeled by a basic relation and all the links in the network are satisfied. Vilain and Kautz (1986) show that the time complexity of such a reasoning task is NP-complete for the interval algebra, where the problem size is the number of intervals. However, this does not prevent people from proposing polynomial algorithms that are approximate for the interval algebra (Allen 1983; Van Beek 1989). Allen's algorithm has $O(n^3)$ time complexity and is shown to be exact only for a subset of the interval algebra (Van Beek 1989). Van Beek then proposes an $O(n^4)$ algorithm which is exact for a larger subset (the subset of the interval algebra that can be translated into the point algebra in (Vilain and Kautz, 1986)). To get more exact results for the full interval algebra, one may have to use some exponential-time algorithms (e.g., Valdés-Pérez 1987).

One problem with these existing algorithms is that they may produce weak results when applied to temporal networks that involve hierarchical constraints (i.e. the decompositions of intervals into low-level subintervals). In Song and Cohen (1991), we proposed a strengthened algorithm for temporal reasoning about hierarchical constraints. The algorithm guarantees that the result is no weaker than that obtained from the existing temporal reasoning algorithms. However, whether it can derive the minimal labeling for the hierarchical constraints depends on the order in which lower-level constraints are combined. In this paper, we present a new order-independent reasoning procedure for hierarchical constraints, along with formal proofs for its associated properties. We further apply our algorithm to the problem of plan recognition, and show that the observed temporal constraints can both reduce the number of candidate plans and make the constraints in the remaining candidate plans more specific.

2. Weak results from the existing algorithms

A hierarchical constraint corresponds to the decomposition of an interval to a set of subintervals. In terms of Allen's interval algebra, this means that the interval temporally includes all the subintervals. Suppose that A is an abstract interval of two subintervals a1 and a2 and initially there is no specific constraint between a1 and a2, as shown in figure 2(a). Then, all we can decide is that A Includes a1 and A Includes a2, where Includes stands for the constraint {si, di, fi, eq}.

[1] Here, "Includes" and "BeforeMeets" are high-level constraints, defined as the sets {si, di, fi, eq} and {b, m}, where b, m, eq are the basic relations "before", "meets", "equal", and si, di, fi are the inverses of "starts", "during", and "finishes" in Allen's interval algebra.

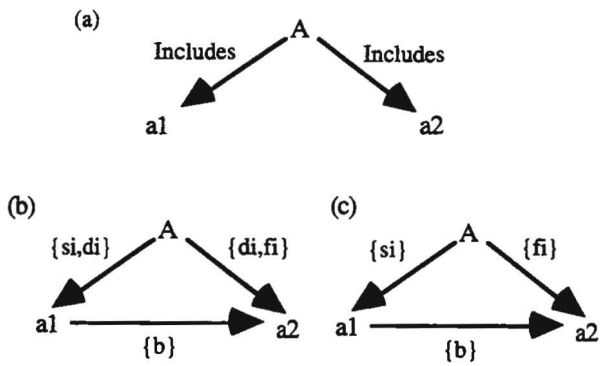


Figure 2: (a) One decomposition, (b) Weak results from Allen's algorithm, and (c) Strong results desired

Now, if we add a new constraint {b} between a1 and a2, then we can use Allen's algorithm to propagate the constraint and produce the results shown in figure 2(b). However, since a1 and a2 are the only subintervals of A and we know that a1 is located before a2, we should be able to decide that a1 is the starting part of A and a2 is the finishing part. In other words, we should get the desired results shown in figure 2(c).

Such weak results can be carried further for networks that consist of more than one decomposition. Suppose that initially we have the network shown in figure 3(a). Later, if we add the constraints a1 {b} a2 and a2 {b} a3, we get the results shown in figure 3(b) using Allen's algorithm, where "Common" stands for the constraint: {o, oi, s, si, d, di, f, fi, eq}. However, using a similar argument as made for the previous example, we should be able to get the stronger results shown in figure 3(c).

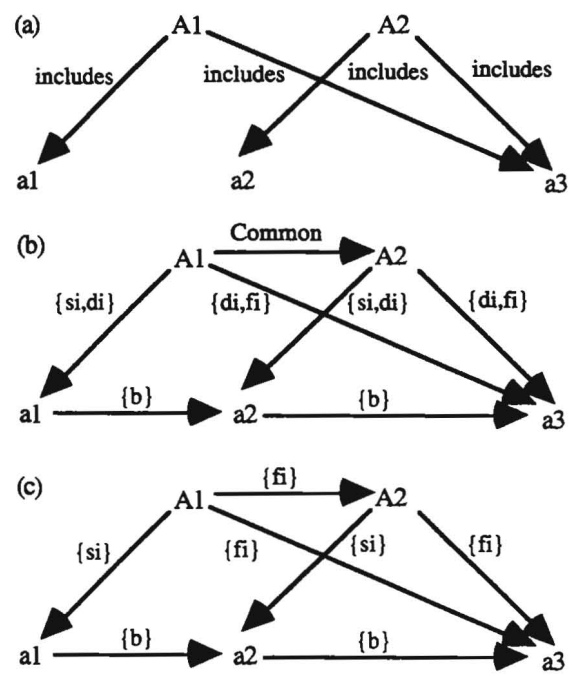


Figure 3: (a) Two decompositions, (b) Weak results from Allen's , and (c) Strong results desired

There are also networks that are considered to be consistent by Allen's algorithm but in fact are not when decompositions are involved. For example, the network in figure 4(a) is regarded as consistent by Allen's algorithm, since we get the same network after applying the algorithm. However, this is actually not true because if a1 and a2 are the only subintervals of A and a1 is located before a2, a2 should be the finishing part of A, not an interior part, as shown in figure 4(b).

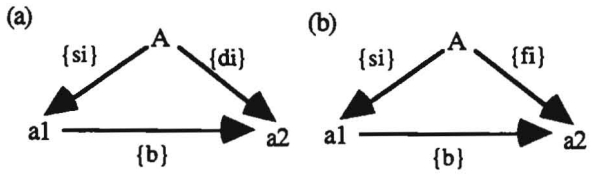


Figure 4: (a) Weak results from Allen's, and (b) Strong results desired

Such weak results are not simply caused by the inexactness of Allen's algorithm. In fact, Allen's algorithm is exact for all these examples since the constraints used fall into a subset of the interval algebra for which Allen's algorithm is guaranteed to find the set of minimal labels (Van Beek 1989). The reason for these weak results is that Allen's algorithm treats all the intervals as independent of each other. This is certainly not true for decompositions, since the abstract intervals are temporally dependent on their subintervals. To make these dependencies explicit in the reasoning process, we need to assume that the decomposition of an abstract interval into its subintervals is complete, that is, no more subintervals can be added to the decomposition. As a result, we can compute how an abstract interval is temporally bounded by its subintervals based on the constraints between all the subintervals. For instance, if there is a linear ordering between all the subintervals, then we can clearly decide that the abstract interval is temporally bounded by the subintervals that occur the earliest and the latest. We say that a decomposition is closed if the constraints between the abstract interval and its subintervals are minimal with respect to the constraints between all the subintervals.

More formally, we describe an abstract interval as the convex hull or the minimal cover of its subintervals, denoted by the equation:

$$A = x_1 + x_2 + \ldots + x_n$$

where A denotes the abstract interval and $x_1, x_2, \ldots, x_n$ denote the subintervals. For the example in figure 3, the two decompositions can be represented as: $A_1 = a_1 + a_3$ and $A_2 = a_2 + a_3$. Closing a decomposition means closing every decomposition edge between the abstract interval and its subintervals, which further implies computing the minimal labels on the decomposition edges. We formally define a closure operation by the following first-order formula:

$$A\,(C_{i1}, C_{i2}, \ldots, C_{im})^c\, x_i \Leftrightarrow$$
$$x_i C_{i1} x_1 \wedge x_i C_{i2} x_2 \wedge \ldots \wedge x_i C_{im} x_m \wedge$$
$$(A = x_1 + x_2 + \ldots + x_m)$$

where $C_{i1}, C_{i2}, \ldots, C_{im}$ are the constraints between the

subinterval x_i and all the subintervals from x_1 to x_m. This formula suggests that a decomposition edge can be closed by using the existing constraints between a subinterval and all the subintervals and the fact that the abstract interval is the minimal cover of all its subintervals.

3. Development of a strengthened temporal reasoning algorithm

To describe our strengthened algorithm for temporal reasoning with hierarchical constraints, we start with the simple case of closing one and two subintervals, and then, we generalize the result to close more than two subintervals. After that, we provide a recursive procedure to close more than one decomposition, and finally, we present the strengthened algorithm that closes all the hierarchical structures in a temporal network.

3.1. Closing one and two subintervals

As described earlier, closing a decomposition means computing the minimal labels on all the decomposition edges. If an abstract interval has only one subinterval, then the minimal label on the decomposition edge is obviously {eq}, that is:

$$A\ (C_{ii})^c x_i \Leftrightarrow x_i C_{ii} x_i \wedge (A = x_i) \Leftrightarrow$$
$$x_i\{eq\}x_i \wedge (A = x_i) \Leftrightarrow A\ \{eq\}\ x_i$$

In the case of two subintervals, we can derive from the definition:

$$A\ (C_{ii}, C_{ij})^c x_i \Leftrightarrow x_i C_{ii} x_i \wedge x_i C_{ij} x_j \wedge (A = x_i + x_j) \Leftrightarrow$$
$$x_i\{eq\}x_i \wedge x_i C_{ij} x_j \wedge (A = x_i + x_j)$$

Since $x_i\{eq\}x_i$ always holds for an interval (the so-called node consistency), we define the basic closure operation on a constraint as: $A\ C_{ij}^c\ x_i \Leftrightarrow x_i C_{ij} x_j \wedge (A = x_i + x_j)$.

Lemma 1: Given C_{ij} as a set of basic relations $R_1, R_2, \ldots, R_m$, the basic closure C_{ij}^c can be computed as $\{R_1^c, R_2^c, \ldots, R_m^c\}$, where R^c is one of the four basic relations: si, di, fi, and eq, as defined in table 1.

Table 1: basic closure on basic relations

R	b	bi	m	mi	o	oi	s	si	d	di	f	fi	eq
R^c	si	fi	si	fi	si	fi	si	eq	di	eq	fi	eq	eq

The validity of table 1 can be easily verified. For example, if x_i {b} x_j, then the closed edge between A and x_i is {si}, since if A consists of only x_i and x_j and x_i is located before x_j, then x_i must be the starting part of A. This basic closure operation also applies to the case of one subinterval, i.e., $(C_{ii})^c \Leftrightarrow C_{ii}^c$, since $A\ C_{ii}^c\ x_i \Leftrightarrow A\ \{eq\}^c\ x_i \Leftrightarrow A\ \{eq\}\ x_i$.

3.2. Closing more than two subintervals

Having defined the basic closure operation, we can now extend it to close a decomposition of more than two subintervals. More specifically, if $C_{i1}, C_{i2}, \ldots, C_{im}$ are the constraints between the subinterval x_i and all the subintervals from x_1 to x_m, including the subinterval x_i, then we can close x_i and another subinterval x_j using the basic closure operation: C_{ij}^c. To get the final closed decomposition edge to x_i, however, we need to somehow combine all of the C_{ij}^c's. It turns out that these C_{ij}^c's can be combined with the normal composition operation in Allen's interval algebra.

Lemma 2: Given the basic relations $R_{i1}, R_{i2}, \ldots, R_{in}$, we have

$$A\ (R_{i1}, R_{i2}, \ldots, R_{in})^c\ x_i \Leftrightarrow A\ (R_{i1}{}^c \times R_{i2}^c \times \ldots \times R_{in}{}^c)\ x_i.$$

Proof. We prove this lemma by induction on the number of subintervals. For n = 1, we showed in the last subsection that $A\ (R_{ii})^c\ x_i \Leftrightarrow A\ R_{ii}^c\ x_i$. For n = 2, we have:

$$A\ (R_{ii}, R_{ij})^c\ x_i \Leftrightarrow A\ R_{ij}{}^c\ x_i$$
$$A\ (R_{ii}^c \times R_{ij}^c)\ x_i \Leftrightarrow A\ (\{eq\}^c \times R_{ij}^c)\ x_i \Leftrightarrow A\ R_{ij}{}^c\ x_i$$

So, the lemma holds for both n = 1 and n = 2.

Assume the lemma holds for n = k, that is, $A'\ (R_{i1}, R_{i2}, \ldots, R_{ik})^c\ x_i \Leftrightarrow A'\ (R_{i1}^c \times R_{i2}^c \times \ldots \times R_{ik}^c)\ x_i$, where $A' = x_1 + x_2 + \ldots + x_k$, we need to prove that the lemma also holds for n = k+1.

We know from table 1 that R_{ij}^c can only be one of the four basic relations: si, di, fi, and eq. By checking table 2, a sub-multiplication table drawn from Allen's (1983a), we see that these four basic relations are closed under multiplication.

Table 2: A Sub-Multiplication Table

×	si	di	fi	eq
si	si	di	di	si
di	di	di	di	di
fi	di	di	fi	fi
eq	si	di	fi	eq

It follows that $R_{i1}^c \times R_{i2}^c \times \ldots \times R_{ik}^c$ can only be one of the four basic relations: si, di, fi, and eq. Let us denote $R_{i1}^c \times R_{i2}^c \times \ldots \times R_{ik}{}^c$ as R, and R_{ik+1}^c as S.

Now, from the definition of the closure operation, we have:

$$A\ (R_{i1}, \ldots, R_{ik}, R_{ik+1})^c\ x_i \Leftrightarrow x_i R_{i1} x_1 \wedge \ldots \wedge x_i R_{ik} x_k \wedge$$
$$x_i R_{ik+1} x_{k+1} \wedge (A = x_1 + \ldots + x_k + x_{k+1})$$

$$\Leftrightarrow x_i R_{i1} x_1 \wedge \ldots \wedge x_i R_{ik} x_k \wedge (A' = x_1 + \ldots + x_k) \wedge$$
$$x_i R_{ik+1} x_{k+1} \wedge (A'' = x_i + x_{k+1}) \wedge (A = A' + A'')$$

$$\Leftrightarrow A'\ (R_{i1}, \ldots, R_{ik})^c\ x_i \wedge A''\ R_{ik+1}^c\ \ x_i \wedge (A = A' + A'')$$

$$\Leftrightarrow A'\ (R_{i1}^c \times \ldots \times R_{ik}{}^c)\ x_i \wedge A''\ R_{ik+1}^c\ \ x_i \wedge$$
$$(A = A' + A'')$$

$$\Leftrightarrow A'\ R\ x_i \wedge A''\ S\ x_i \wedge (A = A' + A'')$$

To further evaluate the above expression, we need to consider the following special cases:

(1) If A' {si} $x_i \wedge$ A" {si} x_i, then we have A {si} x_i, since if x_i is the starting part of both A' and A" and A = A' + A", then x_i should also be the starting part of A.

(2) If A' {fi} $x_i \wedge$ A" {fi} x_i, then we have A {fi} x_i. The reason is similar to case (1) above.

(3) If A' {si} $x_i \wedge$ A" {fi} x_i, then we have A {di} x_i. The reason for this is that if x_i is the starting part of A', then there must be another interval that finishes after x_i. Similarly, if x_i is the finishing part of A", then there must be another interval that starts before x_i. Thus, there are intervals that starts before x_i and finishes after x_i, and x_i must be an interior part of the covering interval A.

(4) If A' {di} $x_i \wedge$ A" S x_i, then we have A {di} x_i, since if x_i is an interior part of A', it is also an interior part of A.

(5) If A' {eq} $x_i \wedge$ A" S x_i, then we have A S x_i. This is obviously true since A' equals x_i.

Since conjunctions are commutative, it is easy to see that these results are exactly the same as table 2 above. In other words, we have proved that:

$$A\ (R_{i1}, \ldots, R_{ik}, R_{ik+1})^c\ x_i \Leftrightarrow A\ (R \times S)\ x_i \Leftrightarrow A\ (R_{i1}^c \times \ldots \times R_{ik}^c \times R_{ik+1}^c)\ x_i.$$

that is, lemma 2 also holds for n = k+1.

Lemma 2 implies that if the constraints between one subinterval and all the subintervals are one of the basic relations, the decomposition edge to the subinterval can be closed by multiplying the basic closures of these constraints.

Theorem 1: Given $C_{i1}, C_{i2}, \ldots, C_{im}$ as the constraints between x_i and all the subintervals from x_1 to x_m, the closed edge between A and x_i can be computed as follows:

$$A\ (C_{i1}, C_{i2}, \ldots, C_{im})^c\ x_i \Leftrightarrow A\ (C_{i1}^c \circ C_{i2}^c \circ \ldots \circ C_{im}^c)\ x_i.$$

Proof. The theorem can be proved by expanding constraints into sets of basic relations, converting the result into disjunctions of conjunctions, and applying lemma 2 to all the conjunctions:

$$A\ (C_{i1}, C_{i2}, \ldots, C_{im})^c\ x_i \Leftrightarrow x_i C_{i1} x_1 \wedge x_i C_{i2} x_2 \wedge \ldots \wedge x_i C_{im} x_m \wedge (A = x_1 + x_2 + \ldots + x_m)$$

$$\Leftrightarrow x_i\ \{R_{11}, R_{12}, \ldots, R_{1n_1}\}\ x_1 \wedge x_i\ \{R_{21}, R_{22}, \ldots, R_{2n_2}\}\ x_2 \wedge \ldots\ldots x_i\ \{R_{m1}, R_{m2}, \ldots, R_{mn_m}\}\ x_m \wedge (A = x_1 + x_2 + \ldots + x_m)$$

$$\Leftrightarrow (x_i R_{11} x_1 \wedge x_i R_{21} x_2 \wedge \ldots \wedge x_i R_{m1} x_m \wedge A = x_1 + x_2 + \ldots + x_m) \vee (x_i R_{11} x_1 \wedge x_i R_{21} x_2 \wedge \ldots \wedge x_i R_{m2} x_m \wedge A = x_1 + x_2 + \ldots + x_m) \vee \ldots\ldots (x_i R_{1n_1} x_1 \wedge x_i R_{2n_2} x_2 \wedge \ldots \wedge x_i R_{mn_m} x_m \wedge A = x_1 + x_2 + \ldots + x_m)$$

$$\Leftrightarrow A\ (R_{11},\ R_{21}, \ldots, R_{m1})^c\ x_i \vee A\ (R_{11},\ R_{21}, \ldots, R_{m2})^c\ x_i \vee \ldots\ldots A\ (R_{1n_1},\ R_{2n_2}, \ldots, R_{mn_m})^c\ x_i$$

$$\Leftrightarrow A\ (R_{11}^c \times R_{21}^c \times \ldots \times R_{m1}^c)\ x_i \vee A\ (R_{11}^c \times R_{21}^c \times \ldots \times R_{m2}^c)\ x_i \vee \ldots\ldots A\ (R_{1n_1}^c \times R_{2n_2}^c \times \ldots \times R_{mn_m}^c)\ x_i$$

$$\Leftrightarrow A\ (C_{i1}^c \circ C_{i2}^c \circ \ldots \circ C_{im}^c)\ x_i.$$

Lemma 3 Given constraints as subsets of {si, di, fi, eq}, the composition is commutative and associative, that is, $C_1 \circ C_2 = C_2 \circ C_1$, and $(C_1 \circ C_2) \circ C_3 = C_2 \circ (C_1 \circ C_3)$.

Proof. Given two subsets of {si, di, fi, eq}, the composition is both commutative and associative since for each pair of the basic relations, the results of multiplications are symmetric, as shown in table 2.

Theorem 2: In closing a decomposition constraint using theorem 1, we get the same result no matter what order we do the compositions.

Proof. This follows directly from lemma 3, since the composition is both commutative and associative for subsets of {si, di, fi, eq}.

Based on theorems 1 and 2, we now present a new procedure for closing a decomposition of any number of subintervals.

```
procedure CLOSE(k, S)
begin
  for each i ∈ S do begin
     t ← {eq}
     for each j ∈ S do
        t ← t ∘ C_ij^c
     t ← t ∩ C_ki
     if t ≠ C_ki then begin
       C_ki ← t
       C_ik ← INVERSE(t)
       Q ← Q ∪ RELATED_PATHS(k, i)
     end
  end
end
```

Figure 5: Procedure for closing a decomposition

The above procedure closes all the decomposition edges in turn, and if a closed edge is more specific than the existing edge, the existing edge will be updated and all the related paths will be queued for further propagation.

Theorem 3. The time complexity of the CLOSE procedure is $O(m^2)$ where m is the number of subintervals in a decomposition.

Proof. Since for each subinterval, we check it with all the subintervals, we need to perform a total of m^2 composition and basic closure operations.

3.3. *Closing all the decompositions in a hierarchical structure*

A hierarchical structure often consists of more than one decomposition. Our strategy is to close a hierarchy in a post-order fashion, since higher-level intervals can be defined in terms of lower-level subintervals. In other words, we start the closing process from the bottom-level decompositions and work our way up until all the decompositions are closed in the hierarchy.

```
procedure CLOSE_ALL (k)
begin
    get a list S of subintervals for k
    if S is not empty then begin
        for each i∈ S do
            CLOSE_ALL (i)
        CLOSE (k, S)
    end
end
```

Figure 6: Closing all the decompositions in a plan

Theorem 4. The time complexity of the CLOSE_ALL procedure is bounded below by O(n) and above by $O(n^2)$, where n is the number of intervals in a plan.

Proof Suppose that there are k decompositions in a plan of n intervals. Then, according to theorem 3, the time complexity for the CLOSE_ALL procedure can be measured as:

$$m_1^2 + m_2^2 + \ldots + m_k^2$$

where m_i is the number of subintervals of the ith decomposition.

For n intervals, there are at most (n-1) decompositions, each of which has one subinterval. In this case, we have: $m_1^2 + m_2^2 + \ldots + m_{n-1}^2 = (n - 1)$.

For n intervals, there is at least one decomposition, which has (n - 1) subintervals. In this case, we have: $m_i^2 = (n - 1)^2$.

Normally, we have the inequalities: $1 \le k \le (n - 1)$ and $m_1 + m_2 + \ldots + m_k = (n - 1)$. As a result, we get the following inequality:

$$m_1^2+m_2^2+\ldots+m_k^2 \le (m_1+m_2+\ldots+m_k)^2=(n-1)^2 ,$$

since $m_i \ge 1$. Similarly, we get another inequality:

$$m_1^2+m_2^2+\ldots+m_k^2 \ge m_1+m_2+\ldots+m_k = (n-1) ,$$

since $m_i^2 \ge m_i$.

Thus, $(n - 1) \le m_1^2 + m_2^2 + \ldots + m_k^2 \le (n - 1)^2$. That is, the time complexity of the CLOSE_ALL procedure is bounded by O(n) and $O(n^2)$.

3.4. *The Strengthened Algorithm*

The CLOSE_ALL procedure closes all the decompositions in a hierarchical structure. To get stronger results for a temporal network, we first use an existing reasoning algorithm to compute the set of constraints to be as specific as possible. Then, for each hierarchical structure in the temporal network, we recursively close all the decompositions using the CLOSE_ALL procedure. After that, some of the decomposition edges may be updated, and we call the temporal reasoning algorithm again to propagate the effects of these new constraints. Thus, we generally need to call interactively an existing reasoning algorithm and our CLOSE_ALL procedure. Such a process will eventually terminate since every time we update a constraint, some of its basic relations will be eliminated and there are at most 13 basic relations in any constraint.

We now give the strengthened algorithm for temporal reasoning with hierarchical constraints:

```
algorithm STRENGTHENED
begin
  Q←{initial paths in a temporal network}
  H←{roots of all hierarchical structures}
  while Q is not empty do begin
      MODIFIED_TR
      foreach k ∈ H do
        CLOSE_ALL (k)
  end
end
```

Figure 7: The strengthened algorithm for temporal reasoning about plans

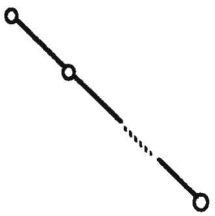
The set H contains the roots of all hierarchical structures, and CLOSE_ALL closes all the decompositions in a hierarchy. The set Q contains those paths whose effects need to be propagated, and MODIFIED_TR is the same as an existing algorithm for temporal reasoning except that the initialization of Q is removed from the procedure.

Theorem 5. The time complexity of our strengthened algorithm is at most $O(T \log_2 n)$, where n is the number of intervals in a temporal network and T is the time complexity of an existing reasoning algorithm (n^3 for the path-consistency procedure, n^4 for Van Beek's procedure, and exponential for some more exact procedures).

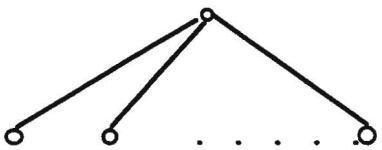
Proof. Our strengthened algorithm interactively calls an existing reasoning procedure and the CLOSE_ALL procedure. For each iteration, the time complexity is measured as $O(T+n^2)$, which can be further reduced to O(T), since T is at least n^3 for most of the existing reasoning algorithms.

To get the factor of $\log_2 n$, we need to identify the worst case where we get the maximum number of iterations for the strengthened algorithm. Such a worst case cannot have more than one plan in the temporal network, since the algorithm only moves to the next iteration when certain decomposition edges are updated, and the effects of closing a decomposition only propagate upwards. Thus, the more the levels of decompositions, the more the number of iterations, and only one plan can provide the maximum levels of decompositions. Assuming only one plan in the temporal network, we can now consider the following different cases.

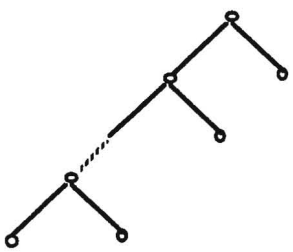
(1) A chain of singular decompositions. Although the maximum levels are achieved in this case, the number of iterations executed by our algorithm is just two: the first pass strengthens all the decomposition edges to {eq}, and the second pass maintains these constraints. Thus, the worst case should deal with decompositions with more than one subinterval.



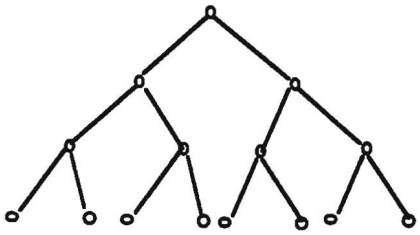
(2) A single decomposition. Again, two iterations are required to get the strengthened results, since there is only one decomposition. These two cases imply that the decompositions in the worst case should have as few subintervals as possible to get the maximum levels, but not just one subinterval. Obviously, only decompositions with two subintervals (we call them binary decompositions) satisfy these conditions.



(3) A chain of binary decompositions. Although this case looks different from the first one, it also requires two iterations of the strengthened algorithm, since all the decompositions are independent of each other: they only rely on the constraints between their subintervals. Thus, the hierarchy in the worst case should be one of a balanced tree so that higher-level decompositions are dependent on lower-level ones, and at the same time, it should have as many levels as possible to get the maximum number of iterations.



(4) A balanced tree of binary decompositions, an example of which with 15 intervals is shown in the figure below. This is the only kind of structure that satisfies all the conditions discussed above for the worst case.



Since this kind of structure generally has $\log_2 n$ levels of decompositions, we conclude that the time complexity for our strengthened algorithm is at most $O(T \log_2 n)$.

4. An application to plan recognition

Plan recognition is the process of inferring an agent's plan based on the observation of the agent's actions. A recognized plan is useful in that it helps to decide an agent's goal and predict the agent's next action. For example, if we observe that John has made the sauce and he is now boiling the noodles, then based on the plan shown in figure 1, we can decide that John's goal is to make a pasta dish and his next action is to put noodles and sauce together. Plan recognition has found applications in such areas as story understanding, psychological modeling, natural language pragmatics, and intelligent interfaces.

Most existing models for plan recognition assume a library of all possible plans that might occur in a particular domain. Then, through some kind of search and matching mechanism, one can find all the plans that contain the observed actions, called candidate plans. Since the observation of an agent's actions is often incomplete and some actions may appear in many different plans of the plan library, it is often difficult to determine the unique plan that an agent is pursuing. Kautz (1987) suggests that one way of reducing the number of candidate plans is to use various kinds of constraints, including the temporal relations explicitly reported in the observations, to further eliminate those inconsistent plans[2]. However, Kautz only adopted a subset of Allen's interval algebra and did not use fully the temporal constraints that correspond to the decomposition edges in a candidate plan.

Our approach to plan recognition is to represent a candidate plan as a temporal network and perform temporal reasoning to eliminate those candidate plans that are inconsistent with the temporal constraints explicitly given in the observations. As illustrated in the following examples, such a reasoning process can produce two useful effects: the given constraints can be used to reduce the number of candidate plans and the given constraints can be made more specific by combining them with the prestored constraints in a candidate plan.

Suppose that our plan library contain two plans for

[2] Other solutions include the use of preference heuristics (Allen, 1983b; Litman, 1985; Carberry, 1986) and probabilities (Goldman and Charniak, 1988).

making GuoTie and JianJiao, two common ways of making fried dumplings in Chinese cooking, as shown in figure 8.

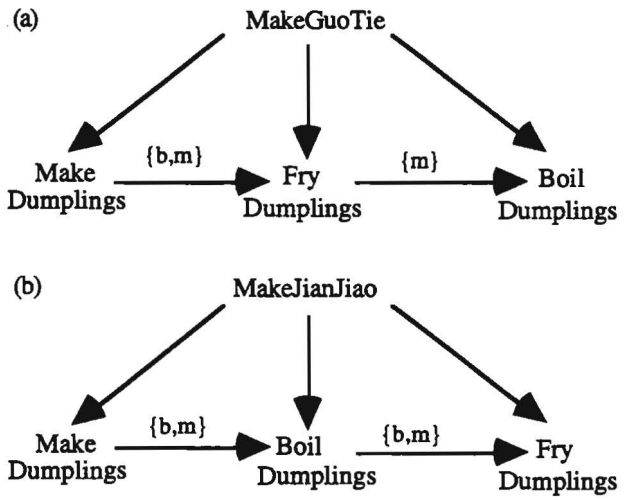
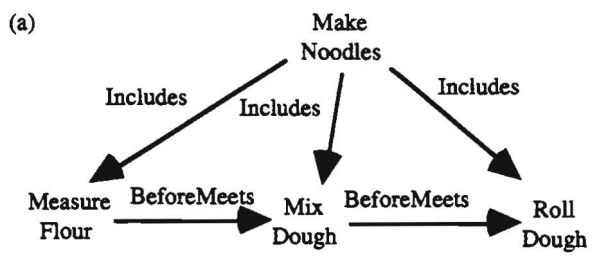


Figure 8: Two simplified plans in a cooking domain

Then, given the observation that BoilDumplings occurs earlier than FryDumplings[3], a plan recognition model that does not use temporal constraints from the input would propose MakeGuoTie and MakeJianJiao as the candidate plans, for both of them contain the two given actions. However, by taking the temporal relations given in the input as a constraint and checking them with those prestored in candidate plans, we find that MakeGuoTie is inconsistent with the given constraint, as BoilDumplings occurs later than FryDumplings in this plan. As a result, our plan recognition model will only propose MakeJianJiao as the plan that the agent is performing.

To illustrate how the results of plan recognition may be improved using our strengthened reasoning algorithm, let us consider an extended version of the plan for making a pasta dish presented in figure 1. As shown in figures 9 (a) and (b), the extended plan contains the further decompositions of MakeNoodles and MakeSauce.



[3] In a natural language setting, such a temporal constraint may be obtained, for example, by linguistically analyzing the input: "I have boiled the dumplings and am now frying them."

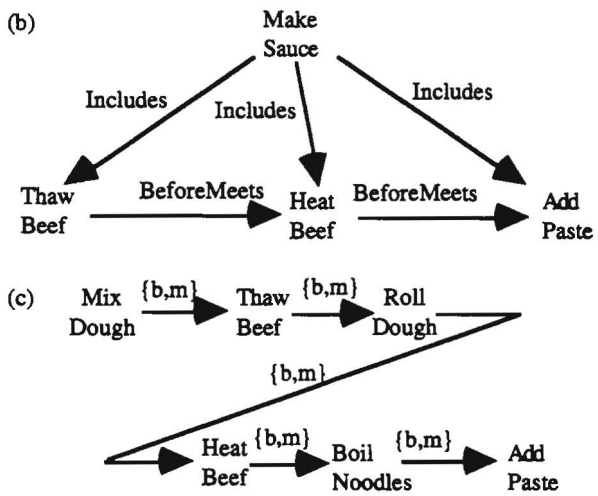


Figure 9: An extended plan for making a pasta dish

Suppose that the observation of an agent's actions is given in figure 9(c). We can then use Allen's algorithm or our strengthened algorithm to make some of the constraints in the plan more specific. Using Allen's algorithm, we can make some of the constraints more specific, as shown in figure 10(a). Using our strengthened algorithm, we can make these constraints even more specific, as shown in figure 10(b).

MakePastaDish {si, di} MakeNoodles
MakePastaDish {si, di} MakeSauce
MakePastaDish {di, fi} PutTogether

MakeNoodles {si, di} MeasureFlour
MakeNoodles {di, fi} RollDough

MakeSauce {si, di} ThawBeef
MakeSauce {di, fi} AddTomatoPaste

MakeNoodles {o, s, d} MakeSauce
MakeNoodles {b, m} BoilNoodles

Figure 10(a): Results from Allen's Algorithms

MakePastaDish {si} MakeNoodles
MakePastaDish {di} MakeSauce
MakePastaDish {fi} PutTogether

MakeNoodles {si} MeasureFlour
MakeNoodles {fi} RollDough

MakeSauce {si} ThawBeef
MakeSauce {fi} AddTomatoPaste

MakeNoodles {o} MakeSauce
MakeNoodles {b} BoilNoodles

Figure 10(b): Results from our strengthened algorithm

5. Conclusion

We presented a strengthened algorithm for temporal reasoning about plans, which improves on straightforward applications of the existing reasoning algorithms for Allen's interval algebra. We view plans as both temporal networks and hierarchical structures. Such a dual view allows us to design a closing procedure which makes as specific as possible the temporal constraints between abstract actions and their subactions. The procedure is then used interactively with an existing reasoning algorithm to help obtain the strengthened results. We applied our algorithm to the problem of plan recognition and showed that such an application can both reduce the number of candidate plans make the constraints in the remaining plans more specific.

One possible area for future work is to improve the efficiency of our algorithm, which calls interactively an existing reasoning algorithm and our closing procedure. Although the strengthened algorithm only adds a factor of $\log_2 n$ to the time complexity of an existing reasoning algorithm, it is worth investigating whether such interactions can be localized and reduced. Some results on localizing the propagation of temporal constraints in Allen's interval algebra have been reported (Koomen 1989). This would form a useful starting point for our future research.

Acknowledgments

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

References

ALLEN, J. F. 1983a. Maintaining knowledge about temporal intervals. Communications of the ACM, **26**: 832-843.

ALLEN, J. F. 1983b. Recognizing intentions from natural language utterances. In *Computational models of discourse.* Edited by M. Brady and R. Berwick. The MIT press, Cambridge, Mass. pp. 107-166.

ALLEN, J. F., and KOOMEN, J. A. 1983. Planning using a temporal world model. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, pp. 741-747.

CARBERRY, S. 1986. Pragmatic modeling in information system interfaces. Ph.D. Dissertation, University of Deleware.

GOLDMAN, R., and CHARNIAK, E. 1988. A probabilistic ATMS for plan recognition. Proceedings of the AAAI Workshop on Plan Recognition.

KAUTZ, H. A. 1987. A formal theory of plan recognition. Ph.D. Dissertation, University of Rochester, Rochester, N.Y.

KOOMEN, J. A. 1989. Localizing temporal constraint propagation. Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Ontario, Canada, pp. 198-202.

LITMAN, D. 1985. Plan recognition and discourse analysis: an integrated approach for understanding dialogues. Ph.D. Dissertation, University of Rochester.

SONG, F. 1991. A processing model for temporal analysis and its application to plan recognition. Ph.D. Dissertation, University of Waterloo, Waterloo, Ontario, Canada.

SONG, F., and COHEN, R. 1991. Temporal reasoning during plan recognition. Proceedings of the Ninth National Conference on Artificial Intelligence, Anaheim, CA, pp. 247-252.

VALDÉS-PÉREZ, R. E. 1987. The satisfiability of temporal constraint networks. Proceedings of the Sixth National Conference on Artificial Intelligence, pp. 256-260.

VAN BEEK, P. 1989. Approximation algorithms for temporal reasoning. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pp. 1291-1296.

VAN BEEK, P. 1990. Exact and approximate reasoning about qualitative temporal relations. Ph.D. Dissertation, University of Waterloo. Available as Technical Report TR90-29, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada.

VILAIN, M., and KAUTZ, H. 1986. Constraint propagation algorithms for temporal reasoning. Proceedings of the Fifth National Conference on Artificial Intelligence, pp. 377-382.

VILAIN, M., KAUTZ, H., and VAN BEEK, P. 1989. Constraint propagation algorithms for temporal reasoning: a revised report. In *Readings in qualitative reasoning about physical systems.* Edited by D.S. Weld and J. de Kleer. Morgan Kaufman, San Mateo, CA, pp. 373-381.

TRL: a formal Language for Temporal References

T.Panayiotopoulos, C.D.Spyropoulos
Institute of Informatics and Telecommunications
N.C.S.R. Demokritos
15310 Aghia Paraskevi,
Athens, Greece
e-mail : themisp@iit.nrcps.ariadne-t.gr e-mail : costass@iit.nrcps.ariadne-t.gr
Tel : +301-6510310, Fax:+301-6532175

Abstract

In this paper we propose an expressively rich formal language for representing temporal references, called TRL. The assumptions about the specifics of time in TRL, and its formal semantics are discussed. Further, the concepts of instant satisfaction, the validity and equivalence in TRL are defined. Many derived properties are proved and illustrative examples are provided which show to some extend the expressive power of TRL. TRL is as expressive as a first-order temporal logic, and is strongly related to reified temporal logics. It introduces and relates the concepts of temporal points, temporal instances and temporal intervals, and has deeply embedded in its semantics the notion of temporal uncertainty.

Keywords : Temporal Logic, temporal semantics, temporal representation

1 Introduction

The problem of representing and reasoning about time in Artificial Intelligence has become an issue of great concern for many researchers during the last few years. The numerous approaches which have been proposed for this reason may be divided into two main categories: numerical [Bar93, Bod93, Ger93, Sti93, Dea88] and symbolic approaches. The symbolic approaches may be further divided in classical logic [Ram88, Kow86], modal logic [Gal87a, Gal87b, Ost89] and reified logic approaches [All84, Rei87, Sho87, Sho88]. Another categorisation concerns the selection of the basic temporal object, which leads to point based approaches [McD82] and interval based approaches [All83, All84, All85, All87, All91, Hay87, Kow86].

In this paper we propose the TRL language as a new approach for the representation of temporal references and introduce its specification, syntax and semantics. We also provide some first results, properties and propositions as well as examples demonstrating its expressive power. TRL may be categorised under the reified approach even if it uses some different syntax for the temporal references.

The paper is structured as follows : In the next section we discuss the motivation behind the development of TRL, as well as the conventions about the specifics of time in TRL. In the following sections we introduce the conceptualisation of the world in TRL, and give some illustrative examples. We then formally introduce the syntax and semantics of TRL. Some preliminary results concerning the semantics are then demonstrated. Definitions concerning instant satisfiability, validity, models, consequence and equivalence, follow, along with some interesting equivalences and properties. Based on these properties, the expressive power of TRL is demonstrated. The paper finishes with the conclusions and some ideas for further work. In the Appendix, there is a collection of selected proofs.

2 The TRL language

2.1 Motivation for the development of the TRL language

Existing temporal logics, either modal or first order, select either a point-based or an interval-based representation scheme. In most systems however, the concept of uncertainty is not considered. An exception is the Tachyon system [Sti93], where uncertainty regards the exact time and duration of events. Fuzzy temporal knowledge is also examined in [Dea88], and [Dub89]. However, the notion of uncertainty in [Dea88] concerns the order of events, and in [Dub89] concerns the introduction of fuzzy temporal measures.

Reified logic approaches tend to keep temporal and non-temporal components separate, and combine the naturalness of expression of modal temporal logics with the efficiency of first order temporal logics [Rei87]. However, reified logics are also based on either point-based algebras [McD82, Sho87, Rei87] or interval-based [All84] and they also do not take into consideration the concept of uncertainty.

It seems that we need a formal logic-based temporal language which should fill in the gap, i.e. provide a common framework for representing both intervals and points with a single notation, consider the notion of temporal

uncertainty, and preserve the important properties of these logics.

Towards this direction, we propose TRL, in which we have made the following assumption about the specifics of time: There are cases in which there is lack of complete information due to the vagueness of natural language or lack of precise information about the exact time of the occurrence of an event. Therefore, we are interested to represent an event that occurred or is going to occur at some moment between two temporal points. Two such temporal points form a temporal instance, i.e. an uncertain temporal point. In order to represent the fact that an event is occurring over a temporal interval, with uncertain start and end points, we must use an uncertain temporal interval, i.e. an interval with temporal instances for its start and end points. In TRL we can represent temporal points, uncertain temporal instances, certain temporal intervals, or even temporal intervals with uncertain start and/or uncertain end. Any such a temporal object is a *temporal reference.*

2.2 Specifications and characteristics of the TRL language

The TRL language is a framework of a temporal logic which uses as the primitive temporal object the uncertain temporal interval which is notated as $<[t_1,t_2],[t_3,t_4]>$. An atom of TRL has the form $<[t_1,t_2],[t_3,t_4]>:\varphi$, where φ is a classical atom of first order logic. The truth value of this atom depends on the truth value of φ during the uncertain temporal interval $<[t_1,t_2],[t_3,t_4]>$. This means that the expression $<[t_1,t_2],[t_3,t_4]>:\varphi$ is true if there is at least one certain temporal interval of the form $<s_1,s_2>$ such that $t_1 \le s_1 \le t_2$, $t_3 \le s_2 \le t_4$, during which φ is true. As it follows from the semantics of TRL, all other temporal references are special cases of an uncertain temporal interval : A *certain temporal instance* t, i.e. a *temporal point*, is the uncertain temporal interval $<[t,t],[t,t]>$. An *uncertain temporal instance* $[t_1,t_2]$, is the uncertain temporal interval $<[t_1,t_2],[t_1,t_2]>$. A *certain temporal interval* $<t_1,t_2>$, is the uncertain temporal interval $<[t_1,t_1][t_2,t_2]>$.

2.3 Representing the world

Assume the set of temporal points T_0. The basic temporal set T consists of the uncertain temporal intervals and is defined as $\mathbf{T=\{<[t_1,t_2],[t_3,t_4]> : t_i \in T_0,\ i=1,...,4,\ t_1 \le t_2,\ t_3 \le t_4,\ t_1 \le t_3,\ t_2 \le t_4\}}$. Notice that $t_2 \le t_3$ is not a necessary condition. We also provide T with a total ordering :

$\forall r,s \in T$, $<[r1,r2],[r3,r4]> = <[s1,s2],[s3,s4]>$
iff $([r1,r2]=[s1,s2] \wedge [r3,r4]=[s3,s4])$.

$\forall r,s \in T$, $<[r1,r2],[r3,r4]> < <[s1,s2],[s3,s4]>$
iff $[([r1,r2]<[s1,s2])$
$\vee ([r1,r2]=[s1,s2] \wedge [r3,r4]<[s3,s4])]$,

where $[r1,r2] = [s1,s2]$ when $(r1=s1$ and $r2=s2)$, and $[r1,r2]<[s1,s2]$ when $(r1<s1$ or $(r1=s1 \wedge r2<s2))$.

Assume D_t, a finite subset of T which includes those moments that are part of our conceptualisation of the world. Assume also D_c, the set of classical objects taking part in the phenomenon we are conceptualising. Following [Gen87], the world is conceptualised by the tuple <D,F,R>, where D is the set of objects, F the set of functions, and R the set of relations. This tuple is called a **TRL-structure**. More specifically, $D=D_c \cup D_t$, $F=F_c \cup F_t$, $R=R_c \cup R_t$, where the subscript 'c' implies the classical view of the world, and the subscript 't' implies the temporal view of the world.

Example 1. This is a similar example to the example appearing in [Kow86].

1. Mary was hired as a lecturer at t_2.
2. Mary left as a professor at some moment between t_5 and t_9.
3. Mike left as a professor at some moment between t_8 and t_{12}.
4. Mary was promoted from lecturer to professor at some moment between t_5 and t_9.
5. Mike was promoted from lecturer to professor some moment between t_6 and t_9.
6. Hiring x as y starts a period of time for which x has rank y.
7. Leaving x as y ends a period of time for which x has rank y.
8. Promoting x from y to z ends a period of time for which x has rank y and starts a consecutive period of time for which x has rank z.

Generally, t_i are symbolic names of dates, e.g. t_1 stands for 10th of May, 1970, t_2 stands for 12th of September, 1970, etc. but for simplicity we make the assumption $t_i=i$, $i=1,...,12$, $t_i<t_{i+1}$, all expressed in the same time unit. The tuple <D,F,R> for this example consists of the following sets :

$D_c=\{mary^i,mike^i,lecturer^i,professor^i\}$, $F_c=\varnothing$, $R_c=\{hire^i,leave^i,promote^i,\ rank^i\}$, D_t takes values for its temporal moments from the set $\{t_1,..,t_{12}\}$, $F_t=\varnothing$, $R_t=\varnothing$. The relations $hire^i$,$leave^i$,$promote^i$, $rank^i$ consist of tuples which indicate the classical and temporal objects for which these relations are true.:

$hire^i$ = $\{<mary^i,lecturer^i,<[t_2,t_2],[t_2,t_2]>>\}$
$leave^i$ = $\{<mary^i,professor^i,<[t_5,t_9],[t_5,t_9]>>,$
$<mike^i,professor^i,<[t_8,t_{12}],[t_8,t_{12}]>>\}$
$promote^i$=
$\{<mary^i,lecturer^i,professor^i,<[t_5,t_9],[t_5,t_9]>>,$
$<mike^i,lecturer^i,professor^i,<[t_6,t_9],[t_6,t_9]>>\}$
$rank^i$=
$\{<mary^i,lecturer^i,<[t_2,t_2],[t_4,t_8]>>,$
$<mary^i,professor^i,<[t_5,t_9],[t_5,t_9]>>,$
$<mike^i,lecturer^i,<[t_1,t_8],[t_5,t_8]>>,$
$<mike^i,professor^i,<[t_6,t_9],[t_7,t_{11}]>>\}$

Notice, that using appropriate uncertain temporal intervals we have compressed our knowledge to

small sets of tuples. Some very interesting tuples appear, however. Two such cases are the tuples expressing the knowledge about the rank of Mike. The first tuple, i.e. $<mike^1,lecturer^1, <[t_1,t_8],[t_5,t_8]>>$, expresses the knowledge : "Mike started having the rank of a lecturer at some temporal point from t_1 to t_8, and finished having that rank at some temporal point between t_5 and t_8". The second tuple, i.e. $<mike^1,professor^i,<[t_6,t_9],[t_7,t_{11}]>>$ is curious too, as the starting uncertain temporal instance overlaps with the finishing uncertain temporal instance. It expresses the knowledge : "Mike started having the rank of a professor at some time point from t_6 to t_9, and finished having that rank at some time point between t_7 and t_{11}". The semantics of TRL take care of the correct interpretation of such pieces of knowledge, e.g. if we select the time point t_9 for the starting point of $<[t_6,t_9],[t_7,t_{11}]>$, then we must select a time point greater than t_9 for the ending point of $<[t_6,t_9],[t_7,t_{11}]>$.

Notice also that the notion of uncertainty as lack of knowledge, is deeply embedded in the primitives of the TRL language.

2.4 The syntax of TRL

The TRL language consists of the set Δ_τ of temporal constants, the set of temporal variables, the set Φ_τ of temporal function symbols , the set P_τ of temporal relation symbols, classical constants Δ_K, classical variables, classical function symbols Φ_K and classical relation symbols P_K. ($\Delta=\Delta_\tau\cup\Delta_K$, $\Phi=\Phi_\tau\cup\Phi_K$, $P=P_\tau\cup P_K$).

The symbols of TRL consist of the symbols of first order predicate logic, and the temporal symbols which we use to represent temporal references: temporal variables (T_0, T_1, ...), temporal reference constructors ([ ] < >), temporal constants (t_0,t_1, ...),a set of temporal function symbols, and a set of temporal relation symbols. A τ-term (temporal term) is defined to be a τ-constant, a τ-variable, or an n-ary τ-function symbol $\pi(\tau1,...,\tau n)$, where $\tau1,\tau2,...,\tau n$ are τ-terms. An atom is either a c-atom, a t-atom or an e-atom. A c-atom is a classical atom, a t-atom is an atom based on a temporal relation symbol and τ-terms. If τ is a τ-term and φ is a classical atom, then $\tau:\varphi$ is an e-atom (extended atom). All atoms take true or false values. The assignment of a truth value to an e-atom depends on the true or false value of the clause φ at, on, or during the temporal reference τ. There are also the standard quantifiers $\forall\tau,\exists\tau$, where τ is a τ-term.

Extended well-formed formulas (ewffs) are defined as follows : An atom is an ewff. If φ, ψ are ewffs then $\neg\varphi$, $\varphi\wedge\psi$, $\varphi\vee\psi$, $\varphi\rightarrow\psi$, $\varphi\leftrightarrow\psi$, $(\forall x)\varphi$, $(\exists x)\varphi$, $<[\tau1,\tau2],[\tau3,\tau4]>:\varphi$, $(\forall<[T1,T2],[T3,T4]>)\varphi$,$(\exists<[T1,T2],[T3,T4]>)\varphi$, are also ewffs.

τ_2:(student(john)$\wedge<[\tau_1,\tau_2],[\tau_3,\tau_4]>$:nice(john)) $\rightarrow[\tau_3,\tau_4]$:likes(ann,john) is an ewff which must be interpreted as "If at t_2 it is true that : John is a student and he is nice during the uncertain interval $<[t_1,t_2],[t_3,t_4]>$, then Mary likes John at some moment(s) between t_3 and t_4.".

2.5. The Semantics of TRL

Definition A temporal interpretation i is a mapping between the elements of TRL, $<\Delta,\Phi,P>$ and the elements of the TRL-structure, $<D,F,R>$, and is defined as follows :

$\tau^i\in D_t$, $\pi^i:{D_t}^m\rightarrow D_t$, $\rho^i\subseteq {D_t}^m$,
where $\tau\in\Delta_\tau$, $\pi\in\Phi_\tau$, $\rho\in P_\tau$,

$\alpha^i\in D_c$, $\pi^i:{D_c}^m x D_t\rightarrow D_c$, $\rho^i\subseteq {D_c}^m x D_t$,
where $\alpha\in\Delta_K$, $\pi\in\Phi_K$, $\rho\in P_K$

We extend the notion of the temporal interpretation by the notion of an assignment. An assignment iu is also a mapping from $<\Delta,\Phi,P>$ to $<D,F,R>$, but takes into account the assignment u of variables to constants :

$iu_\tau(\sigma)=iu(\sigma)=i(\sigma)=\sigma^i$, where $\sigma\in\Delta$,
$iu_\tau(\sigma)=iu(\sigma)=u(\sigma)$, where σ is a variable,
$iu_\tau(\varphi(\alpha_1,\alpha_2,...,\alpha_n)) = f(a_1,a_2,...,a_n)$, where $\varphi\in\Phi$,
$iu_\tau(\varphi)=f$, α_j is a term, $iu(\alpha_j)=a_j$, j=1,..,n.♦

The satisfaction formulae of the TRL language are defined as follows :

(IS1) $\models_{(i,\tau')}\rho(\alpha_1,\alpha_2,...,\alpha_m)[u] \Leftrightarrow$
$<iu_{\tau'}(\alpha_1),...,iu_{\tau'}(\alpha_m),iu_{\tau'}(\tau')>\in\rho^i$,

(IS2) $\models_{(i,\tau')}(\tau=\sigma)[u] \Leftrightarrow iu_{\tau'}(\tau)=iu_{\tau'}(\sigma)$,
$\models_{(i,\tau')}(\tau<\sigma)[u] \Leftrightarrow iu_{\tau'}(\tau)<iu_{\tau'}(\sigma)$

(IS3) $\models_{(i,\tau')}<[\alpha,\beta],[\gamma,\delta]>:\psi[u]\Leftrightarrow$
$\exists T'\in\{\alpha,...,\beta\},\exists T''\in\{\gamma,...,\delta\}$,
$\forall T\in\{T',...,T''\},\models_{(i,T)}\psi[u]$

(IS4) $\models_{(i,\tau)}(\neg\psi)[u] \Leftrightarrow \neg\models_{(i,\tau)}\psi[u]$
$\models_{(i,\tau)}(\wedge\psi_j)[u] \Leftrightarrow (\forall j), \models_{(i,\tau)}\psi_j[u]$
$\models_{(i,\tau)}(\vee\psi_j)[u] \Leftrightarrow (\exists j), \models_{(i,\tau)}\psi_j[u]$
$(\psi_1\rightarrow\psi_2)\equiv(\neg\psi_1\vee\psi_2)$,
$(\psi_1\leftrightarrow\psi_2)\equiv(\psi_1\rightarrow\psi_2)\wedge(\psi_2\rightarrow\psi_1)$

(IS5) $\models_i(\forall x)\psi[u] \Leftrightarrow (\forall c), \models_i\psi[v]$,
v(y)={c if y=x, u(y) otherwise}
$\models_i(\exists x)\psi[u] \Leftrightarrow (\exists c), \models_i\psi[v]$,
v(y)={c if y=x, u(y) otherwise}

(IS6) $\models_{(i,\tau)}(\forall<[T_1,T_2],[T_3,T_4]>)\psi[u] \Leftrightarrow$
$(\forall<[\sigma_1,\sigma_2],[\sigma_3,\sigma_4]>\in\Delta_\tau), \models_{(i,\tau)}\psi[v]$,
v(x)={σ_j if x=T_j, j=1,...,4, u(x) otherwise}
$\models_{(i,\tau)}(\exists<[T_1,T_2],[T_3,T_4]>)\psi[u] \Leftrightarrow$
$(\exists<[\sigma_1,\sigma_2],[\sigma_3,\sigma_4]>\in\Delta_\tau), \models_{(i,\tau)}\psi[v]$,
v(x)={σ_j if x=T_j, j=1,...,4, u(x) otherwise}

3. Preliminary results

3.1 The semantics of other temporal objects

In TRL, the primitive temporal object is the uncertain temporal interval. Other temporal objects are special cases of this object :

Temporal points or points (e.g. $<[\tau_1,\tau_1],[\tau_1,\tau_1]>$), **Certain temporal intervals** or intervals (e.g. $<[\tau_1,\tau_1],[\tau_2,\tau_2]>$), **Temporal instances** or uncertain points (e.g. $<[\tau_1,\tau_2],[\tau_1,\tau_2]>$). According to this convention the semantics for these objects can be obtained from (IS3) :

(a)Temporal points :

$\models_{(i,\tau')} \tau{:}\psi[u] \Leftrightarrow \models_{(i,\tau)} \psi[u]$ **(IS3.a)**

Proof $\models_{(i,\tau')} \tau{:}\psi[u]$

$\Leftrightarrow \models_{(i,\tau')} <[\tau,\tau],[\tau,\tau]>{:}\psi[u]$

$\Leftrightarrow \exists T'\in\{\tau...\tau\}, \exists T''\in\{\tau...\tau\}, \forall T\in\{T'...T''\}, \models_{(i,T)} \psi[u]$

$\Leftrightarrow \forall T\in\{\tau...\tau\}, \models_{(i,T)} \psi[u]$

$\Leftrightarrow \models_{(i,\tau)} \psi[u]$

The intuitive meaning of the proposition $\tau{:}\psi$ is that proposition ψ is true at the temporal point τ. In order to compute the truth value of $\tau{:}\psi$ at a temporal point τ' it is sufficient to compute the truth value of ψ at τ.

(b) Temporal intervals

$\models_{(i,\tau')} <\tau_1,\tau_2>{:}\psi[u] \Leftrightarrow \forall T\in\{\tau_1...\tau_2\}, \models_{(i,T)} \psi[u]$ **(IS3.b)**

Proof $\models_{(i,\tau')} <\tau_1,\tau_2>{:}\psi[u]$

$\Leftrightarrow \models_{(i,\tau')} <[\tau_1,\tau_1],[\tau_2,\tau_2]>{:}\psi[u]$

$\Leftrightarrow \exists T'\in\{\tau_1...\tau_1\}, \exists T''\in\{\tau_2...\tau_2\}, \forall T\in\{T'...T''\}, \models_{(i,T)} \psi[u]$

$\Leftrightarrow \forall T\in\{\tau_1...\tau_2\}, \models_{(i,T)} \psi[u]$

The intuitive meaning of the proposition $<\tau_1,\tau_2>{:}\psi$ is that proposition ψ is true during the temporal interval $<\tau_1,\tau_2>$. The proposition $<\tau_1,\tau_2>{:}\psi$ is true at a temporal point τ' iff ψ is true for all the temporal points in $<\tau_1,\tau_2>$.

(c) Temporal instances :

$\models_{(i,\tau')} [\tau_1,\tau_2]{:}\psi[u] \Leftrightarrow \exists T\in\{\tau_1...\tau_2\}, \models_{(i,T)} \psi[u]$ **(IS3.c)**

Proof $\models_{(i,\tau')} <\tau_1,\tau_2>{:}\psi[u]$

$\Leftrightarrow \models_{(i,\tau')} <[\tau_1,\tau_2],[\tau_1,\tau_2]>{:}\psi[u]$

$\Leftrightarrow \exists T'\in\{\tau_1...\tau_2\}, \exists T''\in\{\tau_1...\tau_2\}, \forall T\in\{T'...T''\}, \models_{(i,T)} \psi[u]$ (obviously $T'\leq T''$)

$\Leftrightarrow \exists T''\in\{\tau_1...\tau_2\}, \exists T''\in\{T'...\tau_2\}, \forall T\in\{T'...T''\}, \models_{(i,T)} \psi[u]$

$\Leftrightarrow (\exists T''\in\{\tau_1...\tau_2\}, \forall T\in\{\tau_1...T''\}, \models_{(i,T)} \psi[u])$
$\vee...\vee$
$(\exists T''\in\{\tau_2...\tau_2\}, \forall T\in\{\tau_2...T''\}, \models_{(i,T)} \psi[u])$

$\Leftrightarrow (\forall T\in\{\tau_1,\tau_1\}, \models_{(i,T)} \psi[u]) \vee$
$(\forall T\in\{\tau_1,\tau_1+1\}, \models_{(i,T)} \psi[u]) \vee...\vee$
$(\forall T\in\{\tau_1...\tau_2\}, \models_{(i,T)} \psi[u]) \vee$
$(\forall T\in\{\tau_1+1,\tau_1+1\}, \models_{(i,T)} \psi[u]) \vee$
$(\forall T\in\{\tau_1+1,\tau_1+2\}, \models_{(i,T)} \psi[u]) \vee...\vee$
$(\forall T\in\{\tau_1+1...\tau_2\}, \models_{(i,T)} \psi[u]) \vee...\vee$
$(\forall T\in\{\tau_2,\tau_2\}, \models_{(i,T)} \psi[u])$

$\Leftrightarrow (T=\tau_1, \models_{(i,T)} \psi[u]) \vee$
$(T=\tau_1+1, \models_{(i,T)} \psi[u]) \vee...\vee$
$(T=\tau_2, \models_{(i,T)} \psi[u])$

$\Leftrightarrow (\models_{(i,\tau 1)} \psi[u]) \vee...\vee (\models_{(i,\tau 2)} \psi[u])$

$\Leftrightarrow \exists T\in\{\tau_1...\tau_2\}, \models_{(i,T)} \psi[u]$

The intuitive meaning of the proposition $[\tau_1,\tau_2]{:}\psi$ is that proposition ψ is true during the temporal instance $[\tau_1,\tau_2]$. The proposition $[\tau_1,\tau_2]{:}\psi$ is true at a temporal point τ' iff ψ is true for at least one temporal point in $[\tau_1,\tau_2]$.

3.2. Properties of Satisfaction

Except from the basic satisfaction axioms, we have also examined relations with conjunctions, disjunctions and negation (see APPENDIX for selected proofs):

A. Properties concerning conjunctions

(PC1) $\models_{(i,\tau')} \tau{:}(\wedge\psi_j)[u] \Leftrightarrow \models_{(i,\tau')} (\wedge\tau{:}\psi_j)[u]$

(PC2) $\models_{(i,\tau')} <\tau_1,\tau_2>{:}(\wedge\psi_j)[u] \Leftrightarrow \models_{(i,\tau')} (\wedge<\tau_1,\tau_2>{:}\psi_j)[u]$

(PC3) $\models_{(i,\tau')} [\tau_1,\tau_2]{:}(\wedge\psi_j)[u] \Rightarrow \models_{(i,\tau')} (\wedge[\tau_1,\tau_2]{:}\psi_j)[u]$

B. Properties concerning disjunctions

(PD1) $\models_{(i,\tau')} \tau{:}(\vee\psi_j)[u] \Leftrightarrow \models_{(i,\tau')} (\vee\tau{:}\psi_j)[u]$

(PD2) $\models_{(i,\tau')} [\tau_1,\tau_2]{:}(\vee\psi_j)[u] \Leftrightarrow \models_{(i,\tau')} (\vee[\tau_1,\tau_2]{:}\psi_j)[u]$

(PD3) $\models_{(i,\tau')} (\vee<\tau_1,\tau_2>{:}\psi_j)[u] \Rightarrow \models_{(i,\tau')} <\tau_1,\tau_2>{:}(\vee\psi_j)[u]$

C. Properties concerning negation

(PN1) $\models_{(i,\tau')} \tau{:}(\neg\psi)[u] \Leftrightarrow \neg \models_{(i,\tau')} \tau{:}\psi[u] \Leftrightarrow \models_{(i,\tau')} (\neg\tau{:}\psi)[u]$

(PN2) $\models_{(i,\tau')} [\tau_1,\tau_2]{:}(\neg\psi)[u] \Leftrightarrow \neg \models_{(i,\tau')} <\tau_1,\tau_2>{:}\psi[u] \Leftrightarrow \models_{(i,\tau')} (\neg<\tau_1,\tau_2>{:}\psi)[u]$

(PN3) $\models_{(i,\tau')} <\tau_1,\tau_2>{:}(\neg\psi)[u] \Leftrightarrow \neg \models_{(i,\tau')} [\tau_1,\tau_2]{:}\psi[u] \Leftrightarrow \models_{(i,\tau')} (\neg[\tau_1,\tau_2]{:}\psi)[u]$

(PN4) $\neg \models_{(i,\tau')} [\tau_1,\tau_2]{:}\psi[u] \Rightarrow \models_{(i,\tau')} [\tau_1,\tau_2]{:}(\neg\psi)[u]$

(PN5) $\models_{(i,\tau')} <\tau_1,\tau_2>{:}(\neg\psi)[u] \Rightarrow \neg \models_{(i,\tau')} <\tau_1,\tau_2>{:}\psi[u]$

(PN6) $\models_{(i,\tau')} (\neg[\tau_1,\tau_2]{:}\psi)[u] \Rightarrow \models_{(i,\tau')} (\neg<\tau_1,\tau_2>{:}\psi)[u]$

(PN7) $\models_{(i,\tau')} [\tau_1,\tau_2]{:}\psi[u] \Rightarrow \neg \models_{(i,\tau')} <\tau_1,\tau_2>{:}\psi[u]$

4. Models, consequence and equivalence

Assume that ψ is an ewff. Then we define :

(D2.1) ψ is *instantly satisfied at* τ by (i,u) iff $\models_{(i,\tau)} \psi[u]$.

(D2.2) (i,τ) is an *instant model* of ψ (at τ) iff $(\forall U) \models_{(i,\tau)} \psi[U]$

ψ is *instantly satisfiable* at τ if it has an instant model at τ.

(D2.3) ψ is *instantly valid at* τ ($\models_\tau \psi$) iff $(\forall I)(\forall U) \models_{(I,\tau)} \psi[U]$

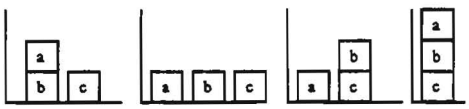
(D2.4) ψ is an instant consequence of φ at τ ($\varphi \models_\tau \psi$) iff $(\forall I)(\forall U)(\models_{(I,\tau)} \varphi[U] \Rightarrow \models_{(I,\tau)} \psi[U])$

(D2.5) Instant equivalence of φ,ψ : $\varphi|=_{\tau}\psi \Leftrightarrow \varphi|=_{\tau}\psi$ and $\psi|=_{\tau}\varphi$ ♦

Example 2. Satisfaction of sentences from the blocks' world.

Assume the scenario of events depicted in figure 1. The conceptualisation C=<D,F,R> is defined as follows : $D=D_c\cup D_t$, where $D_c=\{a^i,b^i,c^i\}$, D_t can be easily constructed from $\{t1,t2,t3,t4,t5,t6,t7,t8\}$, $F=F_c\cup F_t$, where $F_c=\{hat^i\}$, $F_t=\varnothing$, $R=R_c\cup R_t$, where $R_c=\{on^i,above^i,table^i\}$, $R_t=\varnothing$. The function hat and the relation on are interpreted as :

$hat^i=\{<b^i,a^i,<t1,t3>>,$
$<c^i,b^i,<t5,t8>>, <b^i,a^i,<t6,t8>>\}$
$on^i=\{<a^i,b^i,<t1,t3>>,$
$<b^i,c^i,<t5,t8>>, <a^i,b^i,<t6,t8>>\}$



(a) t1,t2,t3 (b) t4 (c) t5 (d) t6,t7,t8

Figure 1. A block's world

Using the satisfaction formulas we can compute that the following sentences are instantly satisfied at any temporal point τ :

(1) $(\forall<T1,T2>)$ <T1,T2>:table(c)
(2) $(\exists[T1,T2])$ [T1,T2]:table(b)
(3) $(\exists T)$ T:¬table(b)
(4) $(\exists T)$ T:table(b)
(5) $(\forall T)$ T:table(c)
(6) $(\forall T)(\forall X)(\forall Y)(\forall Z)$ (T:above(X,Z)←
((T:on(X,Z)∨(T:above(X,Y)∧T:above(Y,Z))))
(7) $(\forall T)(\forall X)(\forall<T1,T2>)$(T:table(X)←
((<T1,T2>:table(X))∧(T1≤T)∧(T≤T2)))

4.1. Propositions and Lemmas

The following propositions can be proved (see APPENDIX for the proof of P3).

(P1) $\varphi|=_{\tau}\psi \Leftrightarrow |=_{\tau}\varphi\rightarrow\psi$
(P2) $\varphi|=|_{\tau}\psi \Leftrightarrow |=_{\tau}\varphi\leftrightarrow\psi$
(P3) $\varphi|=|_{\tau}\psi \Leftrightarrow (\forall I)(\forall U)(\ |=_{(I,\tau)}\varphi[U]$
$\Leftrightarrow |=_{(I,\tau)}\psi[U]\ )$

The following Lemmas can also be proved (see APPENDIX)

(L1) $\neg\ (|=_{(i,\tau')}\tau:\varphi[u]) \Leftrightarrow |=_{(i,\tau')}\neg\tau:\varphi[u]$
$\Leftrightarrow |=_{(i,\tau')}(\tau:\neg\varphi)[u] \Leftrightarrow |=_{(i,\tau)}\neg\varphi[u]$
(L2) $|=_{(i,\tau')}\neg([\tau1,\tau2]:\varphi)[u]$
$\Leftrightarrow |=_{(i,\tau')}<\tau1,\tau2>:(\neg\varphi)[u]$
(L3) $|=_{(i,\tau')}\neg(<\tau1,\tau2>:\varphi)[u]$
$\Leftrightarrow |=_{(i,\tau')}[\tau1,\tau2]:(\neg\varphi)[u]$

4.2. Equivalences and consequences concerning temporal references

Many interesting results have also been proved in the context of consequence and equivalence. (see APPENDIX for selected proofs).

(TE1) $\neg\tau:\varphi|=|_{\tau}\tau:\neg\varphi$, where τ is a temporal point.
(TE2) $\neg[\tau,\sigma]:\varphi|=|_{\tau}<\tau,\sigma>:\neg\varphi$
(TE3) $\neg<\tau,\sigma>:\varphi|=|_{\tau}[\tau,\sigma]:\neg\varphi$
(TE4) $\neg[\tau,\sigma]:\neg\varphi|=|_{\tau}<\tau,\sigma>:\varphi$
(TE5) $\neg<\tau,\sigma>:\neg\varphi|=|_{\tau}[\tau,\sigma]:\varphi$
(TE6) $\tau:(\sigma:\varphi)|=|_{\tau}\sigma:\varphi$, where τ,σ are uncertain temporal intervals.
(TE7) $\tau:(\varphi\vee\sigma:\psi)|=|_{\tau}\tau:\varphi\vee\sigma:\psi$, where τ,σ are uncertain temporal intervals.
(TE8) $\tau:(\varphi\wedge\sigma:\psi)|=|_{\tau}\tau:\varphi\wedge\sigma:\psi$, where τ,σ are uncertain temporal intervals.
(TE9) $\tau:(\varphi\wedge\psi)|=|_{\tau}\tau:\varphi\wedge\tau:\psi$, where τ is a temporal point.
(TE10) $\tau:(\varphi\vee\psi)|=|_{\tau}\tau:\varphi\vee\tau:\psi$, where τ is a temporal point.
(TE11) $<\tau,\sigma>:(\varphi\wedge\psi)|=|_{\tau}<\tau,\sigma>:\varphi\wedge<\tau,\sigma>:\psi$
(TE12) $[\tau,\sigma]:(\varphi\vee\psi)|=|_{\tau}[\tau,\sigma]:\varphi\vee[\tau,\sigma]:\psi$
(TE13) $\tau:(\varphi\rightarrow\psi)|=|_{\tau}\tau:\varphi\rightarrow\tau:\psi$, , where τ is a temporal point.
(TC1) $<\tau,\sigma>:\varphi\vee<\tau,\sigma>:\psi\ |=_{\tau}<\tau,\sigma>:(\varphi\vee\psi)$
(TC2) $[\tau,\sigma]:(\varphi\wedge\psi)|=_{\tau}[\tau,\sigma]:\varphi\wedge[\tau,\sigma]:\psi$
(TC3) $<\tau1,\sigma1>:\varphi|=_{\tau}<\tau2,\sigma2>:\varphi$, $\{\tau2...\sigma2\}\subseteq\{\tau1...\sigma1\}$
(TC4) $<\tau1,\sigma1>:\varphi|=_{\tau}[\tau2,\sigma2]:\varphi$, $\{\tau2...\sigma2\}\cap\{\tau1...\sigma1\}\neq\varnothing$
(TC5) $<\tau,\sigma>:\varphi|=_{\tau}\rho:\varphi$, $\rho\in\{\tau...\sigma\}$
(TC6) $[\tau1,\sigma1]:\varphi|=_{\tau}[\tau2,\sigma2]:\varphi,\{\tau2...\sigma2\}\supseteq\{\tau1...\sigma1\}$
(TC7) $<\tau,\sigma>:(\varphi\rightarrow\psi)|=_{\tau}<\tau,\sigma>:\varphi\rightarrow<\tau,\sigma>:\psi$
(TC8) $[\tau,\sigma]:\varphi\rightarrow[\tau,\sigma]:\psi|=_{\tau}[\tau,\sigma]:(\varphi\rightarrow\psi)$

5. The expressive power of the TRL language

Notice that we haven't demonstrated any properties including temporal quantifiers. We may say that the demonstrated results concern only the propositional temporal case. The expressive power of the TRL language can be demonstrated by its ability to represent sentences expressively rich in temporal information. Some of these sentences are also represented in other logics. In the following paragraphs, we demonstrate the expressive power of TRL taking into account a sample of its properties.

- Properties (TE1), (TE9), (TE10), (TE13) are expected, since TRL behaves as a first-order logic if its formulas are referenced by temporal points.
- Equivalences (TE2), (TE3) remind the properties of universal and existential quantifiers.

- The P,F operators (possibility operator of Modal Logics) can be both expressed in TRL by a temporal instance. In fact, the possibility is better expressed through a temporal instance [τ1,τ2], because one can be more specific about the temporal bounds with a temporal instance than just referring to the possibility in the future (F operator) or the possibility in the past (P operator). A temporal instance may start at sometime in the past and finish at sometime in the future. The H,G operators (necessity operator of Modal Logics) can also be both expressed in TRL by a temporal interval <τ1,τ2>. Again, a temporal interval is more expressive than the H,G operators. Assume the sentence q="John is happy". In Tense Logic [Gal87a], the sentences "John has been happy", "John will be happy", "John has always been happy", "John will always be happy" are expressed by Pq, Fq, Hq, Gq, respectively. In TRL such information can be represented : [first,now]:q, [now,last]:q, <first,now>:q, <now,last>:q, assuming that first,last and now have been defined to be the first moment, the last moment and the current moment of the conceptualisation. However, such sentences are just special cases of even more complex sentences : t:p, (p has been true for only one t), <t-2,t+5>:p, (p has been true for a certain temporal interval <t-5,t+2>), [t-10,t+10]:p, i.e. at some moments between t-10 and t+10, <[t-100,t-1], [t+1,t+100]>:p, i.e. it starts being true at some time point between t-100 and t-1, and finishes being true at some time point between t+1 and t+100.
- Moreover, information concerning temporal points, can be still expressed. It is very important however, that all temporal references are special cases of the uncertain temporal interval. Therefore, a single notation captures the meaning of the operators P,F,G,H.
- Equivalences (TE4), (TE5) resemble to (and generalise) some equivalences of Tense Logic : $Hq \Leftrightarrow \neg P \neg q$;
  $Gq \Leftrightarrow \neg F \neg q$; $\Box q \Leftrightarrow \neg \Diamond \neg q$
- Consequence (TC7) resembles to (and generalises) the following axiom schema:
  $H(p \rightarrow q) \rightarrow (Hp \rightarrow Hq)$;
  $G(p \rightarrow q) \rightarrow (Gp \rightarrow Gq)$
- Consequence (TC8) provides an additional property respective to temporal instances.
- Assume the sentence q="John eats an apple".in the Logic of Occurrence [Gal87b], the sentences "John has been eating an apple", "John is eating an apple", "John will be eating an apple", "John has eaten an apple", "John will eat an apple" can be represented by P Prog q, Prog p, F Prog p, Perf q, Pros p. In TRL such information may for example be represented : <now-t,now-s>:q, where t>s, <now-t,now+s>:q, <now+t,now+s>:q, where t<s, (now-t):q, (now+t):q. However, such sentences are just special cases of even more complex sentences : "John has been eating an apple at some time between 8:00 a.m. and 10:00 a.m.: <[8,8+t],[10-s,10]>:q, where t,s are some positive temporal points, such that 8+t<10-s.
- Allen's Logic [All83, All84, All85, All87], can represent sentences involving the relations after, before, starts, overlaps, meets, etc. In TRL, the temporal set D_t, is augmented with a total ordering. Using this ordering it is easy to built semantics for representing Allen's relations and functions. Moreover, we can extend and generalise these relations to the level of temporal instances and uncertain temporal intervals : starts(1910,<[1905,1915],1940>), before(<[t2,t2],[t3,t4]>,<[s1,s2],[s3,s4]>), overlaps(<[1920,1930],[1950,1960]>, <1940,[1945,1950]>), etc.
- The predicates HOLDS(p,T), IN(t,T), OCCURING(p,T) which appear in Allen's Logic can also be represented in TRL by the sentences <t1,t2>:p, <t1,t2>⊆<T1,T2>, [t1,t2]:p, where ⊆ is an infix predicate testing the set inclusion. Most of the corresponding properties appearing in [All84] are generalised within the TRL framework. Only the predicate OCCUR(e,t) cannot be represented.

6. Conclusions

TRL is a logic based temporal language, which extends a first-order logic to a logic incorporating temporal references. It deals with the notion of temporal uncertainty which is deeply embedded in its semantics, represents both intervals and temporal points (certain and uncertain) with a single notation and generalises notions appearing in many other temporal logics. We have presented the specifications, syntax and semantics of TRL. We have also given some examples and presented some properties illustrating the expressive power of TRL.

TRL introduces four types of temporal references : temporal points, temporal instances, certain temporal intervals and uncertain temporal intervals, but uses the uncertain temporal interval as the primitive temporal object. The uncertain temporal interval, however, is the most general type of temporal references and can be used to represent all other types of temporal references.

The syntax of TRL, supported by the defined semantics, provides a logical framework in which one can prove equivalences and consequences similar to, but more general than, those of Tense Logic, the Logic of Occurrence,

and Allen's Logic. However, TRL's framework is more expressive as it allows the user to represent temporal information not only symbolically, but also numerically.

We have also introduced the concept of temporal quantifiers and we have demonstrated the properties for the propositional case, i.e. without using any temporal quantifiers.

In order to test the semantics of the TRL language we have developed a Prolog program which implements these semantics. In this program we can define the universe of discourse (temporal and classical), the function and relation symbols for each example, and we can then provide the system with ewffs in order to examine their truth value, given the conceptualisation.

Recall that, our final target is to develop a framework equipped with the expressive power of first order logic and powerful temporal extensions which can be used in practical applications such as : planning, scheduling, temporal and deductive databases, etc. In order to complete our goal we are planning in the near future to develop a proof system for TRL. The assumptions which have been made, of a discrete and bounded temporal universe, will be much more practical and computationally tractable for the types of applications we have in mind.

APPENDIX : Selected Proofs

From first order logic we select the following properties :

(H1) $(\forall x)\ (\varphi(x)\wedge\psi(x))\Leftrightarrow(\forall x)\ \varphi(x)\wedge(\forall x)\ \psi(x)$

(H2) $(\exists x)\ (\forall y)\ \varphi(x,y) \Rightarrow(\forall y)\ (\exists x)\ \varphi(x,y)$

(H3) $(\forall x)\ (\exists y)\ \varphi(x,y)\Leftarrow(\exists y)\ (\forall x)\ \varphi(x,y)$

(H4) $(\forall x)\ (\forall y)\ \varphi(x,y)\Leftrightarrow(\forall y)\ (\forall x)\ \varphi(x,y)$

(H5) $(\exists x)\ (\exists y)\ \varphi(x,y)\Leftrightarrow(\exists y)\ (\exists x)\ \varphi(x,y)$

(H6) $(\forall x)\ \neg\varphi(x) \Leftrightarrow \neg\ (\exists x)\ \varphi(x)$

(H7) $\neg(\exists x)\ \varphi(x) \Rightarrow \neg(\forall x)\ \varphi(x)$

A.I Selected proofs of basic properties

We will first prove some helpful propositions :

(A10) $\models_{(i,\tau')} [\tau_1,\tau_2]:(\wedge\psi_j)[u]$

$\Rightarrow (\forall j), \models_{(i,\tau')} [\tau_1,\tau_2]:\psi_j[u]$

Proof : $\models_{(i,\tau')} [\tau_1,\tau_2]:(\wedge\psi_j)[u]$

$\Leftrightarrow^{(IS3.c)} (\exists T\in\{\tau_1...\tau_2\}), \models_{(i,T)} (\wedge\psi_j)[u]$

$\Leftrightarrow^{(IS4)} (\exists T\in\{\tau_1...\tau_2\})(\forall j), \models_{(i,T)} \psi_j[u]$

$\Rightarrow^{(H2)} (\forall j)(\exists T\in\{\tau_1...\tau_2\}), \models_{(i,T)} \psi_j[u]$

$\Leftrightarrow^{(IS3.c)} (\forall j), \models_{(i,\tau')} [\tau_1,\tau_2]:\psi_j[u]$

(A11) $\models_{(i,\tau')} <\tau_1,\tau_2>:(\wedge\psi_j)[u]$

$\Leftrightarrow (\forall j), \models_{(i,\tau')} <\tau_1,\tau_2>:\psi_j[u]$

Proof : $\models_{(i,\tau')} <\tau_1,\tau_2>:(\wedge\psi_j)[u]$

$\Leftrightarrow^{(IS3.b)} (\forall T\in\{\tau_1...\tau_2\}), \models_{(i,T)} (\wedge\psi_j)[u]$

$\Leftrightarrow^{(IS4)} (\forall T\in\{\tau_1...\tau_2\})(\forall j), \models_{(i,T)} \psi_j[u]$

$\Leftrightarrow^{(H4)} (\forall j)(\forall T\in\{\tau_1...\tau_2\}), \models_{(i,T)} \psi_j[u]$

$\Leftrightarrow^{(IS3.c)} (\forall j), \models_{(i,\tau')} <\tau_1,\tau_2>:\psi_j[u]$

(A12) $\models_{(i,\tau')} (\wedge[\tau_1,\tau_2]:\psi_j)[u]$

$\Leftrightarrow (\forall j), \models_{(i,\tau')} [\tau_1,\tau_2]:\psi_j[u]$

Proof : $\models_{(i,\tau')} (\wedge[\tau_1,\tau_2]:\psi_j)[u]$

$\Leftrightarrow^{(IS4)} (\forall j), \models_{(i,\tau')} [\tau_1,\tau_2]:\psi_j[u]$

(A13) $\models_{(i,\tau')} (\wedge<\tau_1,\tau_2>:\psi_j)[u]$

$\Leftrightarrow (\forall j), \models_{(i,\tau')} <\tau_1,\tau_2>:\psi_j[u]$

Proof : $\models_{(i,\tau')} (\wedge<\tau_1,\tau_2>:\psi_j)[u]$

$\Leftrightarrow^{(IS4)} (\forall j), \models_{(i,\tau')} \psi_j[u]$

(A20) $\models_{(i,\tau')} [\tau_1,\tau_2]:(\vee\psi_j)[u]$

$\Leftrightarrow (\exists j), \models_{(i,\tau')} [\tau_1,\tau_2]:\psi_j[u]$

Proof : $\models_{(i,\tau')} [\tau_1,\tau_2]:(\vee\psi_j)[u]$

$\Leftrightarrow^{(IS3.c)} (\exists T\in\{\tau_1...\tau_2\}), \models_{(i,T)} (\vee\psi_j)[u]$

$\Leftrightarrow^{(IS4)} (\exists T\in\{\tau_1...\tau_2\})(\exists j), \models_{(i,T)} \psi_j[u]$

$\Leftrightarrow^{(H5)} (\exists j)(\exists T\in\{\tau_1...\tau_2\}), \models_{(i,T)} \psi_j[u]$

$\Leftrightarrow^{(IS3.c)} (\exists j), \models_{(i,\tau')} [\tau_1,\tau_2]:\psi_j[u]$

(A21) $\models_{(i,\tau')} <\tau_1,\tau_2>:(\vee\psi_j)[u]$

$\Leftarrow (\exists j), \models_{(i,\tau')} <\tau_1,\tau_2>:\psi_j[u]$

Proof : $\models_{(i,\tau')} <\tau_1,\tau_2>:(\vee\psi_j)[u]$

$\Leftrightarrow^{(IS3.b)} (\forall T\in\{\tau_1...\tau_2\}), \models_{(i,T)} (\vee\psi_j)[u]$

$\Leftrightarrow^{(IS4)} (\forall T\in\{\tau_1...\tau_2\})(\exists j), \models_{(i,T)} \psi_j[u]$

$\Leftarrow^{(H3)} (\exists j)(\forall T\in\{\tau_1...\tau_2\}), \models_{(i,T)} \psi_j[u]$

$\Leftrightarrow^{(IS3.b)} (\exists j), \models_{(i,\tau')} <\tau_1,\tau_2>:\psi_j[u]$

(A22) $\models_{(i,\tau')} (\vee[\tau_1,\tau_2]:\psi_j)[u]$

$\Leftrightarrow (\exists j), \models_{(i,\tau')} [\tau_1,\tau_2]:\psi_j[u]$

Proof : $\models_{(i,\tau')} (\vee[\tau_1,\tau_2]:\psi_j)[u]$

$\Leftrightarrow^{(IS4)} (\exists j), \models_{(i,\tau')} [\tau_1,\tau_2]:\psi_j[u]$

(A23) $\models_{(i,\tau')} (\vee<\tau_1,\tau_2>:\psi_j)[u]$

$\Leftrightarrow (\exists j), \models_{(i,\tau')} <\tau_1,\tau_2>:\psi_j[u]$

Proof : $\models_{(i,\tau')} (\vee<\tau_1,\tau_2>:\psi_j)[u]$

$\Leftrightarrow^{(IS4)} (\exists j), \models_{(i,\tau')} \psi_j[u]$

(A30) $\models_{(i,\tau')} [\tau_1,\tau_2]:(\neg\psi)[u]$

$\Leftrightarrow \neg \models_{(i,\tau')} <\tau_1,\tau_2>:\psi[u]$

Proof : $\models_{(i,\tau')} [\tau_1,\tau_2]:(\neg\psi)[u]$

$\Leftrightarrow^{(IS3.c)} (\exists T\in\{\tau_1...\tau_2\}), \models_{(i,T)} (\neg\psi)[u]$

$\Leftrightarrow^{(IS4)} (\exists T\in\{\tau_1...\tau_2\})\neg \models_{(i,T)} \psi[u]$

$\Leftrightarrow^{(H6)} \neg(\forall T\in\{\tau_1...\tau_2\}), \models_{(i,T)} \psi[u]$

$\Leftrightarrow^{(IS3.b)} \neg \models_{(i,\tau')} <\tau_1,\tau_2>:\psi[u]$

(A31) $\models_{(i,\tau')} [\tau_1,\tau_2]:(\neg\psi)[u]$

$\Leftarrow \neg \models_{(i,\tau')} [\tau_1,\tau_2]:\psi[u]$

Proof : $\models_{(i,\tau')} [\tau_1,\tau_2]:(\neg\psi)[u]$

$\Leftrightarrow^{(IS3.c)} (\exists T\in\{\tau_1...\tau_2\}), \models_{(i,T)} (\neg\psi)[u]$

$\Leftrightarrow^{(IS4)} (\exists T\in\{\tau_1...\tau_2\})\neg \models_{(i,T)} \psi[u]$

$\Leftrightarrow^{(H6)} \neg(\forall T\in\{\tau_1...\tau_2\}), \models_{(i,T)} \psi[u]$

$\Leftarrow^{(H7)} \neg(\exists T\in\{\tau_1...\tau_2\}), \models_{(i,T)} \psi[u]$

$\Leftrightarrow^{(IS3.b)} \neg \models_{(i,\tau')} [\tau_1,\tau_2]:\psi[u]$

(A32) $\models_{(i,\tau')} (\neg[\tau_1,\tau_2]:\psi)[u]$

$\Leftrightarrow \neg \models_{(i,\tau')} [\tau_1,\tau_2]:\psi[u]$

Proof : $\models_{(i,\tau')} (\neg[\tau_1,\tau_2]:\psi)[u]$

$\Leftrightarrow^{(IS4)} \neg \models_{(i,\tau')} [\tau_1,\tau_2]:\psi[u]$

(A33) $\models_{(i,\tau')} (\neg<\tau_1,\tau_2>:\psi)[u]$

$\Leftrightarrow \neg \models_{(i,\tau')} <\tau_1,\tau_2>:\psi[u]$

Proof : $\models_{(i,\tau')} (\neg<\tau_1,\tau_2>:\psi)[u]$

$\Leftrightarrow^{(IS4)} \neg\models_{(i,\tau')} <\tau_1,\tau_2>:\psi[u]$

(A34) $\models_{(i,\tau')} <\tau_1,\tau_2>:(\neg\psi)[u]$

$\Leftrightarrow \neg \models_{(i,\tau')} [\tau_1,\tau_2]:\psi[u]$

Proof : $\models_{(i,\tau')} <\tau_1,\tau_2>:(\neg\psi)[u]$

$\Leftrightarrow^{(IS3.b)} (\forall T\in\{\tau_1...\tau_2\}), \models_{(i,T)} (\neg\psi)[u]$

$\Leftrightarrow^{(IS4)} (\forall T\in\{\tau_1...\tau_2\})\neg\models_{(i,T)} \psi[u]$

$\Leftrightarrow^{(H9)} \neg(\exists T\in\{\tau_1...\tau_2\}), \models_{(i,T)} \psi[u]$

$\Leftrightarrow^{(IS3.c)} \neg\models_{(i,\tau')} [\tau_1,\tau_2]:\psi[u]$

A.II Selected proofs concerning conjunctions, disjunctions, negation

(PC2) $\models_{(i,\tau')} <\tau_1,\tau_2>:(\wedge\psi_j)[u]$

$\Leftrightarrow\models_{(i,\tau')} (\wedge<\tau_1,\tau_2>:\psi_j)[u]$, from (A11),(A13).

(PC3) $\models_{(i,\tau')} [\tau_1,\tau_2]:(\wedge\psi_j)[u]$

$\Rightarrow \models_{(i,\tau')} (\wedge[\tau_1,\tau_2]:\psi_j)[u]$, from (A10),(A12).

(PD2) $\models_{(i,\tau')} [\tau_1,\tau_2]:(\vee\psi_j)[u]$

$\Leftrightarrow \models_{(i,\tau')} (\vee[\tau_1,\tau_2]:\psi_j)[u]$, from (A20),(A22).

(PD3) $\models_{(i,\tau')} (\vee<\tau_1,\tau_2>:\psi_j)[u]$

$\Rightarrow\models_{(i,\tau')} <\tau_1,\tau_2>:(\vee\psi_j)[u]$, from (A21),(A23).

(PN2) $\models_{(i,\tau')} [\tau_1,\tau_2]:(\neg\psi)[u]$

$\Leftrightarrow \neg \models_{(i,\tau')} <\tau_1,\tau_2>:\psi[u]$

$\Leftrightarrow \models_{(i,\tau')} (\neg<\tau_1,\tau_2>:\psi)[u]$, from (A30),(A33).

(PN3) $\models_{(i,\tau')} <\tau_1,\tau_2>:(\neg\psi)[u]$

$\Leftrightarrow \neg \models_{(i,\tau')} [\tau_1,\tau_2]:\psi[u]$

$\Leftrightarrow \models_{(i,\tau')} (\neg[\tau_1,\tau_2]:\psi)[u]$, from (A34),(A32).

(PN4) $\neg \models_{(i,\tau')} [\tau_1,\tau_2]:\psi[u]$

$\Rightarrow \models_{(i,\tau')} [\tau_1,\tau_2]:(\neg\psi)[u]$, from (A31)

A.III Selected proofs of propositions and lemmas

(P3) $\varphi\models\!|_\tau\psi \Leftrightarrow$

$(\forall I)(\forall U)(\models_{(I,\tau)}\varphi[U] \Leftrightarrow \models_{(I,\tau)}\psi[U])$

Proof : $\varphi \models\!|_\tau \psi$

$\Leftrightarrow^{(D2.5)} \varphi \models_\tau \psi$ and $\psi \models_\tau \varphi$

$\Leftrightarrow^{(D2.4)} (\forall I)(\forall U)(\models_{(I,\tau)}\varphi[U]\Rightarrow\models_{(I,\tau)}\psi[U])$

and $(\forall I)(\forall U)(\models_{(I,\tau)}\varphi[U]\Rightarrow\models_{(I,\tau)}\psi[U])$

$\Leftrightarrow^{(H1)}$ $(\forall I)(\forall U)((\models_{(I,\tau)}\varphi[U]\Rightarrow\models_{(I,\tau)}\psi[U])$

and $(\models_{(I,\tau)}\varphi[U]\Rightarrow\models_{(I,\tau)}\psi[U]))$

$\Leftrightarrow (\forall I)(\forall U)((\models_{(I,\tau)}\varphi[U]\Leftrightarrow\models_{(I,\tau)}\psi[U]))$

(L1) $\neg(\models_{(i,\tau')} \tau:\varphi[u]) \Leftrightarrow \models_{(i,\tau')}\neg\tau:\varphi[u]$

$\Leftrightarrow \models_{(i,\tau')}(\tau:\neg\varphi)[u] \Leftrightarrow \models_{(i,\tau)}\neg\varphi[u]$

Proof : It can be proved very easily.

(L2) $\models_{(i,\tau')} \neg([\tau1,\tau2]:\varphi)[u]$

$\Leftrightarrow \models_{(i,\tau')} <\tau1,\tau2>:(\neg\varphi)[u]$

Proof : Immediately from (A34),(A32)

(L3) $\models_{(i,\tau')} \neg(<\tau1,\tau2>:\varphi)[u]$

$\Leftrightarrow \models_{(i,\tau')} [\tau1,\tau2]:(\neg\varphi)[u]$

Proof : Immediately from (A30),(A33).

A.IV Selected proofs concerning equivalencies

(TE1) $(\neg\tau{:}\varphi) \models\!|_{\tau} (\tau{:}\neg\varphi)$

$\Leftrightarrow$ (P1) $(\forall I)(\forall U)(\models_{(I,T)} \neg\tau{:}\varphi[U]$

$\Leftrightarrow \models_{(I,T)} \tau{:}\neg\varphi[U])$ which holds due to (L1).

(TE2) $\neg[\tau1,\tau2]{:}\varphi \models\!|_{\tau} <\tau1,\tau2>{:}\neg\varphi$

$\Leftrightarrow$ (P1) $(\forall I)(\forall U)(\models_{(I,\tau')} \neg[\tau1,\tau2]{:}\varphi[U]$

$\Leftrightarrow \models_{(I,\tau')} <\tau1,\tau2>{:}\neg\varphi[U])$ which holds due to (L2).

(TE3) $\neg<\tau1,\tau2>{:}\varphi \models\!|_{\tau} [\tau1,\tau2]{:}\neg\varphi$: Similar to the previous proof, by using (L3)

(TE4) $\neg[\tau1,\tau2]{:}\neg\varphi \models\!|_{\tau} (\neg(\neg<\tau1,\tau2>{:}\varphi))$

$\models\!|_{\tau} <\tau1,\tau2>{:}\varphi$

(TE5) $\neg<\tau1,\tau2>{:}\neg\varphi \models\!|_{\tau} (\neg(\neg[\tau1,\tau2]{:}\varphi))$

$\models\!|_{\tau} [\tau1,\tau2]{:}\varphi$

(TE6) $\tau{:}(\sigma{:}\varphi) \models\!|_{\tau'} \sigma{:}\varphi$, where τ,σ,φ are uncertain temporal intervals

According to (P1) $\tau{:}(\sigma{:}\varphi) \models\!|_{\tau'} \sigma{:}\varphi$

$\Leftrightarrow (\forall I)(\forall U)(\models_{(I,\tau')} \tau{:}(\sigma{:}\varphi)[U]$

$\Leftrightarrow \models_{(I,\tau')} \sigma{:}\varphi[U])$.

If we prove that $\models_{(i,\tau')} \tau{:}(\sigma{:}\varphi)[u]$

$\Leftrightarrow \models_{(i,\tau')} \sigma{:}\varphi[u]$ for some random i,u, then this expression holds for all i,u.

$\models_{(i,\tau')}$

$<[\tau_1,\tau_2],[\tau_3,\tau_4]>{:}(<[\sigma_1,\sigma_2],[\sigma_3,\sigma_4]>{:}\varphi)[u]$

$\Leftrightarrow$ (IS3) $\exists T'\in\{\tau_1...\tau_2\},\exists T''\in\{\tau_3...\tau_4\},$

$\forall T\in\{T'...T''\},\models_{(i,T)}(<[\sigma_1,\sigma_2],[\sigma_3,\sigma_4]>{:}\varphi)[u]$

$\Leftrightarrow$ (IS3) $\exists T'\in\{\tau_1,...,\tau_2\},\exists T''\in\{\tau_3,...,\tau_4\},$

$\forall T\in\{T',...,T''\}, (\exists\Sigma'\in\{\sigma_1...\sigma_2\},\exists\Sigma''\in\{\sigma_3...\sigma_4\},$

$\forall\Sigma\in\{\Sigma'...\Sigma''\},\models_{(i,\Sigma)} \varphi[u])$

but T,T',T'' no longer appear in the formula, therefore

$\Leftrightarrow (\exists\Sigma'\in\{\sigma_1...\sigma_2\},\exists\Sigma''\in\{\sigma_3...\sigma_4\},$

$\forall\Sigma\in\{\Sigma'...\Sigma''\},\models_{(i,\Sigma)} \varphi[u])$

$\Leftrightarrow \models_{(i,\tau')} \sigma{:}\varphi[u]$

A.V Selected proofs concerning consequences

(TC1) $<\tau,\sigma>{:}\varphi \vee <\tau,\sigma>{:}\psi \models_{\tau'} <\tau,\sigma>{:}(\varphi\vee\psi)$

$\Leftrightarrow$ (D2.4) $(\forall I)(\forall U)(\models_{(I,\tau')}(<\tau,\sigma>{:}\varphi \vee$

$<\tau,\sigma>{:}\psi)[U] \Rightarrow \models_{(I,\tau')} <\tau,\sigma>{:}(\varphi\vee\psi)[U])$

However, $\models_{(I,\tau')}(<\tau,\sigma>{:}\varphi \vee <\tau,\sigma>{:}\psi)[U]$

$\Leftrightarrow$(A23) $\models_{(I,\tau')}<\tau,\sigma>{:}\varphi[U]\vee\models_{(I,\tau')}<\tau,\sigma>{:}\psi[U]$

$\Rightarrow$(A21) $\models_{(I,\tau')}<\tau,\sigma>{:}(\varphi\vee\psi)[U]$, QED.

(TC2) $[\tau,\sigma]{:}(\varphi\wedge\psi) \models_{\tau'} [\tau,\sigma]{:}\varphi \wedge [\tau,\sigma]{:}\psi$

$\Leftrightarrow$ (D2.4)$(\forall I)(\forall U)(\models_{(I,\tau')}[\tau,\sigma]{:}(\varphi\wedge\psi)[U] \Rightarrow$

$\models_{(I,\tau')} ([\tau,\sigma]{:}\varphi \wedge [\tau,\sigma]{:}\psi)[U])$

However, $\models_{(I,\tau')}[\tau,\sigma]{:}(\varphi\wedge\psi))[U]$

$\Rightarrow$(A10) $\models_{(I,\tau')}[\tau,\sigma]{:}\varphi[U] \wedge \models_{(I,\tau')}[\tau,\sigma]{:}\psi[U]$

$\Leftrightarrow$(A12) $\models_{(I,\tau')} ([\tau,\sigma]{:}\varphi \wedge [\tau,\sigma]{:}\psi)[U]$, QED.

(TC3) $<\tau1,\sigma1>{:}\varphi \models_{\tau'} <\tau2,\sigma2>{:}\varphi,$

where $\{\tau2...\sigma2\}\subseteq\{\tau1...\sigma1\}$

$\Leftrightarrow$ (D2.4)$(\forall I)(\forall U)(\models_{(I,\tau')}<\tau1,\sigma1>{:}\varphi[U]$

$\Rightarrow\models_{(I,\tau')}<\tau2,\sigma2>{:}\varphi[U]),\{\tau2...\sigma2\}\subseteq\{\tau1...\sigma1\}$

However, $\models_{(i,\tau')}<\tau1,\sigma1>{:}\varphi[u]$

$\Leftrightarrow$ (IS3.b) $(\forall T\in\{\tau1...\sigma1\})\models_{(i,T)} \varphi[u]$

$\Rightarrow (\forall T\in\{\tau2...\sigma2\})\models_{(i,T)} \varphi[u],$

where $\{\tau2...\sigma2\}\subseteq\{\tau1...\sigma1\}$

$\Leftrightarrow$(IS3.b) $\models_{(i,\tau')}<\tau2,\sigma2>{:}\varphi[u]$, QED.

(TC4) $<\tau1,\sigma1>{:}\varphi \models_{\tau'} [\tau2,\sigma2]{:}\varphi,$

where $\{\tau2...\sigma2\}\cap\{\tau1...\sigma1\}\neq\varnothing$

$\Leftrightarrow$(D2.4)$(\forall I)(\forall U)(\models_{(I,\tau')}<\tau1,\sigma1>{:}\varphi[U]$

$\Rightarrow\models_{(I,\tau')}[\tau2,\sigma2]{:}\varphi[U]),$

where $\{\tau2...\sigma2\}\cap\{\tau1...\sigma1\}\neq\varnothing$

$\models_{(i,\tau')}<\tau1,\sigma1>{:}\varphi[u]$

$\Leftrightarrow$(IS3.b)$(\forall T\in\{\tau1...\sigma1\})\models_{(i,T)} \varphi[u],$

and since $\{\tau2...\sigma2\}\cap\{\tau1...\sigma1\}\neq\varnothing$

$\Rightarrow (\exists T\in\{\tau2...\sigma2\})\models_{(i,T)} \varphi[u]$

$\Leftrightarrow$(IS3.b) $\models_{(i,\tau')} [\tau2,\sigma2]{:}\varphi[u]$, QED.

(TC7) $<\tau,\sigma>{:}(\varphi\rightarrow\psi) \models_{\tau} <\tau,\sigma>{:}\varphi\rightarrow<\tau,\sigma>{:}\psi$

$\Leftrightarrow^{(D2.4)}$ $(\forall I)(\forall U)(\models_{(I,\tau)}<\tau,\sigma>:(\varphi\rightarrow\psi)[U]$

$\Rightarrow\models_{(I,\tau)}(<\tau,\sigma>:\varphi\rightarrow<\tau,\sigma>:\psi)[U])$

However, $\models_{(i,\tau)}<\tau1,\tau2>:(\varphi\rightarrow\psi)[u]$

$\Leftrightarrow^{(IS3.b)}(\forall T\in\{\tau1...\tau2\})\models_{(i,T)}(\neg\varphi\vee\psi)[u]$

$\Leftrightarrow^{(IS4)}$

$(\forall T\in\{\tau1...\tau2\})(\models_{(i,T)}(\neg\varphi)[u]\vee\models_{(i,T)}\psi[u])$

$\Leftrightarrow(\models_{(i,\tau1)}(\neg\varphi)[u]\vee\models_{(i,\tau1)}\psi[u])\wedge...\wedge$

$(\models_{(i,\tau2)}(\neg\varphi)[u]\vee\models_{(i,\tau2)}\psi[u])$

But the following $(a_1\vee b)\Rightarrow(a_1\vee a_2\vee...\vee a_n\vee b)$ obviously holds, therefore

$\Rightarrow(\models_{(i,\tau1)}(\neg\varphi)[u]\vee...\vee\models_{(i,\tau2)}(\neg\varphi)[u]\vee\models_{(i,\tau1)}\psi[u])$

$\wedge...\wedge$

$(\models_{(i,\tau1)}(\neg\varphi)[u]\vee...\vee\models_{(i,\tau2)}(\neg\varphi)[u]\vee\models_{(i,\tau2)}\psi[u])$

$\Leftrightarrow(\models_{(i,\tau)}[\tau1,\tau2]:(\neg\varphi)[u]\vee\models_{(i,\tau1)}\psi[u])\wedge...\wedge(\models_{(i,\tau')}<\tau1,\tau2>:(\neg\varphi)[u]\vee\models_{(i,\tau2)}\psi[u])$

$\Leftrightarrow\models_{(i,\tau)}[\tau1,\tau2]:(\neg\varphi)[u]\vee(\models_{(i,\tau1)}\psi[u]\wedge...\wedge\models_{(i,\tau2)}\psi[u])$

$\Leftrightarrow\models_{(i,\tau)}[\tau1,\tau2]:(\neg\varphi)[u]\vee\models_{(i,\tau')}<\tau1,\tau2>:\psi[u]$

$\Leftrightarrow$ (A37a, A33a)

$\models_{(i,\tau)}\neg<\tau1,\tau2>:\varphi[u]\vee\models_{(i,\tau)}<\tau1,\tau2>:\psi[u]$

$\Leftrightarrow\models_{(i,\tau)}(\neg<\tau1,\tau2>:\varphi[u]\vee<\tau1,\tau2>:\psi)[u]$

$\Leftrightarrow\models_{(i,\tau)}(<\tau1,\tau2>:\varphi\rightarrow<\tau1,\tau2>:\psi)[u]$, QED.

(TC8) $[\tau,\sigma]:\varphi\rightarrow[\tau,\sigma]:\psi\models_{\tau}[\tau,\sigma]:(\varphi\rightarrow\psi)$

$\Leftrightarrow^{(D2.4)}(\forall I)(\forall U)(\models_{(I,\tau)}([\tau,\sigma]:\varphi\rightarrow[\tau,\sigma]:\psi)[U]$

$\Rightarrow\models_{(I,\tau)}[\tau,\sigma]:(\varphi\rightarrow\psi)[U])$

However, $\models_{(i,\tau)}([\tau1,\tau2]:\varphi\rightarrow[\tau1,\tau2]:\psi)[u]$

$\Leftrightarrow\models_{(i,\tau)}(\neg[\tau1,\tau2]:\varphi[u]\vee[\tau1,\tau2]:\psi)[u]$

$\Leftrightarrow\models_{(i,\tau)}(\neg[\tau1,\tau2]:\varphi)[u]\vee\models_{(i,\tau)}[\tau1,\tau2]:\psi[u]$

$\Rightarrow$(A36a, A33a')

$\models_{(i,\tau)}([\tau1,\tau2]:(\neg\varphi))[u]\vee\models_{(i,\tau)}[\tau1,\tau2]:\psi[u])$

$\Leftrightarrow^{(A20)}\models_{(i,\tau)}([\tau1,\tau2]:(\neg\varphi\vee\psi)[u]$

$\Leftrightarrow\models_{(i,\tau)}([\tau1,\tau2]:(\varphi\rightarrow\psi)[u]$, QED.

References

[All83] Allen J.F., "Maintaining Knowledge about Temporal Intervals", Communications of ACM, Vol.26, No.11, pp.832-843, 1983.

[All84] Allen, J.F., "Towards a general theory of action and time", Artificial Intelligence, vol 23, pp.123-154, 1984.

[All85] Allen, J.F., Hayes P.J., "A Common Sense Theory of Time", Proceedings of the IJCAI-85 Conference, Los Angeles, CA, pp.528-531, 1985.

[All87] Allen, J.F., Hayes P.J., "Moments and points in an Interval-Based Temporal Logic", Technical Report 180, Departments of Computer Science and Philosophy, The University of Rochester, NY, 1987.

[All91] Allen, J.F., "Planning as Temporal Reasoning", Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, pp.3-14, 1991.

[Bar93] Barber F.A., "A Metric Time-Point and Duration-Based Temporal Model", SIGART Bulletin, Vol.4, No.3, pp.30-49, 1993.

[Bod93] Boddy M., "Temporal Reasoning for Planning and Scheduling", SIGART Bulletin, Vol.4, No.3, pp.17-20, 1993.

[Dea88] Dean T.L., Boddy M., "Reasoning about partially ordered events", Artificial Intelligence, No.36, pp.375-399, 1988.

[Dub89] Dubois D., Prade H., "Processing Fuzzy Temporal Knowledge", IEEE Transactions on Systems,Man and Cybernetics, Vol.19, No.4, pp.729-744, July/August 1989.

[Gal87a] Galton A., "Temporal Logic and Computer Science : An overview", in "Temporal Logics and their applications", Galton A., (Ed), pp.1-52, Academic Press, London, 1987.

[Gal87b] Galton A., "The Logic of Occurrence", in "Temporal Logics and their applications", Galton A., (Ed), pp.169-196, Academic Press, London, 1987.

[Gen87] Genesereth M.R., Nilsson N.J., "Logical Foundations of Artificial Intelligence", Morgan Kaufmann Publishers, Inc., Palo Alto, CA, USA, 1987.

[Ger93] Gerevini A., Schubert L., Schaeffer S., "Temporal reasoning in Timegraph I-II", SIGART Bulletin, Vol.4, No.3, pp.21-25, 1993.

[Hay87] Hayes P.J., Allen, J.F., "Short Time Periods", Proceedings of the IJCAI-87 Conference, pp.981-983, 1987.

[Kow86] Kowalski, R., Sergot, M., "A Logic-based calculus of events", New

Generation Computing, Vol.4, pp.67-95, 1986.

[McD82] McDermott D., "A Temporal Logic for Reasoning about Processes and Plans", Cognitive Science 6, pp.101-155, 1982.

[Ost89] Ostroff J.S., "Temporal Logic for real-time systems", Research Studies Press Ltd., John Wiley & Sons Inc., England, 1989.

[Ram88] Ramsay, A., "Formal Methods in Artificial Intelligence", Cambridge Tracts in Theotetical Computer Science 6, Cambridge University Press, Cambridge, England, 1988.

[Rei87] Reichgelt H., "Semantics for reified temporal logic", Advances in Artificial Intelligence, Hallam J., Mellish C., (Eds), pp.49-62, Wiley 1987.

[Sho87] Shoham, Y., "Temporal Logics in AI: Semantical and Ontological Considerations", Artificial Intelligence, Vol. 33, pp.89-104, 1987.

[Sho88] Shoham, Y., "Reasoning about change. Time and Causation from the Standpoint of Artificial Intelligence", MIT Press, Cambridge, Massachusetts, 1988.

[Sti93] Stillman J., Arthur R., Deitsch A., "Tachyon : A constraint-based temporal reasoning model and its implementation", SIGART Bulletin, Vol.4, No.3, pp.T1-T4, 1993.

Checking satisfiability of $TRIO_{\neq}$ specifications

Emanuele Ciapessoni (*), *Edoardo Corsetti, Ernani Crivelli* (+), *Manlio Migliorati*

CISE - Tecnologie Innovative, S.p.A.

Via Reggio Emilia, 39 - 20090 Segrate (Milano), Italy

(*) kant@sia.cise.it

(+) ENEL S.p.A. / CRA

Via Volta, 1 - Cologno Monzese (Milano), Italy

Abstract*

This paper gives a survey of the execution algorithm for $TRIO_{\neq}$, a temporal logic language to specify real-time systems, supporting the definition of metric and granular constraints. In order to verify the finite satisfiability of a $TRIO_{\neq}$ specification, the algorithm translates it in a first order language, and then proves the satisfiability of the translated formula. The satisfiability proof is performed by an extended analytic tableau in which propositional and predicative rules are coupled with rules to deal with time constraints. Such a kind of rules check the consistency of temporal relations (time constraints including accessibility relations and other temporal relations) held in the translated formula as a (finite) constraint solving problem. The algorithm increases the efficiency of a classical tableau because of, a priori, it reduces the search space for temporal variables.

1. Introduction

This paper presents the execution algorithm for $TRIO_{\neq}$ [Ciapessoni, 93], a layered and metric temporal logic language supporting the specification of *real time systems*. It allows to suitably specify and verify software requirements because of unambiguity, declarativity and formal executability of specifications [Manna,92], [Koymans,89]. $TRIO_{\neq}$ is a multi-modal temporal logic language including a metric temporal operator [Rescher,71], [Ghezzi, 90] and [Alur,93], to state quantitative constraints over a temporal domain.

Furthermore $TRIO_{\neq}$ includes temporal operators to deal with the specification of *granular systems*, that is those systems whose components have dynamic behaviours regulated by different time constants.

A temporal model to represent such a kind of systems, grounded on the temporal entity of time interval, was defined in [Allen,83]. Another temporal model based on a set-theoretic formalization of time is provided by Clifford and Rao [Clifford, 88], in which they define calendric universes, but they do not attempt to relate the truth value of assertions to time instants. The notion of time granularity may be thought as the temporal specialisation of the granularity notion provided in [Hobbes,85]. Hobbes defines a concept of granularity that supports the construction of simple theories out of more complex ones, introducing the basic notions of relevant predicate set, indistinguishability relations, simplification, idealisation and articulation. From the model theoretic point of view of temporal logic, Hayes and Allen [Hayes, 87] introduce a notion of time granularity to cope with short time periods. Galton and Shoham give a significant categorisation of assertions based on their temporal properties, but they do not take into account any notion of time granularity [Galton, 87], [Shoham, 88]. At last, [Montanari, 92] proposes an extension of the Event Calculus to deal with time granularity in a rather general way.

Time granularity proposal defined in [Ciapessoni, 93] is an evolution of previous ones [Corsetti, 91], where the logical language allowed to express but not to reason with time granularity. The last proposal extends the temporal domain of classical temporal logic with the notion of temporal universe: a set of temporal domains, each one referring to a given grain size, and in which three accessibility relations are defined. A set of constraints (hereinafter called Tg i.e. the theory of granularity) are imposed over accessibility relations in order to define calendric (or watch) temporal structure.

The execution algorithm provided with the language is grounded on the translation approach [Ohlbach, 91] because of its generality, in alternative to the non-translational one [Fitting, 83]. As is stated in [Ohlbach,93], the translation approach provides a greater flexibility because of it allows to set free the (definition of the) rules provided in the execution algorithm from set of properties included within Tg. Further, the algorithm includes an extended analytic

* The work was defined within the SPES project, currently ongoing among CISE, Politecnico di Milano and ENEL S.p.A/CRA and it is funded by the Centro Ricerche in Automatica (CRA) of the lectricity Board of Italy (ENEL S.p.A.), and partially by the National Research Council (CNR), within Piano Finalizzato Informatica e Calcolo Parallelo.

tableau [Smullyan, 68], in which beside the propositional and predicative rules, a number of rules are defined in order to tackle the proof of the semantic temporal relations, held in the formula, against Tg. Such a rules are specified so that the consistency of temporal relations is proved as a (finite) constraint satisfaction problem. In such a case the theory Tg is represented as a set of constraints, and the set of temporal relations are represented as the query we want to satisfy [Mackworth, 92].

The paper is organised as follow: section 2 introduces the main features of the language TRIO$_{\neq}$ and section 3 presents the execution algorithm, showing the definition of the translation and sketching the structure of both the tableau and the constraint temporal base.

2. TRIO$_{\neq}$

In this chapter we give a survey of the syntax and the semantics of TRIO$_{\neq}$; for a deep description we refer to [Ciapessoni, 93].

2.1. The syntax of TRIO≠

The alphabet of TRIO≠ includes sorts, variables, constants, functions, predicates and logical constants. All constant, function and predicate symbols are typed as well as variables. The set of sorts includes the sort S_D, called the temporal sort, which is numerical in nature and denotes the set of values of temporal displacements. Depending on the specified system, S_D can be either the set of integers, the set of rational numbers, the set of real numbers, or a subinterval of them.

The alphabet also includes a context sort S_C denoting the set of domains into which the temporal universe is partitioned.

The syntax includes the first order terms and formulae, and those terms and formulae referring to granularity. Apart from usual classical connectives TRIO$_{\neq}$ provides three temporal operators (together with their dual ones):

∇_α *displacement* $\nabla_\alpha\Phi$ ($\Delta_\alpha\Phi =_{def} \neg\nabla_\alpha\neg\Phi$) takes as arguments a formula and a temporal term denoting a distance: ∇_α displace the evaluation instant by the distance α.;

∇^A *context* $\nabla^A\Phi$ ($\Delta^A\Phi =_{def} \neg\nabla^A\neg\Phi$): takes as arguments a formula and a temporal term denoting a temporal domain (i.e., a *granularity*), and restricts the evaluation of Φ to time instants belonging to A;

$\Box$ *correspondence* $\Box\Phi$ ($\Diamond\Phi \equiv \neg\Box\neg\Phi$): takes as argument a formula. $\Box\Phi$ is evaluated to true iff Φ is true in any instant in correspondence with the current one.

The dual operator $\Diamond\Phi$ evaluates to true if Φ is true in at least one related instant.

2.2. The semantics of TRIO$_{\neq}$

As usual in the modal approach to temporal logic the semantics of TRIO$_{\neq}$ is defined according to a generalisation of the notion of *temporal structure* (*frame*).

Formally, a *generalised temporal structure* (the *modal frame of* TRIO$_{\neq}$) is given by:

$$F = \langle \mathfrak{T}, \mathfrak{D}, \mathfrak{C}, +, :, \rightarrow \rangle$$

where:

- $\mathfrak{T}$ is the temporal universe: a set of time instants, partitioned in a number of temporal domains ordered with respect their temporal grain
- $\mathfrak{D}$ is the distance domain over which distance terms are interpreted;
- $\mathfrak{C}$ is the set of domains of the temporal universe $\{T_1,..,T_n\}$ over which context terms are interpreted; temporal domains are totally ordered with respect to the binary granularity relation $\lhd$. Specifically $T_i \lhd T_j$ states that T_i is *coarser* than T_j, or T_j is *finer* than T_i.

Among time instants and temporal domains three accessibility relations are stated. The first one (+) expresses a *displacement* between each couple of instants of a temporal domain, the second one (:) allows to *contextualize* an instant with respect to a temporal domain and the third one ($\rightarrow$) puts into *correspondence* instants belonging to the temporal universe.

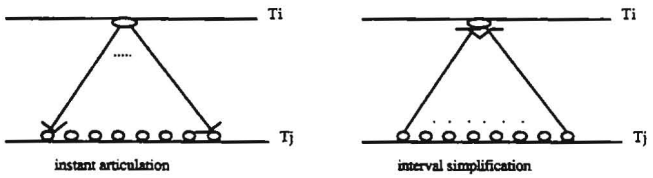
Displacement , correspondence and context accessibility relations are defined as follows:

- \+ is the displacement accessibility relation:
  $$+ : \mathfrak{T} \times \mathfrak{D} \times \mathfrak{T}$$
  +(i,α,j) means that the instant *j* is at a distance α from *i*
- : is the context accessibility relation:
  $$: : \mathfrak{T} \times \mathfrak{C}$$
  i:A means that the instant *i* belongs to the temporal domain *A*
- $\rightarrow$ is the correspondence accessibility relation:
  $$\rightarrow : \mathfrak{T} \times \mathfrak{T}$$
  i→j means that the instant *j* is in correspondence with *i*

In order to represent *granular temporal universe*, we require a number of properties to the relations included within the generalised temporal structures. Such a set of properties is expressed by the first order theory Tg.

In particular the *correspondence* between instants of each couple of temporal domains obeys to the *instant-articulation* and *interval-simplification* rules. Giving two temporal

domains such that $T_i \{ T_j$, each time instant of the coarser one is put into correspondence with an interval of time instants of the finer one, and vice versa:



Then we require that each temporal domain is linearly ordered, that temporal domains partitions temporal universe and that the correspondence preserves the order between instants and intervals between temporal domains.

Properties ***Tg*** of granular temporal universe have been axiomatized in the language, and the correspondence between axioms and properties is proved in [Corsetti, 94].

A *generalised temporal model* M=<F, $\Im$> is a *generalised temporal structure* in which an interpretation function $\Im$ is defined to set a value to terms and time granularity formulae.

First order interpretation rules are the usual ones (we assume rigid designators for variables and non-rigid designators for constants), let us formally define the semantical interpretation of each temporal operator (and its dual one):

$\Im_\iota(\nabla_\alpha\Phi) = true \Leftrightarrow \forall j(+(\iota,\Im_\iota(\alpha),j) \Rightarrow \Im_\iota(\Phi) = true)$

$\Im_\iota(\Delta_\alpha\Phi) = true \Leftrightarrow \exists j(+(\iota,\Im_\iota(\alpha),j) \;\&\; \Im_\iota(\Phi) = true)$

$\Im_\iota(\nabla^A\Phi) = true \Leftrightarrow \iota : \Im_\iota(A) \Rightarrow \Im_\iota(\Phi) = true$

$\Im_\iota(\Delta^A\Phi) = true \Leftrightarrow \iota : \Im_\iota(A) \;\&\; \Im_\iota(\Phi) = true$

$\Im_\iota(\Box\,\Phi) = true \Leftrightarrow \forall j(\iota \rightarrow j \Rightarrow \Im_\iota(\Phi) = true)$

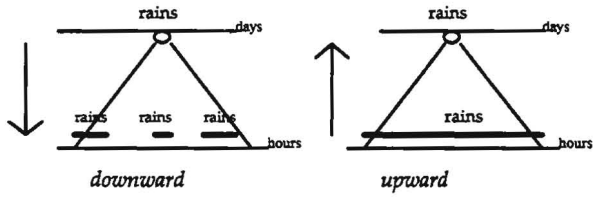
$\Im_\iota(\Diamond\,\Phi) = true \Leftrightarrow \exists j(\iota \rightarrow j \;\&\; \Im_\iota(\Phi) = true)$

where ι is the implicit interpretation instant.

In TRIO$_{\neq}$ the semantical correspondence between letterals asserted over time instants belonging to different domains is stated by *projection rules*.

Projections rules can be classify in *downward projection* (in case the correspondence is stated between a letteral true over an instant, and a letteral true over the instant articulations on finer domains), i.e., *theory articulation*, and *upward projection* (in case the correspondence is stated between a letteral that holds over an interval and a litteral that holds over any interval simplifications on coarser domains), i.e., *theory simplification*.

Downward projection can be specified, in a *punctual* way, stating that if an assertion holds over an instant, it holds at least over one instant of any articulation. Upward projection can be specified, in a *pervasive* way, stating that if an assertion holds over any instant of an interval then it holds over any interval simplification.



Semantically in TRIO$_{\neq}$, apart from granular temporal universe properties, time granularity models must satisfies the *punctual downward projection*[1], that is defined by the following semantical constraint:

$\forall \iota, A, B\ (A \{ B \;\&\; \iota{:}A \supset$
$(\Im_\iota(\Phi) = true \supset \exists j(\iota \rightarrow j \;\&\; j : B \;\&\; \Im_j(\Phi) = true))$

Such a constraint states that for each couple of temporal domains, if a formula Φ holds over an instant of the coarser temporal (A) then it holds over at least an instant of the finer one (B).

The notions of satisfiability and validity of formulae are the usual temporal logic ones, referring to the temporal structure defined above. But such notions can be also localised with respect to temporal domains. Local satisfiability of a formula with respect to a temporal domain means that there exists at least an instant of the temporal domain over which the formula is true; and local validity with respect to a temporal domain means that for each instant of the temporal domain the formula is true.

The language provides an axiomatic system to define both the temporal universe properties and the correspondence rules for first order formulae between temporal domains. The former set of axioms was defined in [Ciapessoni, 93] and its soundness checked in [Corsetti, 94]. The latter set includes several axioms stating the chosen correspondences between temporal domains.

3. The execution algorithm

The execution algorithm of TRIO$_{\neq}$ is based on two steps:

1) the translation of the TRIO$_{\neq}$ formula into a first order one;
2) the proof of the translated formula w.r.t. the theory ***Tg*** performed by an extended first order tableau;

The translation function produces a first order formula in which the extra-logical predicate symbols of TRIO$_{\neq}$ (hereinafter *S_atoms*) and the predicates denoting the accessibility relations of TRIO$_{\neq}$ temporal structure (*C_atoms*) are included.

---

[1]the *pervasive upward projection* is logically equivalent to *punctual downward one*.

The classical first order tableau is extended in order to reduce the computability effort to check the satisfiability of the C_atoms against the theory ***Tg*** by means a constraint solver algorithm. Indeed the theory ***Tg*** is the first order temporal theory with respect to the proofs are performed and the C_atoms are standard by translation function. The C_atom satisfiability problem can be represented as a constraint satisfaction problem (CSP) regarding the theory ***Tg*** as a set of *constraints* and the set of C_atoms as a *query*. Among the CSP techniques we took into account the ones grounded on a priori pruning (i.e., consistency technique), because they reveal a better performance with respect to the ones adopting the a posteriori pruning (i.e., generate and test and standard backtracking) [van Hentenrick,89].

In the following we sketch the translation function and give a description of the classical and the extended tableau. The former includes the classical propositional rules and a refined version of the predicative ones, in order to improve their efficiency. The latter one provides a number of rules in order to deal with the satisfiability proof of temporal relations against ***Tg*** and a rule to check the temporal consistency of the set of TRIO≠ logical symbols. Both the classical and the extended tableau rules are specified by means of a transition system with structured nodes <L,P,V> referring to the current path: where L is the set of currently active formulae, P is the set of used parameters and V is the set of temporal constraints.

3.1 The translation

To make explicit temporal information a TRIO≠ specification is firstly translated from the modal level to the first-order one. The translation function τ is defined by structural induction with respect to the evaluation instant, left implicit in the original formula.

So, for instance, predicates are translated adding a temporal argument:

$$\tau_i(p(t_1,..t_n)) \Rightarrow p(\tau_i(t_1),..,\tau_i(t_n), i)$$

and temporal operators are translated on the base of their semantic:

$$\tau_i(\nabla_\alpha \phi) \Rightarrow \forall j(+(i, \tau_i(\alpha), j) \supset \tau_j(\phi))$$

$$\tau_i(\nabla^A \phi) \Rightarrow i : \tau_i(A) \supset \tau_i(\phi)$$

$$\tau_i(\Box \phi) \Rightarrow \forall j(\rightarrow(i, j) \supset \tau_j(\phi))$$

The function τ produces formulae in which are included the $TRIO_{\neq}$ extra logical symbols (i.e., the above predicate p, referred to as *S_atom*) and a number of predicates which are the $TRIO_{\neq}$ semantical accessibility relations (i.e., the above predicates: +, : and $\rightarrow$, referred to as *C_atoms*).

As the function τ is sound and complete, semantical results gained over any $TRIO_{\neq}$ formula are logically equivalent to those gained with the conjunction between the formula translation and the theory ***Tg*** (soundness and completeness proofs of τ are given in [Scapellato,93]).

3.2 The classical tableau

The analytic tableau works on the classical part of the first order formula. Apart from the classical propositional rules [Smullyan,68] the algorithm includes a revision of the classical γ-rules and δ-rule. The former is specified making explicit the set of parameters with respect to the rule has been already applied. The latter has been splitted in two rules in order to check the finite satisfiability.

- γ-rule:

$$\frac{(\forall^Q x.\phi) \in L, a \in P \setminus Q, L' = L \setminus \{(\forall^Q x.\phi)\}}{\langle L, P, V\rangle \rightarrow \langle L' \cup \{\phi(x/a), \forall^{Q\cup\{a\}} x.\phi\}, P, V\rangle}$$

  the label Q, associated with the universal quantifier, denotes the list of parameters with respect to the universal quantified formula has been already applied; the rule is applied whenever there exists a parameter *a* not included in Q;

- δ-rule: the δ rules couples the classical δ_1 rule:

$$\frac{(\exists x.\phi) \in L, a \notin P, L' = L \setminus \{(\exists x.\phi)\}}{\langle L, P, V\rangle \rightarrow \langle L' \cup \phi(x/a), P, V\rangle}$$

  with a δ_2 rule that instantiates the variable x with an old parameter in Φ; in order to gain decidability of satisfiability proof for $\forall\exists$ of formulae:

$$\frac{(\exists x.\phi) \in L, a \in P, L' = L \setminus \{(\exists x.\phi)\}}{\langle L, P, V\rangle \rightarrow \langle L' \cup \phi(x/a), P, V\rangle}$$

3.3 The extended tableau

The extended tableau provides a number of rules in order to draw out the C_atoms from the translated formula (later than the quantifiers bounding its variables have been tackle by the appropriate rules), and to check their satisfiability against ***Tg*** by the constraint solver algorithm. Let us first of all sketch the C_atom satisfiability problem, and then define the extended tableau rules.

the C_atom satisfiability problem

- the $TRIO_{\neq}$ temporal structure is represented by a number of finite domains:
  - the temporal universe T; (𝔗)
  - the set of temporal domains: T1, .. ,Tn; (𝔊)
  - the domain of distances: T_D; (𝔇)
- the theory ***Tg*** is represented by a set of *constraints*, whose variables range over the finite domains defined above:
  - $C_1, C_2, .. , C_k$

  (where k is the number of properties held within ***Tg***);

- each C_atom has the form (*query*):
  - $c(X_1, X_2, .., X_j)$

  (the variables range over the finite domains and are existentially quantified);
- the satisfiability problem for a set of z-C_atoms can be expressed as follows:

$$C_1, C_2, .., C_k \vdash \exists X_1, X_2, .., X_w \bigwedge_{i=1}^{z} c_i(X_1, X_2, .., X_w)$$

  - where $\exists$ denotes the existential closure of the set of variables $X_1, X_2, .., X_w$;

Thus, the proof can be performed according to an algorithm grounded on *consistency techniques*. Such a kind of constraint solver algorithm prunes the search space using the constraints before a failure is discovered. Whereas, standard backtracking or generate and test strategies reduce the search space later than a failure is discovered. The different strategies between the two techniques raised because the latter one is oriented to recover from failures, while consistency technique is oriented to avoid failures. Moreover, the restriction over finite domains previously provided to consistency techniques is not compulsory. As a matter of fact consistency techniques can be defined independently from the domain cardinality [Le Provost,92].

The extended tableau rules described below refer to the C_atom satisfiability problem with the notation:

$$\mathrm{SAT}_{\mathcal{T}g}\{\mathfrak{Q}\}$$

where $\mathfrak{Q}$ denotes the set of queries, $\mathcal{T}g$ the set of constraints specified over the finite domains and SAT denotes the derivation symbol of the query existential closure from the set of constraints; the notation $\mathrm{UNSAT}_{\mathcal{T}g}\{\mathfrak{Q}\}$ denotes the complementary evaluation result.

the extended tableau rules

- *drawing out C_atoms*: this rule draws out from the set of formulae L those literals α which derives from C_atoms, and puts them in the constraint solver *query*:

$$\frac{\alpha \in L, \quad L' = L \setminus \{\alpha\}}{\langle L,P,V\rangle \rightarrow \langle L',P,V \cup \{\alpha\}\rangle}$$

- *temporal consistency S_atoms*: each couple of potential contradictory literals held within the set of formulae L impose as a new constraint that the temporal instants they refer to must be different:

$$\frac{p(u_1,..,u_n,t) \in L, \quad \neg p(u_1,..,u_n,t') \in L}{\langle L,P,V\rangle \rightarrow \langle L,P,V \cup \{t \neq t'\}\rangle}$$

- *exhaustive rules for universal quantifiers*: because of the satisfiability proof for the universally quantified formulae, whose variables denote instants or distances, must be performed with respect to any domain element a rule must be added to the tableau set. The consequences of such a rule are similar to those of the universal quantifier one, and it is applied whenever the two lists Q (associated to a given universal quantifier) and P share the same elements, and it is possible to find out a new parameter j denoting a value different from the previous parameters in Q (if $a_1, .. a_n$ are the parameters held in Q, let us denote with *new(j)* the following disequalities: $j \neq a_1, j \neq a_2, .., j \neq a_n$.)

  Furthermore in case, it is possible to recognise and gather the set of constraints with respect to the quantified variable is subject within the formula, the parameter must be a new parameter and satisfy this set of constraints. This means that the satisfiability proofs for the universal quantified formula could be performed with respect to those domain elements that satisfy the constraints (in general a subset of the domain).

  Let us consider two version of exhaustive rules: the first one (R∀1) recognises the set of constraints applied to the universally quantified variables and the last one (R∀2) that does not, respectively.

(R∀1)

$$\frac{\left(\forall^{P}_{t.v(t)\supset\phi}\right)\in L, j\notin P, SAT_{\mathcal{T}g}\left(V\cup\left\{new(j), v\left(t/j\right)\right\}\right), L'=L\setminus\left\{\left(\forall^{P}_{t.v(t)\supset\phi}\right)\right\}}{\langle L,P,V\rangle\rightarrow\left\langle L'\cup\left\{\phi\left(t/j\right), \forall^{Q\cup\{j\}}_{t.v(t)\supset\phi}\right\}, P\cup\{j\}, V\cup\left\{new(j), v\left(t/j\right)\right\}\right\rangle}$$

(R∀2)

$$\frac{\left(\forall^{P}_{t.\phi}\right)\in L, j\notin P, SAT_{\mathcal{T}g}\left(V\cup\{new(j))\}\right), L'=L\setminus\left\{\left(\forall^{P}_{t.\supset\phi}\right)\right\}}{\langle L,P,V\rangle\rightarrow\left\langle L'\cup\left\{\phi\left(t/j\right), \forall^{Q\cup\{j\}}_{t.\supset\phi}\right\}, P\cup\{j\}, V\cup\{new(j)\}\right\rangle}$$

  When the applicability conditions of the previous rules fails, the universally quantified formula can be delete from the list L of formulae.

- *the downward projection rule*: as the projection rules are logically equivalent, we define just the downward projection rule, in order to cope with the literals projection between temporal domains. The rule can be applied whenever the set of formulae held a couple of contradictory literals, if they are asserted over two different temporal domains such that one is coarser grain than the other:

$$\frac{(p(t),\neg p(t'))\in L, SAT_{\mathcal{T}g}(V\cup\{t{:}A,t'{:}B,A\prec B,\rightarrow(t,t')\})}{\langle L,P,V\rangle\rightarrow\left\langle L'\cup\{\exists t''{:}B\wedge\rightarrow(t,t'')\wedge p(t'')\}, P, V\cup\{t{:}A,t'{:}B,A\prec B,\rightarrow(t,t')\}\right\rangle}$$

- *path closure rule*: whenever the constraints evaluation fails, the current tableau path is closed. Such a rule provides both the closure rule for the constraints evaluation and the tableau closure rule.

$$\frac{UNSAT_{\mathcal{T}g}(V)}{\langle L,P,V\rangle\rightarrow\langle\perp,P,V\rangle}$$

As usual, the satisfiability proof ends successfully when an open and not more expandable path is found, and the confutability proof ends when every branch in the tree is closed.

4. Conclusions and future works

The effectiveness of a specification language is strongly increased by the availability of a rich and integrated environment of tools. Such tools should allow not only the editing and the managing of specification, but even their execution, with the purpose of early prototyping and verification.

In this paper we sketched the execution algorithm we defined to check satisfiability of TRIO≠ specifications pointing out its main features. Such an algorithm, based on an extension of the proof method of semantic tableaux, allows us to prove the *finite satisfiability* of a formula. The algorithm can be used to verify the consistency of specifications and to perform both simulation and verification of temporal models with respect to a given specification. It also allows to prove any property of the system that can be derived from its specification by verifying if the conjunction of the specification and the negation of the property is unsatisfiable.

We are currently implementing a prototype of an interpreter built on the specified algorithm. This interpreter will be used to performs proofs of consistency (satisfiability) and adequacy (analyses of models) of specifications.

In the future we will improve the execution algorithm with respect to the normal form produced by the translation phase. Next step will be the definition of intensional temporal domains in the constraint solver, improving the generality and flexibility of the interpreter.

Acknowledgements

We would like to thank Giovanna Dondossola, Paolo Mancarella and Dino Pedreschi for the contributions they gave to the argument.

References

[Allen,83] *Maintaining Knowledge about Temporal Intervals*, J.Allen, Communications of the ACM, 26,11,1983.

[Alur,93] *Real-Time Logics: Complexity and Expressiveness* R. Alur, T. Henzinger, Information and Computation, 104, 35-77, 1993.

[Ciapessoni,93] *Embedding time granularity in a logical specification language for synchronous real-time systems*, Ciapessoni E., Corsetti E., Montanari A., Ratto E., San Pietro P., Science of Computer Programming 20 (1993) 141-171.

[Clifford,88] A *simple, general structure for Temporal Domain*, Clifford J., Rao A., in Temporal Aspects in Information Systems, Rolland C., Bodart F., and Leonard, M. (Editors), Elsevier Science Publishers B.V. (North Holland), 1988.

[Corsetti,91] *Dealing with Different Time Granularities in Formal Specification of Real-Time systems*, Corsetti E., Montanari A., Ratto E., The Journal of Real-Time Systems, Vol III, Issue 2, 1991.

[Corsetti,94] *Logical Specification of Real-Time Granular Systems in the Synchronous Case*. Corsetti E., Ciapessoni E., CISE Internal Report 1994.

[Fitting,83] *Proof methods for Modal and Intuitionistic Logic*, Fitting M., D. Reidel Publishing Co., 1983.

[Galton,87] *The logic of Occurence*, A.Galton, in Temporal Logics and their Applications, Academic Press, 1987.

[Ghezzi,90] *TRIO: a Logic Language for Executable specifications of Real-Time Systems*, Ghezzi C., Mandrioli D., Morzenti A., The Journal of Systems and Software, June-July 1990.

[Hayes,87] *Short Time Periods*, P.Hayes, J.Allen, in Proc. 10th IJCAI, Milan, 1987.

[Hobbs,85] *Granularity*, J.Hobbs, in Proc. 9th IJCAI, Los Angeles (USA), 1985.

[Koymans,89] *Specifying Message Passing and Time-Critical Systems wth Temporal Logic*, Koymans R., PhD. Dissertesion, 1989.

[Koymans,90] *Specifying Real-Time Properties with Metric Temporal Logic*, R.Koymans, The Journal of Real_time Systems, vol. II, 1990.

[LePrevost,92] *Domain Independent Propagation*, T. Le Provost, M. Wallace, in Proceedings of FGCS-92, pp 1004-1011.

[Mackworth,92] *The Logic of Constraint Satisfaction*, A.K. Mackworth, in Artificial Intelligence, 58 (1992) pp. 3-20.

[Manna,92] *The Temporal Logic of Reactive and Concurrent Systems*, Z.Manna, A.Pnueli, Springer-Verlag, 1992.

[Montanari,92] *Dealing with Time Granularity in the Event Calculus*, A.Montanari, E.Maim, E.Ciapessoni, E.Ratto, Proc. Int. Conf. on 5th

[Ohlbach,91] *Semantics Based translation methods for modal logics*, Ohlbach H. J., in Journal of Logic and Computation, 1 (5):691-746, 1991.

[Ohlbach,93] *Translation Methods for Non-Classical Logics: An Overview*, H.J. Ohlbach, in Bulletin of the IGPL, Vol. 1, No. 1, pp. 69-89, 1993.

[Rescher,71] *Temporal Logic*, Rescher N., Urquhart A., Springer-Verlag, Wien-New-York, 1971.

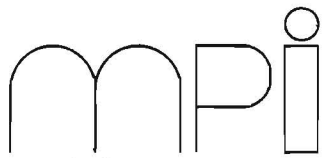
[Shoham,88] *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*, Shoham Y., MIT Press, 1988.

[Scapellato,93] *Theorem prover Grounded on Tableaux Methods for a Layered and Metric Temporal Logic*, Scapellato G., Thesis in Computer Science, Università di Pisa, 1993, (in italian).

[Smullyan,68] *First Order Logic*, Smullyan R. M., Springer Verlag, 1968.

[Van Hentenryck,89] *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, MA, 1989.

Authors Index



Below you find a list of the most recent technical reports of the research group *Logic of Programming* at the Max-Planck-Institut für Informatik. They are available by anonymous ftp from our ftp server `ftp.mpi-sb.mpg.de` under the directory pub/papers/reports. If you have any questions concerning ftp access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Regina Kraemer
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `kraemer@mpi-sb.mpg.de`

MPI-I-94-226	H. J. Ohlbach, D. Gabbay, D. Plaisted	Killer Transformations
MPI-I-94-225	H. J. Ohlbach	Synthesizing Semantics for Extensions of Propositional Logic
MPI-I-94-224	H. Aït-Kaci, M. Hanus, J. J. M. Navarro	Integration of Declarative Paradigms Proceedings of the ICLP'94 Post-Conference Workshop Santa Margherita Ligure, Italy
MPI-I-94-223	D. M. Gabbay	LDS – Labelled Deductive Systems Volume 1 — Foundations
MPI-I-94-218	D. A. Basin	Logic Frameworks for Logic Programs
MPI-I-94-216	P. Barth	Linear 0-1 Inequalities and Extended Clauses
MPI-I-94-209	D. A. Basin, T. Walsh	Termination Orderings for Rippling
MPI-I-94-208	M. Jäger	A probabilistic extension of terminological logics
MPI-I-94-207	A. Bockmayr	Cutting planes in constraint logic programming
MPI-I-94-201	M. Hanus	The Integration of Functions into Logic Programming: A Survey
MPI-I-93-267	L. Bachmair, H. Ganzinger	Associative-Commutative Superposition
MPI-I-93-265	W. Charatonik, L. Pacholski	Negativ set constraints: an easy proof of decidability
MPI-I-93-264	Y. Dimopoulos, A. Torres	Graph theoretical structures in logic programs and default theories
MPI-I-93-260	D. Cvetković	The logic of preference and decision supporting systems
MPI-I-93-257	J. Stuber	Computing Stable Models by Program Transformation
MPI-I-93-256	P. Johann, R. Socher	Solving simplifications ordering constraints
MPI-I-93-250	L. Bachmair, H. Ganzinger	Ordered Chaining for Total Orderings
MPI-I-93-249	L. Bachmair, H. Ganzinger	Rewrite Techniques for Transitive Relations
MPI-I-93-243	S. Antoy, R. Echahed, M. Hanus	A needed narrowing strategy
MPI-I-93-237	R. Socher-Ambrosius	A Refined Version of General E-Unification