A Complete Axiomatization of Interval Temporal Logic with Projection *

Howard Bowman and Simon Thompson

Computing Laboratory,
University of Kent at Canterbury,
Canterbury, Kent, CT2 7NF, United Kingdom
E-mail: {H.Bowman,S.J.Thompson}@ukc.ac.uk

Abstract

This paper presents a complete axiomatisation for propositional interval temporal logic (PITL) with projection. The axiomatisation is based on a tableau procedure for the logic, which in turn is founded upon a normal form for PITL formluae, first introduced in [6].

The construction of the axiomatisation is modular, in the sense that given a normal form for a new connective, axioms can be generated for the connective from the tableau construction. The paper concludes with a discussion of aspects of compositionality for PITL with projection.

1 Introduction

Interval Temporal Logics (ITL) form an important class of temporal logics. Early work on the topic was performed by Moskowski [20] with a number of researchers progressing the topic since then, such as Hale [14], Kono [15], Duan [10], Cau et al [9], Bowman et al [7] and Thompson [26].

Standard temporal logics are defined over models in which time extends infinitely into the future; for example, the models for Manna and Pnueli Linear Time Temporal Logic [18] are infinite state sequences. However, in interval temporal logic the model theory is restricted to finite state sequences, called *intervals*.

There are a number of reasons for being interested in such logics. One reason is that interval temporal logic lends itself to execution. This is apparent from Moskowski's work [20] and its descendants. In addition, a number of interesting and powerful operators arise naturally from ITL. In fact, it is straightforward to derive operators very like the constructs of imperative programming

[0]Travel grants to support the research presented here have been provided by the British Council. During the time that the research presented in this paper was being undertaken the first author was an EU Marie Curie Fellow at Verimag in Grenoble, France.

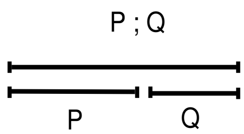


Figure 1: The Chop Operator

(e.g. assignment, conditionals, iteration etc). This then yields the possibility that abstract specifications and concrete implementations can be realised in the same notation, with refinement mappings between being represented by logical implications.

An additional aspect of interval temporal logic is that it provides a very simple real-time model in which one unit of time is passed when moving from state to state. Consequently, timings can be obtained by measuring interval lengths. The ITL operator $\mathbf{len}(n)$ is used for this purpose. This operator is satisfied by any interval with $n+1$ states (transitions between states are counted rather than numbers of states).

Two operators which are characteristic of interval temporal logic are the Chop operator **;** and the projection operator **proj**. The former of these implements a form of sequential composition; an interval will satisfy the formula

$$P \;\mathbf{;}\; Q$$

if it can be divided into two contiguous sub-intervals such that P holds over the first sub-interval and Q holds over the second. The operator is illustrated in figure 1, where line segments depict intervals.

In contrast, the projection operator yields repetitive behaviour; an interval satisfies the formula

$$P \;\mathbf{proj}\; Q$$

if it can be sub-divided into a series of sub-intervals, each of which satisfies P - we call P *the projection formula* - and the new interval formed from the end points of these sub-intervals satisfies Q, which we call *the projected formula*. The operator is illustrated in figure 2, where notice that the interval that Q holds over is not depicted as a line segment, rather it is the concatenation of the depicted sequence of points (each of which represents a state).

The value of the Chop operator should be self evident, however, the motivation behind projection will perhaps be clarified by some examples. An important use of projection is in deriving iteration constructs. For example, *for* loops and *while* loops can be derived using projection. The *for* loop is defined as:

$$\mathbf{for}\ n\ \mathbf{times\ do}\ P \;\equiv\; P \;\;\mathbf{proj}\;\; \mathbf{len}(n)$$

Notice how $\mathbf{len}(n)$ is used to count the number of iterations, by counting endpoints of sub-intervals. In contrast, the *while* loop is defined as:

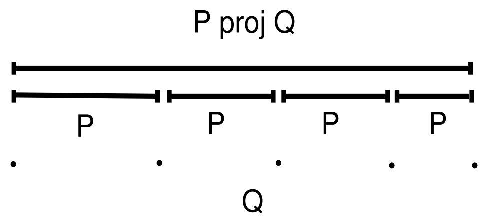


Figure 2: The Projection Operator

$$\textbf{while } P \textbf{ do } Q \;\equiv\; (P \;\wedge\; Q)^* \;\wedge\; \Box(\textbf{len}(0) \Rightarrow \neg P)$$

$\Box$ is the ITL operator 'henceforth', $\Box P$ holds over an interval in which P holds over all suffixes of the interval, and R^* is the star operator which gives arbitrary repetition of R; it is defined as:

$$R^* \;\equiv\; R \textbf{ proj True}$$

In the *while* loop P will typically be a *point formula*, satisfaction of which is determined totally from the first state of the satisfying interval. For example, an atomic proposition, p say, when lifted to the interval level, will hold over any interval in which p is true at the first state of the interval. Point formulas are called *local formulas* in some work [15].

It should also be pointed out that projection has proved a valuable operator in the real-time setting where it can be used to realise temporal abstraction [19] and hence, for example, it can describe speeding up or slowing down real-time presentations [7]. In addition, it has recently been found to be useful for describing goals for cognitive behaviour in the domain of multi-modal human computer interaction [5, 4].

A further reason for being interested in logics containing chop and repetition operators (such as projection) is that they address the problem of the limited expressiveness of classical point-based temporal logics such as linear [18] and branching time [12] temporal logics. It is well known, see e.g. [27], that propositional temporal logic is expressively limited. In fact, propositional temporal logics are only as expressive as F1S (the first order theory of linear orders). In addition, logics with chop, such as Rosner and Pnueli's Choppy logic [25], are similarly expressive [23]. However, addition of the star operator brings the expressive power of S1S (the second order theory of linear orders) [23], i.e. ω-regular languages. Furthermore, as suggested above, projection is at least as expressive as star.

In fact, we cannot claim F1S expressiveness for our logic as we are dealing with finite models. Thus, our logic is expressively incomparable with propositional temporal logic. However, extension of our work (in the style of say [10]) to infinite sequences would yield this result.

Axiomatization in the standard (infinite) temporal logic setting has been extensively investigated and completeness results exist in both the linear time, e.g. [17] and branching time, e.g. [12] case; pointers to these basic completeness results can be found in the bibliographic notes associated with [18]. In addition, a number of axiomatizations have been defined for interval temporal logic, e.g. Moskowski [21], Kono [15], Duan [10], Moskowski [22], but the literature contains no examples of consistent and complete axiomatizations for interval temporal logics with projection.

Of the (ITL) axiomatizations that have been given, the majority do not claim completeness, e.g. [10] makes no such claim. Kono though does claim completeness in an unpublished technical report [16], but one of his rules is unsound. In addition, in [21] Moskowski conjectures completeness but does not provide a full justification for the claim.

The reason for this disparity between the standard temporal logic case and interval temporal logics is that ITL operators are in many senses more difficult to deal with. In particular, inductive definitions of many operators of standard temporal logic are straightforward. For example, the 'until' operator $\mathcal{U}$ is defined to be the least solution of the equation

$$P \,\mathcal{U}\, Q \;\equiv\; Q \;\vee\; (P \;\wedge\; \bigcirc(P \,\mathcal{U}\, Q))$$

In contrast, inductive definitions of chop and projection are inherently more complex. In particular, with chop an inductive definition must cope with its first argument evolving when the operator is unfolded. This issue will become clear when we present our normal forms for chop in the body of the paper.

Research work that is closest to ours is that by Rosner and Pnueli [25], Dutertre [11] and Paech [23]. The logic considered in [25] is a version of standard (infinite) temporal logic which includes the chop operator and they do give a complete axiomatization. However, although this work gives a number of relevant insights it is rather complex and does not handle the projection operator. Dutertre [11] also has a complete axiomatization for a logic based around chop but, like Rosner and Pnueli, he does not consider projection. In some respects Paech's [23] results are the most impressive - she has a complete (Gentzen style) proof system for a branching time logic with chop and the star operator. But, once again projection is not considered.

Thus, one novel element of our work is the identification of completeness results in the presence of the projection operator, which is at least as expressive as star. In addition, central to our strategy is the identification of normal forms for all the operators of our logic. In effect, these normal forms give inductive definitions of the ITL operators in a uniform style. Furthermore because of our emphasis on normal forms (and in contrast to a number of the other approaches just mentioned) we are able to define a tableau decision procedure. This decision procedure is in the style of Wolper [27] and it checks satisfiability of our logic. We relate the tableau decision algorithm to our axiomatisation in order to prove completeness.

An important aspect of our approach is its *modularity*: if formulas containing another operator, such as the **filter** of [7], can be placed in normal form, then that operator is amenable to treatment in exactly the same way as the operators given here. In particular therefore, the decidability and axiomatisation results can be extended to a logic augmented with that operator.

A final advantage of the normal form approach is that it suggests an executable interpretation of the logic. Such interpretations are well known from the Tempura [20] and METATEM [1] work. For simplicity of presentation we work in the propositional setting, but we do not see any fundamental difficulties in extrapolating our theory to the first order case and to obtain "relative" completeness results in the standard fashion.

Structure of the Paper. First, in section 2 we review the basic interval temporal logic approach. Then in section 3 we introduce our normal form, which is at the heart of our strategy. In section 4 we present our tableau decision procedure and in section 5 we prove that the tableau is correct, i.e. that it verifies satisfiability. Then section 6 presents our axiomatization and section 7 verifies that it is indeed a complete axiomatization. Finally, section **??** presents some concluding remarks.

Acknowledgements. We are grateful to Ben Moszkowski, of de Montfort University, for discussions about our work and others' attempts at axiomatisation. This work came out of joint work with Helen Cameron and Peter King of the University of Manitoba on applications of formal methods such as PITL to multimedia specification and design; their collaboration is much appreciated.

2 Interval Temporal Logic

The interval temporal logic that we use is defined in the following subsections. In the context of this paper we will call this logic PITL, for Propositional Interval Temporal Logic. Our approach here is the standard one; we use this section to establish terminology and also to clarify some technical details of the semantics.

We begin by defining intervals, which will give the semantic models for our logic.

2.1 Intervals

PITL is defined over finite state sequences. Each sequence is called an interval and $\mathcal{I}$ denotes the set of all possible intervals; $\sigma \in \mathcal{I}$ has the form:

$$\sigma_0, \sigma_1, ..., \sigma_{|\sigma|}$$

where $|\sigma|$ denotes the length of an interval and σ_i denotes the ith state in an interval. Each state is a set containing all the atomic propositions that are true at that state.

By convention the length of an interval is the number of states minus one and all intervals must have at least one state. We use $[\sigma]^i$ to denote the ith prefix of an interval and $(\sigma)^i$ to denote the ith suffix of an interval. Formally,

$$[\sigma]^i = \sigma_0, ..., \sigma_i$$

$$(\sigma)^i = \sigma_i, ..., \sigma_{|\sigma|}$$

2.2 The Logic

The domain of logical propositions is denoted $\mathcal{P}$ and $P \in \mathcal{P}$ is constructed as follows,

$$P \quad ::= \quad p \mid \textbf{False} \mid \neg P \mid P \ \vee \ P \mid \textbf{empty} \mid \ \bigcirc P \mid P \textbf{ ; } P \mid P \textbf{ proj } P$$

Much of this logic will be well known to a reader familiar with interval temporal logic.

- **False**, $\neg$ and $\vee$ are the usual connectives of classical propositional logic.
- p is chosen from a set of atomic propositions.
- **empty** holds over any interval of length zero, i.e. which has one state.
- $\bigcirc$ is the (strong) next operator. Thus, $\bigcirc P$ holds if and only if P holds over an interval of length one less than the current interval, resulting from moving one state into the future. In particular, $\bigcirc P$ is **False** on an **empty** interval.
- **;** and **proj** are as described in the introduction.

The reader who requires a more detailed discussion of these operators is referred to [20].

Also, we have the following standard derived operators. A wealth of other operators can be defined, see for example [8].

$$\begin{array}{rcll}
P \Rightarrow Q & \equiv & \neg P \ \vee Q & \\
P \ \wedge \ Q & \equiv & \neg(\neg P \ \vee \ \neg Q) & \\
\textbf{len}(0) & \equiv & \textbf{empty} & \\
\textbf{len}(n+1) & \equiv & \bigcirc \textbf{len}(n) & \text{where } n \in \mathcal{N} \\
ⓦ P & \equiv & \bigcirc P \ \vee \ \textbf{empty} & \\
\Diamond P & \equiv & \textbf{True ; } P & \\
\Box P & \equiv & \neg \Diamond \neg P &
\end{array}$$

Thus, in addition, to the standard logical derivations, we have an operator to measure the length of an interval, **len**; a weak next operator, ⓦ; eventually, $\Diamond$, and henceforth, $\Box$.

2.2.1 Interpreting PITL

Our satisfaction relation, $\models$, interprets PITL formulae over intervals. The notation $\sigma \models P$ denotes that the interval σ satisfies (or models) the formula P.

We interpret PITL propositions in the usual way, by induction over the structure of propositions. We consider the satisfaction rule for each class of construct in turn.

Non-temporal Operators.

Atomic propositions are interpreted as follows:

$$\sigma \models p \text{ iff } p \in \sigma_0$$

i.e. a proposition p holds over any interval in which p holds at the first state of the interval. In this way satisfaction of atomic propositions is lifted to the interval level. In propositional interval temporal logic atomic propositions are the basic source of point formula.

The following rules reflect the standard interpretation of propositional logic,

$$\sigma \not\models \textbf{False}$$

$$\sigma \models \neg P \text{ iff } \sigma \not\models P$$

$$\sigma \models P_1 \lor P_2 \text{ iff } \sigma \models P_1 \text{ or } \sigma \models P_2$$

If a formula contains no temporal operators then its value is determined solely by the first state in the interval, σ_0.

Interval Temporal Operators.

empty holds over an interval of length 0 and nothing else.

$$\sigma \models \textbf{empty} \text{ iff } |\sigma| = 0$$

Next moves the start of the interval under consideration one state into the future.

$$\sigma \models \bigcirc P \text{ iff } |\sigma| > 0 \text{ and } (\sigma)^1 \models P$$

This is a 'strong' version of the 'next' operation which forces there to be a next time point; a 'weaker' version also allows the possiblility that the interval is empty.

Chop subdivides the given interval into two sub-intervals the first of which satisfies P_1 and the second of which satisfies P_2.

$$\sigma \models P_1 ; P_2 \text{ iff } \exists k \in \mathcal{N} \ (k \leq |\sigma| \text{ and } [\sigma]^k \models P_1 \text{ and } (\sigma)^k \models P_2)$$

The two sub-intervals arising from chop have one common state, the kth state in the above definition. Thus, the last state of the sub-interval over which P_1

holds is the first state of the sub-interval over which P_2 holds. Note also that k may take the value 0 or $|\sigma|$, so that it is possible to 'chop' an interval into two halves at least one of which is empty.

The semantics of projection are somewhat more complex.

$$\begin{array}{lll} \sigma \models P_1 \ \mathbf{proj} \ P_2 & \text{iff} & \exists m \in \mathcal{N} \text{ and } \exists \tau_0, \tau_1, ..., \tau_m \in \mathcal{N} \\ & & (0 = \tau_0 < \tau_1 < ... < \tau_m = |\sigma| \text{ and} \\ & & \forall j < m \ (([\sigma]^{\tau_{j+1}})^{\tau_j} \models P_1 \text{ and} \\ & & \sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m} \models P_2) \end{array}$$

This rule states that for an interval to satisfy P_1 **proj** P_2 there must exist an increasing sequence of m points[1] (or states) in the interval, $\tau_0, \tau_1, ..., \tau_m$, such that the first and last points bound the interval. This sequence of points effectively divides the interval into a series of m sub-intervals, each of which comprises the states between τ_j and τ_{j+1}. We require that P_1 holds over each of these sub-intervals and, in addition, that P_2 holds when $\sigma_{\tau_0}, \sigma_{\tau_1}, ..., \sigma_{\tau_m}$ is viewed as an interval.

Notice also that P_1 **proj** P_2 holds over an empty (i.e. one state) interval if and only if P_2 holds over the interval. P_1 will be trivially satisfied over such an interval.

Satisfaction and Validity.

We define that an interval σ *satisfies* a proposition P if and only if $\sigma \models P$. In addition, in the usual way, we state that P is *valid* if and only if for all σ in $\mathcal{I}$, $\sigma \models P$. If P is valid we write $\models P$.

3 Normal Forms in Interval Temporal Logic

3.1 A Normal Form for PITL

The use of normal forms in temporal logic is now relatively common, e.g. [1], [13]. Furthermore, a number of authors have proposed normal forms for interval temporal logic, e.g. [20], [15], [10].

Our normal form is based on that of previous workers; it has the general format:

$$(\mathbf{empty} \ \wedge \ P_e) \ \vee \ \bigvee_i (P_i \ \wedge \ \bigcirc P_i')$$

where P_e and P_i are point formulas and P_i' is a general PITL formula. The left disjunct characterises under what circumstances a formula can be satisfied over an empty interval, while the second disjunct characterises the possible ways in which a formula can be satisfied over a non-empty interval, i.e. a point

[1]It is common only to constrain the sequence of end-points to be non-decreasing, e.g. [20], however, this generates a number of pathological cases that we manage to avoid in our definition.

property must hold at the initial state and then an arbitrary property must hold over the remainder of the interval.

Note also this normal form embodies a recipe for evaluating PITL formula. The first disjunct embodies a base case, i.e. what must hold of a one state interval, while the second disjunct embodies an inductive step, i.e. one of the P_i's must hold now and the associated P_i' must hold from the next state onwards.

Observe that the normal forms given here are what might be termed 'head' normal forms. They constrain the top-level structure of the formula, but do not constrain the overall structure, and in particular the P_i' are not themselves required to be in normal form. This has the consequence that the normal form of a particular formula is not in general unique.

We claim that an arbitrary PITL formula can be mapped into this normal form. The next subsection verifies this claim.

3.2 Inductive Definition of the Normal Form

We proceed through PITL highlighting how each construct of the logic can be expressed in our normal form. We thereby give an inductive definition of the normal form. So, assume Q is an arbitrary formula of PITL and assume that P_1, P_2 and P are already in normal form, as follows:

$$P_1 \equiv (\mathbf{empty} \wedge P_{e,1}) \vee \bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1})$$

$$P_2 \equiv (\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2})$$

$$P \equiv (\mathbf{empty} \wedge P_e) \vee \bigvee_i (P_i \wedge \bigcirc P'_i)$$

3.2.1 Non-temporal Operators

Atomic Propositions. If Q is an atomic proposition, p, then,

$$Q \equiv p \equiv (\mathbf{empty} \wedge p) \vee (p \wedge \bigcirc \mathbf{True})$$

Thus, an atomic proposition holds over an interval if either the interval has only one state and p holds at this state or the interval is of an arbitrary size greater than zero and p holds at the first state. This clarifies the nature of atomic propositions as point formulae.

False. If Q is **False** then,

$$Q \equiv \mathbf{False} \equiv (\mathbf{empty} \wedge \mathbf{False}) \vee (\mathbf{False} \wedge \bigcirc \mathbf{True})$$

Or. If Q is $P_1 \vee P_2$ then,

$$Q \equiv P_1 \vee P_2 \equiv (\mathbf{empty} \wedge (P_{e,1} \vee P_{e,2})) \vee \bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1}) \vee \bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2})$$

And. However, if Q is $P_1 \wedge P_2$ then,

$$Q \equiv P_1 \wedge P_2 \equiv (\mathbf{empty} \wedge (P_{e,1} \wedge P_{e,2})) \vee \bigvee_{i,j} ((P_{i,1} \wedge P_{j,2}) \wedge \bigcirc (P'_{i,1} \wedge P'_{j,2}))$$

3.2.2 Partitioned normal forms

Before we cover the normal form for negations of propositions, we take a detour into elementary propositional calculus. We say that a collection of formulas $\langle P_i \rangle_i$ *partitions* **True** when

- $\bigvee_i P_i \;\equiv\; \mathbf{True}$ – the partition is exhaustive; and
- $\bigvee_{i \neq j} (P_i \;\wedge\; P_j) \;\equiv\; \mathbf{False}$ – the partition is exclusive.

It is an exercise to show that in the case that $\langle P_i \rangle_i$ partitions **True** then,

$$\bigvee_i (P_i \;\wedge\; Q_i) \;\equiv\; \bigwedge_i (P_i \;\Rightarrow\; Q_i)$$

and then it is straightforward to see that,

$$\neg \bigwedge_i (P_i \;\Rightarrow\; Q_i) \;\equiv\; \bigwedge_i (P_i \;\Rightarrow\; \neg Q_i)$$

and so in the partitioned case,

$$\neg \bigvee_i (P_i \;\wedge\; Q_i) \;\equiv\; \bigvee_i (P_i \;\wedge\; \neg Q_i)$$

We call a normal form

$$(\mathbf{empty} \;\wedge\; P_e) \;\vee\; \bigvee_i (P_i \;\wedge\; \bigcirc P_i')$$

partitioned when $\langle P_i \rangle_i$ partitions **True**.

We can turn an arbitrary normal form into partitioned form. For notational simplicity choose the case in which there are two disjuncts,

$$(P_1 \;\wedge\; \bigcirc P_1') \;\vee\; (P_2 \;\wedge\; \bigcirc P_2')$$

This generates the partition

$$\langle\, (P_1 \;\wedge\; P_2),\; (P_1 \;\wedge\; \neg P_2),\; (\neg P_1 \;\wedge\; P_2),\; (\neg P_1 \;\wedge\; \neg P_2)\,\rangle$$

and the non-empty part of the normal form becomes

$$\begin{array}{lcl}
(P_1 \;\wedge\; \bigcirc P_1') \;\vee\; (P_2 \;\wedge\; \bigcirc P_2') & \equiv & ((P_1 \;\wedge\; P_2) \;\wedge\; \bigcirc(P_1' \;\vee\; P_2')) \;\vee \\
& & ((P_1 \;\wedge\; \neg P_2) \;\wedge\; \bigcirc P_1') \;\vee \\
& & ((\neg P_1 \;\wedge\; P_2) \;\wedge\; \bigcirc P_2') \;\vee \\
& & ((\neg P_1 \;\wedge\; \neg P_2) \;\wedge\; \bigcirc \mathbf{False})
\end{array}$$

Some logical operations (including conjunction and projection) preserve the property of being partitioned; others, notably disjunction and chop, do not. Using the technique outlined here we can put normal forms into partitioned form should that be either necessary or convenient.

3.2.3 Negation

Using the material in the previous section, if P is in partitioned normal form then

$$Q \equiv \neg P \equiv (\textbf{empty} \wedge \neg P_e) \vee \bigvee_i (P_i \wedge \bigcirc \neg P'_i)$$

Any normal form can be transformed into partitioned normal form, and so can be negated by this method.

3.2.4 Temporal Operators

Empty. If Q is **empty** then,

$$Q \equiv \textbf{empty} \equiv (\textbf{empty} \wedge \textbf{True}) \vee (\textbf{False} \wedge \bigcirc \textbf{True})$$

Thus, **empty** holds over any arbitrary "one state" interval, but fails to hold over any "larger" interval.

Next. If Q is $\bigcirc P$ then,

$$Q \equiv \bigcirc P \equiv (\textbf{empty} \wedge \textbf{False}) \vee \bigvee_i (\textbf{True} \wedge \bigcirc P)$$

Thus, $\bigcirc P$ cannot hold over an empty interval (this is because it is a strong next), while it puts no constraints on the first state of a non-empty interval and requires that P holds over the remainder of the interval.

Note that P is not required to be in normal form here.

Chop. If Q is $P_1 \,; P_2$ then,

$$\begin{aligned} Q \equiv P_1 \,; P_2 \quad \equiv \quad & (\textbf{empty} \wedge (P_{e,1} \wedge P_{e,2})) \vee \\ & \bigvee_i (P_{i,1} \wedge \bigcirc (P'_{i,1} \,; P_2)) \vee \\ & \bigvee_j ((P_{e,1} \wedge P_{j,2}) \wedge \bigcirc P'_{j,2}) \end{aligned}$$

The normal form embodies three cases.

- The first disjunct gives the condition under which the chop formula holds over an empty interval, namely when the two constituent formulas hold over the empty interval;
- the second disjunct is the condition in which P_1 holds over a non-empty interval, and P_2 holds over the remainder; and,
- the third disjunct embodies the case where P_1 holds over an empty interval and P_2 holds over the entire (non-empty) interval.

This explanation forms an informal argument for the correctness of the normal form of the chop; it is formalised in the next section.

The negation of chop is given by transforming the normal form given here into the partitioned normal form, as explained above.

Projection. If Q is $(P_1 \ \mathbf{proj} \ P_2)$ then,

$$Q \equiv P_1 \ \mathbf{proj} \ P_2 \equiv (\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_i \bigvee_j ((P_{i,1} \wedge P_{j,2}) \wedge \bigcirc (P'_{i,1} \ \mathbf{;} \ (P_1 \ \mathbf{proj} \ P'_{j,2})))$$

The normal form embodies two cases.

- The first disjunct gives the condition under which the projection holds over an empty interval, namely when P_2 holds over the interval.
- The other disjunct gives the condition where $(P_1 \ \mathbf{proj} \ P_2)$ holds over a non-empty interval. A choice of initial-state conditions of P_1 and P_2, such as $P_{i,1}$ and $P_{j,2}$, must hold at the first state; over the remaining part of the interval a chop has to hold. First the remainder of the P_1 condition – $P'_{i,1}$ – must hold, then the derivative of the projection – $(P_1 \ \mathbf{proj} \ P'_{j,2})$ – must be valid.

Note that if P_1 and P_2 are in partitioned normal form then this normal form will also be partitioned. The next section formalises this explanation.

3.3 Validity of the Normal Forms

In this section we argue that the normal forms given in the previous section are semantically equivalent to the original formulas; we begin by proving a lemma about the satisfaction of formulas in normal form.

Lemma 1 *If P is the formula*

$$(\mathbf{empty} \wedge P_e) \vee \bigvee_i (P_i \wedge \bigcirc P'_i)$$

then $\sigma \models P$ if and only if either

- $|\sigma| = 0$ *and* $\sigma_0 \models P_e$, *or*
- $|\sigma| > 0$ *and for some* i, $\sigma_0 \models P_i$ *and* $(\sigma)^1 \models P'_i$

Proof
This is a straightforward exercise in checking the semantic definition. □

Theorem 1 *The normal forms derived above are logically equivalent to the original formulas.*

Proof
The proof works by induction over the construction of the formulas; we leave the simpler cases as exercises but in Lemmas 2 and 3 we check the result for chop and projection. □

Lemma 2 *If*

$$P_1 \;\equiv\; (\textbf{empty} \;\wedge\; P_{e,1}) \;\vee\; \bigvee_i (P_{i,1} \;\wedge\; \bigcirc P'_{i,1})$$

and

$$P_2 \;\equiv\; (\textbf{empty} \;\wedge\; P_{e,2}) \;\vee\; \bigvee_j (P_{j,2} \;\wedge\; \bigcirc P'_{j,2})$$

then for all intervals σ

$$\sigma \;\models\; P_1 \,;\, P_2$$

if and only if

$$\begin{array}{ll} \sigma \;\models & (\textbf{empty} \;\wedge\; (P_{e,1} \;\wedge\; P_{e,2})) \;\vee \\ & \bigvee_i (P_{i,1} \;\wedge\; \bigcirc(P'_{i,1} \,;\, P_2)) \;\vee \\ & \bigvee_j ((P_{e,1} \;\wedge\; P_{j,2}) \;\wedge\; \bigcirc P'_{j,2}) \qquad\qquad (RHS) \end{array}$$

Proof
We do a case analysis over the length of σ. Suppose first that $|\sigma| = 0$, then

$$\begin{array}{lll} \sigma \;\models\; P_1 \,;\, P_2 & \text{iff} & \sigma \;\models\; P_1 \text{ and } \sigma \;\models\; P_2 \\ & & \text{(by the semantics of chop)} \\ & \text{iff} & \sigma_0 \;\models\; P_{e,1} \text{ and } \sigma_0 \;\models\; P_{e,2} \\ & & \text{(by Lemma 1)} \\ & \text{iff} & \sigma \;\models\; RHS \\ & & \text{(by Lemma 1 again)} \end{array}$$

Now consider the case that $|\sigma| > 0$. Then

$$\sigma \;\models\; P_1 \,;\, P_2 \quad \text{iff} \quad \exists k \le |\sigma|,\, [\sigma]^k \models P_1 \text{ and } (\sigma)^k \models P_2$$

If this k is 0 then

$$\begin{array}{ll} \text{iff} & [\sigma]^0 \models P_1 \text{ and } \sigma \models P_2 \\ & \text{(since } (\sigma)^0 \text{ is } \sigma) \\ \text{iff} & \sigma_0 \models P_{e,1} \text{ and for some } j,\, \sigma_0 \models P_{j,2} \text{ and } (\sigma)^1 \models P'_{j,2} \\ \text{iff} & \text{for some } j,\, \sigma_0 \models P_{e,1} \wedge P_{j,2} \text{ and } (\sigma)^1 \models P'_{j,2} \\ \text{iff} & \sigma \models \text{last clause of } (RHS) \\ \Rightarrow & \sigma \models (RHS) \end{array}$$

If this $k > 0$ then

$$\text{iff} \quad [\sigma]^k \models P_1 \text{ and } (\sigma)^k \models P_2$$

iff (since this k is positive), for some i,
$\sigma_0 \models P_{i,1}$, $([\sigma]^k)^1 \models P'_{i,1}$
and $(\sigma)^k \models P_2$
iff for some i, $\sigma_0 \models P_{i,1}$ and $(\sigma)^1 \models (P'_{i,1} \mathbin{;} P_2)$
iff $\sigma \models$ second clause of (RHS)
$\Rightarrow$ $\sigma \models (RHS)$

In the other direction, suppose that $\sigma \models (RHS)$. Still under the supposition that $|\sigma| > 0$ by lemma 1 either

- $\sigma_0 \models P_{i,1}$ and $(\sigma)^1 \models (P'_{i,1} \mathbin{;} P_2)$ for some i, or
- $\sigma_0 \models P_{e,1} \wedge P_{j,2}$ and $(\sigma)^1 \models P'_{j,2}$ for some j.

In the first case, suppose that $[(\sigma)^1]^k \models P'_{i,1}$ and $((\sigma)^1)^k \models P_2$. Then $[\sigma]^{k+1} \models P_1$ and $(\sigma)^{k+1} \models P_2$, so $\sigma \models (P_1 \mathbin{;} P_2)$.

In the second case, $[\sigma]^0 \models P_1$ and $(\sigma)^0 \models P_2$; again $\sigma \models (P_1 \mathbin{;} P_2)$. We have therefore the result that $\sigma \models (P_1 \mathbin{;} P_2)$ in all cases. This concludes the proof in the backwards direction for the $|\sigma| > 0$ case, and indeed the proof itself.
$\Box$

Lemma 3 *Under the same hypotheses as Lemma 2, for all intervals σ*

$$\sigma \ \models \ P_1 \ \mathbf{proj} \ P_2$$

if and only if

$$\begin{aligned} \sigma \quad \models \quad & (\mathbf{empty} \ \wedge \ P_{e,2}) \ \vee \\ & \bigvee_i \bigvee_j ((P_{i,1} \ \wedge \ P_{j,2}) \ \wedge \ \bigcirc(P'_{i,1} \mathbin{;} P_1 \ \mathbf{proj} \ P'_{j,2})) \end{aligned}$$

Proof
The semantics of projection are given thus

$$\begin{aligned} \sigma \models P_1 \ \mathbf{proj} \ P_2 \ \text{ iff } \quad & \exists m \in \mathcal{N} \text{ and } \exists \tau_0, \tau_1, ..., \tau_m \in \mathcal{N} \\ & (0 = \tau_0 < \tau_1 < ... < \tau_m = |\sigma| \text{ and} \\ & \forall j < m \ ([\sigma]^{\tau_{j+1}})^{\tau_j} \models P_1 \text{ and} \\ & \sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m} \models P_2) \end{aligned}$$

The case that $m = 0$ is precisely the case that $|\sigma| = 0$, since the requirements

$$0 = \tau_0 < \tau_1 < ... < \tau_m = |\sigma|$$

can only be satisfied by m being 0 and $|\sigma| = 0$. This means that if $|\sigma| = 0$

$$\begin{aligned} \sigma \models (P_1 \ \mathbf{proj} \ P_2) \ & \text{iff} \ \ \sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m} \models P_2 \\ & \text{iff} \ \ \sigma \models P_2 \\ & \text{iff} \ \ \sigma_0 \models P_{e,2} \end{aligned}$$

This observation shows that the 'empty' part of the normal form for (P_1 **proj** P_2) is correct. Now we consider the case that $m > 0$.

In this case we have

$$([\sigma]^{\tau_1})^{\tau_0} \models P_1 \tag{1}$$

and

$$\sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m} \models P_2 \tag{2}$$

Since τ_0 is 0 formula (1) simplifies to

$$[\sigma]^{\tau_1} \models P_1 \tag{3}$$

Since $\tau_1 > \tau_0$ and $|\sigma| \neq 0$ both $[\sigma]^{\tau_1}$ and $\sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m}$ are non-empty. By Lemma 1 and the hypotheses on P_1 and P_2,

$[\sigma]^{\tau_1} \models P_1$ iff for some i $([\sigma]^{\tau_1})_0 \models P_{i,1}$ and $([\sigma]^{\tau_1})^1 \models P'_{i,1}$

$\sigma_{\tau_0}\sigma_{\tau_1}...\sigma_{\tau_m} \models P_2$ iff for some j, $\sigma_0 \models P_{j,2}$ and $\sigma_{\tau_1}...\sigma_{\tau_m} \models P'_{j,2}$

This means that (2) and (3) hold exactly when for some i and j

$$\sigma_0 \models P_{i,1} \wedge P_{j,2} \tag{4}$$

and

$$([\sigma]^{\tau_1})^1 \models P'_{i,1}$$
$$\sigma_{\tau_1}...\sigma_{\tau_m} \models P'_{j,2}$$

and also the semantics of **proj** give us that, for all $0 < j < m$,

$$([\sigma]^{\tau_{j+1}})^{\tau_j} \models P_1$$

Putting these together we have (4) and

$$(\sigma)^1 \models P'_{i,1} \,;\, (P_1 \ \mathbf{proj}\ P'_{j,2}) \tag{5}$$

which shows that for a non-empty σ, $\sigma \models P_1$ **proj** P_2 if and only if

$$\sigma \models \bigvee_i \bigvee_j ((P_{i,1} \ \wedge \ P_{j,2}) \ \wedge \ \bigcirc(P'_{i,1} \,;\, (P_1 \ \mathbf{proj}\ P'_{j,2})))$$

This concludes the part of the proof concerning non-empty intervals, and thus the proof itself. $\Box$

3.4 Examples

This subsection illustrates our normal form by working through a number of examples.

3.4.1 Example 1

Assume that the formula P is in normal form, i.e.

$$P \equiv (\textbf{empty} \wedge P_e) \vee \bigvee_i (P_i \wedge \bigcirc P'_i)$$

and consider the derived formula eventually P, $\Diamond P$. The standard derivation of eventually is from chop,

$$\Diamond P \quad \equiv \quad \textbf{True} \mathbin{\textbf{;}} P$$

Now, using the normal form rule for chop, $\Diamond P$ can be placed in normal form as follows:

$$\begin{aligned} \Diamond P \quad \equiv \quad & (\textbf{empty} \wedge P_e) \\ & \vee (\textbf{True} \wedge \bigcirc(\textbf{True} \mathbin{\textbf{;}} P)) \\ & \vee \bigvee_i (P_i \wedge \bigcirc P'_i) \end{aligned}$$

Notice that using associativity and symmetry of disjunction and **True** as unit of conjunction we can reduce this to:

$$\Diamond P \quad \equiv \quad P \vee \bigcirc \Diamond P$$

which is a more common "inductive" interpretation of $\Diamond$.

3.4.2 Example 2

Assuming again that P is in normal form,

$$P \equiv (\textbf{empty} \wedge P_e) \vee \bigvee_i (P_i \wedge \bigcirc P'_i)$$

and now also assume that $\langle P_i \rangle_i$ is a partition.

Consider always P (the dual of eventually P) which is derived in standard fashion as:

$$\Box P \quad \equiv \quad \neg(\textbf{True} \mathbin{\textbf{;}} \neg P)$$

Now, let us first consider (**True ;** $\neg P$) in normal form, this takes a similar form to our last example:

$$\begin{aligned} \textbf{True} \mathbin{\textbf{;}} \neg P \quad \equiv \quad & (\textbf{empty} \wedge \neg P_e) \\ & \vee (\textbf{True} \wedge \bigcirc(\textbf{True} \mathbin{\textbf{;}} \neg P)) \\ & \vee \bigvee_i (P_i \wedge \bigcirc(\neg P'_i)) \end{aligned}$$

However, before we can negate this normal form and obtain $\Box P$, this must be put into partitioned form, as follows:

$$\begin{array}{rcl} \textbf{True} \; \textbf{;} \, \neg P & \equiv & (\textbf{empty} \;\wedge\; \neg P_e) \\ & & \vee \bigvee_i (P_i \;\wedge\; \bigcirc(\neg P_i' \;\vee\; \textbf{True} \; \textbf{;} \, \neg P)) \end{array}$$

and we can now consider $\Box P$:

$$\begin{array}{rcl} \neg(\textbf{True} \; \textbf{;} \, \neg P) & \equiv & (\textbf{empty} \;\wedge\; \neg\neg P_e) \\ & & \vee \bigvee_i (P_i \;\wedge\; \bigcirc\neg(\neg P_i' \;\vee\; \textbf{True} \; \textbf{;} \, \neg P)) \end{array}$$

Assuming that $ⓦ$ is the weak next operator, the following derivation gives a more familiar expansion of $\Box P$.

$$\begin{array}{ll} & \neg(\textbf{True} \; \textbf{;} \, \neg P) \\ \equiv & \{ \text{ manipulation of above normal form } \} \\ & (\textbf{empty} \;\wedge\; P_e) \;\vee\; \bigvee_i (P_i \;\wedge\; \bigcirc(P_i' \;\wedge\; \neg(\textbf{True} \; \textbf{;} \, \neg P)) \\ \equiv & \{ \text{ properties of next } \} \\ & (\textbf{empty} \;\wedge\; P_e) \;\vee\; \bigvee_i ((P_i \;\wedge\; \bigcirc P_i') \;\wedge\; \bigcirc\neg(\textbf{True} \; \textbf{;} \, \neg P)) \\ \equiv & \{ \text{ normal form } \} \\ & (P \;\wedge\; \textbf{empty}) \;\vee\; (P \;\wedge\; \bigcirc\Box P) \\ \equiv & \{ \textbf{ empty} \;\vee\; \bigcirc\Box P \;\equiv\; ⓦ\Box P \; \} \\ & P \;\wedge\; ⓦ\Box P \end{array}$$

4 The Tableau Method

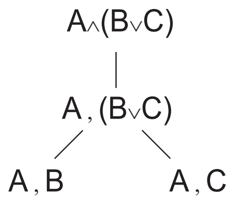
In this section we present a tableau decision procedure for checking the satisfiability of PITL formulae. The decision procedure is influenced by the tableau procedures developed for standard temporal logic, e.g. [3] [24] and [27]. Our work particularly builds from the tableau method defined by P. Wolper [27]. Note, however, that we work with finite rather than infinite models, and so our work has a rather different flavour, particularly when we look at axiomatisation and decidability aspects.

The tableau decision algorithm provides a description of all the models that satisfy a particular formula. It is a graph construction and colouring algorithm under which a formula is systematically decomposed according to the structure of the formula. Models of the formula correspond to *paths* through the graph from the root node to a leaf. The example of $\bigcirc\bigcirc q$ is given later in the Section, on page 24.

Nodes of the graph are sets of PITL formula; these are said to *label the node*. The interpretation of such a labelling is that for the node to be satisfied, it must be possible for all the formulas in the set to be satisfied together. Edges in the graph represent steps to satisfaction of the formulas of a node.

These steps are made according to a set of *tableau (decomposition) rules* which embody how satisfaction of a particular formula can be decomposed into satisfaction of its constituent formulae. Branches in the tableau reflect disjunctive choices.

For standard propositional logic, the tableau procedure consist of a collection of rules which decompose a formula according to its logical structure. For instance, in order to satisfy a conjunction, $A \wedge B$, both conjuncts need to be satisfied, so that the formula can be replaced by its two conjuncts; on the other hand, a formula $A \vee B$ leads to a branch point in the tableau: to satisfy the disjunction, either one of the disjuncts can be satsified. The tableau procedure is completed when all logical operators are expanded out.



In the example shown it can be seen that each model of the formula corresponds to one of the sets of formulae at the leaves of the tree. To satisfy $A \wedge (B \vee C)$ it is sufficient to satisfy either the set $\{A, B\}$ or to satisfy the set $\{A, C\}$ (and it is necessary to satisfy at least one of them).

An important element of the tableau rules for temporal formulas is that they subdivide the requirements imposed by temporal formulas into requirements on the present (i.e. the first state of an interval) and requirements on the remainder of the interval, the latter being embodied in a formula of the form $\bigcirc P$. This is in fact exactly the subdivision in our normal form rules. Accordingly our normal forms will play a central role in these tableau rules.

Elaboration of the logical structure of the requirements on the present is done just as in the non-temporal case, whereas a temporal *step rule* is required to elaborate requirements on the future. A path from the root to a leaf will contain a number of occurrences of the step rule, each representing the passage of a unit of time. To satisfy the temporal formula at the root, it is necessary to meet the propositional requirements along the path, which correspond to requirements on the various states in the interval which models the formula.

A set of *graph construction* rules build the graph using the tableau rules. In particular, the graph construction rules generate time passing steps in the tableau that unravel next formulas.

The final stage of the tableau algorithm is a *graph colouring* algorithm[2]

[2]In [6] we employed a graph reduction algorithm whereby unsatisfiable nodes were removed

which systematically colours unsatisfiable nodes in the graph. A formula can then be deduced to be satisfiable if it appears at an uncoloured node.

The tableau algorithm provides a decision procedure for propositional interval temporal logic, but more importantly it also provides a means of deriving a complete axiomatisation of this logic, a topic which is discussed in Sections 6 and 7.

We will work through the different stages of the tableau algorithm in turn, but first give some notation.

Notation. The following notation is related to that used in [27].

- The formula being evaluated is called the *initial formula.*
- The *labelling of node* n is denoted T_n.
- A formula P is called *elementary* if
    1. it is an atomic proposition or the negation of an atomic proposition;
    2. it is **empty** or ¬**empty**; or
    3. it has $\bigcirc$ as the main connective.
- A node containing only elementary formulae is called a *state.*
- A *pre-state* is a node that is either initial or not a state.
- A node is *final* if it has no successor nodes (whether coloured or not) in the tableau.

Tableau Rules. The *tableau rules* (also called *decomposition rules*) drive the unfolding of formulae into their components. They map a formula into a set of sets of formulas, i.e. they have the form:

$$P \longrightarrow \{\ S_1, S_2, ..., S_n\ \}$$

where S_i is a set of PITL formulas. The interpretation that such rules embody is that P can be satisfied if and only if there exists a j $(1 \leq j \leq n)$ such that all formulas in S_j can together be satisfied. This assertion is a component of the proof in Section 5 that the tableau procedure is complete.

Our first set of rules are standard and are unchanged from those in [27]:

$$\begin{array}{rcl}
\neg\neg P & \longrightarrow & \{\{\ P\ \}\} \\
P_1\ \wedge\ P_2 & \longrightarrow & \{\{\ P_1\ ,\ P_2\ \}\} \\
\neg(P_1\ \wedge\ P_2) & \longrightarrow & \{\{\ \neg P_1\ \}, \{\ \neg P_2\ \}\} \\
P_1\ \vee\ P_2 & \longrightarrow & \{\{\ P_1\ \}, \{\ P_2\ \}\} \\
\neg(P_1\ \vee\ P_2) & \longrightarrow & \{\{\ \neg P_1\ ,\ \neg P_2\ \}\}
\end{array}$$

rather than coloured. However, the colouring approach helps our later proof of completeness and the two schemes are clearly equivalent.

We also need the following rule which enables the next operator to be moved to the top level through negation. Notice that since we are using a strong next operator, $\bigcirc P$ will fail over any **empty** interval.

$$\neg \bigcirc P \quad \longrightarrow \quad \{\{\ \textbf{empty}\ \}, \{\ \bigcirc \neg P\ \}\}$$

The remaining rules are direct extrapolations from our normal form rules. They all assume that P_1 and P_2 are in the normal forms we highlighted earlier. The next two rules decompose chop and projection:

$$\begin{array}{rcl} P_1 \ ;\ P_2 & \longrightarrow & \{\{\ (\textbf{empty}\ \wedge\ (P_{e,1}\ \wedge\ P_{e,2}))\ \vee \\ & & \bigvee_i (P_{i,1}\ \wedge\ \bigcirc(P'_{i,1}\ ;\ P_2))\ \vee \\ & & \bigvee_j ((P_{e,1}\ \wedge\ P_{j,2})\ \wedge\ \bigcirc P'_{j,2})\ \}\} \\ P_1\ \textbf{proj}\ P_2 & \longrightarrow & \{\{\ (\textbf{empty}\ \wedge\ P_{e,2})\ \vee \\ & & \bigvee_i \bigvee_j ((P_{i,1}\ \wedge\ P_{j,2})\ \wedge\ \bigcirc(P'_{i,1}\ ;\ P_1\ \textbf{proj}\ P'_{j,2}))\ \}\} \end{array}$$

These rules will give rise to branch points in the tableau when elaborated with the disjunction rule.

$\neg(P_1\ ;\ P_2)$ and $\neg(P_1\ \textbf{proj}\ P_2)$ can be obtained by the negation of normal forms previously highlighted. In fact, negation of all the operators could be obtained from negation of the normal forms, but in the case of the propositional operators and strong next it is more straightforward to handle negation outside the normal form expansion.

The following two rules encapsulate the required normal form negation. Both rules assume that $\langle P_{i,1}\rangle_i$ and $\langle P_{j,2}\rangle_j$ are in partition form.

$$\begin{array}{rcl} \neg(P_1\ ;\ P_2) & \longrightarrow & \{\{\ (\textbf{empty}\ \wedge\ (\neg P_{e,1}\ \vee\ \neg P_{e,2}))\ \vee \\ & & \bigvee_i ((\neg P_{e,1}\ \wedge\ P_{i,1})\ \wedge\ \bigcirc\neg(P'_{i,1}\ ;\ P_2))\ \vee \\ & & \bigvee_i \bigvee_j ((P_{e,1}\ \wedge\ P_{i,1}\ \wedge\ P_{j,2})\ \wedge\ \bigcirc(\neg P'_{j,2}\ \wedge\ \neg(P'_{i,1}\ ;\ P_2)))\ \}\} \end{array}$$

$$\begin{array}{rcl} \neg(P_1\ \textbf{proj}\ P_2) & \longrightarrow & \{\{\ (\textbf{empty}\ \wedge\ \neg P_{e,2})\ \vee \\ & & \bigvee_i \bigvee_j ((P_{i,1}\ \wedge\ P_{j,2})\ \wedge\ \bigcirc\neg(P'_{i,1}\ ;\ P_1\ \textbf{proj}\ P'_{j,2}))\ \}\} \end{array}$$

We have the following lemma.

Lemma 4 *All the decomposition rules preserve satisfaction. Thus, for a rule of the form,*

$$P \longrightarrow \{\ S_i\ \}$$

the following holds:

$$\sigma \models P \;\; \textit{iff} \;\; \exists j \,.\, \sigma \models \bigwedge S_j \qquad \textit{where} \;\bigwedge\{P_1, ..., P_n\} = P_1 \;\wedge\; ... \;\wedge\; P_n$$

Proof
The propositional rules and the rule for $\neg \bigcirc P$ are straightforward. The rules for **;**, **proj** , $\neg$ **;** and $\neg$ **proj** follow from theorem 1 in which the validity of the normal forms was verified. □

The Graph Construction. The tableau graph is constructed using the following rules:

1. Where P is the formula to test, label the *initial node* with { P , **True ; empty** } and apply steps (2) and (3) repeatedly until neither can be applied to any node in the graph.

   Note that when we create a son node in rules (2) and (3) with labelling T, we only add a new node if a node labelled T does not already exist, otherwise we add an edge back to the original node.

2. If a node n contains a non-elementary formula P and the tableau rule for P is $P \longrightarrow \{S_i\}$, then,

   For all i create a son of n labelled:
   $(T_n - \{P\}) \cup S_i$

   There is some room for choice in the order that these decomposition rules are applied. It is the accepted wisdom that branching rules should be applied as late as possible.

3. If a node n contains only:

   - elementary formulas, and
   - does not contain empty.

   Then create a son of n labelled by the next formulas of T_n with outermost nexts removed.

We call rule 2 *the decomposition rule* and rule 3 *the step rule.*

So, we start with the formula under test and we add $\Diamond$**empty** to the initial node. All models in interval temporal logic are finite, and so satisfy the formula $\Diamond$**empty**. Moreover, the presence of **empty** in a node will signal the fact that no further temporal steps need to be taken; **empty** can be seen as a flag which indicates that the path representing a particular model ends at the node in question. Note that, because of the decomposition and step rules, a marker for termination of a path is included in every state, since $\Diamond$**empty** is either satisfied immediately and a path is terminated or it is preserved in the next state, i.e. $\Diamond$**empty** arises in all descendent states; lemma 5 encapsulates this fact.

Graph Colouring. The next stage in the tableau algorithm is to colour unsatisfiable nodes by repeatedly applying the following rules:

- (E1) Colour nodes containing formula P and $\neg P$.
- (E2) If a node has no uncoloured successors and does not contain **empty**, then colour it.
- (E3) If a node contains **empty** and $\bigcirc P$, then colour it.
- (E4) If for a particular pre-state there is no uncoloured node containing **empty** reachable from it, then colour the pre-state.

This procedure terminates when all unsatisfiable nodes have been eliminated. The first and the third of these rules embody straightforward cases of an unsatisfiable node, and the second rule elminates nodes all of whose successors are unsatisfiable.

The fourth rule is more subtle. It is required because cycles can arise in the tableau, see e.g. figure 3 and figure 5, and some unsatisfiable **;** and **proj** formulae will generate cycles with no path to a final node (i.e. a satisfiable node containing **empty**). The formula $\Box \bigcirc$**True** is an example (see figure 5). Cycles of this variety would not be coloured through application of rules (E1), (E2) and (E3). However since all invervals must, by definition, be finite, we can colour any node from which we cannot reach a final node. This is the purpose of rule (E4).

Note also that our approach here is a simplification of the approach we introduced in [6]. It is clear that rule (E4) simplifies the algorithm by removing the spurious extra complexity of searching for accepting paths, which was the approach in [6].

We claim that the initial formula is satisfiable if and only if we can generate a tableau according to the graph construction rules, colour it according to rules (E1) - (E4) and the initial formula is not coloured. We prove this result in the next section.

Discussion. The tableau algorithm that we present here is simpler in a number of respects than that defined in [27]. For example, a central concept in Wolper's work is that of a *fulfilling path*. For any temporal formula, other than $\bigcirc$, to be satisfied a path which witnesses the satisfaction of the formula must exist. Rule (E4) plays the corresponding role in our model. However, many of the complexities associated with fulfilling paths do not arise in our work since we are searching for *finite* models and thus, we can use **empty** as a marker for successful completion of such a satisfying model. In addition, Wolper includes a marking mechanism (quite different from our colouring mechanism), whereby formulas are marked when they are being considered for fulfillment.

Examples. We present three example tableau evaluations.

Example 1. The tableau expansion of $\bigcirc\bigcirc q$, where q is an atomic proposition, is shown in figure 3. The tableau uses the normal form for **True ; empty** which after a series of applications of the decomposition rule reduces to:

$$\textbf{empty} \;\vee\; \bigcirc(\textbf{True ; empty})$$

In order to simplify the presented tableau, we do not show the intermediate decomposition steps.

In order to clarify the terminology that we have been using, note that nodes: 1, 4, 7, and 10 are pre-states and nodes: 2, 3, 5, 6, 8, 9, 11 and 12 are states.

Application of tableau colouring will result in the colouring of nodes: 2 and 5. However, the initial node will remain uncoloured, so the formula is correctly shown to be satisfiable.

This example indicates the effect of including **True ; empty** in the initial node of the tableau. Specifically, it enables a path to be terminated at any point, by offering the possibility of **empty** at every state. However, any such termination which fails to satisfy the remaining formulae under consideration, will be coloured during the tableau colouring phase, as happens with nodes 2 and 5.

Paths through the tableau to the remaining final nodes correspond to possible models for the formulae under consideration.

- The path from the root to node 8 contains two steps and thus three states. The presence of q at node 8 requires that q is true in the final state of the interval.
- The direct paths from the root to node 11 consists of four states; node 9 requires that q is true at the third of these.
- Paths from the root to node 11 may also follow the loop $10 \rightarrow 12 \rightarrow 10$ any number of times. Each transition of the loop adds a time step and thus a further state to the interval; the only other requirement is imposed by node 10 which requires that q be true at the third state.

The three different cases correspond to three different classes of model of $\bigcirc\bigcirc q$: those of length two, three and greater than three; the colouring of nodes 2 and 5 corresponds to the fact that there are no models of this formula of length zero or one.

Example 2. The tableau expansion for $(\mathbf{len}(1)\ ;\ \bigcirc q) \wedge \mathbf{len}(1)$ is shown in figure 4. Once again the decomposition arrows shown reflect multiple applications of the decomposition rules. These multiple applications will simplify the normal form expansions, yielding the following relationships:

$$
\begin{array}{rcl}
\bigcirc\mathbf{empty}\ ;\ \bigcirc q & \equiv & \bigcirc(\mathbf{empty}\ ;\ \bigcirc q) \\
\mathbf{True}\ ;\mathbf{empty} & \equiv & \mathbf{empty}\ \vee\ \bigcirc(\mathbf{True}\ ;\mathbf{empty}) \\
\mathbf{empty}\ ;\ \bigcirc q & \equiv & \bigcirc q
\end{array}
$$

It turns out that all the final nodes here, i.e. those containing **empty**, will be coloured because they contain unsatisfied next state formulae. The colouring of all leaf nodes will cause all nodes at one step from the leaves (which neither contain **empty** or have any successors) to be coloured which in turn will have

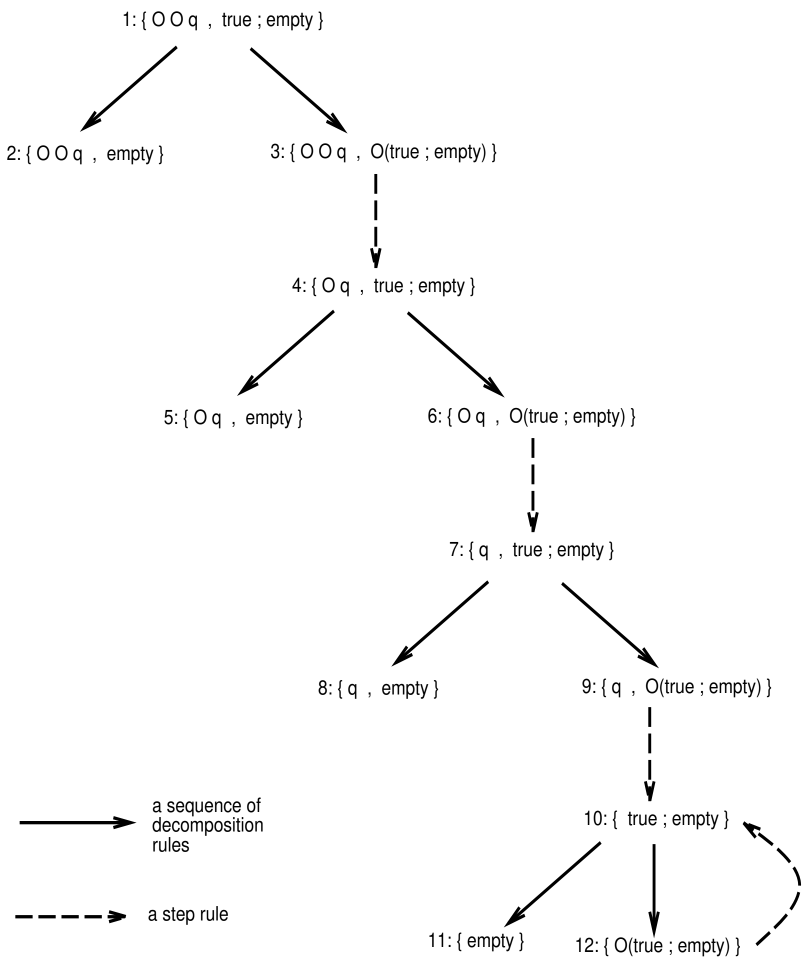


Figure 3: Tableau expansion of $\bigcirc\bigcirc q$

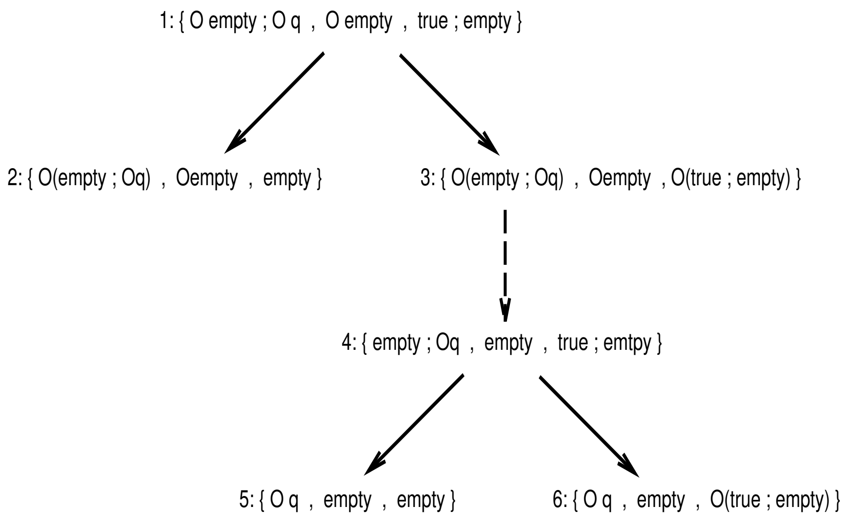


Figure 4: Tableau expansion of $\mathbf{len}(1)\ \mathbf{;}\ \bigcirc q\ \wedge\ \mathbf{len}(1)$

a knock on effect on the nodes two steps from the leaves and so on, until the initial node has been coloured. This shows that the formula

$$(\mathbf{len}(1)\ \mathbf{;}\ \bigcirc q)\ \wedge\ \mathbf{len}(1)$$

is unsatisfiable.

Note that in examples 1 and 2 only the rules $(E1)$, $(E2)$ and $(E3)$ have been used. Our final example illustrates a situation in which $(E4)$ comes into play.

Example 3. We also include the tableau expansion for $\Box\bigcirc\mathbf{True}$, which is equivalent to $\neg(\mathbf{True}\ \mathbf{;}\ \neg\bigcirc\mathbf{True})$. Our decomposition rules would realise the following rewrites:-

$$\begin{array}{lll}
\neg(\mathbf{True}\ \mathbf{;}\ \neg\bigcirc\mathbf{True}) & \rightarrow & \{\text{Normal Forms}\} \\
 & & \neg(((\mathbf{empty}\ \wedge\ \mathbf{True})\ \vee\ (\mathbf{True}\ \wedge\ \bigcirc\mathbf{True}))\ \mathbf{;} \\
 & & ((\mathbf{empty}\ \wedge\ \mathbf{True})\ \vee\ (\mathbf{True}\ \wedge\ \bigcirc\neg\mathbf{True}))) \\
 & \rightarrow & \{\text{Decomposition of } \neg \text{ chop}\} \\
 & & (\mathbf{empty}\ \wedge\ (\neg\mathbf{True}\ \vee\ \neg\mathbf{True}))\ \vee \\
 & & ((\neg\mathbf{True}\ \wedge\ \mathbf{True})\ \wedge\ \bigcirc\neg(\mathbf{True}\ \mathbf{;}\ \neg\bigcirc\mathbf{True}))\ \vee \\
 & & ((\mathbf{True}\ \wedge\ \mathbf{True}\ \wedge\ \mathbf{True})\ \wedge \\
 & & \bigcirc(\neg\neg\mathbf{True}\ \wedge\ \neg(\mathbf{True}\ \mathbf{;}\ \neg\bigcirc\mathbf{True}))) \\
 & \rightarrow & \{\text{Propositional simplifications}\} \\
 & & \bigcirc\neg(\mathbf{True}\ \mathbf{;}\ \neg\bigcirc\mathbf{True})
\end{array}$$

which yields the tableau expansion shown in figure 5. None of the rules $(E1)$,

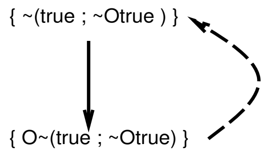


Figure 5: Tableau expansion of $\Box \bigcirc$**True**

$(E2)$ or $(E3)$ applies to any of the nodes in this graph, but it can be seen that $(E4)$ can be applied to colour all the nodes in question, since no node contains **empty**.

5 Correctness of the Tableau

This section investigates correctness of the tableau algorithm. First we need a small lemma which clarifies how the $\Diamond$**empty** behaves in the initial formula.

Lemma 5 *All states in the tableau contain either* **empty** *or* $\bigcirc$(**True ; empty**).

Proof
In this proof we will refer to the normal form for **True ; empty** which can be determined by substitution from the derivation example 3.4. The normal form is:

$$\begin{array}{rcl} \textbf{True ; empty} & \equiv & (\textbf{empty} \ \wedge \ \textbf{True}) \ \vee \\ & & (\textbf{True} \ \wedge \ \bigcirc(\textbf{True ; empty})) \end{array}$$

Now the proof proceeds by induction over the graph.

Base Case. By rule 1 of graph construction **True ; empty** will be in the initial node. In addition, all initial states (i.e. states reached from the initial node without applying the step rule) will be reached by applying tableau rules alone. By inspection of the normal form for **True ; empty** we can see that such an initial state must either contain **empty** (and **True**) or $\bigcirc$(**True ; empty**) (and **True**). Both cases are as required.

Inductive Step. For an arbitrary node n assume that either **empty** or $\bigcirc$(**True ; empty**) is present. We need to show that either **empty** or $\bigcirc$(**True ; empty**) is present in all states accessible from node n.

- *Case (i)* - **empty** *is present.* Since we are at a state we cannot apply the decomposition rule, but in addition, we cannot apply the step rule because **empty** is present. Thus, the node will have no successors and we have what we require vacuously.

- *Case (ii) -* $\bigcirc$(**True ; empty**) *is present.* Since we are at a state we will only be able to apply the step rule. However, all resulting pre-states will contain **True ; empty**. Now in order to reach a new state from such a pre-state, decomposition rules will be applied. In particular, these rule applications will have the effect of reducing **True ; empty** according to its normal form. Now, by inspection of the normal forms, we can see that resulting states must either contain **empty** (and **True**) or $\bigcirc$(**True ; empty**) (and **True**). As was the case with the base condition, both of these are as required.

$\square$

We also need to verify that our tableau expansion process will always terminate.

Lemma 6 *For any particular initial state, repeated application of the decomposition and step rules will terminate.*

Proof

We prove this statement by showing that from any finite set G of ITL formulas there is a finite set of formulas containing G which is closed under application of the tableau decomposition and step rules. Since there is only a finite number of subsets of this set and each node in the tableau is characterised by such a subset, the tableau must itself be finite.

We first observe that given any finite set of formulas there will only be finitely many formulas generated by applying the decomposition rules alone. This is clear since the rules decompose formulas into subformulas. The nub of the proof therefore consists of an analysis of the effect of the step rules.

The crucial issue is thus the complexity of the temporal formulas generated by stripping off a $\bigcirc$, which, in turn, depends upon the normal forms given earlier. Suppose that we have a normal form

$$P \;\equiv\; (\textbf{empty} \;\wedge\; P_e) \;\vee\; \bigvee_i (P_i \;\wedge\; \bigcirc P_i')$$

A combination of a number of decomposition rules and a single step rule lead from the temporal formula P to the temporal formulas P_i'; we call these the **residuals** of P and write

$$P_i' \lhd P$$

to indicate this. We also write $\lhd^*$ ($\unlhd^*$) for the (reflexive) transitive closue of $\lhd$. The crucial property that is needed to prove termination is that for any formula R, the set

$$Res(R) =_{df} \{S | S \lhd^* R\}$$

is finite. Note that this is not the same as claiming that $\lhd^*$ is well-founded: there will be cases of P where $P \lhd^* P$.

The construction of the set $\bigcup_{R\in X}\{S|S \lhd^* R\}$ is analogous to building the Fischer-Ladner closure [] of a set X; in both cases a construction is given which shows that the (relevant) universe of discourse is finite.

In the remainder of the proof we use the notation $Res_=(R)$ for $Res(R) \cup R$; for finite sets R this will be finite if and only if $Res(R)$ is.

It is now a matter of describing the residual relation. When doing this we assume that the formulas P_1 and P_2 have the following normal forms:

$$P_1 \equiv (\textbf{empty} \wedge P_{e,1}) \vee \bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1})$$

$$P_2 \equiv (\textbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2})$$

and that these are indeed negation normal forms. Analysis of the relation $\lhd$ according to the structure of the formulas gives the following characterisation:

$$\begin{array}{rcll}
\textbf{True} & \lhd & \textbf{True} & \\
\textbf{True} & \lhd & \textbf{False} & \\
\textbf{True} & \lhd & p & \\
P' & \lhd & P \vee Q & if\ P' \lhd P \\
Q' & \lhd & P \vee Q & if\ Q' \lhd Q \\
\neg P' & \lhd & \neg(P \vee Q) & if\ P' \lhd P \\
\neg Q' & \lhd & \neg(P \vee Q) & if\ Q' \lhd Q \\
P & \lhd & \bigcirc P & \\
\neg P & \lhd & \neg \bigcirc P & \\
P'_{i,1};P_2 & \lhd & P_1;P_2 & \\
P'_{j,2} & \lhd & P_1;P_2 & \\
\neg(P'_{i,1};P_2) & \lhd & \neg(P_1;P_2) & \\
\neg P'_{j,2} & \lhd & \neg(P_1;P_2) & \\
P'_{i,1};(P_1\ \textbf{proj}\ P'_{j,2}) & \lhd & P_1\ \textbf{proj}\ P_2 & \\
\neg(P'_{i,1};(P_1\ \textbf{proj}\ P'_{j,2})) & \lhd & \neg(P_1\ \textbf{proj}\ P_2) &
\end{array}$$

it is now a matter of proving that $Res(R)$ is finite for all finite sets R. Given that $Res(R) = \bigcup_{P\in R} Res(\{P\})$ it is sufficient to look at residual sets of the form $Res(\{P\})$; where no comfusion can arise these will be written $Res(P)$.

We show that these sets of residuals are finite by an **induction** over the nesting of **proj** operators within the formula.

The **base case** examines **proj** -free formulas, and proceeds by induction over the size of the formula. The only non-trivial case involves 'chop'.

$$\begin{array}{rcl}
Res(P_1;P_2) & = & \bigcup\limits_{P'_{i,1} \lhd P_1} Res_=(P'_{i,1};P_2) \cup \\
& & \bigcup\limits_{P'_{j,2} \lhd P_2} Res_=(P'_{j,2})
\end{array}$$

which is a finite union of finite sets, and thus is finite. The case of a negated 'chop' formula is similar.

In the **induction step** we proceed by a subsidiary induction over the size of the formula. This induction works much as does the base case, except that the **proj** case needs to be addressed. Consider the formula

$$P_1 \ \mathbf{proj}\ P_2$$

In this formula the **proj** nesting in P_1 and P_2 will be smaller than in the formula itself, so that we may assume that the sets $Res(P_1)$ and $Res(P_2)$ are finite. How is the set of residuals characterised?

$$
\begin{aligned}
&Res(P_1 \ \mathbf{proj}\ P_2)\\
&\quad= \bigcup_{\substack{P'_{i,1} \lhd P_1 \\ P'_{j,2} \lhd P_2}} Res_{=}(P'_{i,1}\mathbf{;}(P_1 \ \mathbf{proj}\ P'_{j,2}))\\
&\quad= \{Q\mathbf{;}(P_1 \ \mathbf{proj}\ P'_{j,2}) \mid Q \in Res_{=}(P'_{i,1}),\ P'_{i,1} \lhd P_1,\ P'_{j,2} \lhd P_2\} \cup\\
&\qquad \bigcup_{P'_{j,2} \lhd P_2} Res_{=}(P_1 \ \mathbf{proj}\ P'_{j,2})\\
&\quad= \{Q\mathbf{;}(P_1 \ \mathbf{proj}\ P'_{j,2}) \mid Q \in Res_{=}(P'_{i,1}),\ P'_{i,1} \lhd P_1,\ P'_{j,2} \lhd P_2\} \cup\\
&\qquad \{P_1 \ \mathbf{proj}\ P'_{j,2} \mid P'_{j,2} \lhd P_2\} \cup\\
&\qquad \bigcup_{P'_{j,2} \lhd P_2} Res(P_1 \ \mathbf{proj}\ P'_{j,2}) \qquad\qquad (\dagger)
\end{aligned}
$$

Now, by our inductive hypothesis, the first two components of the union

$$
\begin{aligned}
&\{Q\mathbf{;}(P_1 \ \mathbf{proj}\ P'_{j,2}) \mid Q \in Res_{=}(P'_{i,1}),\ P'_{i,1} \lhd P_1,\ P'_{j,2} \lhd P_2\} \cup\\
&\{P_1 \ \mathbf{proj}\ P'_{j,2} \mid P'_{j,2} \lhd P_2\}
\end{aligned}
$$

are plainly finite, but the third is not so easily analysed.

As an aside, observe that the expressions P_1 **proj** $P'_{j,2}$ need be no less complex than P_1 **proj** P_2 itself. Consider two examples: when P_2 is **True** then the projection formula represents P_1^* which is a residual of itself (since **True** $\lhd$ **True**); in the case that P_2 is Q^* then P_1 **proj** $(Q'\mathbf{;}Q^*)$ will be a residual of P_1 **proj** P_2 whenever $Q' \lhd Q$.

The equation ($\dagger$) characterises $Res(P_1$ **proj** $P_2)$ in terms of other sets of residuals, perhaps including $Res(P_1$ **proj** $P_2)$ itself. Such recursive equations can be problematic unless we are able to deduce other properties that they might have.

In this case, consider the **proj** -nesting in the formulas P_1 **proj** $P'_{j,2}$; these formluas are no more complex than P_1 **proj** P_2 itself, since residuals of P_2 have no more complex **proj** nesting than P_2 itself. This means that the definitions of the residual sets of P_1 **proj** P_2 and all its residuals will form a *finite* collection of recursion equations of the same form as (†); such a collection of equations, defined from finite unions, will have finite (least) solutions. This argument shows that the residual set for a projection formula is finite; a similar argument applies to the negated case, and will complete the proof that the tableau procedure terminates in all cases. □

The following theorem shows that our tableau algorithm does indeed check satisfiability.

Theorem 2 *P is satisfiable if and only if the initial node of the graph generated by the tableau method just presented is not coloured.*

Proof

($\Longrightarrow$) For this we prove the contrapositive:

The initial node is coloured implies that P is unsatisfiable.

First we prove the general result that a node, n say, labelled $\{P_1, .., P_m\}$ is coloured implies $P_1 \wedge P_2 \wedge ... \wedge P_m$ is unsatisfiable. We prove this by induction on the structure of the graph. Thus, we assume the result for all successors of node n. Cases (E1) and (E3) are the base cases.

We consider in turn the possible reasons for node n being coloured.

- (E1) Clearly, if a proposition and its negation are both in T_n then the conjunction of the propositions in T_n is unsatisfiable.

- (E2.1) Assume n is coloured by (E2) but is not a state. Since n is not a state (and hence decomposition rules can be applied) we know that n must have had successors in the tableau construction. Thus, the sons of n were created using one of the tableau rules, $P \longrightarrow \{S_i\}$ say. From lemma 4 we know that $\Phi \cup \{P\}$ (where Φ is an arbitrary set of formulae) is satisfiable if and only if there exists a j such that $\Phi \cup \{\bigwedge S_j\}$ is satisfiable. However, the fact that no child node containing $\Phi \cup \{S_i\}$ is uncoloured implies that (by our inductive hypothesis) for all i, $\Phi \ \cup \ \{\bigwedge S_i\}$ is unsatisfiable and hence T_n is unsatisfiable.

- (E2.2) Assume n is coloured by (E2) and n is a state. Now by lemma 5 we know that n either contains $\bigcirc$(**True ; empty**) or **empty**. However, in this case it must be the former. This implies that node n would have had successor nodes resulting from application of the step rule, which have since been coloured. This implies that the conjunction of all the P's such that $\bigcirc P$ is in T_n is unsatisfiable (by the inductive hypothesis). By the semantics of $\bigcirc$, it follows that n is unsatisfiable.

- (E3) By the semantics of $\bigcirc$ and **empty** it is clear that for all P, **empty** $\wedge$ $\bigcirc P$ is unsatisfiable.

- (E4) Assume m is a pre-state and is coloured by (E4). We need to prove the following:

A node containing **empty** *cannot be reached from* T_n*, implies that* $P_1 \ \wedge \ P_2 \ \wedge ... \wedge \ P_m$ *is unsatisfiable.*

In order to prove this we use the terminology that for a particular formula in a particular pre-state a path in the tableau to an uncoloured node containing **empty** is called an accepting path. Thus, we can rephrase the property we need as,

There is no accepting path from T_n *implies that* $P_1 \ \wedge \ P_2 \ \wedge ... \wedge \ P_m$ *is unsatisfiable.*

We prove this in the contrapostive, i.e. we verify:

$$\exists \sigma \,.\, \sigma \models P_1 \ \wedge \ ... \ \wedge \ P_m \implies \forall i (1 \leq i \leq m) \,.\, P_i \textit{ has an accepting path.}$$

Now from our above assumption and properties of conjunction we know that $\forall i (1 \leq i \leq m) \,.\, \sigma \models P_i$. So, take an arbitrary $P \in \{ P_1, P_2, ..., P_m \}$; we need to show that P has an accepting path. We verify this by considering P at the normal form level, i.e

$$P \ \equiv \ (\textbf{empty} \ \wedge \ P_e) \ \vee \ \bigvee_i (P_i \ \wedge \ \bigcirc P_i')$$

and by performing induction on the length of σ.

Base Case. Assume $|\sigma| = 0$. Now from lemma 1 we know that $\sigma \models P_e$. In addition, on applying the decomposition rule to n we will be able to reach a state containing P_e and **empty**. The route to this state serves as an accepting path for P, as required.

Inductive Step. Assume $|\sigma| > 0$. Now by lemma 1 we know that $\sigma \models P_i$ and $(\sigma)^1 \models P_i'$ for some i. But, we know by our inductive hypothesis that an accepting path for P_i' exists. Furthermore, our graph construction rules ensure that there is a path from node n through a node containing P_i and a node containing P_i' (note: the step rule will be applied once in this path). So, we can concatenate this path with the accepting path for P_i' to get an accepting path for P, as required.

This completes our proof that a node n is coloured implies that the conjunction of its constituent formulae is unsatisfiable. The required property concerning the initial node is a special case of this result.

($\Longleftarrow$) We need to verify the property:

The initial node of the graph is not coloured implies P *is satisfiable*

First note that if the initial node of the graph is not coloured then an accepting path for P must exist in the (coloured) tableau. Our strategy will be to derive a model for P from this accepting path.

Assume P has an accepting path. Now from that path extract the *subaccepting* path which contains just the states in the accepting path, i.e. all pre-states are removed[3]. We will argue by induction on the size of this subaccepting path that an interval σ exists. In fact, the induction states that σ corresponds exactly to this subaccepting path.

Base Case. Assume the subaccepting path has length 0, i.e. there is just one state (which contains **empty**). However, if this is the case the state cannot contain any temporal formulas since if it did (E3) would have coloured the node. So, it must have been reached by applying non-temporal decomposition rules. These rules implement a standard propositional logic tableau and will only fail to generate a contradiction if P is satisfiable. The result follows.

Inductive Step. Assume P has a subaccepting path of length $n+1$. Now if we consider P in normal form:

$$P \;\equiv\; (P_e \;\wedge\; \textbf{empty}) \;\vee\; \bigvee_i (P_i \;\wedge\; \bigcirc P_i')$$

it is clear that the first node in the subaccepting path must "fulfil" P_i and contain $\bigcirc P_i'$ for some i. Furthermore, the tail of the subaccepting path will "fulfil" P_i' and will be of length n. But, now we have a subaccepting path of length n and we can apply our inductive hypothesis to give us an interval, σ say (where $|\sigma| = n$), and $\sigma \models P_i'$. In addition, by *cons*ing the first node of the original subaccepting path on the front of σ we obtain an interval of length $n+1$ that satisfies P.

The result follows. $\Box$

6 Axiomatization

Now we present out axiomatization of PITL, which we call Λ. In the conventional manner, $\vdash$ denotes provability.

Our logic contains the following axioms, where P, Q, R are arbitrary PITL formulae and p is a propositional formula.

$\vdash\; \psi$ where ψ is a propsitional tautology (PropTaut)

$\vdash\; \neg\textbf{empty} \Leftrightarrow \bigcirc\textbf{True}$ (NotEmpty)

$\vdash\; \bigcirc P \Rightarrow \neg\bigcirc\neg P$ (NextImplic)

$\vdash\; \bigcirc(P \Rightarrow Q) \Rightarrow \bigcirc P \Rightarrow \bigcirc Q$ (NextImp)

[3] Note that an accepting path must contain at least one state since the terminating node of the path will be a state. This is because all nodes containing non-elementary formulae must have successors as tableau rules can be applied and the colouring rules ensure that only final nodes can end accepting paths.

$\vdash\ \bigcirc P\ ;\ Q\ \Leftrightarrow\ \bigcirc(P\ ;\ Q)$	(NextChop)
$\vdash\ (P\ \vee\ Q)\ ;\ R\ \Leftrightarrow\ P\ ;\ R\ \vee\ Q\ ;\ R$	(ChopDistr1)
$\vdash\ P\ ;\ (Q\ \vee\ R)\ \Leftrightarrow\ P\ ;\ Q\ \vee\ P\ ;\ R$	(ChopDistr2)
$\vdash\ P\ ;\ (Q\ ;\ R)\ \Leftrightarrow\ (P\ ;\ Q)\ ;\ R$	(ChopAssoc)
$\vdash\ (p\ \wedge\ P)\ ;\ Q\ \Leftrightarrow\ p\ \wedge\ (P\ ;\ Q)$	(ChopProp)
$\vdash\ \mathbf{empty}\ ;\ P\ \Leftrightarrow\ P\ ;\ \mathbf{empty}\ \Leftrightarrow\ P$	(ChopIdent)
$\vdash\ P\ \mathbf{proj}\ \mathbf{empty}\ \Leftrightarrow\ \mathbf{empty}$	(ProjZero)
$\vdash\ P\ \mathbf{proj}\ (Q\ \vee\ R)\ \Leftrightarrow\ (P\ \mathbf{proj}\ Q)\ \vee\ (P\ \mathbf{proj}\ R)$	(ProjDistr1)
$\vdash\ P\ \mathbf{proj}\ (p\ \wedge\ Q)\ \Leftrightarrow\ p\ \wedge\ P\ \mathbf{proj}\ Q$	(ProjProp)
$\vdash\ P\ \mathbf{proj}\ \bigcirc R\ \Leftrightarrow\ (P\ \wedge\ \neg\mathbf{empty})\ ;\ P\ \mathbf{proj}\ R$	(ProjInduc)

In addition, the logic contains the following inference rules where, in standard fashion, the notation $P_1, ..., P_n \vdash P$ states that P is a provable consequence of the assumptions $P_1, ..., P_n$. We use Γ to abbreviate such sequences.

$$\text{(NextInt)} \qquad \frac{P_1, ..., P_n \vdash P}{\bigcirc P_1, ..., \bigcirc P_n \vdash \bigcirc P}$$

$$\text{(MP)} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash P \Rightarrow Q}{\Gamma \vdash Q}$$

$$\text{(Finite-k)} \qquad \frac{\vdash P \Rightarrow \bigcirc^k P}{\vdash \neg P}$$

We leave it as an exercise for the reader to verify that the axioms and rules of our system are sound. The following lemmas verify some simple consequences of our axiomatization.

Lemma 7 $\quad\vdash\ \mathbf{True}\ \Leftrightarrow\ \mathbf{empty}\ \vee\ \bigcirc\mathbf{True}$

Proof

1.	$\vdash$	$\bigcirc\mathbf{True}\ \vee\ \neg\bigcirc\mathbf{True}$	(PropTaut - Excluded Middle)
2.	$\vdash$	$\bigcirc\mathbf{True}\ \vee\ \neg\neg\mathbf{empty}$	(1., NotEmpty)
3.	$\vdash$	$\bigcirc\mathbf{True}\ \vee\ \mathbf{empty}$	(2., PropTaut)

□

Lemma 8 $\quad\vdash\ \mathbf{True}\ ;\ \mathbf{empty}$

Proof

1.	$\vdash$	$\mathbf{True}$	(PropTaut)
2.	$\vdash$	$\mathbf{True}\ ;\ \mathbf{empty}$	(1., ChopIdent)

□

Lemma 9 $\quad\vdash\ Q\ \mathbf{proj}\ (\mathbf{empty}\ \wedge\ p)\ \Leftrightarrow\ \mathbf{empty}\ \wedge\ p$

Proof

$$
\begin{array}{ll}
& Q \ \mathbf{proj} \ (\mathbf{empty} \ \wedge \ p) \\
\Leftrightarrow & \{ \ (\text{ProjProp}) \ \} \\
& p \ \wedge \ (Q \ \mathbf{proj} \ \mathbf{empty}) \\
\Leftrightarrow & \{ \ (\text{ProjZero}) \ \} \\
& p \ \wedge \ \mathbf{empty} \quad \Box
\end{array}
$$

Lemma 10 $\vdash \ \bigcirc P \wedge \bigcirc Q \Leftrightarrow \bigcirc(P \wedge Q)$

Proof

1.	$P \wedge Q \vdash P$	(PropTaut)
2.	$\bigcirc(P \wedge Q) \vdash \bigcirc P$	(1., NextInt)
3.	$\vdash \bigcirc(P \wedge Q) \Rightarrow \bigcirc P$	(2., PropTaut)
4.	$P \wedge Q \vdash Q$	(PropTaut)
5.	$\bigcirc(P \wedge Q) \vdash \bigcirc Q$	(2., NextInt)
6.	$\vdash \bigcirc(P \wedge Q) \Rightarrow \bigcirc Q$	(3., PropTaut)
7.	$\vdash \bigcirc(P \wedge Q) \Rightarrow \bigcirc P \wedge \bigcirc Q$	(3.,6., PropTaut)
8.	$P, Q \vdash P \wedge Q$	(PropTaut)
9.	$\bigcirc P, \bigcirc Q \vdash \bigcirc(P \wedge Q)$	(8., NextInt)
10.	$\vdash \bigcirc P \Rightarrow (\bigcirc Q \Rightarrow \bigcirc(P \wedge Q))$	(9., PropTaut)
11.	$\vdash \bigcirc P \wedge \bigcirc Q \Rightarrow \bigcirc(P \wedge Q)$	(10., PropTaut)
12.	$\vdash \bigcirc P \wedge \bigcirc Q \Leftrightarrow \bigcirc(P \wedge Q)$	(7.,11., PropTaut)

$\Box$

Lemma 11

(i) $\vdash \ \bigcirc P \wedge \neg\mathbf{empty} \Leftrightarrow \bigcirc P$

(ii) $\vdash \ \mathbf{empty} \wedge \bigcirc P \Leftrightarrow \mathbf{False}$

(iii) $\vdash \ \bigcirc\mathbf{False} \Leftrightarrow \mathbf{False}$

Proof

(i)

1.	$P \vdash P$	(PropTaut)
2.	$P, \mathbf{True} \vdash P$	(1., PropTaut)
3.	$\bigcirc P, \bigcirc\mathbf{True} \vdash \bigcirc P$	(2., NextInt)
4.	$\vdash \bigcirc P \Rightarrow (\bigcirc\mathbf{True} \Rightarrow \bigcirc P)$	(3., PropTaut)
5.	$\vdash (\bigcirc P \wedge \bigcirc\mathbf{True}) \Rightarrow \bigcirc P$	(4., PropTaut)
6.	$\vdash (\bigcirc P \wedge \neg\mathbf{empty}) \Rightarrow \bigcirc P$	(5., NotEmpty)
7.	$P \vdash P \wedge \mathbf{True}$	(PropTaut)
8.	$\bigcirc P \vdash \bigcirc(P \wedge \mathbf{True})$	(7., NextInt)

9. $\vdash \bigcirc P \;\Rightarrow\; \bigcirc(P \;\wedge\; \mathbf{True})$ (8., PropTaut)
10. $\vdash \bigcirc P \;\Rightarrow\; (\bigcirc P \;\wedge\; \bigcirc\mathbf{True})$ (9., lemma 10)
11. $\vdash \bigcirc P \;\Rightarrow\; (\bigcirc P \;\wedge\; \neg\mathbf{empty})$ (10,. NotEmpty)

12. $\vdash \bigcirc P \;\Leftrightarrow\; (\bigcirc P \;\wedge\; \neg\mathbf{empty})$ (6.,11., PropTaut)

(ii)
1. $\vdash\; \mathbf{empty} \;\wedge\; \bigcirc P \;\Leftrightarrow\; \mathbf{empty} \;\wedge\; \bigcirc P$ (PropTaut)
2. $\vdash\; \mathbf{empty} \;\wedge\; \bigcirc P \;\Leftrightarrow\; \mathbf{empty} \;\wedge\; (\bigcirc P \;\wedge\; \neg\mathbf{empty})$ (1., *(i)* of this lemma)
3. $\vdash\; \mathbf{empty} \;\wedge\; \bigcirc P \;\Leftrightarrow\; \mathbf{False}$ (3., PropTaut)

(iii)
1. $\bigcirc\mathbf{False} \;\vdash\; \bigcirc\mathbf{False}$ (PropTaut)
2. $\bigcirc\mathbf{False} \;\vdash\; \bigcirc\mathbf{False} \;\wedge\; \neg\mathbf{empty}$ (1., *(i)* of this lemma)
3. $\bigcirc\mathbf{False} \;\vdash\; \bigcirc\mathbf{False} \;\wedge\; \bigcirc\mathbf{True}$ (2., NotEmpty)
4. $\bigcirc\mathbf{False} \;\vdash\; \bigcirc\mathbf{False} \;\wedge\; \neg\bigcirc\mathbf{False}$ (3., NextImplic)
5. $\bigcirc\mathbf{False} \;\vdash\; \mathbf{False}$ (4., PropTaut)

6. $\mathbf{False} \;\vdash\; \bigcirc\mathbf{False}$ (PropTaut)

7. $\vdash\; \mathbf{False} \;\Leftrightarrow\; \bigcirc\mathbf{False}$ (5.,6., PropTaut)

$\square$

Thus far in this subsection we have examined some of the formulae which are conseqences of the axiomatisation Λ. We can also show that some deduction rules are derivable.

Lemma 12 *The rule*

$$\frac{\vdash P}{\vdash \square P}$$

is derivable in the system Λ.

Proof

Take ϕ to be the formula $\neg\square P$, and assume that $\vdash P$. $\square$ is defined in the system from 'chop' and so $\neg\square P$ is in fact the formula $(\mathbf{True};\neg P)$. We want to derive $\phi \Rightarrow \bigcirc\phi$ since then we can apply (Finite-1) to give $\vdash \neg\phi$ which, since $\neg\phi$ is equivalent to $\square P$, gives $\vdash \square P$.

Using the deduction theorem of Section 7.2 it is sufficient to prove $\phi \vdash \bigcirc\phi$. By lemma 1 we have that

$$(\mathbf{True};\neg\phi) \equiv \neg\phi \vee \bigcirc(\mathbf{True};\neg\phi)$$

but given a proof that $\vdash \phi$ it is easy to deduce that

$$(\mathbf{True};\neg\phi) \vdash \bigcirc(\mathbf{True};\neg\phi)$$

as required. This gives the result $\square P$ and completes the proof. $\square$

Other derivable rules are dicussed in Section 8 after the completeness theorem is proved in the next section.

7 Completeness

There are three stages to our proof of completeness. The first is to show that our normal form expansions corresponds to theorems of Λ. The second is to verify that the deduction theorem holds for our particular system and the third is to prove the completeness of the axiomatization with reference to our tableau decision procedure. This latter is couched in terms of our normal forms, hence the necessity for the first stage; the proof of completeness uses the deduction theorem.

We consider these three stages in separate subsections.

7.1 Normal Form Verification

The following lemmas systemmatically show that each of our normal form expansions is provable from Λ. Once again assume the following normal form expansions:-

$$P_1 \equiv (\mathbf{empty} \wedge P_{e,1}) \vee \bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1})$$

$$P_2 \equiv (\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2})$$

$$P \equiv (\mathbf{empty} \wedge P_e) \vee \bigvee_i (P_i \wedge \bigcirc P'_i)$$

Lemma 13
$\vdash \ p \Leftrightarrow (\mathbf{empty} \wedge p) \vee (p \wedge \bigcirc\mathbf{True})$

where p is a propositional variable or **False**.

Proof

$$\begin{array}{ll} & p \\ \Leftrightarrow & \{ \text{ PropTaut } \} \\ & p \wedge \mathbf{True} \\ \Leftrightarrow & \{ \text{ Lemma 7 } \} \\ & p \wedge (\mathbf{empty} \vee \bigcirc\mathbf{True}) \\ \Leftrightarrow & \{ \text{ PropTaut } \} \\ & (p \wedge \mathbf{empty}) \vee (p \wedge \bigcirc\mathbf{True}) \quad \Box \end{array}$$

Lemma 14

$$\vdash \ P_1 \vee P_2 \Leftrightarrow (\mathbf{empty} \wedge (P_{e,1} \vee P_{e,2})) \vee \bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1}) \vee \bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2})$$

Proof
Repeated application of $\wedge$ and $\vee$ distributivity and symmetry. $\Box$

Lemma 15

$$\vdash \quad (\textbf{empty} \wedge P_e) \vee \bigvee_i (P_i \wedge \bigcirc P'_i) \Leftrightarrow (\textbf{empty} \wedge P_e) \vee \bigvee_j (P_j \wedge \bigcirc P'_j)$$

where, $\langle P_j \rangle_j$ partitions **True** *and is obtained from* P *by the construction presented in section 3. In addition,*

$$\vdash \quad \neg((\textbf{empty} \wedge P_e) \vee \bigvee_j (P_j \wedge \bigcirc P'_j)) \Leftrightarrow ((\textbf{empty} \wedge \neg P_e) \vee \bigvee_j (P_j \wedge \bigcirc \neg P'_j))$$

where, $\langle P_j \rangle_j$ partitions **True***.*

Proof
Propositional reasoning can justify this result. □

Lemma 16

$$\vdash \quad \textbf{empty} \Leftrightarrow (\textbf{empty} \wedge \textbf{True}) \vee (\textbf{False} \wedge \bigcirc \textbf{True})$$

Proof

$$\begin{array}{ll} & \textbf{empty} \\ \Leftrightarrow & \{ \text{ PropTaut } \} \\ & \textbf{empty} \wedge \textbf{True} \\ \Leftrightarrow & \{ \text{ PropTaut } \} \\ & (\textbf{empty} \wedge \textbf{True}) \vee (\textbf{False}) \\ \Leftrightarrow & \{ \text{ PropTaut } \} \\ & (\textbf{empty} \wedge \textbf{True}) \vee (\textbf{False} \wedge \bigcirc \textbf{True}) \quad \Box \end{array}$$

Lemma 17

$$\begin{array}{lll} \vdash \quad P_1 \,;\, P_2 & \Leftrightarrow & (\textbf{empty} \wedge (P_{e,1} \wedge P_{e,2})) \vee \\ & & \bigvee_i (P_{i,1} \wedge \bigcirc (P'_{i,1} \,;\, P_2)) \vee \\ & & \bigvee_j ((P_{e,1} \wedge P_{j,2}) \wedge \bigcirc P'_{j,2}) \end{array}$$

Proof

$$\begin{array}{ll} & ((\textbf{empty} \wedge P_{e,1}) \vee \bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1})) \,; \\ & ((\textbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2})) \\ \Leftrightarrow & \{ \text{ ChopDistr1 } \} \\ & ((\textbf{empty} \wedge P_{e,1}) \,; ((\textbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2}))) \vee \\ & (\bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1}) \,; P_2) \\ \Leftrightarrow & \{ \text{ ChopDistr2 } \} \end{array}$$

$$
\begin{array}{ll}
& ((\mathbf{empty} \wedge P_{e,1})\,\mathbf{;}\,(\mathbf{empty} \wedge P_{e,2})) \vee \\
& ((\mathbf{empty} \wedge P_{e,1})\,\mathbf{;}\,\bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2})) \vee \\
& (\bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1})\,\mathbf{;}\,P_2) \\
\Leftrightarrow & \{\ \text{ChopDistr1 ; ChopDistr2}\ \} \\
& ((\mathbf{empty} \wedge P_{e,1})\,\mathbf{;}\,(\mathbf{empty} \wedge P_{e,2})) \vee \\
& \bigvee_j ((\mathbf{empty} \wedge P_{e,1})\,\mathbf{;}\,(P_{j,2} \wedge \bigcirc P'_{j,2})) \vee \\
& \bigvee_i ((P_{i,1} \wedge \bigcirc P'_{i,1})\,\mathbf{;}\,P_2) \\
\Leftrightarrow & \{\ \text{ChopProp}\ \} \\
& (P_{e,1} \wedge (\mathbf{empty}\,\mathbf{;}\,(\mathbf{empty} \wedge P_{e,2}))) \vee \\
& \bigvee_j ((\mathbf{empty} \wedge P_{e,1})\,\mathbf{;}\,(P_{j,2} \wedge \bigcirc P'_{j,2})) \vee \\
& \bigvee_i ((P_{i,1} \wedge \bigcirc P'_{i,1})\,\mathbf{;}\,P_2) \\
\Leftrightarrow & \{\ \text{ChopIdent}\ \} \\
& (P_{e,1} \wedge (\mathbf{empty} \wedge P_{e,2})) \vee \\
& \bigvee_j ((\mathbf{empty} \wedge P_{e,1})\,\mathbf{;}\,(P_{j,2} \wedge \bigcirc P'_{j,2})) \vee \\
& \bigvee_i ((P_{i,1} \wedge \bigcirc P'_{i,1})\,\mathbf{;}\,P_2) \\
\Leftrightarrow & \{\ \text{PropTaut ; ChopProp ; ChopIdent}\ \} \\
& ((P_{e,1} \wedge P_{e,2}) \wedge \mathbf{empty}) \vee \\
& \bigvee_j (P_{e,1} \wedge P_{j,2} \wedge \bigcirc P'_{j,2}) \vee \\
& \bigvee_i ((P_{i,1} \wedge \bigcirc P'_{i,1})\,\mathbf{;}\,P_2) \\
\Leftrightarrow & \{\ \text{ChopProp}\ \} \\
& ((P_{e,1} \wedge P_{e,2}) \wedge \mathbf{empty}) \vee \\
& \bigvee_j (P_{e,1} \wedge P_{j,2} \wedge \bigcirc P'_{j,2}) \vee \\
& \bigvee_i (P_{i,1} \wedge (\bigcirc P'_{i,1}\,\mathbf{;}\,P_2)) \\
\Leftrightarrow & \{\ \text{NextChop}\ \} \\
& ((P_{e,1} \wedge P_{e,2}) \wedge \mathbf{empty}) \vee \\
& \bigvee_j (P_{e,1} \wedge P_{j,2} \wedge \bigcirc P'_{j,2}) \vee \\
& \bigvee_i (P_{i,1} \wedge \bigcirc (P'_{i,1}\,\mathbf{;}\,P_2)) \quad \Box
\end{array}
$$

Lemma 18

$$
\begin{array}{lll}
\vdash \ P_1 \ \mathbf{proj}\ P_2 & \Leftrightarrow & (\mathbf{empty} \wedge P_{e,2}) \ \vee \\
& & \bigvee_i \bigvee_j ((P_{i,1} \wedge P_{j,2}) \wedge \bigcirc (P'_{i,1}\,\mathbf{;}\,(P_1 \ \mathbf{proj}\ P'_{j,2})))
\end{array}
$$

Proof

P_1 **proj** $((\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2}))$

$\Leftrightarrow$ { ProjDistr1 }

$(P_1 \ \mathbf{proj}\ (\mathbf{empty} \wedge P_{e,2})) \vee (P_1 \ \mathbf{proj} \bigvee_j (P_{j,2} \wedge \bigcirc P'_{j,2}))$

$\Leftrightarrow$ { ProjDistr1 }

$(P_1 \ \mathbf{proj}\ (\mathbf{empty} \wedge P_{e,2})) \vee \bigvee_j (P_1 \ \mathbf{proj}\ (P_{j,2} \wedge \bigcirc P'_{j,2}))$

$\Leftrightarrow$ { lemma 9 }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_1 \ \mathbf{proj}\ (P_{j,2} \wedge \bigcirc P'_{j,2}))$

$\Leftrightarrow$ { ProjProp }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge (P_1 \ \mathbf{proj}\ \bigcirc P'_{j,2}))$

$\Leftrightarrow$ { ProjInduc }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge (P_1 \wedge \neg\mathbf{empty})\ \mathbf{;}\ (P_1 \ \mathbf{proj}\ P'_{j,2}))$

$\Leftrightarrow$ { Expansion of P_1 }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge (((\mathbf{empty} \wedge P_{e,1}) \vee$
$\bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1})) \wedge \neg\mathbf{empty})\ \mathbf{;}\ (P_1 \ \mathbf{proj}\ P'_{j,2}))$

$\Leftrightarrow$ { PropTaut }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge ((\mathbf{empty} \wedge P_{e,1} \wedge \neg\mathbf{empty}) \vee$
$\bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1} \wedge \neg\mathbf{empty}))\ \mathbf{;}\ (P_1 \ \mathbf{proj}\ P'_{j,2}))$

$\Leftrightarrow$ { PropTaut }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge$
$\bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1} \wedge \neg\mathbf{empty})\ \mathbf{;}\ (P_1 \ \mathbf{proj}\ P'_{j,2}))$

$\Leftrightarrow$ { Lemma 11 }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge \bigvee_i (P_{i,1} \wedge \bigcirc P'_{i,1})\ \mathbf{;}\ (P_1 \ \mathbf{proj}\ P'_{j,2}))$

$\Leftrightarrow$ { ChopDistr1 }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j (P_{j,2} \wedge \bigvee_i ((P_{i,1} \wedge \bigcirc P'_{i,1})\ \mathbf{;}\ (P_1 \ \mathbf{proj}\ P'_{j,2})))$

$\Leftrightarrow$ { PropTaut }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j \bigvee_i (P_{j,2} \wedge (P_{i,1} \wedge \bigcirc P'_{i,1})\ \mathbf{;}\ (P_1 \ \mathbf{proj}\ P'_{j,2}))$

$\Leftrightarrow$ { ChopProp }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j \bigvee_i (P_{j,2} \wedge P_{i,1} \wedge (\bigcirc P'_{i,1}\ \mathbf{;}\ (P_1 \ \mathbf{proj}\ P'_{j,2})))$

$\Leftrightarrow$ { NextChop }

$(\mathbf{empty} \wedge P_{e,2}) \vee \bigvee_j \bigvee_i (P_{j,2} \wedge P_{i,1} \wedge \bigcirc(P'_{i,1}\ \mathbf{;}\ (P_1 \ \mathbf{proj}\ P'_{j,2})))$ $\Box$

7.2 The deduction theorem

The deduction theorem gives a mechanism by which implications can be introduced, and thus complements *modus ponens* by which implications are eliminated. Proving the deduction theorem (DT) is a standard exercise; a useful reference for the propositional DT and also propositional tableaux is [2].

Theorem 3 ***The Deduction Theorem*** *For all PITL formulae ϕ and ψ and for all sets of PITL formulae Γ, if*

$$\Gamma, \phi \vdash \psi$$

then

$$\Gamma \vdash \phi \Rightarrow \psi$$

Proof

In order to avoid confusion between the object and meta levels we will use the term **derivation** for a proof in the logic Λ. We assume that derivations are presented as in the previous section: a derivation is a linear sequence of lines; each line introduces an assumption or an axiom, or is justified by applying one of the rules of the logic to one or more earlier lines.

The proof of the deduction theorem proceeds by induction over the lengths of derivations in the logic Λ. The cases in which the last line of the derivation consists of a use of

- an assumption;
- an axiom;
- the rule of Modus Ponens

are standard; see [2] for example. Our proof here addresses the other rules of the logic

Case (NextInt):

Suppose that the last line of the derivation is an application of the rule (NextInt). It is thus an application of

$$\frac{\Gamma, \phi \vdash \psi}{\bigcirc\Gamma, \bigcirc\phi \vdash \bigcirc\psi}$$

and we are required to find a derivation of $\bigcirc\Gamma \vdash \bigcirc\phi \Rightarrow \bigcirc\psi$.

The derivation of $\Gamma, \phi \vdash \psi$ is shorter and thus the deduction theorem applies to this result, giving a derivation of

$$\Gamma \vdash \phi \Rightarrow \psi$$

Applying (NextInt) to this gives a derivation of

$$\bigcirc\Gamma \vdash \bigcirc(\phi \Rightarrow \psi)$$

The Axiom (NextImp) has the instance

$$\bigcirc(\phi \Rightarrow \psi) \Rightarrow \bigcirc\phi \Rightarrow \bigcirc\psi$$

Applying Modus Ponens to this and the derivation of $\bigcirc(\phi \Rightarrow \psi)$ gives

$$\bigcirc\Gamma \vdash \bigcirc\phi \Rightarrow \bigcirc\psi$$

as required.

Case Other rules:
The other rules of the system have the form

$$\frac{\vdash \psi}{\vdash \psi'}$$

It will never be the case that a rule of this form is the last rule to be applied in a derivation of $\Gamma, \phi \vdash \psi$ since the antecedent of the rule requires that it is only applied in a situation that the derivation in question has no assumptions.

This completes the cases and thus the proof of the deduction theorem for the logic Λ. $\Box$

7.3 Completeness Theorem

Now we prove the completeness result that we seek.

Theorem 4 *The logic Λ is complete, i.e. for all PITL formulae ϕ,*

$$\models \phi \quad \textit{if and only if} \quad \vdash \phi$$

Proof
The "top level" line of reasoning of the proof is,

$\models \phi$
iff { Definition of $\models$; logic }
$\neg\exists\sigma \,.\, \sigma \models \neg\phi$
iff { Theorem 2 }
The tableau algorithm colours the inital node $\{\neg\phi, \Diamond\mathbf{empty}\}$.
iff { (*) }
$\neg\phi \ \vdash \ \psi \ \wedge \ \neg\psi$
iff { PropTaut (deduction theorem) }
$\vdash \ \neg\phi \ \Rightarrow \ (\psi \ \wedge \ \neg\psi)$
iff { PropTaut }
$\vdash \ (\phi \ \vee \ \psi) \ \wedge \ (\phi \ \vee \ \neg\psi)$

iff $\{$ PropTaut $\}$

$\vdash (\phi \Leftarrow \neg\psi) \wedge (\phi \Leftarrow \psi)$

iff $\{$ PropTaut - $(p \Rightarrow r) \wedge (\neg p \Rightarrow r) \Leftrightarrow r$ $\}$

$\vdash \phi$

All of these steps are already justified apart from (*) and it is this step we turn to now. In fact, we prove the following more general result:-

> For any node, n say, in the tableau, where $T_n = \{\phi_1, ..., \phi_m\}$, if the node is coloured, then $\phi_1 \wedge ... \wedge \phi_m$ is contradictory, i.e. $\exists\psi . \phi_1 \wedge ... \wedge \phi_m \vdash \psi \wedge \neg\psi$.

We prove this result by induction over the strongly connected components of the tableau[4].

Base Cases. These are given by (E1) and (E3) in the tableau colouring algorithm.

(E1) $\Phi, p, \neg p \vdash p \wedge \neg p$ immediately (where $\Phi, p, \neg p = \phi_1, ..., \phi_m$);

(E3) This is straightforward as well.

$$\begin{array}{ll} & \Phi, \mathbf{empty}, \bigcirc p \vdash \mathbf{empty} \wedge \bigcirc p \\ \Leftrightarrow & \{ \text{ lemma 11 (ii) } \} \\ & \Phi, \mathbf{empty}, \bigcirc p \vdash \mathbf{False} \end{array}$$

Inductive Cases. These are given by (E2) and (E4).

(E2) There are two cases to consider here - if the node is a pre-state or a state. We consider these in turn.

Pre-state. If n is a pre-state then all its successors will be labelled according to one of the decomposition rules. Also, note that since n does not contain **empty**, it will have successors.

Now it is straightforward to show that for all the propositional decomposition rules, if the resulting sucessor nodes are contradictory (this is our inductive hypothesis) then n is also contradictory. For example, if $T_n = \{\phi, \phi_1, ..., \phi_s\}$ and $\phi = \neg(P_1 \wedge P_2)$ then there exist successor nodes, r and r' say, such that,

$$T_r = \{\neg P_1, \phi_1, ..., \phi_s\} \text{ and } T_{r'} = \{\neg P_2, \phi_1, ..., \phi_s\}$$

Now we can argue from our inductive hypothesis as follows:-

$$\neg P_1, \phi_1, ..., \phi_s \vdash \mathbf{False} \;\text{ and }\; \neg P_2, \phi_1, ..., \phi_s \vdash \mathbf{False}$$

[4]This is in the sense that we treate strongly connected components as single items.

iff { Deduction theorem }

$\phi_1, ..., \phi_s \ \vdash \ (\neg P_1 \Rightarrow \textbf{False})$ and $\phi_1, ..., \phi_s \ \vdash \ (\neg P_2 \Rightarrow \textbf{False})$

And we can argue as follows from T_n,

$\neg(P_1 \ \wedge \ P_2), \phi_1, ..., \phi_s \ \vdash \ \neg(P_1 \ \wedge \ P_2)$

iff { PropTaut }

$\neg(P_1 \ \wedge \ P_2), \phi_1, ..., \phi_s \ \vdash \ \neg P_1 \ \vee \ \neg P_2$

iff { PropTaut ; above derivation from inductive hypothesis }

$\neg(P_1 \ \wedge \ P_2), \phi_1, ..., \phi_s \ \vdash \ \textbf{False}$

Furthermore, if the rule applied decomposes $\neg \bigcirc P$ (into **empty** or $\bigcirc \neg P$) we can argue as follows:

$\neg \bigcirc P, \phi_1, ..., \phi_s \ \vdash \ \neg \bigcirc P$

iff { lemma 11 (i) }

$\neg \bigcirc P, \phi_1, ..., \phi_s \ \vdash \ \neg(\bigcirc P \ \wedge \ \neg \textbf{empty})$

iff { PropTaut }

$\neg \bigcirc P, \phi_1, ..., \phi_s \ \vdash \ \neg \bigcirc P \ \vee \ \textbf{empty}$

implies{ NextImplic }

$\neg \bigcirc P, \phi_1, ..., \phi_s \ \vdash \ \bigcirc \neg P \ \vee \ \textbf{empty}$

iff { Inductive hypothesis }

$\neg \bigcirc P, \phi_1, ..., \phi_s \ \vdash \ \textbf{False}$

Finally, if the rule applied decomposes a chop, projection or one of their respective negations, then we can use the lemmas proved in the previous section to obtain the result required. For example, if $T_n = \{\phi, \phi_1, ..., \phi_s\}$ and $\phi = P_1 \ ; P_2$ then we can argue as follows:

$P_1 \ ; P_2, \phi_1, ..., \phi_s \ \vdash \ P_1 \ ; P_2$

iff { lemma 17 }

$P_1 \ ; P_2, \phi_1, ..., \phi_s \ \vdash$
$(\textbf{empty} \ \wedge \ (P_{e,1} \ \wedge \ P_{e,2})) \ \vee$
$\bigvee_i (P_{i,1} \ \wedge \ \bigcirc(P'_{i,1} \ ; P_2)) \ \vee$
$\bigvee_j ((P_{e,1} \ \wedge \ P_{j,2}) \ \wedge \ \bigcirc P'_{j,2})$

iff { Inductive Hypothesis }

$P_1 \ ; P_2, \phi_1, ..., \phi_s \ \vdash \ \textbf{False}$

which is as required.

State. If n is a state then it will have one successor, say m. Now assume that $T_n = \{\alpha_1, ..., \alpha_s, \bigcirc\phi_1, ..., \bigcirc\phi_r\}$. We can argue from our inductive hypothesis as follows:

$$
\begin{array}{ll}
 & \phi_1, ..., \phi_r \ \vdash \ \psi_1 \ \wedge \ \neg\psi_2 \\
\text{iff} & \{ \ \text{NextInt} \ \} \\
 & \bigcirc\phi_1, ..., \bigcirc\phi_r \ \vdash \ \bigcirc(\psi_1 \ \wedge \ \neg\psi_2) \\
\text{iff} & \{ \ \text{Lemma 10} \ \} \\
 & \bigcirc\phi_1, ..., \bigcirc\phi_r \ \vdash \ \bigcirc\psi_1 \ \wedge \ \bigcirc\neg\psi_2 \\
\text{implies} & \{ \ \text{NextImplic} \ ; \text{PropTaut} \ \} \\
 & \alpha_1, ..., \alpha_s, \bigcirc\phi_1, ..., \bigcirc\phi_r \ \vdash \ \bigcirc\psi_1 \ \wedge \ \neg\bigcirc\psi_2
\end{array}
$$

which is as required.

(E4) Consider node n eliminated by this rule and assume WLOG that there are no uncoloured strongly connected components (other then the one that n could be in) reachable from n in the tableau. This is a reasonable assumption since at any point in the tableaus colouring it must have a set of minimal uncoloured SCC and colouring will reveal a new set of minimal uncoloured SCC. Thus, all nodes eliminated by this rule will eventually be made minimal by the colouring process.

We consider two cases dependent upon whether or not n is in a SCC.

1. If n is not in a SCC then we know the tableau reachable from it is a tree, since we have already assumed there are no SCCs below n. Now since there is no uncoloured final node reachable in the tree (by the definition of (E4)) the tree will progressively be coloured by application of rule (E2) until n has no uncoloured successors at which point (E4) reduces to rule (E2) and hence we can apply our (E2) argument to obtain that,

   $$\phi_1, ..., \phi_m \ \vdash \ \psi \ \wedge \ \neg\psi$$

   which is as required.

2. Now if n is in a SCC we have to work a little harder. Firstly, notice that there cannot be any uncoloured nodes reachable from n apart from those in the SCC. This is because these nodes would be coloured by rule (E2). Furthermore, our inductive hypothesis ensures that all such coloured nodes are contraditory.

   Now a finite set of states must be reachable from n (call them $m_1, ..., m_r$) and if $T_n = \{\phi_1, ..., \phi_m\}$ and $T_{m_i} = \{\psi_{1,i}, ..., \psi_{z,i}\}$ we know from "provability" of our normal forms / tableau rules that,

   $$\vdash \ \bigwedge_k \phi_k \ \Leftrightarrow \ \bigvee_h \bigwedge_r \psi_{r,h}$$

and furthermore each $\{\psi_{i,1}, ..., \psi_{z,i}\}$ will be partitioned into two sets - point formulae and formulae with $\bigcirc$ as the outermost connective. This is because each T_{m_i} is a state.
Thus, we can argue as follows:

$$
\begin{aligned}
& \bigwedge_k \phi_k \\
\Leftrightarrow \quad & \{ \text{ Derived above } \} \\
& \bigvee_h \bigwedge_r \psi_{r,h} \\
\Leftrightarrow \quad & \{ \text{ Above argument } \} \\
& \bigvee_h (\bigwedge_p \beta_{p,h} \ \wedge \ \bigwedge_q \bigcirc \alpha_{q,h}) \\
\Rightarrow \quad & \{ \text{ PropTaut, i.e. } A \ \wedge \ B \Rightarrow A \ \vee \ C, E \Rightarrow F \ \wedge \ G \Rightarrow F \ \vdash E \ \vee \ G \ \Rightarrow \ F \ \} \\
& \bigvee_h \bigwedge_q \bigcirc \alpha_{q,h} \\
\Leftrightarrow \quad & \{ \text{ Lemma 10 } \} \\
& \bigvee_h \bigcirc \bigwedge_q \alpha_{q,h}
\end{aligned}
$$

Now notice that for all h the set $\{\alpha_{q,h}\}$ labels a pre-state and there are two possibilities for the pre-state - it is either below the SCC or it is a node in the SCC. If it is the former then using our inductive hypothesis we can argue as follows:-

$$
\begin{aligned}
& \vdash \ \bigwedge_q \alpha_{q,h} \ \Rightarrow \ \mathbf{False} \\
\Leftrightarrow \quad & \{ \text{ NextInt } \} \\
& \vdash \ \bigcirc \bigwedge_q \alpha_{q,h} \ \Rightarrow \ \bigcirc \mathbf{False} \\
\Leftrightarrow \quad & \{ \text{ Lemma 11 (iii) } \} \\
& \vdash \ \bigcirc \bigwedge_q \alpha_{q,h} \ \Rightarrow \ \mathbf{False}
\end{aligned}
$$

Thus, if we put our last two derivations together we obtain,

$$\vdash \ \bigwedge_k \phi_k \ \Rightarrow \ \bigvee_w \bigcirc \bigwedge_q \alpha_{q,w}$$

where for all w the set $\{\alpha_{q,w}\}$ is a pre-state in the SCC.
Now since we are in a SCC the sets $\{\alpha_{q,w}\}$ are either equal to $\{\phi_k\}$ or after a series of similar steps will reach $\{\phi_k\}$. Furthermore, all derivations of all $\{\alpha_{q,w}\}$ are also provable consequences of $\bigwedge_q \alpha_{q,w}$ using similar arguments to those that we have just used and all these consequences can be related back to $\bigcirc \bigwedge_q \alpha_{q,w}$ (and hence $\bigwedge_k \phi_k$) using (NextInt). In summary, (with some applications of lemma 10) we have the following situation:-

$$\vdash \bigwedge_k \phi_k \;\Rightarrow\; \bigvee_w \bigcirc^{x_w} \bigwedge_k \phi_k$$

where $x_i \in \mathcal{N}$ and $\bigcirc^1 \equiv \bigcirc$ and $\forall n > 1 \,.\, \bigcirc^n = \bigcirc\bigcirc^{n-1}$.
If s is the least common multiple of the x_w then it is not hard to see that

$$\vdash \bigvee_w \bigcirc^{x_w} \bigwedge_k \phi_k \Rightarrow \bigcirc^{s} \bigwedge_k \phi_k$$

and so

$$\vdash \bigwedge_k \phi_k \;\Rightarrow\; \bigcirc^{s} \bigwedge_k \phi_k$$

Using the axiom (Finite-s) we have

$$\vdash \bigwedge_k \phi_k \;\Rightarrow\; \neg \bigwedge_k \phi_k$$

which is the contradiction required.

This completes the last part of the proof and thus completes the proof itself. □

8 Other considerations

This section brings together discussions of a number of aspects of the logic developed in this paper.

8.1 Finiteness

Standard modal and temporal logics are modelled by structures in which the number of possible worlds or the time domain is infinite. In particular, in standard linear time temporal logic (LTL) time extends infinitely into the future.

What modalities are available for expressing the informal proposition 'P holds whatever the case'? In LTL the answer is $\Box$, which expresses the semantic fact that P holds on every final segment of every (linear) model. This is reflected by the inference rule

$$\frac{\vdash P}{\vdash \Box P}$$

which, in a sense, *internalises* the theoremhood of P by prefixing it with the modality $\Box$.

In an interval temporal logic, the situation is different. 'P holds whatever the case' can be interpreted in a stronger fashion. It means that P will hold over not simply the final segments of the interval, but also the initial segments and indeed arbitrary contiguous segments. These are reflected in the inference rules

$$\frac{\vdash P}{\vdash \boxed{\mathrm{i}}\, P} \qquad \frac{\vdash P}{\vdash \boxed{\mathrm{a}}\, P}$$

for the modalities $\boxed{\text{i}}$ and $\boxed{\text{a}}$. These modalities are definable in the logic of chop, thus:

$$\boxed{\text{i}}P \equiv \neg(\neg P;\mathbf{True})$$

$$\boxed{\text{a}}P \equiv \neg(\mathbf{True};\neg P;\mathbf{True})$$

If projection is added to the language, then it is possible to express a stronger modality still: P should hold over an arbitrary sub-interval, and not simply over a contiguous sub-interval. This can be expressed thus:

$$\boxed{\text{a}}(\neg(\mathbf{True\ proj}\ \neg P))$$

since if an interval σ models the proposition $\neg(\mathbf{True\ proj}\ \neg P)$ this means that P will hold over any – not necessarily contiguous – sub-interval of σ whose endpoints coincide with the endpoints of σ itself. The $\boxed{\text{a}}$ has the effect of allowing the endpoints of the sub-interval to be chosen arbitrarily within σ. Within the logic for **proj** the strongest internal reflection of theoremhood is therefore provided by the inference rule

$$\frac{\vdash P}{\vdash \boxed{\text{a}}(\neg(\mathbf{True\ proj}\ \neg P))}$$

It is an important question, therefore, to ask whether these rules are admissible in the logic Λ. We return to this question after examining a related point, namely that of symmetry.

8.2 Symmetry

We have seen in Section 8.1 that a finite interval temporal logic has symmetry properties — between $\boxed{}$ and $\boxed{\text{i}}$ for instance — beyond those of standard LTL. The normal form approach advocated in this paper is, however, directed: time is examined forward, as it were, from the earliest point in an interval to the latest. Stated another way, the effect of a formula P is unravelled by looking at its effect now and its (residual) effect after a single unit of time has elapsed.

This approach makes perfect intuitive sense. It also makes formal sense, and underpins the completeness result given earlier in the paper. It does, however, make for a formal distinction between final and initial segments, say, and so between $\boxed{}$ and $\boxed{\text{i}}$ and so on. It was relatively straightforward to derive the rule

$$\frac{\vdash P}{\vdash \Box P}$$

in Section 6, whilst it is less straightforward to derive

$$\frac{\vdash P}{\vdash \boxed{\text{i}}P}$$

This we examine next.

8.3 Derived rules

How are rules which deduce modalities such as $\boxed{\text{i}}P$ from proofs of P to be derived? A number of possibilities present themselves.

- It is possible to use the completeness theorem to give a non- constructive derivation. If we assume that $\vdash P$ then by the soundness theorem for Λ then $\models P$. This means that P holds over every interval, and so it is easy to deduce that also $\models \boxed{\text{i}}P$. The completeness theorem then gives $\vdash \boxed{\text{i}}P$.
- It is also possible to use the tableau mechanism to give a proof. If we know that $\vdash P$ then the root node of the tableau for $\neg P$ will be coloured. Taking a partitioned normal form for $\neg P$, this gives a tableau which branches at the top level to nodes labelled:

  $$\begin{array}{l} \neg P_e,\ \textbf{empty} \\ P_i,\ \bigcirc\neg P'_i \end{array}$$

  Examining the tableau for $(\neg P\textbf{;True})$ we have

  $$\begin{array}{l} \neg P_e,\ \textbf{empty} \\ \neg P_e,\ \bigcirc\textbf{True} \\ P_i,\ \bigcirc\neg(P'_i\textbf{;True}) \end{array}$$

  and it can be seen that

  - The node $\neg P_e$, **empty** is coloured in the tableau for $\neg P$. This means that from $\neg P_e$ alone a contradiction can be proved, and so that the node $\neg P_e, \bigcirc\textbf{True}$ will also be blocked in the tableau for $(\neg P\textbf{;True})$.
  - After a time step in the two tableaux there are nodes labelled

    $$\neg P'_i$$

    and

    $$\neg P'_i\textbf{;True}$$

    respectively. The subtableaux for $\neg P'_i$ are blocked; an appeal to induction gives blocked subtableaux for $\neg P'_i\textbf{;True}$; the induction is over the same ordering as that given in the termination proof of the tableau procedure in Section 5.

Similar arguments will apply for the other modalities; indeed it is easy to see that appeal to the competeness theorem has the desired effect in all cases. Note, however, that the nature of the proofs of $\vdash \Box P$ and $\vdash \boxed{\text{i}}P$ are different: in the former case the proof is the same for all P – it is uniform, in other words – whilst in the latter case its structure depends upon the normal form of the formula P.

8.4 Design of logical systems

The design of a compositional system for reasoning about temporal logic requires some thought; see [21] for an earlier discussion of compositionality in the context of ITL.

As was alluded to in Section 8.1, the modalities $\boxed{}$, $\boxed{\text{i}}$ and so forth can be seen as internal reflections of provability. In looking at rules for 'chop' two options are possible. One can say

$$\frac{\vdash P \Rightarrow Q}{\vdash (R;P) \Rightarrow (R;Q)}$$

or frame this as an axiom:

$$\Box(P \Rightarrow Q) \Rightarrow ((R;P) \Rightarrow (R;Q))$$

The modality here reflects the fact that P is replaced by Q on a final segment of the interval. In a similar way, one can embody substitution in the first position by

$$\frac{\vdash P \Rightarrow Q}{\vdash (P;R) \Rightarrow (Q;R)}$$

or by the axiom

$$\boxed{\text{i}}(P \Rightarrow Q) \Rightarrow ((P;R) \Rightarrow (Q;R))$$

since substitution is over an initial segment of the interval. It is evident from these examples that different modalities encapsulate the substitutability of formulas in different positions within temporal formulas.

One can then ask what are the modalities corresponding to substitution within a projection formula. As rules we have

$$\frac{\vdash P \Rightarrow Q}{\vdash (P\ \mathbf{proj}\ R) \Rightarrow (Q\ \mathbf{proj}\ R)} \qquad \frac{\vdash P \Rightarrow Q}{\vdash (R\ \mathbf{proj}\ P) \Rightarrow (R\ \mathbf{proj}\ Q)}$$

To replace the first rule by an axiom we require

$$\boxed{\text{a}}(P \Rightarrow Q) \Rightarrow (P\ \mathbf{proj}\ R) \Rightarrow (Q\ \mathbf{proj}\ R)$$

as the left-hand formula might be replaced over any contiguous subinterval. How about the second position in a projection formula? Here the formula holds over a potentially arbitrary sub interval, so we require

$$\neg(\mathbf{True\ proj}\ \neg(P \Rightarrow Q)) \Rightarrow (R\ \mathbf{proj}\ P) \Rightarrow (R\ \mathbf{proj}\ Q)$$

where the modality $\neg(\mathbf{True\ proj}\ \neg S)$ represents the assertion that S holds over an arbitrary – not-necessarily contiguous – subinterval (with endpoints shared with the original interval).

It is a matter of taste whether it is more straightforward to work with rules or with axioms using a variety of modalities, each reflecting the relationship required to ensure substitution at a certain position within a formula.

A semantic argument based on the completeness theorem gives the derivability of the rules

$$\frac{\vdash P \Rightarrow Q}{\vdash (R;P) \Rightarrow (R;Q)}(ChopSubL)$$

$$\frac{\vdash P \Rightarrow Q}{\vdash (R;P) \Rightarrow (R;Q)}(ChopSubR)$$

$$\frac{\vdash P \Rightarrow Q}{\vdash (P \ \mathbf{proj} \ R) \Rightarrow (Q \ \mathbf{proj} \ R)}(ProjSubL)$$

$$\frac{\vdash P \Rightarrow Q}{\vdash (R \ \mathbf{proj} \ P) \Rightarrow (R \ \mathbf{proj} \ Q)}(ProjSubR)$$

References

[1] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM : A framework for programming in temporal logic. In *Lecture Notes in Artificial Intelligence, vol. 430*. Springer–Verlag, 1989.

[2] J. L. Bell and M. Machover. *A Course in Mathematical Logic*. North-Holland, 1977.

[3] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. In *8th ACM Symposium on Principles of Programming Languages*, pages 164–176. ACM, 1981.

[4] H. Bowman. An interpretation of cognitive theory in concurreny theory (long version). Technical Report 8-98, Computing Laboratory, University of Kent at Canterbury, 1998. http://www.cs.ukc.ac.uk/pubs/1998/646/index.local.

[5] H. Bowman and G. Faconti. Analysing cognitive behaviour using LOTOS and Mexitl. *Formal Aspects of Computing, 11*, 1999. to appear.

[6] H. Bowman and S. J. Thompson. A tableaux method for interval temporal logic with projection. In *TABLEAUX'98, International Conference on Analytic Tableaux and Related Methods*, volume 1397 of *Lecture Notes in AI*, pages 108–123. Springer-Verlag, May 1998.

[7] Howard Bowman, Helen Cameron, Peter King, and Simon Thompson. Specification and prototyping of structured multimedia documents using interval temporal logic. In *ICTL'97, International Conference on Temporal Logic, Manchester*. Kluwer, Applied Logic Series, 1997.

[8] Howard Bowman, Helen Cameron, Peter King, and Simon Thompson. Specification and prototyping of structured multimedia documents using interval temporal logic. Technical Report 3-97 (Kent), Computing Laboratory, University of Kent, 1997.

[9] A. Cau and H. Zedan. Refining interval temporal logic specifications. In *Fourth AMAST Workshop on Real-Time Systems, Concurrent, and Distributed Softward (ARTS'97), LNCS 1231*, May 1997.

[10] Z. H. Duan. *An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming.* PhD thesis, University of Newcastle Upon Tyne, May 1996.

[11] B. Dutertre. Complete proof systems for first order interval temporal logic. In D. Kozen, editor, *(LICS 10), Logic in Computer Science*, pages 36–43. IEEE, 1995.

[12] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of 14th Annual ACM Symposium on Theory of Computing*, pages 169–180. ACM, 1982.

[13] Dov Gabbay. The declarative past and imperative future. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification.* Lecture Notes In Computer Science 389, Springer-Verlag, 1989.

[14] R. Hale. Using temporal logic for prototyping: the design of a lift controller. In *Lecture Notes in Computer Science, vol. 379*, pages 375–408. Springer–Verlag, 1989.

[15] S. Kono. A combination of clausal and non-clausal temporal logic programs. In *Lecture Notes in Artificial Intelligence, vol. 897*, pages 40–57. Springer–Verlag, 1993.

[16] Shinji Kono. Automatic verification of interval temporal logic. Technical Report SCSL-TM-92-007, Sony Computer Science Laboratory, 1992.

[17] O. Lichtensteim, A. Pnueli, and L. Zuck. The glory of the past. In *Proceedings of Conference on Logics of Programs, LNCS 193.* Springer-Verlag, 1985.

[18] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.

[19] T. Melham. *Higher Order Logic and Hardware Verification.* Cambridge Tracts in Theoretical Computer Science (31), 1993.

[20] B. Moskowski. *Executing Temporal Logic.* Cambridge University Press, 1986.

[21] B. Moskowski. Some very compositional temporal properties. In *Proceedings of Programming Concepts, Methods and Calculi, IFIP Transactions A-56*, pages 307–326. North-Holland, 1994.

[22] B. Moskowski. Using temporal fixpoints to compositionally reason about liveness. In *Proceedings of the 7th Refinement Workshop, electronic Workshops in Computing.* Springer–Verlag, 1996.

[23] B. Paech. Gentzen-systems for propositional temporal logics. In *Fourth Proceedings of the 2nd Workshop on Computer Science Logic*, pages 240–253. Lecture Notes in Computer Science 385, Springer-Verlag, 1988.

[24] N. Rescher and A. Urquart. *Temporal Logic.* Springer-Verlag, 1971.

[25] R. Rosner and A. Pnueli. A choppy logic. In *Proceedings of 1st IEEE Symposium on Logic in Computer Science*, pages 306–314. IEEE, 1986.

[26] Simon Thompson. Constructive interval temporal logic in alf. In *ICTL'97, International Conference on Temporal Logic, Manchester.* Kluwer, Applied Logic Series, 1997.

[27] P. Wolper. Temporal logic can be more expressive. *Information and Computation*, 56:72–99, 1983.