Partial State Progression: An Extension to the Bacchus-Kabanza Algorithm, with Applications to Prediction and MITL Consistency

P@trik Haslum
Department of Computer Science, Linköping University
pahas@ida.liu.se

Abstract

The Metric Interval Temporal Logic (MITL) progression algorithm invented by Bacchus and Kabanza (1996) is extended to work also for the case when the duration and content of states are only partially specified (constrained). This is motivated by an approach to prediction, but also leads to a new, tableaux style, algorithm for deciding the consistency of an MITL formula. The algorithm is not yet fully developed.

1. Introduction

Metric Interval Temporal Logic (MITL) (Alur, Feder, & Henzinger 1996), like LTL (Emerson 1990), is a tense-modal logic expressing properties of infinite sequences of propositional states. The logics differ in that state sequences in MITL models are *timed*: each state in the sequence has a starting and ending timepoint, which maps to an underlying real timeline, and modal operators in MITL are qualified with constraints on state durations.

For applications working with MITL, the progression algorithm of Bacchus and Kabanza (1996) is a powerful tool. The algorithm "pushes" the truth criteria for an MITL formula forwards over the states of a timed model, one at a time: input is an MITL formula ϕ, and a state q with duration $D(q)$, and the algorithm returns an MITL formula ϕ' such that ϕ is T in q *iff* ϕ' is T in the succeeding state.

Of state succeeding q, nothing needs to be known: the returned formula is a condition that can be checked for contradiction, for truth in the succeeding state when it becomes known, or further progressed. The progression algorithm as presented, however, requires all properties of the state q, in particular the duration $D(q)$, to be known.

Motivated by the use of MITL to represent knowledge in a predictive model, I have extended the progression algorithm to work with only partial knowledge about the duration of state q, expressed by a set of constraints on variables representing state starting and ending times. As it turns out, the further extension to working with only partial knowledge about the state q itself is fairly straightforward, and potentially applicable to the problem MITL consistency checking.

The next two sections introduce background material: first, the approach to prediction that motivated this line of inquiry, then in more detail MITL and Bacchus' and Kabanza's progression algorithm. Section *4* presents the extension to partial knowledge about state timing.

Section *5*, finally, describes the further extension to the progression algorithm and how it may be applied to MITL consistency. This is still work in progress: the consistency checking algorithm presented is sound, but not complete.

2. Background I: Prediction as a Knowledge Representation Problem

Prediction is a central component in many control and reasoning tasks, *e.g.* state estimation/diagnosis, interpretation of sensor data (most notably the problem of reidentification), tracking control and planning. All of prediction relies on *models*: it is only with the knowledge encoded in a model, whatever form it takes, that a predictive system can conclude anything stronger than a tautology.

In a previous paper (Haslum 2001), I argued that there are for any application involving prediction many possible "model designs", by which I mean broadly the ontology, representation, acquisistion and computational methods associated with a model, and for the comparative investigation of different alternatives. As a case study, I have designed two different solutions to a prediction problem encountered in the WITAS UAV project[1]: one is a discrete event model and uses a schema-like representation of "normality", while the other is based on a Markov process. Both models use continuous time.

The role of MITL is in the first model design, as a language to formulate *expectations*, expressing beliefs about what will hold true, over time, in the normal case. Expectations are arranged in a heirarchy reflecting the strength of belief. A possible future development is represented by a timed sequence of events, finite since the prediction horizon is bounded. By checking which formulas of the hierarchy, if any, the development necessarily violates, it is assigned a "normality rating", representing a measure of the percieved relative likelihood of that development occuring.

[1] The WITAS project studies architectures and techniques for intelligent autonomous systems, centered around an unmanned aerial vehicle (UAV) for traffic surveillance. For an overview, see *e.g.* (Doherty *et al.* 2000) or http://www.ida.liu.se/ext/witas/. The problem considered is to predict the movements of a vehicle in a road network, and appears in the context of the task of planning a search strategy.

To make enumeration of even finite developments possible, it is necessary to adopt a constraint based representation of state starting and ending times, since events are distributed along a dense time line. This, and the use of progression as the tool to check for formula violation in finite developments, motivated the first step in extending the progression algorithm.

3. Background II: Timed Automata, MITL and Progression

Timed automata and MITL both have their roots in the research area of formal specification and verification of reactive hardware/software systems. Introductions can be found in *e.g.* Alur (1999) and Emerson (1990).

Timed Automata Timed automata (Alur & Dill 1994) are essentially finite state automata, augmented with time constraints of two kinds: a transition can have a time window in which it is possible to make the transition and a state can have a maximal time that the system may remain in the state before it has to exit by some transition.

Let $\mathbb{R}^+$ denote real numbers $\geqslant 0$, with a special symbol ∞ for infinity.

Definition 1 (Timed Automaton)
A timed automaton, $A = (Q, R, C, L)$, consists of a set of *states* Q, a *transition relation*

$$R \subseteq \Sigma \times Q \times Q \times \mathbb{R}^+ \times \mathbb{R}^+$$

where Σ is some set of *event* labels, a *state constraint* function $C : Q \longrightarrow \mathbb{R}^+$, and state labelling function $L : Q \longrightarrow 2^P$, where P is some set of propositional symbols (often the set of states is 2^P, *i.e.* a state is defined by its properties).

As usual, if $(a, q, q', t, t') \in R$, the system may transit from state q to q' in response to the event a, but only in the time interval $[t, t']$ relative to the time that the system entered q (time constraints of the first kind), and the system may remain in state q for a time at most equal to $C(q)$ (time constraints of the second kind)[2].

Like a finite automaton accepts a set of strings over its alphabet, a timed automaton accepts a set of histories.

Definition 2 (Development)
A development is a sequence of alternating states and events marking state transitions, $d = q_0, a_0, q_1, a_1, \ldots$, with an associated function $T : d \longrightarrow \mathbb{R}^+$ that tells the starting time of each state, such that

(*i*) for $i \geqslant 0$, there exists $t, t' \in \mathbb{R}^+$ such that $R(a_i, q_i, q_{i+1}, t, t')$ and $T(q_i)+t \leqslant T(q_{i+1}) \leqslant T(q_i)+t'$, and

(*ii*) for $i \geqslant 0$, $T(q_{i+1}) \leqslant T(q_i) + C(q_i)$.

The time interval through which state q_i lasts is $[T(q_i), T(q_{i+1}))$, *i.e.* closed at the beginning and open at the end[3]. The duration of a state q_i is denoted $D(q_i) = T(q_{i+1}) - T(q_i)$.

Two additional properties are usually required of a timed automaton: *executability*, which is the requirement that any finite prefix satisfying conditions (*i*) and (*ii*) of definition 2 can be extended to an infinite development, and *non-zenoness*, which is the requirement that the automaton does not make an infinite number of transitions in finite time.

Even when only finite development prefixes starting in a specific state q_0 are considered, the set of possible developments is uncountable, since the starting time of any state in a development can change by an arbitrarily small amount. For finite developments to be enumerable, a more compact representation has to be adopted: a set of developments that differ only on state starting times are represented by a single sequence of states and events, $d = q_0, a_0, \ldots, q_n$, and a set of constraints on the starting times $T(q_0), \ldots, T(q_n)$, managed in a temporal constraint network (TCN). Throughout, the TCN is assumed to be *simple*, *i.e.* containing only upper and lower bounds on the difference between pairs of temporal variables. This ensures that there are efficient algorithms for checking the consistency of the TCN, and for extracting minimal and maxmimal bounds on variable differences (Dechter, Meiri, & Pearl 1991; Brusoni, Console, & Terenziani 1995).

Metric Interval Temporal Logic

The Metric Interval Temporal Logic (MITL) is a so called “tense modal logic”, and was developed as a language for specifying properties of real-time, reactive systems (Alur, Feder, & Henzinger 1996).

Definition 3 (MITL Syntax)
The language of propositional MITL consists of a set of atoms P, propositional connectives and four temporal operators: $\Box_{[t,t']}\varphi$ (*always* φ), $\Diamond_{[t,t']}\varphi$ (*eventually* φ), $\bigcirc_{[t,t']}\varphi$ (*next* φ) and $\varphi\,\mathcal{U}_{[t,t']}\,\psi$ (φ *until* ψ). The intervals adjoined to the operators express metric temporal restrictions, and take point values in $\mathbb{R}^+$.

Formulas in MITL are evaluated over an infinite timed development (d, T). Since MITL formulas only reference present and future states, any suffix of a development is, for the purpose of evaluating formulas, also a development: thus a formula holds in a state q_i, if it holds in the development suffix beginning with q_i.

Definition 4 (MITL Semantics)
Let d^i denote the suffix of d starting with the ith state.

A formula φ not containing any temporal operator holds in d^i iff φ evaluates to T in the state q_i. The truth conditions for temporal formulas are

[2] The normal way to define timed automata is to augment standard automata with a set of real-valued “clock variables”, and express time constraints in a language of inequalities (Alur 1999). Definition 1 is less general, but it is sufficient for MITL satisfiability and simplifies some of what follows.

[3] This choice is rather arbitrary: the reverse convention could be made as well.

- $\Box_{[t,t']}\varphi$ holds in d^i iff φ holds in every d^k such that the intersection of time intervals $[T(q_k), T(q_{k+1}))$ and $[T(q_i) + t, T(q_i) + t']$ is non-empty (note that $k \geqslant i$ is implied by the fact that $t, t' \geqslant 0$ and that time increases along a development).
- $\Diamond_{[t,t']}\varphi$ holds in d^i iff there exists a q_k such that $[T(q_k), T(q_{k+1}))$ and $[T(q_i) + t, T(q_i) + t']$ intersect and φ holds in d^k.
- $\bigcirc_{[t,t']}\varphi$ holds in d^i if φ holds in q_{i+1} and $T(q_i) + t \leqslant T(q_{i+1}) \leqslant T(q_i) + t'$.
- $\varphi \mathcal{U}_{[t,t']}\, \psi$ holds in d^i iff there exists a q_k such that $[T(q_k), T(q_{k+1}))$ and $[T(q_i) + t, T(q_i) + t']$ intersect, ψ holds in d^k and φ holds for all d^j with $i \leqslant j < k$.

Connectives are interpreted as in ordinary logic.

It should be clear that the definitions can be extended to allow open or half-open operator time intervals, but to avoid an explosion of cases in definitions and algorithms, only closed intervals are considered in the remainder of the paper.

The MITL Progression Algorithm

The MITL progression algorithm provides a way to evaluate MITL formulas "incrementally" over a finite prefix of a development. Thus, the algorithm does not always return TRUE or FALSE, but often a condition to be further progressed through remaining states in the development.

Algorithm 5 (MITL Progression)
Let φ and q be the input formula and state, respectively, and φ' the returned formula. The progresion algorithm works recursively, by cases depending on the form of the input formula:

(*i*) If φ contains no temporal operators, $\varphi' =$ TRUE if φ is true in q and $\varphi' =$ FALSE if not (note that TRUE and FALSE are formula constants, not truth values).

(*ii*) If φ combines one or more subformulas with a propositional connective, φ' is the result of likewise combining the result of progressing each subformula. For example, if $\varphi = \alpha \wedge \beta$ then $\varphi' = \alpha' \wedge \beta'$.

(*iii*) If $\varphi = \bigcirc_{[t,t']}\psi$, then

(*iii.a*) if $D(q) < t$ or $t' < D(q)$, then $\varphi' =$ FALSE, and

(*iii.b*) if $t \leqslant D(q) \leqslant t'$, then $\varphi' = \psi$.

(*iv*) If $\varphi = \Box_{[t,t']}\psi$, then

(*iv.a*) if $D(q) < t$, then $\varphi' = \Box_{[t-D(q),t'-D(q)]}\psi$,

(*iv.b*) if $t \leqslant D(q) \leqslant t'$, then $\varphi' = \psi' \wedge \Box_{[0,t'-D(q)]}\psi$, and

(*iv.c*) if $t' < D(q)$, then $\varphi' = \psi'$,

where ψ' is the result of progressing ψ.

(*v*) If $\varphi = \Diamond_{[t,t']}\psi$, then

(*v.a*) if $D(q) < t$, then $\varphi' = \Diamond_{[t-D(q),t'-D(q)]}\psi$,

(*v.b*) if $t \leqslant D(q) \leqslant t'$, then $\varphi' = \psi' \vee \Diamond_{[0,t'-D(q)]}\psi$,

(*v.c*) if $t' < D(q)$, then $\varphi' = \psi'$,

where ψ' is the result of progressing ψ.

(*vi*) if $\varphi = \chi \mathcal{U}_{[t,t']}\, \psi$, then

(*vi.a*) if $D(q) < t$, then $\varphi' = \chi' \wedge (\chi \mathcal{U}_{[t-D(q),t'-D(q)]}\, \psi)$,

(*vi.b*) if $t \leqslant D(q) \leqslant t'$, $\varphi' = \psi' \vee (\chi' \wedge (\chi \mathcal{U}_{[0,t'-D(q)]}\, \psi))$ and

(*vi.c*) if $t' < D(q)$, then $\varphi' = \psi'$,

where χ' and ψ' are the result of progressing χ and ψ, respectively.

4. The Extended Progression Algorithm

Algorithm 5 assumes that the duration of the input state, $D(q)$, is a number, but as explained above sets of developments have to be represented by a combination of state/event sequence and time constraints to achieve enumerability. Consequently, the progression algorithm has to take as input a set of time constraints, C, and return the set of *all* possible progressions, $\{(\varphi'_1, C'_1), \ldots, (\varphi'_k, C'_k)\}$, where each C'_i is a set of additional time constraints consistent with C and φ'_i is the result of progressing the input formula φ under constraints $C \cup C'_i$.

The extension is conceptually straightforward, though somewhat complicated in practice. Notice that for each temporal operator algorithm 5 branches depending on the duration of the input state, and that the returned formula is built up according to the recursive path of branches taken. In general, an MITL formula defines a tree structure of possible progressions, with time constraints associated to the branches and resulting formulas at the leaves.

For the formal definition of the progression tree, a special form of each temporal operator has to be introduced: a *relative interval* is written $[X : t, t']$, where X is a TCN variable and $t, t' \in \mathbb{R}^+$, and interpreted as $[X + t, X + t']$. Relative temporal operators are obtained by adjoining a relative interval to any of the standard temporal operators.

Definition 6 (Progression Tree)
For an MITL formula φ and a state q with starting and ending times denoted by variables $X_S(q)$ and $X_E(q)$, the *progression tree* $\mathcal{T}_P(\varphi)$ is defined as follows: edges in the tree are labeled with constraints (involving $X_S(q)$, $X_E(q)$, and possibly other TCN variables) and leaf nodes are labeled with formulas.

(*i*) If φ is a state formula (*i.e.* contains no temporal operators), $\mathcal{T}_P(\varphi)$ consists only of a leaf, labeled with TRUE if φ holds in q and FALSE if not.

(*ii*) $\mathcal{T}_P(\neg\alpha)$ is constructed from $\mathcal{T}_P(\alpha)$ by replacing every leaf label α' with $\neg\alpha'$.

(*iii*) To construct $\mathcal{T}_P(\alpha \wedge \beta)$, start with $\mathcal{T}_P(\alpha)$. For every leaf, let α' be the leafs label: replace the leaf with a copy of $\mathcal{T}_P(\beta)$, and within that copy replace every leaf label β' by $\alpha' \wedge \beta'$. The construction is illustrated in figure 1.
The construction of $\mathcal{T}_P(\alpha \vee \beta)$ is analogous.

(*iv*) $\mathcal{T}_P(\bigcirc_{[t,t']}\psi)$ has three branches:

(*iv.a*) a branch labeled $X_E(q) - X_S(q) < t$ leads to a leaf labeled by FALSE,

(*iv.b*) a branch labeled $X_E(q) - X_S(q) > t'$, also to a leaf labeled FALSE, and

(*iv.c*) a branch labeled $t \leqslant X_E(q) - X_S(q) \leqslant t'$ leads to the root of $\mathcal{T}_P(\psi)$.

(*v*) $\mathcal{T}_P(\Box_{[X:t,t']}\psi)$ also has three branches:

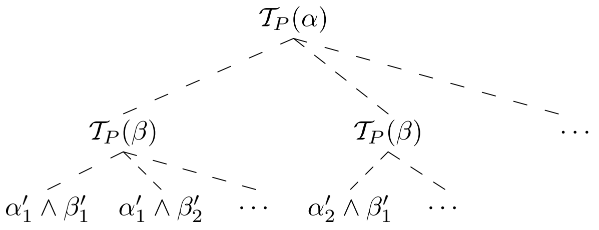

Figure 1: Construction of $\mathcal{T}_P(\alpha \wedge \beta)$

(*v.a*) at the first branch, labeled $X_E(q) - X < t$, is a leaf labeled $\Box_{[X:t,t']}\psi$,

(*v.b*) at the second branch, labeled $t \leqslant X_E(q) - X \leqslant t'$, is a copy of $\mathcal{T}_P(\psi)$, and within that copy every leaf label ψ' is replaced with $(\Box_{[X:t,t']}\psi) \wedge \psi'$, and

(*v.c*) at the third branch, labeled $t' < X_E(q) - X$, is an unmodified copy of $\mathcal{T}_P(\psi)$.

(*vi*) $\mathcal{T}_P(\Box_{[t,t']}\psi)$ is equal to $\mathcal{T}_P(\Box_{[X_S(q):t,t']}\psi)$.

(*vii*) $\mathcal{T}_P(\Diamond_{[X:t,t']}\psi)$ has three branches:

(*vii.a*) at the first branch, labeled $X_E(q) - X < t$, is a leaf labeled $\Diamond_{[X:t,t']}\psi$,

(*vii.b*) at the second branch, labeled $t \leqslant X_E(q) - X \leqslant t'$, is a copy of $\mathcal{T}_P(\psi)$ with every leaf label ψ' replaced by $(\Diamond_{[X:t,t']}\psi) \vee \psi'$, and

(*vii.c*) at the third branch, labeled $t' < X_E(q) - X$, is a copy of $\mathcal{T}_P(\psi)$.

(*viii*) $\mathcal{T}_P(\Diamond_{[t,t']}\psi)$ is equal to $\mathcal{T}_P(\Diamond_{[X_S(q):t,t']}\psi)$.

(*ix*) $\mathcal{T}_P(\chi\mathcal{U}_{[X:t,t']}\psi)$ has three branches:

(*ix.a*) at the first branch, labeled $X_E(q) - X < t$, is a leaf labeled $\chi\mathcal{U}_{[X:t,t']}\psi$,

(*ix.b*) at the second branch, labeled $t \leqslant X_E(q) - X \leqslant t'$, is $\mathcal{T}_P(\psi)$, but every leaf, with label ψ', is replaced a copy of $\mathcal{T}_P(\chi)$, and in this copy, every leaf label χ' is replaced by $(\chi' \wedge (\chi\mathcal{U}_{[X:t,t']}\psi)) \vee \psi'$, and

(*ix.c*) at the third branch, labeled $t' < X_E(q) - X$, is only $\mathcal{T}_P(\psi)$.

(*x*) $\mathcal{T}_P(\chi\mathcal{U}_{[t,t']}\psi)$ equals $\mathcal{T}_P(\chi\mathcal{U}_{[X_S(q):t,t']}\psi)$.

The construction of the progression tree parallels progression algoritm 5, but time constraints are captured in the edge labels instead of the bounds of intervals adjoined to temporal operators in the formulas labelling the leafs. The introduction of relative temporal operators serves to ensure that all time constraints found in the progression tree remain simple. The algorithm for progression now becomes a simple matter of tree traversal:

Algorithm 7 (Extended MITL Progression)
Let φ and q be the input MITL formula and state, respectively, and C a set of (simple) time constraints.

Construct $\mathcal{T}_P(\varphi)$. For every leaf l in the progression tree, collect the set of constraints C_l found along the path to l: if $C \cup C_l$ is consistent, (φ'_l, C'_l) is included in the set of progressions returned.

The method of traversing the progression tree is not important, as long as every leaf at the end of a consistent path is eventually found. The most efficient method appears to be to search the tree depth first and check the set of constraints incrementally, at each node.

Example 1 Consider the formula $\varphi = \Box_{[5,9]} \bigcirc_{[0,4]} p$. The progression tree is shown in figure 2. If the input set of constraints is $C = \{0 < X_E - X_S < 7\}$, there are two consistent solution paths:

(*a*) $X_E - X_S < 5$, with the result $\Box_{[X_S:5,9]} \bigcirc_{[0,4]} p$.

(*b*) $5 \leqslant X_E - X_S \leqslant 9$, $X_E - X_S > 4$, with the result FALSE.

$X_E - X_S > 9$ and $X_E - X_S < 0$ both contradict C, while $0 \leqslant X_E - X_S \leqslant 4$ is inconsistent with the constraint $5 \leqslant X_E - X_S$ labelling the branch above.

Note that solutions returned by the algorithm may contain relative temporal operators: these only make sense in the context of a particular constraint set, which may be viewed as conjoined to the returned formula. Consequently, if the formula of a returned solution (φ', C') is to be further progressed through another state, the whole constraint set, $C \cup C'$, must be passed as input to the next progression.

5. MITL Consistency

Alur *et al.* (1996) present a decision procedure for the consistency of an MITL formula, but with several restrictions on the formula: there is no *next* operator, the time intervals adjoined to temporal operators must have rational constant endpoints, and must be non-singular, *i.e.* not of the form $[t, t]$. The restriction to non-singular intervals is necessary, since the consistency problem is provably undecidable if singular intervals are allowed (Alur & Henzinger 1994).

The algorithm constructs a timed automaton accepting exactly the developments that are models of the formula. The automaton is then checked for emptiness (*i.e.* whether it accepts at least one development), *e.g.* using the procedure of Alur and Dill (1994).

Suppose that the progression algorithm is extended, so that the input state q may be unspecified *w.r.t.* propositions as well as starting and ending time, and the solutions returned contain constraints on both. Then, if progressing an MITL formula through a sequence of completely unspecified states (except for the constraint $D(q) \geqslant 0$, for each state), results in TRUE, any sequence of states that satisfies the set of constraints collected along the way is a model for the formula. This idea leads to what is essentially a tableaux algorithm for MITL consistency.

The decision procedure for (non-real time) Linear Temporal Logic (LTL) is also tableaux based, but constructs a Büchi automaton equivalent to the formula to be checked (Wolper 1989; Gerth *et al.* 1995). A variant described by

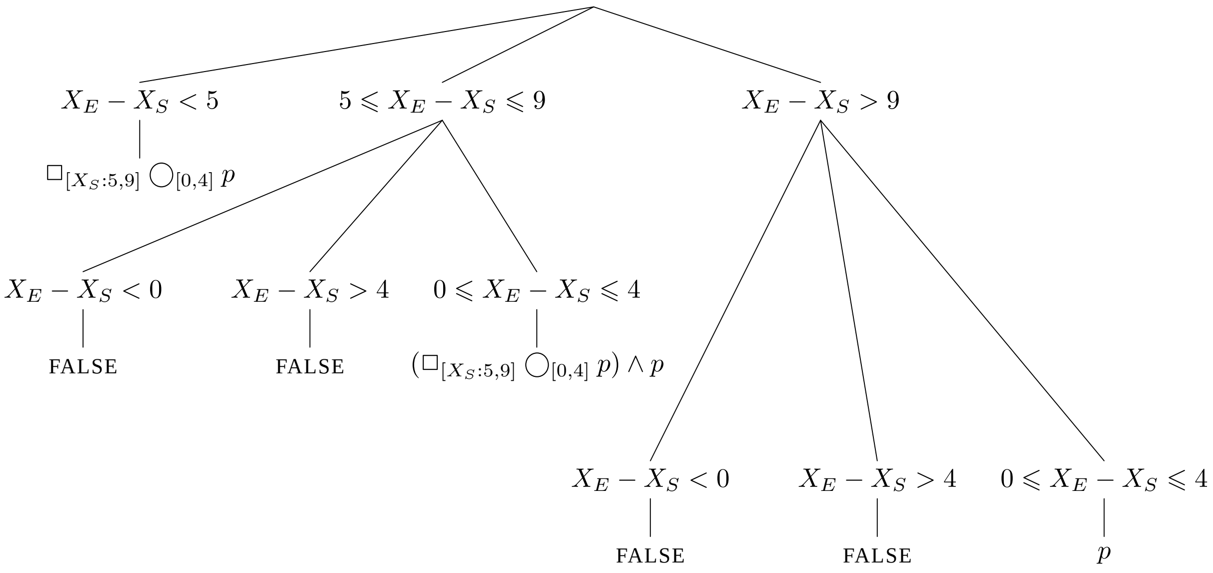
$X_E - X_S < 5$
$5 \leqslant X_E - X_S \leqslant 9$
$X_E - X_S > 9$
$\Box_{[X_S:5,9]} \bigcirc_{[0,4]} p$
$X_E - X_S < 0$
$X_E - X_S > 4$
$0 \leqslant X_E - X_S \leqslant 4$
FALSE
FALSE
$(\Box_{[X_S:5,9]} \bigcirc_{[0,4]} p) \wedge p$
$X_E - X_S < 0$
$X_E - X_S > 4$
$0 \leqslant X_E - X_S \leqslant 4$
FALSE
FALSE
p

Figure 2: $\mathcal{T}_P(\Box_{[5,9]} \bigcirc_{[0,4]} p)$

Schwendimann (1998) is remarkably similar to the algorithm developed here, except it is formulated in terms of inference rules rather than progression, and applies of course only to LTL.

Partial State Progression

For the progression algorithm to work with partially or completely unspecified states requires only a small change in the progression tree: case (*i*) is replaced by

(*i'*) For a single proposition p, $\mathcal{T}_P(p)$ has two branches, labeled by $p = \mathrm{T}$ and $p = \mathrm{F}$, ending in leafs labeled TRUE and FALSE, respectively.

The branch labels are constraints on the value of p, while the leaf labels are formula constants. Because there are no disjunctions, the set of propositional constraints essentially corresponds to a partial assignment, so consistency can be easily determined.

Tableaux Construction

The tableaux is a tree constructed by repeated progression of the MITL formula to be checked for consistency. Each node in the tree is labeled by a formula and a constraint set. The tree also represents a set of developments: each node corresponds to a state, and each path from the root downwards to a development.

Definition 8 (Tableaux Tree)
Let φ be an MITL formula. The *tableaux tree* of φ, $\mathcal{T}_T(\varphi)$ is defined inductively by:

(*i*) The root, r, is labeled by φ and $\{X_E(r) - X_S(r) \geqslant 0\}$.

(*ii*) Let n be a node labeled with formula φ_n and constraint set C_n. For each solution (φ', C') found by progressing φ_n with input constraints C_n, n has a successor node, n', labeled with φ' and $C_n \cup C' \cup \{X_S(n') = X_E(n), X_E(n') - X_S(n') \geqslant 0\}$.

Because constraints are only passed down in the tree, variables can be named by the depth of the node only, *i.e.* the root node starts at X_0 and ends at X_1, its successors all start at X_1 and end at X_2, *etc.* The constraint $X_{i+1} - X_i \geqslant 0$, placed on every node, ensures that the duration of the corresponding state is non-negative.

Example 2 Figure 3 shows the tableaux tree for the formula $\Box_{[5,9]} \bigcirc_{[0,4]} p$ (formula labels only). Nodes marked "zeno cycle" correspond to non-executable developments, as explained in the next section.

Definition 9 (Closed, Satisfying)
A node n in $\mathcal{T}_T(\varphi)$ is *closed* iff (*a*) the formula labeling the node is TRUE or FALSE, or (*b*) the constraint set stored in the node is inconsistent. A node is *satisfying* if it is labeled with TRUE.

To decide the consistency of a formula φ, $\mathcal{T}_T(\varphi)$ is searched for a satisfying node: if one is found, φ is consistent, and the sequence of states along the path from the root to the node is a model. The subtree beneath a closed node does not have to be examined, and if every leaf node is closed and not satisfying, φ is inconsistent.

Closing Cycles

For many MITL formulas, the tableaux tree contains infinite branches without a closing node, and thus a straightforward search in the tableaux tree may fail to terminate. Consider for example $\Box_{[0,\infty]} \Diamond_{[0,10]} p$: this formula generates, among others, an infinite sequence of progressions identical to the

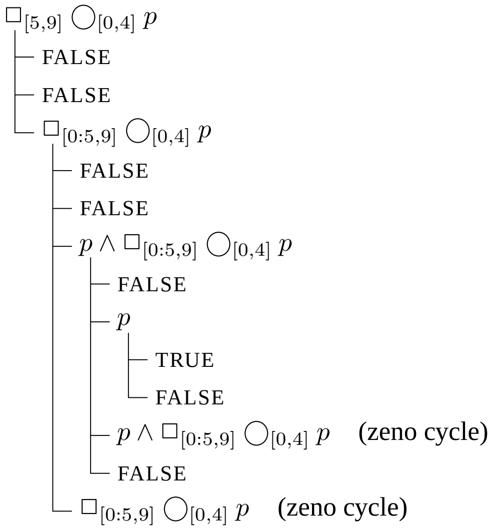
$\Box_{[5,9]} \bigcirc_{[0,4]} p$
FALSE
FALSE
$\Box_{[0:5,9]} \bigcirc_{[0,4]} p$
FALSE
FALSE
$p \wedge \Box_{[0:5,9]} \bigcirc_{[0,4]} p$
FALSE
p
TRUE
FALSE
$p \wedge \Box_{[0:5,9]} \bigcirc_{[0,4]} p$ (zeno cycle)
FALSE
$\Box_{[0:5,9]} \bigcirc_{[0,4]} p$ (zeno cycle)

Figure 3: $\mathcal{T}_T(\Box_{[5,9]} \bigcirc_{[0,4]} p)$

input formula (except for the $\Box$ operator being relative to X_0).

If a node n is labeled by the same formula as a node n' found along the path from the root to n, and the constraint set of the ancestor node n' subsumes, *i.e.* is less restrictive than, that of n, then all the possible successors of n are already contained in the subtree beneath n', and it should not be necessary to search further below n. Thus, definition 9 must be amended:

Definition 10 (Cycle, Zeno Cycle)
A node n in $\mathcal{T}_T(\varphi)$ for which there exists an ancestor node n' labeled by an identical formula and such that the constraint set of n' subsumes that of n, is a *cycle*. A cycle node is also closed.

If in addition, the maximal value of $X_S(n) - X_S(r)$, where r is the root, is bounded by the set of constraints (*i.e.* not infinite), node n is a *zeno cycle*.

Because time constraints are only added as the formula is progressed down the tableaux tree, the temporal constraint set of a node always subsumes those of its successors. Subsumption for state constraints, however, must still be checked.

A cycle node n represents an infinite development, consisting of the states along the path from the root to the ancestor node n', followed by an infinite number of repetitions of the states from n' to, but not including, n. If the maximal starting time of n (which is also the ending time of the cycle) is bounded, the time in this development can not diverge beyond that bound, *i.e.* it will pass through an infinite number of states in finite time: hence the name "zeno cycle".

Evaluation in Cycles That a node is a cycle does not mean that it does not satisfy the formula labeling the node. For example, $\mathcal{T}_T(\Box_{[0,\infty]}\Diamond_{[0,10]}p)$ contains a cycle node at depth 2, labeled by $\Box_{[X_0:0,\infty]}\Diamond_{[0,10]}p$ and, among others, the constraints $\{X_2 - X_1 \leqslant 10, p_2 = \mathrm{T}\}$. This represents a development consisting of an infinite sequence of states, each with duration at most 10, in all of which p is true, clearly a model for the formula.

For any node n that is a cycle but not a zeno cycle, if the formula labeling the node at the start of the cycle (node n' in definition 10) holds in the corresponding infinite development, according to the standard MITL semantics, n is satisfying. Even though the development is infinite, the number of distinct states it visits is finite, which makes evaluation possible, although not without complications: the distance, in time, between different states in the cycle may be "stretched", though not arbitrarily, by inserting repetitions of the cycle.

Improved Cycle Detection The condition for cycle detection defined above is too weak to ensure that the search for a satisfying node in the tableaux tree terminates. Two examples illustrate the problem:

Example 3 $\mathcal{T}_T(\Box_{[0,\infty]}(\Diamond_{[0,\infty]}p \wedge \Diamond_{[0,\infty]}\neg p))$ contains an infinite branch beginning with

$$\begin{aligned}
&n_0 : \Box_{[0,\infty]}(\Diamond_{[0,\infty]}\neg p \wedge \Diamond_{[0,\infty]}p)\\
&n_1 : \quad \Diamond_{[X_0:0,\infty]}\neg p \wedge \Box_{[X_0:0,\infty]}(\Diamond_{[0,\infty]}\neg p \wedge \Diamond_{[0,\infty]}p)\\
&n_2 : \qquad \Diamond_{[X_1:0,\infty]}p \wedge \Box_{[X_0:0,\infty]}(\Diamond_{[0,\infty]}\neg p \wedge \Diamond_{[0,\infty]}p)\\
&n_3 : \qquad\quad \Diamond_{[X_2:0,\infty]}\neg p \wedge \Box_{[X_0:0,\infty]}(\Diamond_{[0,\infty]}\neg p \wedge \Diamond_{[0,\infty]}p)\\
&\qquad\qquad\vdots
\end{aligned}$$

corresponding to a sequence of states in which p alternates between T and F.

Example 4 $\mathcal{T}_T(\Box_{[0,\infty]}\Diamond_{[0,\infty]}(p \wedge \neg p))$ contains only infinite branches of the form

$$n_0 : \Box_{[0,\infty]}\Diamond_{[0,\infty]}(p \wedge \neg p)$$

$$\begin{aligned}
&n_1 : \quad \Diamond_{[X_0:0,\infty]}(p \wedge \neg p) \wedge \Box_{[X_0:0,\infty]}\Diamond_{[0,\infty]}(p \wedge \neg p)\\
&n_2 : \qquad \Diamond_{[X_0:0,\infty]}(p \wedge \neg p) \wedge \Diamond_{[X_1:0,\infty]}(p \wedge \neg p) \wedge\\
&\qquad\qquad \Box_{[X_0:0,\infty]}\Diamond_{[0,\infty]}(p \wedge \neg p)\\
&\qquad\qquad\vdots
\end{aligned}$$

since the state formula $p \wedge \neg p$ will never be true.

In example 3, the formulas labelling nodes n_3 and n_1 are identical, except for the reference time point of the relative interval adjoined to the $\Diamond$ operator. If, however, the constraints on the time variable X_0 at n_1 are not more restrictive than those on X_2 at n_3, the set of successors, *i.e.* progressions, of n_3 will also be identical to those of n_1, except for this difference: in particular, if there exists a satisfying node among the successors of n_3, the "same" satisfying node must be found by the same sequence of progressions from n_1. To state this condition precisely requires some extra definitions:

Definition 11 (Formula Shift)
The (backwards) *shift by k* of a formula ϕ is obtained by replacing every occurrence of every time variable X_i with $i \geqslant k$ by X_{i-k} in ϕ.

Definition 12 (Subsumes with Shift)
Let C and C' be constraint sets, where $C \subseteq C'$. Then C *subsumes C' with shift by k* iff

- $X_i - X_j$ in C subsumes $X_i - X_j$ in C', for all $i \leqslant j < k$,
- $X_i - X_j$ in C subsumes $X_i - X_{j+k}$ in C', for all $i < k \leqslant j$,
- $X_i - X_j$ in C subsumes $X_{i+k} - X_{j+k}$ in C', for all $k < i \leqslant j$.

The meaning of "$X_i - X_j$ in C subsumes $X_u - X_v$ in C'" is that the bounds on $X_i - X_j$ set by C are less restrictive than those placed on $X_u - X_v$ by C'.

Definition 13 (Extended Cycle)
A node n starting at X_j in $\mathcal{T}_T(\varphi)$ for which there exists an ancestor node n' starting at X_i such that the formula labelling n' equals the formula labelling n backwards shifted by $j - i$ and such that the constraint set at n' subsumes that at n with shift $j - i$, is also a cycle.

By the extended definition, node n_3 in example 3 is a cycle so the branch is closed. The formula labelling node n_1 at the start of the cycle is true in the corresponding infinite development.

This, however, is not enough to close the branch in example 4, because the formula resulting from progression is redundant. To fix this, the progression result is rewritten into an equivalent, simpler, form, based on the notion of "weak implication":

Definition 14 (Weakly Implies)
Formula α *weakly implies β*, *w.r.t.* constraint set C, in case

(*i*) α and β are identical (*i.e.* a formula always weakly implies itself), or

(*ii*) $\alpha = \Box_{[X:s,t]}\phi$, $\beta = \Box_{[X':s',t']}\phi'$, $[X : s, t]$ necessarily contains $[X' : s', t']$ given C and ϕ weakly implies ϕ', or

(*iii*) $\alpha = \Diamond_{[X:s,t]}\phi$, $\beta = \Diamond_{[X':s',t']}\phi'$, $[X : s, t]$ is necessarily contained in $[X' : s', t']$ given C and ϕ weakly implies ϕ'.

The rewrite rules applied to the formula resulting from progression are:

R1. Eliminate from a conjunction every conjunct that is weakly implied by another conjunct.

R2. Eliminate from a disjunction every disjunct that weakly implies another disjunct.

Weak implication is determined *w.r.t.* the constraint set input to progression and the set returned along with the formula. Since weak implication entails ordinary implication, the rules preserve equivalence.

Because of the constraint $X_1 - X_0 \geqslant 0$, $[X_1 : 0, \infty]$ must be contained in $[X_0 : 0, \infty]$, so $\Diamond_{[X_1:0,\infty]}(p \wedge \neg p)$ weakly implies $\Diamond_{[X_0:0,\infty]}(p \wedge \neg p)$, and the formula labelling node n_2 in example 4 can be simplified to

$$\Diamond_{[X_1:0,\infty]}(p \wedge \neg p) \wedge \Box_{[X_0:0,\infty]}\Diamond_{[0,\infty]}(p \wedge \neg p)$$

using rewrite rule *R1*, which makes the node a cycle (by the extended condition).

Correctness and Complexity

The tableaux method is clearly sound, in the sense that whenever a satisfying node is found, the path leading to that node (with infinite repetition if it is a cycle node) is a model for the formula. Likewise, if only the basic cycle definition is applied and all branches are closed and not satisfying, the formula is inconsistent: the argument is that the progression algorithm is exhaustive and that the development corresponding to a closed node can not be a model for the formula. The extended cycle definition appears, intuitively, to be correct, but since it is so complicated, intuition is not quite reliable and it should be proved formally.

Even the extended cycle detection, however, is not strong enough to ensure termination, *e.g.* the tableaux tree for the formula $\Box_{[0,\infty]}(\Diamond_{[0,10]}\neg p \wedge \Diamond_{[0,10]}p)$ contains infinite branches. The reason, in this case, is that the time limit on the $\Diamond$ operator causes weak implication to fail: $\Diamond_{[X_1:0,10]}p$ does not weakly imply $\Diamond_{[X_0:0,10]}p$ unless constraints entail $X_1 = X_0$, since only then is $[X_1 : 0, 10]$ necessarily contained in $[X_0 : 0, 10]$.

Deciding MITL consistency is EXPSPACE-complete (Alur, Feder, & Henzinger 1996). The algorithm by Alur *et al.* requires time exponential in the number of connectives and the largest integer constant appearing in the formula.

The tableaux method may not be able to do too much better. Consider the number of consistent progressions of a formula φ, in the worst case: The number of temporal branch nodes in the progression tree is bounded by the number of temporal operators. At each node, the difference $X_k - X_i$, where k is the index of the state through which the formula is progressed (equal to the tableaux depth), and $i < k$, is compared to a constant interval (the restriction interval of a temporal operator). With n branch nodes, the comparisons involve at most $2n$ different constants, which when ordered yield $2n + 1$ different intervals that each $X_k - X_i$ may fall into. Denoting the number of distinct variable differences occuring in constraints in the tree by $d \leqslant \max(k, n)$, the number of consistent paths through the temporal branch nodes is at most

$$\frac{(2n + d)!}{d!(2n)!} \leqslant (2n + 1)^d,$$

since $X_{i+1} - X_i \geqslant 0$ for all i, and therefore $X_k - X_j \leqslant X_k - X_i$ whenever $j > i$. The number of consistent truth value assignments is of course 2^p, where p is the number of distinct propositions occuring in the formula.

Thus, the worst case branching factor in the tableaux tree is polynomial in the number of temporal operators in the formula, but exponential in the number of distinct propositions and in the tableaux depth. As for how deeply the tree may have to be searched to find a satisfying node or close all branches (assuming the search terminates at all), I have currently no idea.

6. Concluding Remarks

MITL is a powerful language for expressing properties over time: it has been used to express requirements in formal verification, goals and control rules in planning, and knowledge in predictive models. Likewise, progression is a powerful tool for working with MITL. The extended algorithm enables it to be used also with a compact representation of sets of developments, which in turn enables enumeration of the finite development prefixes generated by a timed automaton.

The tableaux algorithm for deciding MITL consistency shows some promise: with extended cycle detection and simplification rewriting, it manages to prove the formula $\Box_{[0,\infty]}\Diamond_{[0,\infty]}p \wedge \Diamond_{[0,\infty]}\Box_{[0,\infty]}\neg p$ unsatisfiable. In difference to the method by Alur *et al.*, it does not depend on time constants being integral, or even rational, except as far as constraint management does. Also, it appears simpler, which is not an unimportant property.

Still, it is very much work in progress: besides making it complete, the extended cycle detection, possibly strengthened, needs to be proved correct, and a more thorough analysis of the algorithms complexity to be done. Although it certainly requires both exponential time and space, it may be exponential in different variables than the existing algorithm: for example, it is hard to see that the size of the constants in the intervals adjoining the temporal operators should play a part, and this may make a difference in practice.

That aside, rewriting based on weak implication is inelegant and *ad hoc*. It may be seen as imposing a "weak normal form" on formulas, but exactly what this form is, and what set of rewrite rules is sufficient to obtain it, needs to be clarified.

References

Alur, R., and Dill, D. 1994. A theory of timed automata. *Theoretical Computer Science* 126(2):183 – 236.

Alur, R., and Henzinger, T. 1994. A really temporal logic. *Journal of the ACM* 41(1):181 – 204.

Alur, R.; Feder, T.; and Henzinger, T. 1996. The benefits of relaxing punctuality. *Journal of the ACM* 43(1):116 – 146.

Alur, R. 1999. Timed automata. In *NATO-ASI Summer School on Verification of Digital and Hybrid Systems*. Available at http://www.cis.upenn.edu/∼alur/Nato97.ps.gz.

Bacchus, F., and Kabanza, F. 1996. Planning for temporally extended goals. In *Proc. 13th National Conference on Artificial Intelligence (AAAI'96)*.

Brusoni, V.; Console, L.; and Terenziani, P. 1995. On the computational complexity of querying bounds on differences constraints. *Artificial Intelligence* 74:367 – 379.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61 – 95.

Doherty, P.; Granlund, G.; Kuchcinski, K.; Sandewall, E.; Nordberg, K.; Skarman, E.; and Wiklund, J. 2000. The WITAS unmanned aerial vehicle project. In *Proc. European Conference on Artificial Intelligence*.

Emerson, E. 1990. Temporal and modal logic. In *Handbook of Theoretical Computer Science*. Elsevier. 997 – 1072.

Gerth, R.; Peled, D.; Vardi, M.; and Wolper, P. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th Workshop on Protocol Specification, Testing and Verification*. North-Holland.

Haslum, P. 2001. Models for prediction. In *Proc. IJCAI 2001 workshop on Planning under Uncertainty*.

Schwendimann, S. 1998. A new one-pass tableau calculus for PLTL. In de Swart, H., ed., *Proc. International Conference on Theorem Proving with Analytic Tableaux and Related Methods*, volume 1397 of *Lecture Notes in Artificial Intelligence*, 277 – 291. Springer Verlag.

Wolper, P. 1989. On the relation of programs and computations to models of temporal logic. In Banieqbal, B.; Barringer, H.; and Pnueli, A., eds., *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, 75 – 123. Springer Verlag.