ELSEVIER Theoretical Computer Science 170 (1996) 1–46

Theoretical Computer Science

Fundamental Study

Interval logics and their decision procedures

Part II: A real-time interval logic[1]

Y.S. Ramakrishna*, P.M. Melliar-Smith, L.E. Moser, L.K. Dillon, G. Kutty[2]

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA

Received September 1994; revised December 1995
Communicated by M. Nivat

Abstract

In a companion paper, we presented an interval logic, and showed that it is elementarily decidable. In this paper we extend the logic to allow reasoning about real-time properties of concurrent systems; we call this logic real-time future interval logic (RTFIL). We model time by the real numbers, and allow our syntax to state the bounds on the duration of an interval. RTFIL possesses the "real-time interpolation property," which appears to be the natural quantitative counterpart of invariance under finite stuttering. As the main result of this paper, we show that RTFIL is decidable; the decision algorithm is slightly more expensive than for the untimed logic. Our decidability proof is based on the reduction of the satisfiability problem for the logic to the emptiness problem for timed Büchi automata. The latter problem was shown decidable by Alur and Dill in a landmark paper, in which this real-time extension of ω-automata was introduced. Finally, we consider an extension of the logic that allows intervals to be constructed by means of "real-time offsets", and show that even this simple extension renders the logic highly undecidable.

* Corresponding address. Computer Science Department, SUNY Stony Brook, NY 11794, USA.; e-mail: ysr@cs.sunysb.edu.

[1] Research partially supported by NSF/ARPA grant CCR-9014382. Portions of this paper have appeared in preliminary form as: A real-time interval logic and its decision procedure, in: *Proc. 13th Found. Softw. Tech. and Theoret. Comput. Sci.*, Lecture Notes in Computer Science, Vol. 761 (Springer, Berlin, 1993) 173–192. Most of the results also appear in the first author's Ph.D. Thesis, Interval Logics for Temporal Specification and Verification, Dept. of Elec. and Comput. Eng., University of California, Santa Barbara, 1993.

[2] Current address: G.K. Mathew, GE Medical Systems, P.O. Box 414, W-657, Milwaukee, WI 53201-0414, USA.

Contents

1. Introduction

Much of the elegance and ease of using temporal logic derives from the abstraction away from physical time, concentrating instead on the essential property of causal ordering of events in a concurrent system. While many systems and algorithms are amenable to this form of analysis, there are many situations where the correctness of the system depends not only on the causal ordering of events, but also on their relative real-time delays. In such cases one must be able to reason about the real-time values of these delays in order to establish correct behavior.

There is growing consensus among practitioners that physical time should *not* be treated as just another state variable – that it is special enough and used frequently enough to require the development of explicit mechanisms for its manipulation. The growing body of recent work in real time specification and verification more than attests to this fact; as a small sampling of this work, we mention [1,4,6,13,20,23,28,29].

In this paper, we investigate an extension, with real time, of the purely qualitative Future Interval Logic (FIL) that we presented in a companion paper [27]. We do so by introducing a special duration predicate, parameterized with two rational constants, which can be applied to intervals. This simple extension, as we show in the sequel, not only gives us reasonable expressiveness, but also preserves decidability, thus making the logic amenable to automation. Moreover, the resulting logic, in keeping with the tradition in temporal logic, "hides" the time variable, thus preventing its improper manipulation. We call this logic Real-Time Future Interval Logic (RTFIL).

The remainder of this paper approximately parallels our presentation of FIL in [27]. We start, in Section 2, with a brief informal description of the logic, then formally introduce the extensions to FIL that yield RTFIL. After first introducing some

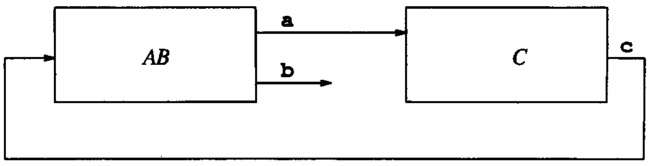


Fig. 1. The system for the example of Section 2.

preliminary background and machinery, we describe the decision procedure in Section 3. We briefly discuss complexity issues at the end of that section. We then consider an extension of RTFIL, and comment on related decidability issues in Section 4. Finally, we compare our logic with several other dense real-time logics and comment briefly on related results. Section 6 contains concluding remarks and states some open problems. An appendix gives details of proofs that do not appear in the body of the paper.

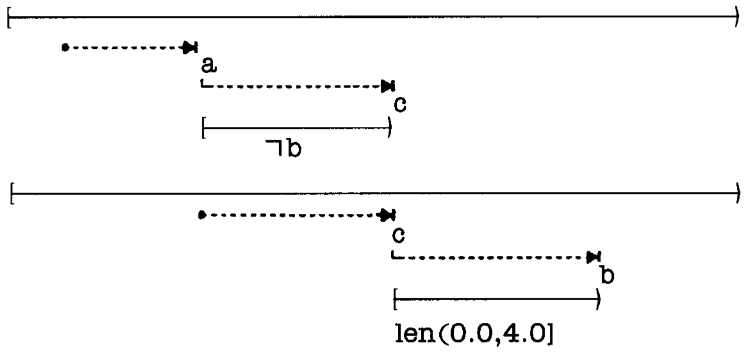
Note: We make heavy reference, in this paper, to [27], where the untimed logic FIL was introduced. In the remainder of the paper, we shall use "Part I" to mean the companion paper [27].

2. The extension to real time

We introduce the real-time features of the logic with a simple example, expressed in the graphical representation for RTFIL. We shall assume that the reader is already familiar with the graphical representation for FIL introduced in Part I, Section 1.

Consider two interacting systems *AB* and *C* connected as shown in Fig. 1. System *AB* requests permission to perform an action by raising signal *a*, and system *C* authorizes the action by raising signal *c* and *AB* performs the action by raising signal *b*. Many examples of such request/response protocols occur in real-time systems. The wires *a* and *b* are outputs from *AB*, and the wire *c* is an output of *C*. Wires *a* and *c* also serve, respectively, as inputs to systems *C* and *AB*.

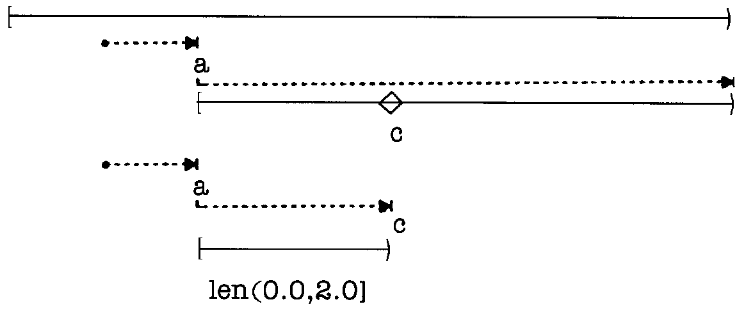
The external specification of *AB* is that whenever *a* is asserted, *b* remains false at least until the input *c* becomes true. Moreover, whenever the input *c* is asserted, if the module *AB* asserts its output *b*, it must do so within 4.0 time units. This specification is represented graphically in the following formula:



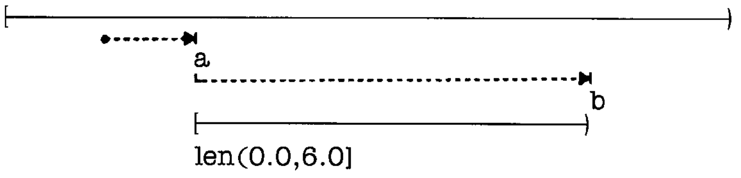
(1)

Note how the real-time property is stated by asserting that the duration of every interval from a c-state to a b-state is more than 0.0 and at most 4.0 time units. In general, the predicate $\mathsf{len}(d_1,d_2]$ is true at the initial point of an interval if the duration of the interval is more than d_1, and at most d_2, time units. This duration predicate on intervals is the only real-time construct in the syntax of RTFIL.

The following specification of C states that whenever input a is asserted, the output c becomes true within 2.0 time units. The property is stated as a conjunction of two formulae, the first stating a qualitative reactivity property, and the second specifying the real-time requirement:

 (2)

The following example is a property of the overall system that can be deduced from the preceding specifications; it states that the duration of every interval starting with a and ending with the first subsequent b is at most 6.0 time units:

 (3)

The deduction is shown in Fig. 2, with the endpoints of the intervals appropriately aligned to illustrate better the underlying temporal intuitions.

For the remainder of the paper, we revert to the more compact textual representation of formulae that we introduced in Part I. Recall that, using the textual representation, the formula (2) above would be written

$$\Box([\rightarrow a \mid \ \rightarrow) \diamond c \wedge [\rightarrow a \mid \ \rightarrow a, \rightarrow c)\,\mathsf{len}(0,2])$$

2.1. Syntax

The syntax for RTFIL is simply that of FIL in Part I, with the extra primitives $\mathsf{len}(0,d], d$ any non-negative rational, introduced at the level of propositions.

Thus, for the sake of recapitulation, the syntax of RTFIL is given by the BNF grammar for FIL (Section 2.2, Part I), except that we also have duration

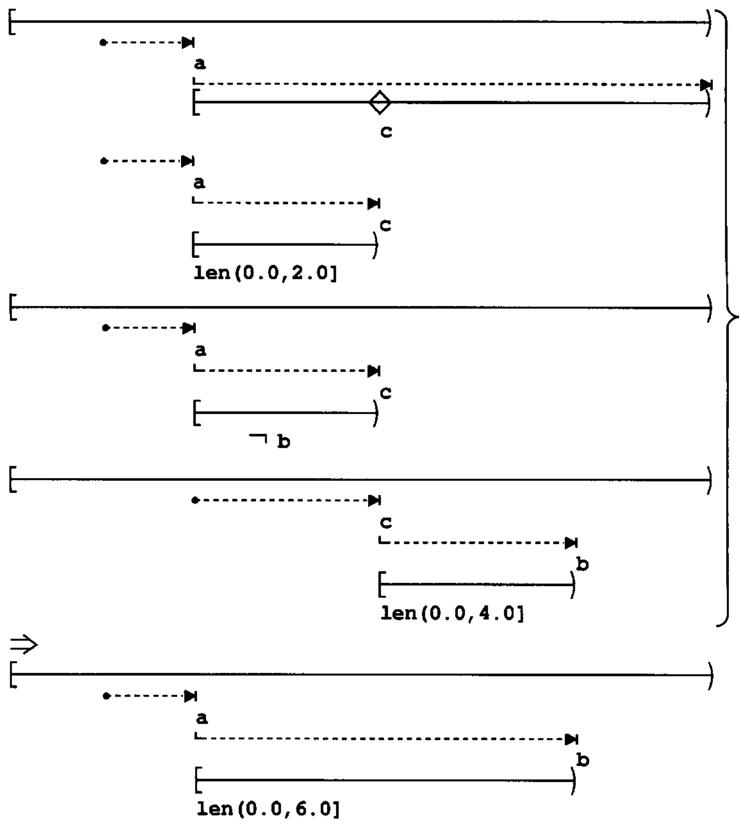


Fig. 2. Graphical representation of a deduction in RTFIL.

predicates.

$$f ::= \mathsf{true} \mid p \mid \mathsf{len}(0,d] \mid \neg f \mid f_1 \wedge f_2 \mid If$$
$$I ::= [-|\theta) \mid [\theta| \rightarrow) \mid [\theta_1|\theta_2)$$
$$\theta ::= \rightarrow f \mid \rightarrow f, \theta$$

Note that any formula can be used to define the target of a search. The predicate $\mathsf{len}(d_1,\infty)$ is defined by $\neg\mathsf{len}(0,d_1]$, within any context, and consequently $\mathsf{len}(d_1,d_2]$ is simply $\mathsf{len}(0,d_2] \wedge \mathsf{len}(d_1,\infty)$. We shall shortly comment upon our choice of the half-open bound rather than all four different possibilities.

2.2. Models

We introduce real time into our semantic model by considering computations over a dense time domain. Thus, RTFIL models are *dense traces*, providing a valuation to every proposition at every instant $t \in R$, where our time domain is the set R of non-negative real numbers. Note that we could have chosen Q, the non-negative rationals, as our time domain, requiring only denseness, but not completeness. However, it is convenient to assume completeness as this simplifies the presentation of the semantics. An RTFIL model is thus an element of $(2^{\mathscr{P}})^R$, where $\mathscr{P}$ is a set of primitive propositions, as before.

Since the time domain is now no longer order-isomorphic with ω, as was the case with the untimed logic, FIL, we must restrict our models appropriately so as to rule out non-Zeno behaviours (or equivalently, so that we enforce finite variability). Finite variability ensures that if any finite segment of R there are only finitely many state changes of the system. This automatically ensures non-Zenoness, which requires that in any infinite computation, time must progress beyond any bound.

We also make another restriction, which is not strictly necessary for our decidability results, but which is natural in a state-based view of the world. We do not admit models in which there are instantaneous states, i.e. states with no duration. Another logic that explicitly makes this restriction is the Duration Calculus [13]. Kurshan [18] also advocates this restriction in the context of modelling and verifying asynchronous systems of processes. In the context of RTFIL, without this restriction, we could have the anomalous situation of two events (marking, respectively, the transitions into and out of an instantaneous state) occurring at the same real time, yet being ordered one after the other. As indicated previously, such a semantics can be quite unintuitive. Note, however, that the time spent in a state can be arbitrarily small as long as that time is non-zero.

It is often convenient, for proofs by successive refinement, to use a logic that satisfies a property that, after Schneider [29], we call temporal interpolation: Between the current instant and the next instant at which the system is in a different state, the system resides in the current state. We therefore require our models to be right-continuous, a property that we define formally below. We complete the resulting model by closing the interval on the left, i.e. for every state there is a first instant at which the system is in that state (although, because of the above, there is no corresponding last instant).[3]

The following definition of admissibility is relative to an arbitrary discrete, and *possibly partial,* valuation function on R.

Definition 2.1 (*Admissibility*). A function $F: R \to X \uplus \{\bot\}$ is[4]

- *finitely variable* iff for any two elements $t_1 < t_2$ in R, there are only finitely many changes between t_1 and t_2;
- *right continuous* iff for any $t \in R$, $\lim_{t' \to t+} F(t') = F(t)$;
- *domain clopen* iff $\operatorname{dom} F = \{t \in R \mid F(t) \neq \bot\}$ is a left-closed right-open, not necessarily proper, segment of R;
- *image finite* iff $\operatorname{im} F = \{x \in X \mid \exists t \in R. F(t) = x\}$ is finite.

F is admissible iff it is all of the above.

[3] Note how each state resembles our overall domain R in which there is a first instant but no last instant. In the terminology of [16] this gives RTFIL the properties of homogeneity and reflection, which appear to be desirable while doing proofs by successive refinement.

[4] As in Part I, $\uplus$ is used to denote disjoint union.

With the above definition, an RTFIL formula may be interpreted on any admissible partial function $\mathcal{M}: R \rightarrow 2^{\mathscr{P}}$. When the model represents a computation, however, we also require it to be total, i.e. $\operatorname{dom} \mathcal{M} = R$.

Notice that an admissible computation $\mathcal{M}$ partitions the real line into at most ω contiguous segments $[0,t_1),[t_1,t_2),\ldots$ such that $\mathcal{M}$ is constant over each segment, and differs between successive segments. Our representation is therefore equivalent to the timed ω-string representation used elsewhere in the literature (see, for instance, [1]), an equivalence that we shall often implicitly exploit in the sequel. A formal definition of timed ω-strings appears in Section 3.1.1.

2.3. Semantics

The semantics that follow are a natural extension of the semantics for FIL, generalized to the dense domain R with the usual metric. We use the "locator" function λ for locating the result of a search and the "constructor" function $\mathscr{C}$ for constructing the subinterval, given the current interval and the states located by the searches. For brevity, we use $R_\perp$ to denote $R \cup \{\perp\}$, $R_{\perp\infty}$ to denote $R \cup \{\perp, \infty\}$, and $\perp_\mathcal{M}$ to denote the null model with $\operatorname{dom} \perp_\mathcal{M} = \emptyset$.

Definition 2.2. The search-locator function

$$\lambda: \mathbf{srchp}(\mathscr{P}) \times ((2^{\mathscr{P}})_\perp)^R \times R_\perp \rightarrow R_{\perp,\infty}$$

is defined by

- if $\mathcal{M} = \perp_\mathcal{M}$ or $t = \perp$ then $\lambda(\theta, \langle \mathcal{M}, t\rangle) = \perp$
- if $\mathcal{M} \neq \perp_\mathcal{M}$ and $t \neq \perp$ then

$$\lambda(-, \langle \mathcal{M}, t\rangle) = t$$

$$\lambda(\rightarrow, \langle \mathcal{M}, t\rangle) = \sup \operatorname{dom} \mathcal{M}$$

$$\lambda(\rightarrow a, \langle \mathcal{M}, t\rangle) = \begin{cases} \perp & \text{if } \langle \mathcal{M}, t'\rangle \not\models a \\ & \quad \text{for all } t' \geqslant t, t' \in \operatorname{dom} \mathcal{M} \\ \inf\{t' \mid t' \geqslant t, \langle \mathcal{M}, t'\rangle \models a\} & \text{otherwise} \end{cases}$$

$$\lambda(\rightarrow a, \theta, \langle \mathcal{M}, t\rangle) = \lambda(\theta, \langle \mathcal{M}, \lambda(\rightarrow a, \langle \mathcal{M}, t\rangle)\rangle)$$

The model-constructor function

$$\mathscr{C}: \mathbf{imod}(\mathscr{P}) \times ((2^{\mathscr{P}})_\perp)^R \times R \rightarrow ((2^{\mathscr{P}})_\perp)^R$$

is defined by

$$\mathscr{C}([\theta_1 | \theta_2), \langle \mathcal{M}, t\rangle) = \mathcal{M}_{[\lambda(\theta_1, \langle \mathcal{M}, t\rangle), \lambda(\theta_2, \langle \mathcal{M}, t\rangle))}$$

where $\mathcal{M}_{[t_1,t_2)}$ with $t_1, t_2 \in R_{\perp,\infty}$, represents the subcontext model defined by

$$\mathcal{M}_{[t_1,t_2)} = \perp_\mathcal{M} \quad \text{if } t_1 = \perp \text{ or } t_2 = \perp \text{ or } t_1 \geqslant t_2$$

and otherwise, $\mathcal{M}_{[t_1,t_2)}$ is the restriction of $\mathcal{M}$ to $[t_1, t_2)$.

Definition 2.3 (*Semantics*). The valuation of an RTFIL formula is defined at a point $t \in R$ in an admissible model $\mathscr{M} \in ((2^{\mathscr{P}})_\perp)^R$ using the satisfaction relation defined below:

- if $\mathscr{M} = \perp_{\mathscr{M}}$ then $\langle \mathscr{M}, t\rangle \models f$;
- if $\mathscr{M} \neq \perp_{\mathscr{M}}$ and $t \in \operatorname{dom} \mathscr{M}$ then

$$\begin{aligned}
&\langle \mathscr{M}, t\rangle \models \mathsf{true}\\
&\langle \mathscr{M}, t\rangle \models p \quad \text{for} p \in \mathscr{P} \text{ iff } p \in \mathscr{M}(t)\\
&\langle \mathscr{M}, t\rangle \models \neg f \quad \text{iff} \quad \langle \mathscr{M}, t\rangle \not\models f\\
&\langle \mathscr{M}, t\rangle \models f \wedge g \quad \text{iff} \quad \langle \mathscr{M}, t\rangle \models f \text{ and } \langle \mathscr{M}, t\rangle \models g\\
&\langle \mathscr{M}, t\rangle \models \mathsf{len}(0,d] \quad \text{iff} \quad t < \sup \operatorname{dom} \mathscr{M} \leqslant t + d\\
&\langle \mathscr{M}, t\rangle \models If \quad \text{iff} \quad \langle \mathscr{M}', \inf \operatorname{dom} \mathscr{M}'\rangle \models f \quad \text{where } \mathscr{M}' = \mathscr{C}(I, \langle \mathscr{M}, t\rangle)
\end{aligned}$$

A formula f is *satisfiable* iff there exists a total admissible model $\mathscr{M} \in (2^{\mathscr{P}})^R$ such that $\langle \mathscr{M}, 0\rangle \models f$. A formula f is *valid* iff every total admissible model is a satisfying model for f.

Theorem 2.4. *RTFIL is a conservative extension of FIL.*

Proof. We show that given any wff f of FIL (which is, therefore, also a wff of RTFIL), f is FIL-satisfiable iff it is RTFIL-satisfiable. For this purpose, we give two mappings, one that takes an FIL model $\mathscr{M}_{\mathrm{FIL}}$ for f and produces a corresponding RTFIL model $\mathscr{M}_{\mathrm{RTFIL}}$ that satisfies f, and another that takes us in the reverse direction. But these mappings are trivial.

($\Rightarrow$) We simply let $\mathscr{M}_{\mathrm{RTFIL}}(t) = \mathscr{M}_{\mathrm{FIL}}(\lfloor t \rfloor)$ for all $t \in R$, where $\lfloor t \rfloor$ is the greatest integer less than or equal to t.

($\Leftarrow$) From the admissibility of the RTFIL model, there exists a monotonically increasing sequence $\langle t_i \rangle_{i \in \omega}$ such that $\lim_{i\to\infty} t_i = \infty$, which partitions R so that $\mathscr{M}_{\mathrm{RTFIL}}(t)$ is constant over each $[t_i, t_{i+1})$. We simply let $\mathscr{M}_{\mathrm{FIL}}(i) = \mathscr{M}_{\mathrm{RTFIL}}(t_i)$ for each $i \in \omega$.

That each of these mappings preserves satisfaction is proved by induction on the structure of formulae and on the sequence using the semantics of each of the logics, a routine and tedious exercise. □

The mapping we gave above for going from an FIL model for f to an RTFIL model for f shows how we could have given an alternative dense time semantics to FIL. In fact, this can be done for any logic that is invariant under stuttering [10].[5]

[5] As is clear from the preceding discussion, our requirement of finite variability, makes each RTFIL model "isomorphic" to a timed ω-string (Definition 3.1).

2.4. On the choice of timing primitives

The following theorem gives RTFIL a property which, after [29], we call "temporal interpolation." Intuitively, this means that a system continues to remain in the "current" state until it undergoes some observable change, when it enters the "next" state. Thus, if a timed ω-string (see Section 3.1.1) is a model for an RTFIL formula, so is any other timed ω-string in which the only difference is the insertion of finitely many copies of a state in the old string, so long as the time stamps[6] for the newly inserted states lie between the time stamps of the previous state and the next state. The newly inserted states then comprise stuttering states. This property appears to have the same significance for proofs by successive refinement within a real-time framework as does the property of invariance under stuttering in a non-realtime framework.

The following theorem is a corollary to Theorem 3.10 which appears in Section 3.1.2.

Theorem 2.5. *Let f be any RTFIL formula and let $\mathcal{M}$ be an admissible model. Then for any $t \in R, \langle \mathcal{M}, t\rangle \models f$ iff there exists $\varepsilon > 0$ such that for all $t \leqslant t' < t + \varepsilon, \langle \mathcal{M}, t'\rangle \models f$.*

The theorem strengthens our observation made earlier regarding right continuity. It implies that the valuation of *any* RTFIL formula interpreted over an admissible model partitions the real line into a sequence of contiguous left-closed right-open intervals, over each of which the valuation is constant. Of course, this partition is at least as fine as that induced by the valuations for each of the primitive propositions mentioned in the formula, and often might be finer (when the formula mentions duration predicates).

The theorem also motivates our choice of $\mathsf{len}(0,d]$ and, by negation, $\mathsf{len}(d,\infty)$ as timing primitives. Had we, for instance, chosen $\mathsf{len}[d,\infty)$ (with the intuitive semantics) as a basic timing primitive, then the RTFIL model $\mathcal{M}$ defined on $\mathcal{P} = \{p\}$ by

$$\mathcal{M}(t) = \begin{cases} \emptyset & \text{for } t \in [0,1) \\ \{p\} & \text{otherwise} \end{cases}$$

violates Theorem 2.5 since

$$\langle \mathcal{M}, 0\rangle \models [-| \rightarrow p)\,\mathsf{len}[1,\infty)$$

but

$$\langle \mathcal{M}, t\rangle \not\models [-| \rightarrow p)\,\mathsf{len}[1,\infty)$$

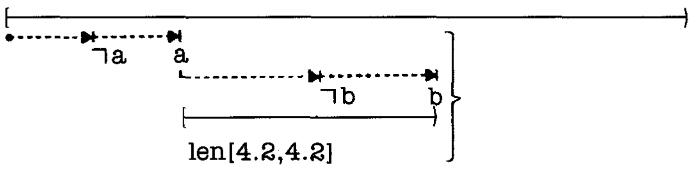
for any $t, 0 < t < 1$.

Theorem 2.5 is useful in proofs by successive refinement, and it appears to have the same significance for real-time temporal logics as does the qualitative notion of stuttering invariance for non-real-time temporal logics; see, for instance, [9, 18], where

[6] The *time stamp* of the nth state of the timed ω-string $\langle(\sigma_i, t_i)\rangle_i$ is t_n.

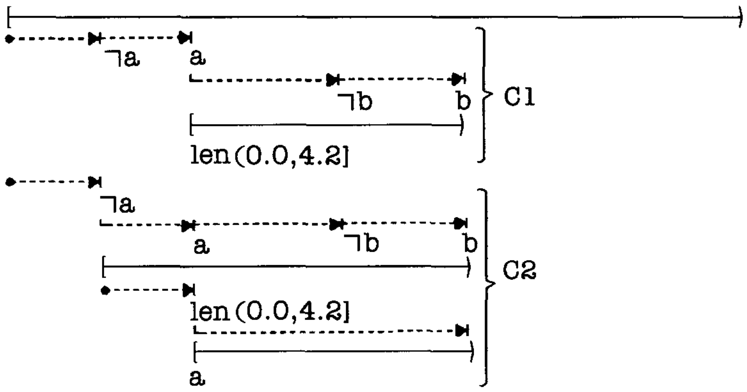
some general models, methods and calculi for refinement (and its inverse, reduction) are presented. Theorem 2.5 implies that RTFIL should fit smoothly into such a framework, since we are automatically ensured that any of the refinements that these methodologies permit would still preserve all of the RTFIL properties that were proved at the higher, more abstract level.

Because of the above choice of timing primitives, the logic might appear to lack the ability to specify, for instance, that the duration from the next positive transition of a to the subsequent positive transition of b is precisely 4.2. Recall that, in the syntax that we have just presented, the following formula is not allowed, since len$[d,d]$ is not a legal primitive:



(4)

However, the following formula does express the required condition:



(5)

where the conjunct C1 ensures that the first positive b-transition, following the a-transition, does not occur too late, and the conjunct C2 ensures that the a-transition occurs at the right moment. Note that both conjuncts are required to state the required condition, and neither suffices by itself.

However, whether or not RTFIL has the ability to state this property for arbitrary *temporal* formulae, a and b, is an open question.[7] It appears from our experience that such a general property is usually not needed in practice.

We note here, for the record, that even if we relax our notion of admissibility and/or allow the primitive len$[d,\infty)$, in addition to len$(0,d]$ (thus allowing the formula

[7] It is easy to see that such a property can be stated for arbitrary temporal formulae if one uses auxiliary variables.

$\mathsf{len}[d,d]$ to be expressible), the resulting logic is still decidable, by an appropriate (very minor) modification of the method that we present shortly, and with essentially no change in complexity. However, for reasons already stated, such a logic might have some drawbacks while using a refinement-based proof methodology, and its semantics would not be quite as natural.

3. Decision procedure

Since RTFIL is a conservative extension of FIL, the only new feature in the decision strategy is that required to deal with timed formulae, i.e. formulae involving the duration predicate. Here the careful groundwork of Part I, where we set up our notion of syntactic reductions between interval formulae, is useful again. Consider, for the moment, replacing each occurrence of a timing primitive by a fresh primitive proposition symbol and running the decision strategy for FIL on the resulting formula. If the formula is not satisfiable, then clearly the original formula is not satisfiable, either. However, if the rewritten formula is satisfiable, we must now check the effect of the presence of timing constraints in the original formula. At this juncture, Alur and Dill's timed Büchi automata (TBA) [3] come into play. Thus, the automata for our decision strategy are now TBAs rather than simply BAs.

A formal definition of TBAs is postponed until the next section. Intuitively, however, a TBA is like a BA, except that it is also equipped with a finite set of clocks. The TBA can activate a clock (to start with the value 0) on a transition, as well as check the reading (of active clocks) before taking a transition. All active clocks, however, progress at the same rate, representing the flow of real time.

With this brief description of TBAs, we sketch, using an example RTFIL formula, a strategy that such an automaton can follow to verify satisfiability. For simplicity, assume that we are given the formula

$$f \overset{\text{def}}{\equiv} [\rightarrow a| \rightarrow b)\,\mathsf{len}(2.0, 4.2]$$

and are asked to verify whether $\Box f$ holds on a given timed string.

Since we know how to deal with negated formulae, and since conjunctions and disjunctions can be handled, respectively, by a product of constituent automata or by non-deterministic choice between them, the general algorithm is a simple generalization of the given strategy. Moreover, since we have already discussed in Part I, informally as well as formally, the constructibility of contexts, here we shall ignore that issue, and let the reader fill in the (untimed) details.

Observe that f is equivalent to the condition that every a-state, at which b does not hold, must be separated from the first subsequent b-state by more than 2.0, but at most 4.2, time units. Of course, since this a-state will itself "persist" for some time (because of our assumption of right continuity), the same condition must be satisfied at *every* point in the intervening interval, call it S, before b becomes true. Thus, this condition must be verified at uncountably many points. However, observe that f is really the conjunction of two conditions:

- $f_1 \overset{\text{def}}{\equiv} [\rightarrow a \mid \ \rightarrow b)\,\mathsf{len}(0.0, 4.2]$, asserting that the upper bound holds, and
- $f_2 \overset{\text{def}}{\equiv} [\rightarrow a \mid \ \rightarrow b)\neg\mathsf{len}(0.0, 2.0]$, stating that the lower bound holds.

Observe that if f_1 holds at the "start" of S, as described above, it must also hold at *every point* in S. Thus, verifying, f_1 for the point marking the beginning of S suffices to verify that f_1 holds also for all the uncountably many points in S. But, this can be done simply by starting an upper-bound timer as soon as we enter S, and checking that its value does not exceed 4.2 when the first subsequent b-state is encountered (that is, when we exit S).

For verifying a lower-bound condition we argue as follows. If f_2 holds at the "end" of S, as described above, it must have held at *every point* in S. Thus, verifying the formula for the point marking the end of S subsumes its verification for all points within S. This argument works recursively – so if f_2 holds in the duration S' succeeding S, and b does not hold at S', then its verification for S' subsumes its verification for S. Our strategy therefore works as follows. If a holds in a state for which we must verify f_2 this is equivalent to verifying $f_3 \overset{\text{def}}{\equiv} [- \mid \ \rightarrow b)\neg\mathsf{len}(0.0, 2.0]$ at that state. But condition f_3 which states that the first subsequent b state is at least 2.0 in the future, must stop holding at some point in the future, when we get sufficiently close to the said b-state, but before we reach a b-state. Thus, before reaching that b-state, our automaton non-deterministically guesses at some point that the b-state is precisely 2.0 in the future – this is the first point at which $[- \mid \ \rightarrow b)\,\mathsf{len}(0.0, 2.0]$ starts holding (and continues to hold until we reach the b-state). At this point we start a "lower-bound" clock and verify, when b happens, that the clock reads precisely 2.0 time units.

For this particular example, it is clear that two clocks, one for timing the upper-bound requirement, and one for timing the lower-bound requirement, will suffice. However, we shall need more clocks if the right end of the interval is located through a series of searches. We encourage the reader to try out the case of, for instance, the formula $[- \mid \ \rightarrow b, \rightarrow c)\,\mathsf{len}(0, d]$, for which we shall need two clocks to verify this condition of every state at which it holds. When intervals are nested, these clocks must now be kept local to each active context within which a timing condition is being verified. This increases the number of timers even further.

Although the situation becomes more complicated – with a fair bit of timer juggling needed to ensure that we use only finitely many, but not too many, timers – it still remains "tractable." Also, even though the number of timers increases with the length of the search patterns in, and nesting depth of, the interval modalities surrounding a duration predicate, as we show in the sequel, this number does not grow too fast (each new search in an outermost context adds to the number by an additive factor, and each nesting increases the number by a multiplicative factor). In the next few sections we give an algorithm to execute a general strategy, a simple instance of which we sketched above.

3.1. Some preliminaries

Much of this section is a generalization of similar concepts for FIL, introduced in Section 3.2 of Part I. The important difference is that we must now also deal with the timing primitives $\mathsf{len}(0,d]$, and must explicitly consider *timed* strings, while extending those concepts to automata operating on strings representing the extensions of RTFIL models. Except for this difference, the concepts of subformula closure, reductions and Hintikka sets remain much the same as before. Therefore, rather than repeat the formal definitions in this context, we shall often simply refer the reader to the definitions from Section 3.2 of Part I and, where necessary, simply state the modifications or generalizations required for those definitions.

The initial portion of this section also introduces TBAs formally, and states related results, most of which can be found in [1], and which are used for the decision procedure.

3.1.1. Timed Büchi automata and timed ω-strings

As we mentioned earlier, the decision strategy for an arbitrary RTFIL formula can be executed by a finite-state strategy, using a finite set of real-valued timers. The concrete machine model capable of executing this strategy is the timed Büchi automaton (TBA) of Alur and Dill [3]. Thus, we reduce an arbitrary RTFIL formula to such an automaton, in the sense that the formula is satisfiable iff the language of the corresponding TBA is non-empty. Note, in this connection, that an admissible RTFIL model is essentially a timed string, in the sense of [3]. In fact, in some of our subsequent proofs we shall use the timed ω-string representation for RTFIL models rather than the dense map representation used in the previous section.[8]

Definition 3.1 (*Timed ω-string*). A timed ω-string over the alphabet Σ is an infinite sequence $\langle(\sigma_i, t_i)\rangle_{i\in\omega}$ in $(\Sigma \times R)^{\omega}$ such that $\langle t_i\rangle_{i\in\omega}$ is an unbounded, strictly monotonically increasing sequence, with $t_0 > 0$.

Note, thus, that timed ω-strings are non-Zeno. It is easy to see that a total admissible RTFIL model can be represented as a timed ω-string. Since our strings are always infinite, we shall usually skip the qualification ω and simply say "timed string" for a timed ω-string.

The following definition of a timed Büchi automaton (TBA) is a slightly specialized version of the TBA defined in [3]. The automata defined below are special in the sense that the only timer conditions employed are conjunctions of conditions of the form $c \leqslant t$ and $c = t$ (however, see also a related discussion preceding Theorem 3.5).

Definition 3.2 (*Timed Büchi Automation*). A timed Büchi automation $\mathscr{A}$ is a tuple $\langle \Sigma, \mathrm{S}, \mathrm{C}, \rho, \mathrm{S_I}, \mathrm{S_F}, \rangle$ where

[8] The presentation of TBAs and timed ω-strings that follows is slightly different from the form in which it appears in [25]. We have found the present approach technically more convenient.

- Σ is a finite input alphabet,
- $\mathbf{S}$ is a finite set of states,
- $\mathbf{C}$ is a finite set of clocks,
- $\rho : \mathbf{S} \times \Sigma \longrightarrow 2^{\mathbf{S} \times 2^{\mathbf{C}} \times 2^{\Phi(\mathbf{C})}}$ is the transition function, where $\Phi(\mathbf{C})$, the set of clock conditions, is the set of inequalities of the form $c \leqslant t$ and $c = t$, for $c \in \mathbf{C}$ and $t \in Q$,
- $\mathbf{S}_\mathrm{I} \subseteq \mathbf{S}$ is the set of possible initial states,
- $\mathbf{S}_\mathrm{F} \subseteq \mathbf{S}$ is the set of accepting states.

The transition function ρ defines, for each state $s \in \mathbf{S}$ and input $\sigma \in \Sigma$, a set of triples, where each triple $\langle s', C', \varphi \rangle \in \rho(s, \sigma)$ specifies a next state s', a set C' of clocks reset with that transition and a set φ of clock conditions that must be satisfied at the moment of the transition. We say that a clock assignment $\gamma \in R^{\mathbf{C}}$ *satisfies* a set of clock conditions $\varphi \subseteq \Phi(\mathbf{C})$ iff the set of inequalities $\varphi[c \leftarrow \gamma(c)]$ obtained by replacing each clock variable c in φ by the corresponding value $\gamma(c)$ is satisfied.[9] If $\langle s', C', \varphi \rangle \in \rho(s, \sigma)$, we say that ρ allows the transition $s \overset{\sigma, C', \varphi}{\longrightarrow} s'$.

A *run* of $\mathscr{A}$ on a timed ω-string $\sigma = \langle (\sigma_i, t_i) \rangle_i$ over Σ is an ω-string $\mathscr{R}(\mathscr{A}, \sigma) = \langle (s_i, \gamma_i) \rangle_i \in (\mathbf{S} \times R^{\mathbf{C}})^\omega$ satisfying

- *Initiality*: $s_0 \in \mathbf{S}_\mathrm{I}$, and for all $c \in \mathbf{C}$, $\gamma_0(c) = 0$,
- *Transitions*: for each i, there is a set $C_i \subseteq \mathbf{C}$ of clocks and a finite set $\varphi_i \subset \Phi(\mathbf{C})$ of clock conditions such that
  - ρ allows the transition $s_i \overset{\sigma_i, C_i \varphi_i}{\longrightarrow} s_{i+1}$,
  - the inequalities in $\varphi_i[C \leftarrow \gamma_i(c) + t_i - t_{i-1}]_{c \in \mathbf{C}}$ are satisfied, where $t_{-1} = 0$,
  - $\gamma_{i+1}(c) = 0$ for all $c \in C_i$,
  - $\gamma_{i+1}(c) = \gamma_i(c) + t_i - t_{i+1}$ for all $c \in \mathbf{C} \backslash C_i$.

We write $(s_0, \gamma_0) \overset{\sigma_0, t_0}{\longrightarrow} (s_1, \gamma_1) \overset{\sigma_1, t_1}{\longrightarrow} \cdots$ when these conditions hold. Such a run is *accepting* iff the set $\{i \mid s_i \in \mathbf{S}_\mathrm{F}\}$, is infinite. The language of a TBA is non-empty iff there is a timed ω-string over its alphabet on which it has an accepting run.

Intuitively, a TBA reads a timed ω-string over its alphabet and makes transitions satisfying its transition function. It has a finite set of clocks, which proceed at the same rate, and which it can reset with a transition or compare with rational constants. Transitions must satisfy the associated clock conditions for the input string to be consumed. The operational intuition for the run shown above is that the automaton stays in state s_i for all $t \in [t_{i-1}, t_i)$. At time t_i it moves into state s_{i+1} resetting the clocks in C_i. The remaining clocks have meanwhile advanced by the time spent in s_i. The input string σ intuitively represents the admissible model $\mathscr{M}_\sigma$ satisfying, for all $i \in \omega$, and all $t \in R$ such that $t_{i-1} \leqslant t < t_i$, $\mathscr{M}_\sigma(t) = \sigma_i$. We say that the TBA $\mathscr{A}$ *consumes* a timed Σ-string when there exists a run of $\mathscr{A}$ on the string and that it *accepts* the string when some such run is accepting. Since we disallow ε-moves by our automaton, the ω-trace of states of the automaton as it consumes a timed ω-string can also be regarded as an admissible function from R to $\mathbf{S}$.

[9] As usual the empty set of conditions imposes no conditions and, therefore, is always satisfied.

Observe that a TBA with no clocks is simply the BA we defined in Section 3.17 in Part I. When the set of clocks of a TBA is empty, its transition function can be regarded as a function $\rho : S \times \Sigma \longrightarrow 2^S$, and it ignores the timing information on a timed ω-string.

Definition 3.3 (*Untiming*). We define a polymorphic untiming function which, when given a

- timed ω-string $\langle \sigma_I, t_i \rangle_{i \in \omega}$ returns the untimed ω-string

$$\mathbf{untime}(\langle \sigma_i, t_i \rangle_{i \in \omega}) = \langle \sigma_i \rangle_{i \in \omega}$$

- TBA $\mathscr{A} = \langle \Sigma, \mathrm{S}, \mathrm{C}, \rho, \mathrm{S_I}, \mathrm{S_F} \rangle$ returns the BA

$$\mathbf{untime}(\mathscr{A}) = \langle \Sigma, \mathrm{S}, \rho', \mathrm{S_I}, \mathrm{S_F} \rangle$$

  where the transition function $\rho' : \mathrm{S} \times \Sigma \longrightarrow 2^{\mathrm{S}}$ is defined by

$$\rho'(s, \sigma) = \{s' \mid \text{for some } C, \varphi, \langle s', C, \varphi \rangle \in \rho(s, \sigma)\}$$

The following lemma is immediate from the above definition; it is easy to see that its converse is not valid.

Lemma 3.4. *For a timed ω-string σ and TBA $\mathscr{A}$, if $\mathscr{A}$ accepts σ then* **untime**($\mathscr{A}$) *accepts* **untime**(σ).

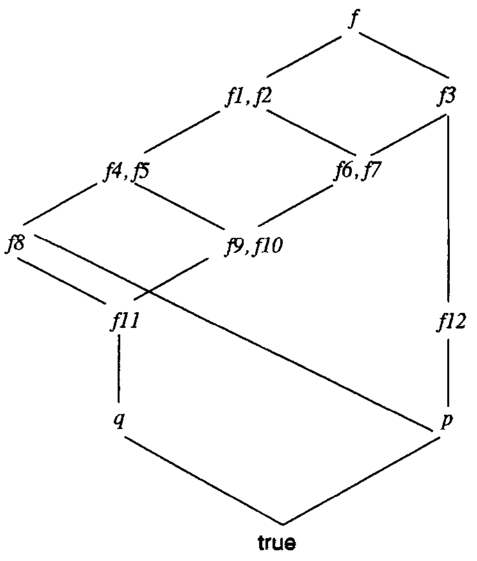
Observe that the admissibility requirement on timed strings makes the acceptance criterion for our TBAs slightly more restrictive than that in [3]. However, by a simple modification of our TBAs, namely by introducing a new “admissibility checking clock”, which is always active, and which is reset at every transition after checking for a positive reading of the clock (verifying that some non-zero time has passed since the last transition was made), we can use the emptiness algorithm of [3] without any modification.

Theorem 3.5 (Alur and Dill [3]). *It is decidable whether the language of a TBA is empty.*

3.1.2. Subformulae, reductions and extensions

The concepts of subformula closure set, reductor set and reductions on interval formulae for FIL, introduced in Part I, suffice for RTFIL, relativized now to the language of RTFIL, and with the obvious, but important, rider[10] that duration predicates $\mathsf{len}(0, d]$ are not purely propositional in the sense of Section 2.2 in Part I. Clearly, the truth of a duration predicate depends not only on the truth of primitive propositions at the point of evaluation, but also on the temporal context of the evaluation.

[10] Required only for clause 9 in Definition 3.1 of Part I.



f	$[\rightarrow p \mid \rightarrow p, \rightarrow q)\neg\mathsf{len}(0,3]$
f_1	$[- \mid \rightarrow p, \rightarrow q)\neg\mathsf{len}(0,3]$
f_2	$[- \mid \rightarrow p, \rightarrow q)\mathsf{len}(0,3]$
f_3	$[\rightarrow p \mid \rightarrow q)\neg\mathsf{len}(0,3]$
f_4	$[- \mid \rightarrow p, \rightarrow q)\mathsf{false}$
f_5	$[- \mid \rightarrow p, \rightarrow q)\mathsf{true}$
f_6	$[- \mid \rightarrow q)\neg\mathsf{len}(0,3]$
f_7	$[- \mid \rightarrow q)\mathsf{len}(0,3]$
f_8	$[\rightarrow p, \rightarrow q \mid \rightarrow)\mathsf{false}$
f_9	$[- \mid \rightarrow q)\mathsf{false}$
f_{10}	$[- \mid \rightarrow q)\mathsf{true}$
f_{11}	$[\rightarrow q \mid \rightarrow)\mathsf{false}$
f_{12}	$[\rightarrow p \mid \rightarrow)\mathsf{false}$

Fig. 3. Example illustrating the subformula closure definition for RTFIL. Edges in the Hasse diagram are implicitly directed downward, and denote strict inclusion of subformula closure sets.

The following examples illustrate the definitions in the context of a typical formula containing timing primitives.

Example 3.6. Let f be the formula $[\rightarrow p \mid \rightarrow p, \rightarrow q)\neg\mathsf{len}(0,3]$ where $p, q \in \mathscr{P}$ and let $f_1, \ldots, f_{12}$ represent the subformulae shown in Fig. 3. The subformula closure, **scl**(f), consists of precisely the formulae f, $f_1, \ldots, f_{12}, p, q$, true and all their negations. This is shown in Fig. 3 in the form of a Hasse diagram.

That Lemma 3.4 of Part I extends also to RTFIL should be fairly clear from the preceding example. The definitions of size and depth for FIL are extended to RTFIL by stipulating[11]

$$\text{size}(\mathsf{len}(0,d]) = \text{depth}(\mathsf{len}(0,d]) = 1$$

We shall treat the complexity arising from the presence of the constants d separately later.

As a result of the above, Lemma 3.6 of Part I also extends to RTFIL; see Lemma 3.20.

Reductor sets, reducibility and reductions retain their definitions, now relativized to RTFIL's syntax and the corresponding definition of **scl**. These are illustrated in the following series of examples.

[11] For the purposes of induction on the structure of formulae , however we shall assume, as in Part I, that the size of true as well as that of each logical connective is 1, and that of duration predicates, like that of primitive propositions, is 2. Recall that this choice gives us the property that whenever **scl**$(f_1) \subset$ **scl**(f_2), either size$(f_1) <$ size(f_2) or depth$(f_1) <$ depth(f_2).

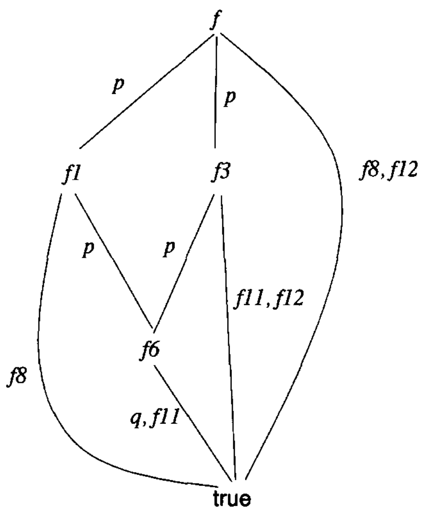


Fig. 4. Example illustrating reductions for RTFIL. The edges are implictly directed downward.

Example 3.7. Continuing with Example 3.6 in Fig. 4, if a formula f' is reachable from a formula f'' above it by a direct edge labelled with a formula a, then $f' \prec_a f''$. Thus, the fanout labels of a node f' are precisely the formulae in **red**(f'). For instance, f is p-reducible but q-irreducible. Moreover, p transitively reduces f to f_6. This reduced formula is now q-reducible, so that $\mathsf{true} \prec^*_{\{p,q\}} f$. Note also that f_8 directly reduces f to true.

Recall that for a wff a, the parameterized reduction operator $\prec_a$ on wff, has been defined so that $f' \prec_a f$ guarantees that $a \Rightarrow (f' \equiv f)$ as well as $\mathbf{scl}(a) \subset \mathbf{scl}(f)$ and $\mathbf{scl}(f') \subset \mathbf{scl}(f)$. RTFIL satisfies the following counterpart of Lemma 3.11 in Part I, obtained by the obvious modification of replacing the index i by the time $t \in R$. The proof follows that of Lemma 3.11 of Part I.

Lemma 3.8. *Let f, f' and a be formulae, and let $\mathcal{M}$ be a model such that $\langle \mathcal{M}, t\rangle \models a$ and $f' \prec_a f$. Then $\langle \mathcal{M}, t\rangle \models f$ iff $\langle \mathcal{M}, t\rangle \models f'$.*

Example 3.9. Thus, for our running example, $p \Rightarrow (f \equiv f_1)$, $(p \wedge q) \Rightarrow f$ and $f_8 \Rightarrow f$. Note also that, for any formula f, the formulae which are the (transitive) reducts of f give rise to a complete lattice under the relation "is a reduct of."

The concept of model extension introduced in Part I, continues to have its intuitive meaning, so that for an RTFIL model $\mathcal{M}$, we have for all $t \in R$, $\mathcal{M}^f(t) = \{f_1 \in \mathbf{scl}(f) \mid \langle \mathcal{M}, t\rangle \models f_1\}$. Note that $\mathcal{M}^f$ exists and is unique. We do not need to be able to construct it, just to be able to use the fact that it exists. However, for the case of an admissible model, we can give a straightforward recursive construction by exploiting the fact that such a model can be represented as a timed ω-string. The next theorem shows that, in fact, the extension of a model is also admissible and, thus, also representable as a timed ω-string, over the *extended alphabet* $2^{\mathbf{scl}(f)}$. This theorem plays a crucial role not only in providing intuition for why the method of automata works for RTFIL, but also in the proof of correctness, since it allows us, as in the case of FIL, to carry out all of the arguments in terms of extended models, rather than the models themselves.

Theorem 3.10. *Admissibility of (total) models is preserved under extension.*

Recall that the real line is partitioned by any primitive proposition p into a sequence of segments over each of which the valuation of p is constant. We may extend this concept to formulae in $\mathbf{scl}(f)$, such that two points $t_1 \leqslant t_2 \in R$ are in the same equivalence class iff all points t such that $t_1 \leqslant t \leqslant t_2$ yield the same valuation for all formulae in the set. Our proof (which appears in an appendix) of Theorem 3.10 makes use of the fact that the partition of the real-line induced by any RTFIL formula f, not involving duration predicates, is at most as fine as the coarset partition that refines the partitions induced by the formulae in $\mathbf{scl}(f) \backslash \{f, \neg f\}$.

Our definition of reductions yields Lemma 3.11, which is the RTFIL counterpart of Lemma 3.12 in Part I for FIL. The first two clauses of both lemmas are essentially the same; the last two clauses of Lemma 3.11 below relate the timing constraints between consecutive states, and capture the essence of the modification that must be made in the decision algorithm of FIL to obtain a decision algorithm for RTFIL. This motivates directly the construction of the untimed automation $\mathscr{A}_t$ (Definition 3.23).

Lemma 3.11. *Let $\mathscr{M}$ be an admissible model, let $t, t' \in \mathrm{dom}\mathscr{M}$, and let $f_1 \in \mathbf{scl}(f)$ be $\mathscr{M}^f(t)$-irreducible. If there is a least $t' > t$ such that $\mathscr{M}^f(t) \neq \mathscr{M}^f(t')$, then*

1. *if f_1 is $\mathscr{I}[\theta_1|\theta_2\rangle f_2$ where θ_1 is not $-$ then $\langle\mathscr{M}, t\rangle \models f_1$ iff $\langle\mathscr{M}, t'\rangle \models f_1$;*
2. *if f_1 is $\mathscr{I}\neg[\theta_1|\theta_2\rangle f_2$ where θ_1 is not $-$ then $\langle\mathscr{M}, t\rangle \models f_1$ iff both $\langle\mathscr{M}, t'\rangle \models f_1$ and $\langle\mathscr{M}, t'\rangle \not\models \mathscr{I}\mathsf{false}$;*
3. *if f_1 is $\mathscr{I}\mathsf{len}(0,d]$ and $\langle\mathscr{M}, t\rangle \models f_1$, then $\langle\mathscr{M}, t'\rangle \models \mathscr{I}\neg\mathsf{len}(0,d]$ iff $\langle\mathscr{M}, t'\rangle \models \mathscr{I}\mathsf{false}$;*
4. *if f_1 is $\mathscr{I}\neg\mathsf{len}(0,d]$ and $\langle\mathscr{M}, t\rangle \models f_1$, then $\langle\mathscr{M}, t'\rangle \not\models \mathscr{I}\mathsf{false}$.*

Intuitively, in the first case, if $\mathscr{I}[\theta_1|\theta_2\rangle$ can be constructed, it lies in the strict future of t, and therefore in the reflexive future of t'. In the second case, $[\theta_1|\theta_2\rangle$ can be constructed within $\mathscr{I}$ (its surrounding context), so $\mathscr{I}$ cannot collapse at t'. For the third case, $\mathscr{I}$ must collapse at t' since its suffix cannot have a longer duration. Finally, for the last case, $\mathscr{I}$ cannot collapse before its duration becomes less than d (at the earliest such point $\mathscr{I}$ $\mathsf{len}(0,d]$ must hold).

Proof of Lemma 3.11. The proofs of the first two clauses are quite similar to those for Lemma 3.12 in Part I. We consider below only the last two clauses.

Clause 3. We proceed by induction on the depth of $\mathscr{I}$. For the base case, let $\langle \mathscr{M}, t\rangle \models \mathsf{len}(0,d]$. We have $\sup\operatorname{dom}\mathscr{M} \leqslant t+d$. Since $t' > t$, clearly $\sup\operatorname{dom}\mathscr{M} \leqslant t'+d$, so that $\langle \mathscr{M}, t'\rangle \models \mathsf{len}(0,d]$ and $\langle \mathscr{M}, t'\rangle \not\models \mathsf{false}$.

For the induction step, let $f_1 = [-|\theta)\mathscr{I}_n\mathsf{len}(0,d]$, where the nesting depth of $\mathscr{I}_n$ is equal to n, and that of $[-|\theta)$ is at most n. Since f_1 is $\mathscr{M}^f(t)$-irreducible, we have $\langle \mathscr{M}, t\rangle \not\models a$ for all $a \in \mathbf{red}(f_1)$ and, as in the proof of clause 1 of Lemma 3.12 of Part I, we can conclude that $t'' = \lambda(\theta, \langle \mathscr{M}, t\rangle) = \lambda(\theta, \langle \mathscr{M}, t'\rangle) \geqslant t'$. Denoting $\mathscr{M}' = \mathscr{C}([-|\theta), \langle \mathscr{M}, t\rangle)$ and $\mathscr{M}'' = \mathscr{C}([-|\theta), \langle \mathscr{M}, t'\rangle)$, we can conclude that $\mathscr{M}''$ is a "suffix" of $\mathscr{M}'$, i.e. $\operatorname{dom}\mathscr{M}' = [t,t'')$ and $\operatorname{dom}\mathscr{M}'' = [t',t'')$. Using the semantics, we may conclude that, for any formula g,

$$\langle \mathscr{M}', t'\rangle \models g \quad \text{iff} \quad \langle \mathscr{M}'', t'\rangle \models g \tag{$*$}$$

Recall, from our definition of reductors, that $b \in \mathbf{red}(\mathscr{I}_n\mathsf{len}(0,d])$ iff $[-|\theta)b \in \mathbf{red}(f_1)$. Moreover, from the semantics, for any formula b, we have $\langle \mathscr{M}', t\rangle \models b$ iff $\langle \mathscr{M}, t\rangle \models [-|\theta)b$, and $\langle \mathscr{M}'', t'\rangle \models b$ iff $\langle \mathscr{M}, t'\rangle \models [-|\theta)b$. From the irreducibility of f_1 at $\langle \mathscr{M}, t\rangle$, we have the irreducibility of $\mathscr{I}_n\mathsf{len}(0,d]$ at $\langle \mathscr{M}', t\rangle$. Since $\langle \mathscr{M}, t\rangle \models f_1$, we have $\langle \mathscr{M}', t\rangle \models \mathscr{I}_n\mathsf{len}(0,d]$. Using the induction hypothesis, we can conclude that $\langle \mathscr{M}', t'\rangle \models \mathscr{I}_n\neg\mathsf{len}(0,d]$ iff $\langle \mathscr{M}', t'\rangle \models \mathscr{I}_n\mathsf{false}$. From $(*)$ above, we can conclude that $\langle \mathscr{M}'', t'\rangle \models \mathscr{I}_n\neg\mathsf{len}(0,d]$ iff $\langle \mathscr{M}'', t'\rangle \models \mathscr{I}_n\mathsf{false}$, and the result now follows, using the semantics.

Clause 4. The argument for the inductive step is similar to that in the previous clause, with a suitably modified induction hypothesis. For the base case, assume that $\langle \mathscr{M}, t\rangle \models \neg\mathsf{len}(0,d]$; so $t+d < \sup\operatorname{dom}\mathscr{M}$. We have two cases: either $\sup\operatorname{dom}\mathscr{M}$ is infinite or finite. If it is infinite, then every $t'' > t$ is in $\operatorname{dom}\mathscr{M}$; in particular, $t' \in \operatorname{dom}\mathscr{M}$, so $\langle \mathscr{M}, t'\rangle \not\models \mathsf{false}$. On the other hand, if $\sup\operatorname{dom}\mathscr{M}$ is finite, then there exists a t'' satisfying $t < t'' < \sup\operatorname{dom}\mathscr{M}$ (for example, choose $t'' = \sup\operatorname{dom}\mathscr{M} - d/2$), and $\langle \mathscr{M}, t''\rangle \models \mathsf{len}(0,d]$. Since $\mathscr{M}^f(t'') \neq \mathscr{M}^f(t)$, we conclude that $(t<)t' \leqslant t''$ $(< \sup\operatorname{dom}\mathscr{M})$. Thus, $\langle \mathscr{M}, t'\rangle \not\models \mathsf{false}$ in this case also. □

Example 3.12. Let $\mathscr{M}$ be defined by $\mathscr{M}(t) = \emptyset$ for $t \in [0,1)$, $\mathscr{M}(t) = \{p\}$ for $t \in [1,7)$, and $\mathscr{M}(t) = \{p,q\}$ for $t \in [7,\infty)$. The reader can verify that $\mathscr{M}^f(t)$ is defined by the matrix shown in Table 1, where for a given row, denoting an interval I of R, a formula appearing in a column is in $\mathscr{M}^f(t)$, $t \in I$, iff the entry in that column is a 1, and its negation is in $\mathscr{M}^f(t)$ iff the entry in that column is a 0. The example also illustrates the ideas in Lemmas 3.8 and 3.11.

Finally, the reader should recall from Part I, Definition 3.13, the definition of the *basis* $\langle f\rangle_S$ of a formula f with respect to a set S of formulae . As before, this will be useful in the description of the eventuality automation.

Example 3.13. For the case of Example 3.12, for instance, $f_6 = \langle f\rangle_{\mathscr{M}^f(t)}$ for $t \in [1,7)$ and $\mathsf{true} = \langle f\rangle_{\mathscr{M}^f(t)}$ for $t \in [7,\infty)$. Note also that f is irreducible at $t \in [0,1)$ and is (trivially) its own basis with respect to $\mathscr{M}^f(t)$, $t \in [0,1)$.

Table 1
Example illustrating model extension

	true	p	q	f	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
$[0,1)$	1	0	0	1	1	0	1	0	1	1	0	0	0	1	0	0
$[1,4)$	1	1	0	1	1	0	1	0	1	1	0	0	0	1	0	0
$[4,7)$	1	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0
$[7,\infty)$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0

3.1.3. Interval reductions, clocks and conditions

In Example 3.12 there are no formulae involving nested interval modalities. However, in general, a formula may involve nested modalities, so that for ease in describing the decision procedure, we require the more general machinery below.

The unit of manipulation by the TBA is a timed current interval formula of the form $\mathscr{I}\,\mathsf{len}(0,d]$ or $\mathscr{I}\neg\mathsf{len}(0,d]$, where $\mathscr{I} = [-|\theta_1)[-|\theta_2)\cdots[-|\theta_n)$ is a sequence of zero or more current interval modalities. For the case of such formulae , we also need the concept of an interval reduct. Interval reduction is a (purely syntactic) relation on strings of current interval modalities and is parameterized by a set of formulae .

Definition 3.14 (*Interval reduction*). Let $\mathscr{I}$ and $\mathscr{I}'$ denote strings of current interval modalities and let S be a set of RTFIL formulae . The $\mathscr{I}'\sqsubset_S\mathscr{I}$ iff $\mathscr{I}'\mathsf{true} \prec_S \mathscr{I}\mathsf{true}$. The transitive closure of $\sqsubset_S$ is represented by $\sqsubset_S^+$, and the reflexive closure of $\sqsubset_S^+$ by $\sqsubset_S^*$. When $\mathscr{I}'\sqsubset_S^+\mathscr{I}$, we say that $\mathscr{I}$ is S-reducible and, moreover, that $\mathscr{I}'$ is an *interval reduct* of $\mathscr{I}$ with respect to S.

Note that $\mathscr{I}'$ above may be the "empty" sequence of modalities (which we suppress), which is always irreducible. Often we shall simply say "$\mathscr{I}'$ is a reduct of $\mathscr{I}$" instead of "$\mathscr{I}'$ is an interval reduct of $\mathscr{I}$," where there is no confusion.

Among the possible reductions on an interval modality is a special kind of reduction called a *collapsing reduction.* A collapsing reduction may trigger the checking of clock conditions on a transition that was just taken, and so our procedure must treat it differently from a non-collapsing reduction. This will become clear later when we describe the TBA construction (also see remarks below, regarding the role of reductions in the timer construction).

Definition 3.15 (*Collapsing reductions*). Let $\mathscr{I} = I_1I_2\cdots I_n$ and $\mathscr{I}' = I_1'I_2'\cdots I_{n-1}'$ be such that $\mathscr{I}'\sqsubset_S^+\mathscr{I}$. Then $\mathscr{I}'$ is a *collapsed reduct* of $\mathscr{I}$ and the corresponding operation is a *collapsing reduction*, written $\sqsubset_S^\downarrow$.

Intuitively, in the above definition (when S represents a point in the extension of a model), if $I_n = [-|\rightarrow a_1,\cdots,\rightarrow a_k)$, then $I_1\cdots I_{n-1}a_i \in S$ for each $i \in \{1\cdots k\}$. In other words, the nth nested context I_n collapses.

The important property of interval reductions that we require for the sequel is as follows. Suppose $\mathscr{M}$ is admissible, $t \in R$ and $\mathscr{I}$ is $\mathscr{M}^f(t)$-irreducible. Suppose further that there is a next (least) time $t' > t$ such that $\mathscr{M}^f(t') \neq \mathscr{M}^f(t)$. Let $\mathscr{I}$ be of the form $\mathscr{I}_1[-| \rightarrow a, \theta)\mathscr{I}_2$, and let $\mathscr{I}_1 a \in \mathscr{M}^f(t')$. Then $\mathscr{I}$ is $\mathscr{M}^f(t')$-reducible to $\mathscr{I}_1[-|\theta)\mathscr{I}_2$. Intuitively, at time t', the modality $\mathscr{I}$ is "equivalent to" the syntactically simpler modality $\mathscr{I}_1[-|\theta)\mathscr{I}_2$. Moreover, when $\mathscr{I}$ is of the form $\mathscr{I}_1[-|\rightarrow a)$, then the reduction is collapsing, $\mathscr{I}_1 \sqsubset^{\downarrow}_{\mathscr{M}^f(t')} \mathscr{I}$, and $\mathscr{I}$ yields the empty subcontext at t' in $\mathscr{M}$ (recall the subcontext function of Definition 2.2).

Example 3.16. Continuing with Example 3.12, the modality $[-|\rightarrow q)$ collapses at all $t \in [7, \infty)$. The modality $[-|\rightarrow p, \rightarrow q)$ reduces to $[-|\rightarrow q)$ at $t \in [1, \infty)$ and collapses at $t \in [7, \infty)$. In each case, the set with respect to which the collapse or reduction occurs is $\mathscr{M}^f(t)$ for the appropriate t.

These syntactic reductions on intervals are used by the automaton to activate clocks and to keep track of the "remaining searches" in an interval that is being timed by an active clock.

The clock-closure and clock-condition sets, defined below, represent the clocks and associated conditions required by a TBA during the satisfiability procedure. Thus, while deciding a formula f, the automation $\mathscr{A}(f)$ never needs any timers other than those in **clocks**(f) and the conditions appearing on its transitions are all contained in the set **clkconds**(f).

Definition 3.17 (*Clock set*). Given a formula f its *clock set*, denoted **clocks**(f), is the set

$$\{c_i^{\gamma,\mathscr{I},d} \mid \mathscr{I}\mathrm{len}(0,d] \in \mathbf{scl}(f), i \in \{1,\ldots,n_{\mathscr{I}}\}, \gamma \in \{\alpha,\beta\}, n_{\mathscr{I}} = \mathrm{card}\{\mathscr{I}' \mid \mathscr{I}' \sqsubset^*_{\mathbf{scl}(f)} \mathscr{I}\}\}$$

Definition 3.18 (*Clock conditions set*). Given a formula f, its *clock condition set*, **clkconds**(f) is the set of conditions of the form
- $c \leqslant d$ for all $c = c_i^{\alpha,\mathscr{I},d} \in$ **clocks**(f),
- $c = d$ for all $c = c_i^{\beta,\mathscr{I},d} \in$ **clocks**(f).

In the sequel, α-clocks are used to enforce upper-bound constraints and β-clocks to enforce lower-bound constraints. States in the TBA for a formula will contain "clock-activity sets," which indicate the clocks that are active. Furthermore, with each active clock $c_i^{\gamma,\mathscr{I} d}$ (where γ is either α or β), we shall associate a *context-tag*, ranging over $\{\mathscr{I}' \mid \mathscr{I}' \sqsubset^*_{\mathbf{scl}(f)} \mathscr{I}\}$. Loosely speaking, the clock $c_i^{\gamma,\mathscr{I},d}$ will be made active at a state when it is necessary to time the context $\mathscr{I}$. The context tag for this clock is initialized to $\mathscr{I}$ when the clock is made active, and it is updated at each transition, to reflect the *remaining context* that is being timed; interval reductions are useful for carrying out these tag upadates. When a transition causes a collapsing reduction of the tag for its clock $c_i^{\gamma,\mathscr{I},d}$, the clock will be compared to the upper or lower bound of d.

Example 3.19. Let f be $[\rightarrow p| \rightarrow p, \rightarrow q)\neg\mathsf{len}(0,3]$. Then **clocks**$(f)$ contains the clocks, $c_1^{\alpha,[-|\rightarrow p,\rightarrow q),3} c_2^{\alpha,[-|\rightarrow p,\rightarrow q),3}, c_1^{\alpha,[-|\rightarrow q),3}$, and their β counterparts. The clock condition associated with $c = c_1^{\alpha,[-|\rightarrow q),3}$ is $c \leqslant 3$ and with its β-counterpart $c' = c_1^{\beta,[-|\rightarrow q),3}$ is $c' = 3$.

Lemma 3.20. *For a formula f of size n and depth k, $|$**scl**$(f)| = \mathrm{O}(n^k)$ and $|$**clocks**$(f)| = \mathrm{O}(n^{2k})$.*

3.2. Decision procedure

We now have most of the formal machinery required to describe the construction of the TBA $\mathscr{A}_{\mathrm{m}}(f)$ corresponding to a formula f, whose satisfiability we are interested in checking. The construction of $\mathscr{A}_{\mathrm{m}}$ is described below in four steps.

In the first step, we construct a BA $\mathscr{A}_{\mathrm{u}}(f)$ whose states are subsets of **scl**(f). This part of the construction is quite similar to the construction of the local automaton for a formula in the untimed logic FIL (Part I, Section 3.3.1). Intuitively, the automaton, that is produced in this first step ensures that all safety conditions that are independent of real time are correctly handled. This automaton also checks some simple consistency conditions relating to real time. More precisely, $\mathscr{A}_{\mathrm{u}}(f)$ accepts the untiming of any timed string corresponding to a model of f. However, since $\mathscr{A}_{\mathrm{u}}(f)$ does not fully take into account the real-time constraints imposed by f, it may also accept many other strings. The states of $\mathscr{A}_{\mathrm{u}}(f)$ are, however, annotated with formulae involving duration predicates. These formulae encode real-time constraints imposed by f, and are used in the next step to augment the automaton with real-time recognition capabilities.

The second step comprises the heart of the construction. In this step the timing assertions, of the form $\mathscr{I}\mathsf{len}(0,d]$ and $\mathscr{I}\neg\mathsf{len}(0,d]$, annotating the states of $\mathscr{A}_{\mathrm{u}}(f)$ are used to construct a TBA $\mathscr{A}_{\mathrm{t}}(f)$ in such a manner that all timing constraints are encoded into the timer-related actions of the TBA. Each state of $\mathscr{A}_{\mathrm{t}}(f)$ has a set of "active clocks," a subset of **clocks**(f), that it uses to enforce the timing assertions. Associated with each active clock is an appropriate context tag which, loosely speaking, represents the remaining suffix of a context being timed by that clock. The edges of $\mathscr{A}_{\mathrm{t}}(f)$ have clock resetting and comparison actions, and the transitions of $\mathscr{A}_{\mathrm{t}}(f)$ ensure that context tags associated with clocks are updated in a consistent fashion. Thus, $\mathscr{A}_{\mathrm{t}}(f)$ ensures that all safety and timing-dependent properties are checked. In this connection, it is useful to note that a time-bounded liveness property is really a safety property – the time bound must not pass before the liveness property is satisfied. That the requisite time must eventually pass – the condition of non-Zenoness – is essentially an *implicit* liveness condition.

To take care of the timeless liveness conditions, we construct the eventuality automation $\mathscr{A}_{\mathrm{e}}(f)$ in the third step of the construction. This eventuality automation is a pure BA, without any timers. It is constructed in much the same manner as for FIL (Section 3.3.2, Part I).

The final automaton $\mathscr{A}_{\mathrm{m}}(f)$ is the product of $\mathscr{A}_{\mathrm{t}}(f)$ and $\mathscr{A}_{\mathrm{e}}(f)$. The formula f is satisfiable iff the TBA $\mathscr{A}_{\mathrm{m}}(f)$ accepts some timed string. The latter question can be answered by a celebrated result of Alur and Dill [3].

The construction reveals an interesting and, from the point of view of expressibility, important aspect of RTFIL: the local automaton $\mathscr{A}_{\mathrm{t}}(f)$ might consume non-Zero runs, but $\mathscr{A}_{\mathrm{m}}(f)$ does not. This is because, in RTFIL, unlike, for instance, in MITL [4], *there is an implicit liveness condition associated with every timing constraint*, namely, that the right endpoint of the interval satisfying the timing constraint is eventually found. This allows us to dispense with the "progressiveness check" that Alur and Dill [3] require. In effect, our decision procedure requires only the timing consistency algorithm of [14]. Observe also the expressiveness implications: In RTFIL, a time-bounded eventuality is stated as a conjunction of an unbounded eventuality and the time bounds within which that eventuality must be satisfied.

3.2.1. *Hintikka sets*

As before, we first restrict our attention to a specific type of subsets of $\mathbf{scl}(f)$, called Hintikka sets, with the property that any state in an extension of a model for f is a Hintikka set.

Definition 3.21 (*Hintikka sets*). We refer the reader to the definition of Hintikka sets for FIL (Section 3.15, Part I). A Hintikka set for RTFIL is constructed using the same rules, modulo our extended definitions for **scl** and $\prec$, and the following additional rule: [12]

8. for all $\mathsf{len}(0,d] \in \mathbf{scl}(f), \neg\mathsf{len}(0,d] \in \mathrm{s}$.

To apply correctly Clause 5 of Definition 3.15 of Part I in the new setting, recall our rider that $\mathsf{len}(0,d]$ is not purely propositional. As before, let $\mathbf{H}(f)$ denote the set of all Hintikka sets for an RTFIL formula.

It is easy to show, following Part I, that Lemma 3.16 of Part I holds for RTFIL in the new setting.

Example 3.22. In Example 3.12, when $\mathscr{M}^f$ is constant throughout the interval $[t_1,t_2)$, let $\mathscr{M}^f[t_1,t_2)$ denote its value in that interval. It is clear that the sets $S_1 = \mathscr{M}^f[0,1)$, $S_2 = \mathscr{M}^f[1,4), S_3 = \mathscr{M}^f[4,7)$, $S_4 = \mathscr{M}^f[7,\infty)$ are Hintikka. Each of these Hintikka sets is satisfiable. However, consider the set $S_5 = (S_1\backslash\{\neg f_{11}\})\cup\{f_{11}\}$. This is Hintikka by our definition above, but is not satisfiable, because the conjunction of $\neg f_8$ and f_{11} cannot be satisfied in any model. Such "temporal conflicts" are detected by the consecution and acceptance conditions of $\mathscr{A}_{\mathrm{e}}(f)$ and $\mathscr{A}_{\mathrm{t}}(f)$, as will become clear in the sequel.

[12] Intuitively, this rule is a consequence of the assumption that tim eventually exceeds any finite bound.

$$\begin{tikzcd}
{} \arrow[r] & S1 \arrow[loop above, "S1"] \arrow[r, "S1"] & S2 \arrow[loop above, "S2"] \arrow[r, "S2"] & S3 \arrow[loop above, "S3"] \arrow[r, "S3"] & S4 \arrow[loop above, "S4"]
\end{tikzcd}$$

Fig. 5. Example of an accepting run of $\mathscr{A}_{\mathrm{u}}([\rightarrow p \mid \rightarrow p, \rightarrow p)\mathsf{len}(3.0,\infty))$.

3.2.2. *Untimed construction*

Having obtained the candidate states for $\mathscr{A}_{\mathrm{u}}(f)$ as Hintikka sets above, we must now connect them together appropriately. Compared to FIL (see Part I), the only new feature is the presence of formulae of the form $\mathscr{I}\mathsf{len}(0,d]$ and $\mathscr{I}\neg\mathsf{len}(0,d]$. Reductions on such formulae in a given state are essentially as before. However, consecution of two different states imposes further conditions on the timing assertions that these two states may contain, in addition to the reducibility of non-current interval formulae from one state to the next.

Definition 3.23 (*Untimed construction*). $\mathscr{A}_{\mathrm{u}}(f)$ is the BA with

- Input alphabet $2^{\mathbf{scl}(f)}$;
- State set $\mathbf{H}(f)$;
- Non-deterministic transition function ρ_{u} defined on $\mathbf{H}(f)\times 2^{\mathbf{scl}(f)}$ such that ρ_{u} allows $\mathbf{s} \xrightarrow{\mathbf{i}} \mathbf{t}$ iff
  1. $\mathbf{i} = \mathbf{s}$,
  2. if $\mathscr{I}[\theta_1|\theta_2)f_1 \in \mathbf{s}$ is s-irreducible and θ_1 is not $-$, then $\mathscr{I}[\theta_1|\theta_2)f_1 \in \mathbf{t}$,
  3. if $\mathscr{I}\neg[\theta_1|\theta_2)f_1 \in \mathbf{s}$ is s-irreducible and θ_1 is not $-$, then $\mathscr{I}\neg[\theta_1|\theta_2)f_1 \in \mathbf{t}$ and $\mathscr{I}\mathsf{false} \notin \mathbf{t}$,
  4. if $\mathscr{I}\mathsf{len}(0,d] \in \mathbf{s}$ is s-irreducible, then if $\mathscr{I}\neg\mathsf{len}(0,d] \in \mathbf{t}$ then $\mathscr{I}$ has a collapsing reduction in $\mathbf{t}$,
  5. if $\mathscr{I}\neg\mathsf{len}(0,d] \in \mathbf{s}$ is s-irreducible, then $\mathscr{I}\mathsf{false} \notin \mathbf{t}$;
- Accepting state set $\mathbf{H}(f)$;
- Initial state set $\{\mathbf{s} \in \mathbf{H}(f) \mid f \in \mathbf{s}\}$.

The first transition rule ensures that the automaton consumes only Hintikka sets. The remaining transition rules reflect the conditions stated in Lemma 3.11. Observe that ρ_{u} is reflexive, allowing $\mathscr{A}_{\mathrm{u}}$ to (non-deterministically) stay in state $\mathbf{s}$ when input with $\mathbf{i} = \mathbf{s}$.

Example 3.24. Consider the Hintikka sets $S_1,\ldots,S_4$ of the last example, and $\mathscr{M}^f$ of Example 3.12. If we untime $\mathscr{M}^f$ and feed it to $\mathscr{A}_{\mathrm{u}}(f)$ as an untimed ω-string, then the resulting run is shown in Fig. 5. The vertices represent states of the automaton and the edge labels represent letters of the input string.

Note that the automaton $\mathscr{A}_{\mathrm{u}}(f)$ has many other states and transitions, but for brevity only those in the locus of this run are shown in the figure. The reader can verify that the transition conditions given in the definition of $\mathscr{A}_{\mathrm{u}}$ are satisfied for each transition shown.

3.2.3. Timing augmentation

The timing augmentation systematically examines each state of the automaton built above, starting from an initial state, adding activity indicators to its states, associating a context tag with each active clock, augmenting its transitions with appropriate clock conditions, and splitting states where necessary. State splitting occurs when different paths from an initial state to some state of $\mathscr{A}_u(f)$ require different sets of timers to be active, or associate different context tags with the active timers. The resulting automation is the required local TBA, denoted $\mathscr{A}_t(f)$.

The augmentation is described here in two steps. First, we replicate the states of $\mathscr{A}_u(f)$, pairing the replicas with subsets of $\mathbf{clocks}(f)$, and further associating with each resulting replica a map from its active clocks to context tags – we, thus, obtain the states of $\mathscr{A}_t(f)$. Thus, each state of $\mathscr{A}_t(f)$ is a triple $\langle \mathbf{s}, \mathbf{a}_\mathrm{s}, \mathrm{tag}_\mathrm{s} \rangle$, where

- $\mathbf{s}$ is a subset of $\mathbf{scl}(f)$, representing the state of $\mathscr{A}_u(f)$ "corresponding" to this state of $\mathscr{A}_t(f)$
- $\mathbf{a}_\mathrm{s}$ is a subset of $\mathbf{clocks}(f)$, representing the set of clocks active in this replica of state $\mathbf{s}$ of $\mathscr{A}_u(f)$
- tag_s is a function associating with each clock $c_i^{\gamma,\mathscr{I},d} \in \mathbf{a}_\mathrm{s}$ an element in the set $\{\mathscr{I}' \mid \mathscr{I}' \sqsubseteq_\mathrm{s}^* \mathscr{I}\}$, representing the remaining contexts being timed by the active clocks.

Next, we define the transition function of $\mathscr{A}_t(f)$ to ensure that, of all the possible transitions resulting from this replication process, only the "legal" transitions are allowed by $\mathscr{A}_t(f)$. While this style of exposition clarifies the underlying mechanics, it is generally more expedient to perform a breadth-first traversal of $\mathscr{A}_u(f)$, adding clock-activity sets to its states, associating context tags with active clocks, and splitting states to create replicas only as required. Although the worst-case behaviour of this "on-the-fly" procedure may be as bad as the naïve method used in our description, in general, the latter procedure never creates many unreachable replicas.

Observe also that the concept of clock-activity sets that we use here does not appear either in the original [1] or our own definition of TBAs given earlier. It is easy, however, to modify our definition as well as the emptiness algorithm to handle this in a straightforward manner; see, for instance, [14] where a similar concept is used. In particular, the intuitive complexity of the emptiness algorithm is much reduced because activity indicators allow us to ignore the values of inactive clocks and thus cut down substantially on the size of the "configuration space" of the timed automaton that we need to explore.

Notation. Let κ represent the set of context tags $\{\mathscr{I} \mid \mathscr{I}\,\mathsf{len}(0,d] \in \mathbf{scl}(f)\}$. Using $\perp$ to represent an undefined value, we use $\kappa_\perp$ for $\kappa \uplus \{\perp\}$. For tag $\in \kappa_\perp^{\mathbf{clocks}(f)}$, we let $\mathrm{dom}\,\mathrm{tag} = \{c \in \mathbf{clocks}(f) \mid \mathrm{tag}(c) \neq \perp\}$ represent the domain of the partial function tag from $\mathbf{clocks}(f)$ to κ. We represent by $\perp_\kappa$ the null map $\perp_\kappa \in \kappa_\perp^{\mathbf{clocks}(f)}$ satisfying $\mathrm{dom}\perp_\kappa = \emptyset$.

We now define the automaton $\mathscr{A}_t(f)$.

Definition 3.25 (*Timing augmentation*). Let $\mathscr{A}_u(f)$ be an untimed automaton such as obtained above. Then its *timing augmentation*, denoted $\mathscr{A}_t(f)$, is the TBA with

- state set $\mathbf{H}(f) \times 2^{\mathbf{clocks}(f)} \times \kappa_{\perp}^{\mathbf{clocks}(f)}$;
- input Alphabet $2^{\mathbf{scl}(f)}$;
- clock set $\mathbf{clocks}(f)$;
- Non-deterministic transition function

$$\rho_t : X \times (2^{\mathbf{scl}(f)} \times 2^{\mathbf{clocks}(f)} \times 2^{\mathbf{clkconds}(f)}) \to 2^X$$

where X represents the state set defined above, such that ρ_t allows the transition $\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle \xrightarrow{i, C, \varphi} \langle \mathbf{t}, \mathbf{a}_t, \mathrm{tag}_t \rangle$ iff

1. $\mathbf{s} \xrightarrow{i} \mathbf{t}$ is allowed by ρ_u,
2. for any pair of β-clocks $c_j^{\beta, \mathscr{I}, d} \in \mathbf{a}_s$ and $c_k^{\beta, \mathscr{I}, d} \in \mathbf{a}_s$, with $j \neq k$, there is no $\mathscr{I}'$ such that $\mathscr{I}' \sqsubset_t^* \mathrm{tag}_s(c_k^{\beta, \mathscr{I}, d})$ and $\mathscr{I}' \sqsubset_t^* \mathrm{tag}_s(c_j^{\beta, \mathscr{I}, d})$,
3. $\mathbf{a}_t = (\mathbf{a}_s \backslash \mathrm{deact}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t})) \cup \mathrm{act}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t})$, where the set of deactivated clocks is given by

$$\begin{aligned}
&\mathrm{deact}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t}) \\
&= \{c_j^{\gamma, \mathscr{I}, d} \in \mathbf{a}_s \mid \gamma \in \{\alpha, \beta\}, \exists \mathscr{I}'. \mathscr{I}' \sqsubset_t^{\downarrow} \mathrm{tag}_s(c_j^{\gamma, \mathscr{I}, d})\} \\
&\quad \cup \{c_j^{\alpha, \mathscr{I}, d} \in \mathbf{a}_s \mid \exists k. k \neq j \wedge c_k^{\alpha, \mathscr{I}, d} \in \mathbf{a}_s \wedge \mathrm{tag}_s(c_k^{\alpha, \mathscr{I}, d}) \sqsubset_t^* \mathrm{tag}_s(c_j^{\alpha, \mathscr{I}, d})\}
\end{aligned}$$

and the set of newly activated clocks is given by

$$\begin{aligned}
&\mathrm{act}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t}) \\
&= \left\{ c_j^{\alpha, \mathscr{I}, d} \;\middle|\; \begin{array}{l} \mathscr{I}\,\mathsf{len}(0, d] \in \mathbf{t},\ \mathscr{I} \text{ irreducible in } \mathbf{t}, \\ \text{if } \mathscr{I}\,\mathsf{len}(0, d] \in \mathbf{s} \text{ then } \mathscr{I} \text{ reducible in } \mathbf{s}, \text{ and} \\ j = \min\{i \mid c_i^{\alpha, \mathscr{I}, d} \in \mathbf{clocks}(f) \backslash (\mathbf{a}_s \backslash \mathrm{deact}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t}))\} \end{array} \right\} \\
&\quad \cup \left\{ c_j^{\beta, \mathscr{I}, d} \;\middle|\; \begin{array}{l} \mathscr{I} \neg \mathsf{len}(0, d] \in \mathbf{s},\ \mathscr{I}\mathsf{len}(0, d] \in \mathbf{t}, \\ \mathscr{I} \text{ reducible in both } \mathbf{s} \text{ and } \mathbf{t}, \text{ and} \\ j = \min\{i \mid c_i^{\beta, \mathscr{I}, d} \in \mathbf{clocks}(f) \backslash (\mathbf{a}_s \backslash \mathrm{deact}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t}))\} \end{array} \right\}
\end{aligned}$$

4. $C = \mathrm{act}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t})$ where act is as defined above;
5. $\varphi = \{c \leqslant d \mid c = c_j^{\alpha, \mathscr{I}, d} \in \mathbf{a}_s\} \cup \{c = d \mid c = c_j^{\beta, \mathscr{I}, d} \in \mathbf{a}_s \cap \mathrm{deact}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t})\}$ where deact is as defined above;
6. tag_t satisfying $\mathrm{dom}\, \mathrm{tag}_t = \mathbf{a}_t$ is defined for $c_j^{\gamma, \mathscr{I}, d} \in \mathbf{a}_t$ by

$$\mathrm{tag}_t(c_j^{\gamma, \mathscr{I}, d}) = \begin{cases} \mathscr{I} & \text{if } c_j^{\gamma, \mathscr{I}, d} \in \mathrm{act}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t}) \\ \mathscr{I}' & \text{if } c_j^{\gamma, \mathscr{I}, d} \notin \mathrm{act}(\langle \mathbf{s}, \mathbf{a}_s, \mathrm{tag}_s \rangle, \mathbf{t}),\ \mathscr{I}' \sqsubset_t^* \mathrm{tag}_s(c_j^{\gamma, \mathscr{I}, d}), \\ & \quad \text{and } \mathscr{I}' \text{ is } \mathbf{t}\text{-irreducible} \end{cases}$$

- Initial state set consisting of the set of triples of the form $\langle \mathbf{s}, \mathbf{a}_\mathrm{s}, \mathrm{tag}_\mathrm{s}\rangle$ satisfying
  - $f \in \mathbf{s}$,
  - $\mathbf{a}_\mathrm{s} = \{c_1^{\alpha,\mathscr{I},d} \mid \mathscr{I}\mathsf{len}(0,d] \in \mathbf{s},\ \mathscr{I} \text{ is s-irreducible}\}$,
  - tag_s with $\mathrm{dom}\,\mathrm{tag}_\mathrm{s} = \mathbf{a}_\mathrm{s}$ satisfying $\mathrm{tag}_\mathrm{s}(c_1^{\alpha,\mathscr{I},d}) = \mathscr{I}$,
- Accepting state set $\mathbf{H}(f) \times 2^{\mathbf{clocks}(f)} \times \kappa_\perp^{\mathbf{clocks}(f)}$.

The intuition behind the augmentation procedure is as follows.

Rule 1 ensures that any model of $\mathscr{A}_\mathrm{t}(f)$, when untimed, is accepted by $\mathscr{A}_\mathrm{u}(f)$.

Rule 2 is a consistency condition, which we call the *β-clock consistency condition*, stating that if two lower-bound timers were started in the past to time two presumably different instances of a context $\mathscr{I}$, then they cannot both end at the same point.

Rule 3 shows how the set of clocks active in the next state are computed. This set of clocks comprises all those clocks active before the transition that were not deactivated by the transition, together with all the newly activated clocks. The set of clocks deactivated by the transition consists of all those active upper-bound timers for which either the context tag, representing the remaining context, collapsed, or the verification condition was properly subsumed by the verification condition for another upper-bound timer. As for clock activation, upper-bound timers are started as soon as there is a "new" upper-bound condition to verify, representing the start of a new context with an upper-bound constraint. Lower-bound timers are started at the precise point of transition when the duration of the remaining context $\mathscr{I}$ goes from being more than d (prior to the transition) to no more than d (following the transition).

Rule 4 ensures that all newly activated clocks are reset with the transition.

Rule 5 ensures that timing conditions are correctly verified. For the case of an α-clock, the value is compared against the prescribed upper bound at every transition preceding the location of the right endpoint of the context that it is timing. For the case of a β-clock, the value is compared against the lower bound as soon as the right endpoint of its context is located.

Rule 6 ensures that the context tags associated with old clocks are appropriately updated following the transition, and that the newly activated clocks have their context tags set to the appropriate value (the context they are timing).

The only problem with the above definition is that act might be undefined, because either of the sets

$$S_{\alpha,\mathscr{I},d} = \{i \mid c_i^{\alpha,\mathscr{I},d} \in \mathbf{clocks}(f)\backslash(\mathbf{a}_\mathrm{s}\backslash \mathrm{deact}(\langle \mathbf{s}, \mathbf{a}_\mathrm{s}, \mathrm{tag}_\mathrm{s}\rangle, \mathbf{t}))\}$$

$$S_{\beta,\mathscr{I},d} = \{i \mid c_i^{\beta,\mathscr{I},d} \in \mathbf{clocks}(f)\backslash(\mathbf{a}_\mathrm{s}\backslash \mathrm{deact}(\langle \mathbf{s}, \mathbf{a}_\mathrm{s}, \mathrm{tag}_\mathrm{s}\rangle, \mathbf{t}))\}$$

might be empty. To show that act is well-defined, we need only show the following:

- $S_{\alpha,\mathscr{I},d}$ is non-empty, whenever it is the case that $\mathscr{I}\,\mathsf{len}(0,d] \in \mathbf{t}$, $\mathscr{I}$ is irreducible in $\mathbf{t}$ and if $\mathscr{I}\,\mathsf{len}(0,d] \in \mathbf{s}$ then $\mathscr{I}$ is reducible in $\mathbf{s}$;
- $S_{\beta,\mathscr{I},d}$ is non-empty, whenever it is the case that $\mathscr{I}\neg\mathsf{len}(0,d] \in \mathbf{s}$, $\mathscr{I}\,\mathsf{len}(0,d] \in \mathbf{t}$ and $\mathscr{I}$ is irreducible in both $\mathbf{s}$ and $\mathbf{t}$.

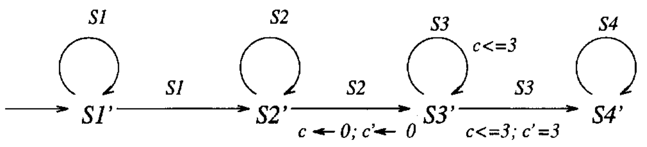


Fig. 6. Example of an accepting run of $\mathscr{A}_t$ for $[\rightarrow p \mid \rightarrow p, \rightarrow p)\mathsf{len}(3.0, \infty)$.

But this is quite straightforward. Assume, for the first case, that S_α is empty. This implies that $\mathbf{a}_s$ includes the set of clocks $C^{\alpha,\mathscr{I},d} = \{c_j^{\alpha,\mathscr{I},d}\}_{j\in[n]}$, where $n = \text{card}\,\{\mathscr{I}' \mid \mathscr{I}' \sqsubset^* \mathscr{I}\}$. Clearly, for any two distinct clocks $c_1, c_2 \in C^{\alpha,\mathscr{I},d}$, it is the case that $\text{tag}_s(c_1) \neq \text{tag}_s(c_2)$, for otherwise one of the two could not have been active in $\mathbf{a}_s$.[13] Consider now the clock $c \in C^{\alpha,\mathscr{I},d}$ such that $\text{tag}_s(c) = \mathscr{I}$. We shall assume now that $\mathscr{I}\,\mathsf{len}(0,d] \in \mathbf{t}$, $\mathscr{I}$ is irreducible in $\mathbf{t}$, and if $\mathscr{I}\,\mathsf{len}(0,d] \in \mathbf{s}$ then $\mathscr{I}$ is reducible in $\mathbf{s}$, and exhibit a contradiction. Since $\text{tag}_s(c) = \mathscr{I}$, clearly $\mathscr{I}$ is irreducible in $\mathbf{s}$, so $\mathscr{I}\,\mathsf{len}(0,d] \notin \mathbf{s}$. This implies, by the definition of Hintikka sets, that $\mathscr{I}\neg\mathsf{len}(0,d] \in \mathbf{s}$. But the fact that c is active in $\mathbf{s}$, means that it was activated in some prior state which contained $\mathscr{I}\,\mathsf{len}(0,d]$. This implies, by the transition rules of $\mathscr{A}_u$, that $\mathscr{I}\neg\mathsf{len}(0,d]$ could not have become true across a subsequent transition unless $\mathscr{I}$ had a reduction. Clearly, $\mathscr{I}\neg\mathsf{len}(0,d] \notin \mathbf{s}$, giving us a contradiction. The case of S_β is similar.

Example 3.26. Recall Example 3.24, where we illustrated an accepting run of $\mathscr{A}_u(f)$. Fig. 6 shows the corresponding accepting run of $\mathscr{A}_t(f)$ on our now familiar $\mathscr{M}^f$. The states of $\mathscr{A}_t(f)$ shown in the figure are $S_1' = \langle S_1, \emptyset, \bot_\kappa\rangle$, $S_2' = \langle S_2, \emptyset, \bot_\kappa\rangle$, $S_3' = \langle S_3, \{c, c'\}, \text{tag}_{S_3}\rangle$, $S_4' = \langle S_4, \emptyset, \bot_\kappa\rangle$, where c and c' are the clocks of Example 3.19, $\text{dom}\,\text{tag}_{S_3} = \{c, c'\}$ and $\text{tag}_{S_3}(c) = \text{tag}_{S_3}(c') = [-|\rightarrow q)$. The edge labels also indicate associated clock conditions and/or clock actions.

Although the role of clock c is superfluous in the example run shown above, it may be required in general – for instance, when $\langle \mathscr{M}, 0\rangle \models p$, requiring the verification of $[-| \rightarrow q)\,\mathsf{len}(0.0, 3.0]$ starting from an initial state.

3.2.4. *Eventuality automaton*

The construction of the eventually automaton for an RTFIL formula is essentially the same as that for an FIL formula, relativized to the new definition of **scl**. We refer the reader to Part I (Section 3.3.2) for details and for intuition. Note that the eventually automaton is a BA (see our comments following Definition 3.2, regarding TBAs with no clocks).

As we noted earlier, $\mathscr{A}_e(f)$ handles only unbounded liveness conditions. Time-bounded liveness conditions are handled by the combination of $\mathscr{A}_e(f)$ and $\mathscr{A}_t(f)$; $\mathscr{A}_e(f)$ ensures that the required state is eventually reached (without regard to real time) and $\mathscr{A}_t(f)$ ensures that the related timing constraints are met when the state is

[13] This can be established by a routine inductive argument, starting from an initial state of $\mathscr{A}_t$, an argument that we skip for the sake of brevity.

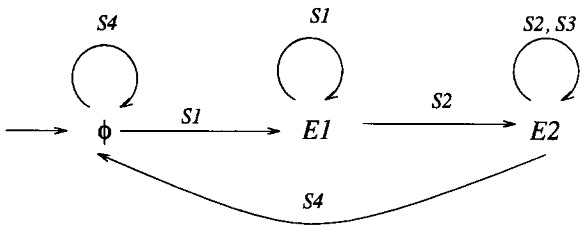


Fig. 7. Example of an accepting run of $\mathscr{A}_\mathrm{e}$ for $[\rightarrow p \mid \rightarrow p, \rightarrow p)\mathrm{len}(3.0, \infty)$.

reached. A similar "communication" (via the "input" string) also occurs in the purely untimed case of FIL while dealing with eventualities that are bounded within intervals (Section 3.3.2, Part I).

Example 3.27. In our running example, we have $\mathbf{E}(f) = \{\neg f_8, \neg f_{11}, \neg f_{12}\}$. As in the previous two examples, we illustrate the accepting run of $\mathscr{A}_\mathrm{e}(f)$ on (untimed) $\mathscr{M}^f$ in Fig. 7. The states shown are $\emptyset$ (the only accepting state of $\mathscr{A}_\mathrm{e}(f)$), $E_1 = \{\neg f_8, \neg f_{11}, \neg f_{12}\}$, and $E_2 = \{\neg f_{11}\}$.

3.2.5. *Combining the automata*

The decision procedure is now straightforward. We construct $\mathscr{A}_\mathrm{u}(f)$ and augment it using the timing construction to obtain $\mathscr{A}_\mathrm{t}(f)$. We then take the product of $\mathscr{A}_\mathrm{t}(f)$ with the eventuality automaton $\mathscr{A}_\mathrm{e}(f)$ (the $\mathscr{A}_\mathrm{e}$ component of the product ignores timing information in the input). Finally, we check the emptiness of the resulting automaton $\mathscr{A}_\mathrm{m}(f)$, using the emptiness algorithm of [3]. We thus have our main theorem.

Theorem 3.28 (Decision procedure). *Given an RTFIL formula f, it is decidable whether or not f is satisfiable.*

The main lemma required in the proof of Theorem 3.28 is

Lemma 3.29. *The language of $\mathscr{A}_\mathrm{m}(f)$ is empty iff f is not satisfiable.*

Proof. The proof follows from the following lemmas, which are proved in the next section.

Lemma 3.30 (Completeness). *Let f be an RTFIL formula and $\mathscr{M}$ a satisfying model for it. Then $\mathscr{M}^f$ is accepted by $\mathscr{A}_\mathrm{m}(f)$.*

Lemma 3.31 (Soundness). *Let f be an RTFIL formula, and $\langle \sigma_i, t_i \rangle_{i \in \omega}$ a timed string accepted by $\mathscr{A}_\mathrm{m}(f)$. Then there is a model $\mathscr{M}$, such that $\mathscr{M} \models f$.*

The construction given above for our decision procedure shows that RTFIL is invariant under finite infinitesimal timed stuttering. This property was stated and proved directly in Theorem 2.5, but is further clarified by noting that the local TBA $\mathscr{A}_t(f)$ has a reflexive transition relation with the self-loops containing only edge conditions of the form $c \leqslant d$ and no clock resetting actions.

3.3. *Proof of correctness*

We devote the next two sections to proving the Soundness and Completeness Lemmas. Since many of the details for the "non-real-time component" of the proof are similar to the case of FIL, we shall here emphasize the handling of the real-time constructs of the logic.

3.3.1. *Completeness*

Throughout this subsection we assume that $\mathscr{M}^f$ is the extension of a satisfying model for f, as stated in the Completeness Lemma above. Moreover, we use the timed ω-string representation for $\mathscr{M}^f$. It is easy to see that the admissibility of $\mathscr{M}^f$ implies that there is a timed ω-string representation for it. However, for convenience, we use a "canonical" representation, with $\mathscr{M}^f$ represented by the timed ω-string $\langle \sigma_i, t_i \rangle_{i \in \omega}$, defined inductively as follows for all i (let $t_{-1} = 0$):

$$\sigma_i = \mathscr{M}^f(t_{i-1})$$

$$t_i = \inf(\{t > t_{i-1} \mid \mathscr{M}^f(t) \neq \mathscr{M}^f(t_{i-1}\} \cup \{\lfloor t_{i-1} \rfloor + 1\})$$

where $\lfloor t \rfloor$ represents the largest integer in t.

The proof of the Completeness Lemma follows from the Lemmas 3.33 and 3.35.

Lemma 3.32. *$\mathscr{A}_u(f)$ accepts* **untime**$(\mathscr{M}^f)$.

Proof. The proof is similar to that for Lemma 3.21 of Part I, with minor changes to account for the denseness of the time domain. In the induction step, we invoke Lemma 3.11 in the place of Lemma 3.12 of Part I. □

As in Part I, it is easy to see that the accepting run is, in fact, unique. This is useful for the proof of the next lemma.

Lemma 3.33. *$\mathscr{A}_t(f)$ accepts $\mathscr{M}^f$.*

Proof. From the previous lemma, $\mathscr{A}_u$ must accept the untimed string $\langle \sigma_i \rangle_i$. We know from our earlier observations (see the proof of Lemma 3.21 in Part I, and Lemma 3.32) that the run of $\mathscr{A}_u$ on $\langle \sigma_i \rangle_i$ is unique and is, in fact, $\langle \sigma_i \rangle_{i \in \omega}$ itself.

We can now build an accepting run of $\mathscr{A}_t$, as follows. Recall that a state of $\mathscr{A}_t$ consists of three components: a Hintikka component, a second component consisting of the active clocks, and a third component associating a context with each active clock.

The "Hintikka component" of the run $\mathscr{A}_t$ on $\langle \sigma_i, t_i \rangle_i$ is simply the sequence $\langle \sigma_i \rangle_i$. From the construction in Definition 3.25, given a transition from the state $\langle \sigma_i, \mathbf{a}_i, \text{tag}_i \rangle$ to the state $\langle \sigma_{i+1}, \mathbf{a}_{i+1}, \text{tag}_{i+1} \rangle$, $\mathbf{a}_{i+1}$ is a (deterministic) function of σ_i and σ_{i+1}, and, in turn, tag_{i+1} is a (deterministic) function of σ_i, σ_{i+1} and $\mathbf{a}_{i+1}$. Moreover, for the initial state, given σ_0, $\mathbf{a}_0$ is determined uniquely by σ_0, and tag_0 is then determined uniquely from $\mathbf{a}_0$.

We only need to show that, for the above run, the following hold:

- $\langle \sigma_0, \mathbf{a}_0, \text{tag}_0 \rangle$ is an initial state of $\mathscr{A}_t$,
- for every $i \in \omega, \mathscr{A}_t$ allows a transition from the state $\langle \sigma_i, \mathbf{a}_i, \text{tag}_i \rangle$ to the state $\langle \sigma_{i+1}, \mathbf{a}_{i+1}, \text{tag}_{i+1} \rangle$.

That σ_0 is an initial state of $\mathscr{A}_t$ is immediate from the proof of Lemma 3.32, and our construction of the run. So we need only show that the β-clock consistency criterion and the clock conditions associated with the transition relation to $\mathscr{A}_t$ are always respected by the above run. We consider the two cases below.

Case 1 (*β-clock consistency*). Let t_v be the least time at which the above run violates the β-clock consistency condition. We show that this leads to a contradiction. Observe first that if two clocks $c_i^{\beta,\mathscr{I},d}$ and $c_j^{\beta,\mathscr{I},d}, i \neq j$, are simultaneously active then, by Definition 3.25, they must must have been activated at different transitions. Let the times of activation be, respectively, $t_i, t_j < t_v$, with $t_i \neq t_j$. Let $\mathscr{I} = [-|\theta_1)[-|\theta_2) \cdots [-|\theta_n)$. Then the irreducibility of $\mathscr{I}$ at t_i implies that for the sequence of models $\mathscr{M}_i^k$ defined by $\mathscr{M}_i^k = \mathscr{C}([-|\theta_k), \langle \mathscr{M}_i^{k-1}, t_i \rangle)$ for all $k \in [n] = \{1, \ldots, n\}$ (let $\mathscr{M}_i^0 = \mathscr{M}$), we have for all $k \in [n]$,

- $\mathscr{M}_i^k \neq \perp_{\mathscr{M}}$,
- $\inf \operatorname{dom} \mathscr{I}_k = t_i$,
- $\sup \operatorname{dom} \mathscr{I}_k \leqslant \sup \operatorname{dom} \mathscr{I}_{k-1}$.

In fact, defining $\text{first}(\theta)$ to be the target of the first search in the search pattern θ (for instance, $\text{first}\,(\rightarrow a, \rightarrow b) = a$) it is easy to see the following:

Fact 3.34. *If $t' \geqslant t_i$ is the least time at which the first reduction occurs on $\mathscr{I}$ then, for some $k \in [n]$, it is the case that $\lambda(\text{first}(\theta_k), \langle \mathscr{M}_i^{k-1}, t_i \rangle) = t'$ and, for all $l \in [n]$, it is the case that $\lambda(\text{first}(\theta_l), \langle \mathscr{M}_i^{l-1}, t_i \rangle) \geqslant t'$.*

By repeatedly using this fact on each resulting pattern sufficiently many times, we can reach the least point where a collapsing reduction occurs. Since $\sup \operatorname{dom} \mathscr{M}_i^n < \sup \operatorname{dom} \mathscr{M}_i^k$ for all $k \in [n-1]$, we can conclude that the least time at which a series of interval reductions starting with $\mathscr{I}$ at t_i leads to a collapse is the point $\sup \operatorname{dom} \mathscr{M}_i^n = T_i$ (say).

Consider now the point t_v where, for some $\mathscr{I}'$, we have $\mathscr{I}' \sqsubset_{\sigma_v}^{*} \text{tag}_{v-1}(c_i^{\beta,\mathscr{I},d})$, $\mathscr{I}'$ being σ_v-irreducible. Using arguments as above, it follows that $\sup \operatorname{dom} \mathscr{M}_v^n = T_i$ where, as above, we let $\mathscr{I}' = [-|\theta_1') \cdots [-|\theta_n')$ and denote, for all $k \in [n], \mathscr{M}_v^k = \mathscr{C}([-|\theta_k'), \langle \mathscr{M}_v^{k-1}, t_v \rangle)$ with $\mathscr{M}_v^{k-1} = \mathscr{M}$.

On the other hand, since $\langle \mathscr{M}, t \rangle \models \mathscr{I} \neg \mathsf{len}(0, d]$ for all $t \in [t_{i-1}, t_i)$ it follows that $t + d < \sup \operatorname{dom} \mathscr{C}(\mathscr{I}, \langle \mathscr{M}, t \rangle) = \sup \operatorname{dom} \mathscr{C}(\mathscr{I}, \langle \mathscr{M}, t_i \rangle)$ for all $t \in [t_{i-1}, t_i)$. Since

$\langle \mathcal{M}, t_i \rangle \models \mathcal{I}\mathsf{len}(0,d]$, we have $t_i + d \geqslant \sup \mathrm{dom} \mathscr{C}(\mathcal{I}, \langle \mathcal{M}, t_i \rangle) = T_i$, and it follows that $t_i + d = T_i$.

Using an identical series of arguments with the index i uniformly replaced by j, we can conclude, similarly, that $\sup \mathrm{dom}\, \mathcal{M}_v^n = T_j$. Thus $T_j = T_i$. Moreover, as above, we also obtain $t_j + d = T_j$. So $t_i = t_j$, giving us the required contradiction.

Case 2 (*Clock conditions*). Assume that a clock condition of the form $c_j^{\beta,\mathcal{I},d} = d$ is present on the vth transition. Let t_j be the time of the most recent transition that activated $c_j^{\beta,\mathcal{I},d}$, so the value of the clock at the vth transition is $t_v - t_j$. Arguing as in the last case, we have $t_v = \sup \mathrm{dom}\, \mathcal{M}_j^n$, as well as $t_j + d = \sup \mathrm{dom}\, \mathcal{M}_j^n$, giving us $t_v - t_j = d$, so that the clock condition is met.

A similar argument can be used to show that α-clock conditions are also met. □

Lemma 3.35. *$\mathcal{A}_{\mathrm{e}}(f)$ accepts* **untime**$(\mathcal{M}^f)$.

Proof. The proof is along the lines of that of Lemma 3.22 in Part I. □

3.3.2. Soundness

The proof parallels the corresponding proof for FIL. We shall here concentrate on the real-time constructs. We show that, given a (timed) string in the language of $\mathcal{A}_{\mathrm{m}}$, one can construct a satisfying model $\mathcal{M}$ for f. Let $\langle \sigma_i, t_i \rangle_{i \in \omega}$ be a string in the language of $\mathcal{A}_{\mathrm{m}}$. Let $\langle \sigma_i, \mathbf{a}_i, \mathrm{tag}_i \rangle_{i \in \omega}$ be the accepting run of $\mathcal{A}_{\mathrm{t}}$ on the timed string $\langle \sigma_i, t_i \rangle_i$, and let $\langle \sigma_i^{\mathrm{e}} \rangle_i$ be the corresponding accepting run of $\mathcal{A}_{\mathrm{e}}$. Let $\mathcal{M}'$ be defined by

$$\mathcal{M}'(t) = \{ f_1 \in \mathbf{scl}(f) \mid f_1 \in \sigma_i,\ t \in [t_{i-1}, t_i) \}$$

where we have assumed $t_{-1} = 0$. Moreover, let $\mathcal{M}$ be defined by

$$\mathcal{M}(t) = \{ p \in \mathcal{P} \mid p \in \mathcal{M}'(t) \}$$

To prove the lemma, we must show that $\langle \mathcal{M}, 0 \rangle \models f$. In fact, we show the following stronger result.

Lemma 3.36. *For any $t \in R$ and $f_1 \in \mathbf{scl}(f)$, $f_1 \in \mathcal{M}'(t)$ iff $\langle \mathcal{M}, t \rangle \models f_1$.*

Proof. The proof is substantially along the lines of that of Lemma 3.24 of Part I. As before, we induct, for an arbitrary t, on the inclusion order induced by **scl** on the formulae. Let $t \in [t_{i-1}, t_i)$ as defined above.

The base case for the primitive propositions remains the same. For the base case of timing formulae of the form $\mathsf{len}(0,d] \in \mathbf{scl}(f)$, we have by construction that $\mathsf{len}(0,d] \notin \sigma_i$, and from the semantics that $\langle \mathcal{M}, t \rangle \not\models \mathsf{len}(0,d]$.

For the inductive step, we consider below the sample case of a timing assertion $\mathcal{I}\mathsf{len}(0,d]$, call it f'. For the forward direction, assume that $f' \in \sigma_i$. We have two subcases depending on whether or not $\mathcal{I}\mathsf{len}(0,d]$ is reducible with respect to σ_i.

For the subcase in which it is reducible, we have for some formula $a \in \sigma_i$, $f'' \prec_a f'$, and by construction then $f'' \in \sigma_i$. By the induction hypothesis, we have $\langle \mathscr{M}, t\rangle \models f''$, as well as $\langle \mathscr{M}, t\rangle \models a$. By the RTFIL counterpart of Lemma 3.11 of Part I, we then have $\langle \mathscr{M}, t\rangle \models f'$.

For the subcase in which f' is irreducible, so is $\mathscr{I}$, and therefore there is a k, such that $c_k^{\alpha,\mathscr{I},d} \in \mathbf{a}_i$, and $\mathrm{tag}_i(c_k^{\alpha,\mathscr{I},d}) = \mathscr{I}$. By an argument similar to that in the proof of Lemma 3.33, it follows that $\sup \mathrm{dom}\,\mathscr{C}(\mathscr{I}, \langle \mathscr{M}, t_{i-1}\rangle) = t^{\downarrow}$, where $t^{\downarrow}$ is the least time greater than t_{i-1}, at which a series of reductions starting with $\mathscr{I}$ at t_{i-1} leads to a collapse. (To show that such a $t^{\downarrow}$ must exist, we make use of the irreducibility of $\mathscr{I}\mathsf{len}(0,d]$ with respect to σ_i, along with the acceptance criteria for $\mathscr{A}_e$, just as we did in the proof of the Soundness Lemma for FIL in Part I.) Let $t_j \leqslant t_{i-1}$ be the time of the most recent activation of $c_k^{\alpha,\mathscr{I},d}$. It follows from our construction that $t^{\downarrow} - t_j \leqslant d$, so $t^{\downarrow} - t_{i-1} \leqslant d$, and therefore $\langle \mathscr{M}, t_{i-1}\rangle \models \mathscr{I}\mathsf{len}(0,d]$. Now, since $\mathscr{M}^f$ is constant in $[t_{i-1}, t_i)$, it follows by the inductive assumption and the irreducibility of $\mathscr{I}\mathsf{len}(0,d]$ at t_{i-1} that $\langle \mathscr{M}, t \not\models a$ for all $a \in \mathbf{red}(\mathscr{I}\mathsf{len}(0,d])$. By Fact 3.34 and the semantics, we also have $\langle \mathscr{M}, t\rangle \models \mathscr{I}\mathsf{len}(0,d]$ for any $t \in [t_{i-1}, t_i)$.

For the backward direction, we prove its contrapositive. By construction, $\mathscr{I}\mathsf{len}(0,d] \notin \sigma_i$ implies $\neg\mathscr{I}\mathsf{len}(0,d] \in \sigma_i$. The case where $\neg\mathscr{I}\mathsf{len}(0,d]$ is reducible is straightforward. When it is not reducible, we have by construction that $\mathscr{I}\neg\mathsf{len}(0,d] \in \sigma_i$. We now use Lemma 3.11 and arguments regarding β-blocks, analogous to those used above for α-clocks, to establish the result. □

The soundness lemma follows since f is in $\mathscr{M}'(0)$.

3.4. Complexity of the decision problem

Let f be an RTFIL formula of size n and depth k, and let T be the size of the binary encoding of the timing constant appearing in f. By Lemma 3.20, $|\mathbf{scl}(f)| = \mathrm{O}(n^k)$. Clearly, $\mathscr{A}_u(f)$ and $\mathscr{A}_e(f)$ can have at most $2^{\mathrm{O}(n^k)}$ states each. The timing augmentation can introduce $\mathrm{O}(n^{2k})$ clocks, and the number of states of $\mathscr{A}_t(f)$ is $2^{\mathrm{O}(n^{2k}\cdot k \log n)}$. Thus, $\mathscr{A}_m(f)$ can have at most $2^{\mathrm{O}(n^{2k}\cdot k \log n)}$ states and $\mathrm{O}(n^{2k})$ clocks. The final emptiness check has a complexity of $\mathrm{O}(C! \cdot (S+E)\cdot 2^{T\log T})$, where C is the size of the clock set, S and E are the number of states and edges in the TBA, and T is the size of the binary encoding of the timing constants appearing on the edge conditions of the TBA [3]. The overall complexity of the decision procedure is thus $2^{\mathrm{O}(n^{2k}\cdot k\log n + T\log T)}$.

The main source of the blow-up is due to the large number of clocks. Note, however, that usually the number of clocks will be much less than that indicated by the large upper bound because timing conditions in specifications will generally involve relations between a few simple predicates rather than long sequences of events. As a result the overall complexity will be closer to $2^{\mathrm{O}(n^k + C\cdot k\log n + T\log T + C\log C)}$, where C is the number of clocks introduced in the timing augmentation. Comparing this with the $2^{\mathrm{O}(n^k)}$ upper bound for FIL, the price for real time is seen to be an additional factor exponential in the number of timers and the constants appearing in the specification. However, the decision procedure is still doubly exponential (deterministic time), essentially the same as for the timeless logic FIL [27].

In fact, we can show, using a method similar to that used for FIL, that the decision problem can be solved in *EXPSPACE*. Observe that a configuration of (the region automaton [3] corresponding to) $\mathscr{A}_{\mathrm{m}}$ can be encoded as a tuple consisting of

- the set of formulae in the state,
- the set of clocks active in the state,
- for each active clock, the associated context tag,
- for each active clock, its integer value, when it is less than the value it will be compared against (i.e. when it is less than d for the clock $c_j^{\alpha,\mathscr{I},d}$),
- the relative ordering of the fractional parts of each active clock.

The number $O(n^k)$ of formulae and the number $O(n^{2k})$ of clocks in a state is at most exponential in the size n of the input formula (since k is $O(n)$), so a configuration can be represented in space exponential in n. Our TM begins by guessing an initial configuration, and verifying that the conditions of Definition 3.25 are satisfied. At each subsequent stage it now guesses the next configuration and verifies that all the transition conditions are satisfied. This now includes verifying that the ordering of the fractional parts of the clocks that are active across the transition, and which do not change their integral value, is the same in both configurations, and that the fractional part of every newly activated clock, and every clock for which there is an increase in the integral value, is zero. The remaining arguments are analogous to the case of FIL; in particular, the values of the two counters can be represented in exponential space, since the number of regions of the region automaton are bounded above by a triple exponential.

We have, however, not been able to show a matching lower bound for the problem. The best lower bound that we have is the *PSPACE*-hardness of Part I. (Recall that there is also an exponential gap between the upper and lower bounds we gave, in Part I, for FIL for the general case.) The rather clever encoding of the computation of an *EXPSPACE*-bounded Turing machine in MITL [2] does not extend, as far as we can see, to our logic, since RTFIL lacks the ability (see Section 5) to relate states separated by a given duration.

The satisfiability procedure can be adapted, in a straightforward manner, to obtain a model-checking algorithm for RTFIL. This is done by checking the emptiness of the product of the automaton for the negation of the formula with that representing the model. The resulting algorithm runs in time doubly exponential in the input formula (the same as the satisfiability procedure) and linear in the size of the input model (given in the form of a fair timed transition system).

Analogous to the result for FIL, it is easy to show that if we bound the largest constant appearing in a formula and the largest depth of nesting of interval modalities, then this bounded version of satisfiability for RTFIL is *PSPACE*-complete in the size of the formula. This result may be more indicative of the type of scaling behaviour one might expect for the logic.

3.5. Tableaux: Reducing average-case complexity

A tableau-based analogue of the algorithm presented above can result in a reduction in the average-case complexity. The construction of the tableau is based on the following observations: it is not necessary to build all the subsets of $\mathbf{scl}(f)$ while constructing $\mathscr{A}_{\mathrm{m}}(f)$, and thus immediately pay an almost worst-case penalty. Firstly, there may be many states that are unreachable from any initial state. It may be possible to avoid exploring such states. Secondly, there may be states that are *trace-equivalent*, in the sense that the language of the automaton starting at either state is identical. It may be possible to save the duplication in effort involved in exploring these states separately.

Before we make our third observation, we need to make a brief digression into the emptiness algorithm for TBAs. Recall our statements in Section 3.4 regarding the number of "regions" of the region automaton corresponding to a TBA. The complexity in checking the emptiness of the TBA stems, to a large extent, from the need to explore all the regions defined by the TBA. However, in many cases, the timed-language accepted by a TBA, starting in different regions, may be the same. When such regions are contiguous (and their union is convex) it may be possible to explore this set of regions simultaneously by considering their union. This optimization can be done using a method proposed by Dill et al. [2, 14].

Finally, the emptiness checking need not wait until the entire graph of the region automaton has been constructed. On-the-fly methods exist for maintaining the strongly connected components of the tableau, as it is "grown" from its initial node. The method terminates upon finding a reachable bottom strongly connected component that is timing-consistent and eventuality-fulfilling.

Using these simple heuristics, it is possible to obtain a tableau-based refinement of the automata-theoretic algorithm that, in many cases, terminates much faster than the automata-theoretic method. In [24] we present a version of such a method. This method also underlies an implementation of a proof-assistant for the logic [21,22,26]. Wolper [31] gives a good overview of tableau-based methods for non-quantitative temporal logics.

4. An undecidable extension

In [20] an extension of (qualitative) Interval Logic to real time was suggested that makes use of two main constructs. The first construct allows the language to specify bounds on the duration of intervals. This is precisely what we have in RTFIL. The second construct that was suggested is a real-time offset operator, $\rightarrow +d$, much like the search operator, which moves a point of reference by a given real time from the point where it began. These extensions were, of course, suggested in the context of an interval logic different from ours. Several examples were given showing the expressiveness and naturalness of such a construct. We show in the next section that the addition of such a construct to RTFIL, which has a dense notion of time, makes the logic undecidable.

The results of the next section underscore our earlier points regarding the difficulty of obtaining an expressive dense-time logic without sacrificing decidability.

4.1. Adding the "$\rightarrow +$" construct

We extend RTFIL to the logic RTFIL+ by adding an offset construct similar to the one mentioned in [20]. The syntax of RTFIL+ consists of that defined by the BNF grammar for RTFIL in Section 2.1 except that wfsps have an extra terminal $\rightarrow +d$ with $d \in Q$, i.e.

$$\theta \ ::== \ \rightarrow f \mid \ \rightarrow +d \mid \ \rightarrow f, \theta \mid \ \rightarrow +d, \theta$$

The semantics of RTFIL+ are defined by the semantics for RTFIL in Section 2.3 and the following additional rule for interpreting offset searches, which extends the definition of the search-locator function for the case $\mathcal{M} \neq \bot_{\mathcal{M}}$ and $t \neq \bot$:

$$\lambda(\rightarrow +d, \langle \mathcal{M}, t \rangle) = \begin{cases} \bot & \text{if } t + d > \sup \operatorname{dom} \mathcal{M} \\ t + d & \text{otherwise} \end{cases}$$

RTFIL+ thus allows natural expression of constructs that RTFIL cannot express. For instance, to require the occurrence of a q-state precisely 4.2 time units from every p-state, one can simply assert

$$\Box(p \Rightarrow [\rightarrow +4.2 | \rightarrow \rangle q)$$

Note that this construct also preserves right-continuity of models under extension. Unfortunately, the augmentation of RTFIL with this construct leads to undecidability.

Theorem 4.1 (Undecidability of RTFIL+). *The satisfiability problem for RTFIL+ is undecidable.*

Our undecidability proof, which appears in the appendix, is by reduction from the halting problem for two-counter Minsky machines. That is, we give an encoding f from a given input program x for a two-counter machine to an RTFIL formula $f(x)$, such that $f(x)$ is satisfiable in RTFIL iff the machine has a halting computation on x. This gives us undecidability since a two-counter machine is universal [15, p. 172]. The strategy and the idea of the encoding are similar to those used in the proof of undecidability of Metric Interval Temporal Logic with singular intervals, which appears in [4]. In fact, the proof is trivially modified to give a Π_1^1 lower-bound for the validity problem, showing that the proposed extension is not even axiomatizable.

5. Related work

Our approach of extending a qualitative temporal logic to real time is not new, having been introduced into temporal logic by Koymans [17], and falls roughly into the category of bounded-operator temporal logics. Indeed, the precise construct that we use is one of two suggested by Melliar-Smith [20], in the context of the Interval Logic of [30]. Our main contribution has been in formalizing the syntax and semantics of the logic, and in giving a decision procedure for it.

Proposals for real time interval logics also appear in [23,28]. However, neither of these papers provides a decision procedure for the proposed logic. In fact, the logic of Razouk and Gorlick [28] is so powerful that it is highly undecidable. The logics of Aaby and Narayana [23] and of Melliar-Smith [20] allow the expression of the forbidden "punctuality" construct of [4], so that they can be shown to be undecidable if interpreted over a dense time domain, in much the same way as we do for RTFIL+ in the appendix (see proof of Theorem 4.1).

The Duration Calculus [13] differs from RTFIL in that it treats intervals as primitive semantic objects. It is well-suited to describing and reasoning about cumulative behaviour, a feature especially useful for hybrid systems. The operator $\int$ in that logic, for instance, allows one to bound the duration of a fragment of a computation during which a predicate holds. This ability to integrate over non-convex intervals, combined with the "non-local" character of the logic, makes it very expressive. However, as is shown in [12], over dense time even the simplest real time fragment of the calculus is undecidable and, even without real time, the simplest fragment is non-elementary.

Many of the recent advances in dense real time specification and verification theory spring from the important paper [3] of Alur and Dill, where a very expressive concrete model of real time, in the form of *timed automata*, was first presented. The usefulness of their model derives from its expressiveness and the fact that the emptiness problem for these automata is solvable. Its expressiveness allows many quite powerful real time logics to be interpreted in that model, as is the case for (qualitative) temporal logics vis-à-vis ω-automata. The solvability of the emptiness problem for timed automata then yields decision procedures for the real time logics thus interpreted. One can, therefore, expect these automata to play the same central role in real time temporal logic decision procedures that Büchi automata (and their many variants) have played in qualitative temporal logics.[14]

Decidable dense real time logics are relatively rare because a dense real time logic must tread the fine line between expressiveness and undecidability. Indeed, RTFIL and the Metric Interval Temporal Logic (MITL) of Alur et al. [4] are two of the few real time temporal logics known today that admit a dense notion of time and yet are decidable. The logics RTFIL and MITL adopt different compromises and neither, we believe, is as expressive as the other. MITL appears to have no direct means of expressing RTFIL formulae that constrain the length of an interval defined between the endpoints of a sequence of (more than two) searches. Correspondingly, RTFIL cannot

[14] The only stumbling block may be that, unlike untimed automata, these automata are not closed under complementation. However, recently, Alur, et al. [5] have identified a large determinizable subclass of TBAs.

express the MITL construct $p\mathcal{U}_I q$, which requires q to occur within the time bounds denoted by I (while not constraining its occurrence outside that interval), and p to hold until that occurrence.[15]

In effect, RTFIL defines events in relation to other events, and then imposes real time constraints on their relative occurrence. In contrast, MITL first defines real time intervals and then requires events within those intervals, possibly in relation to other events. Thus, it appears that MITL will be found more satisfactory for reasoning about synchronous real time systems (where the synchronization is by real time), whereas RTFIL may be more effective for reasoning about asynchronous real time systems. A natural question, then, is whether there is a reasonable combination of the two logics that retains decidability. We conjecture that the answer is in the affirmative, and that a decision procedure for the combination would follow from a suitable "composition" of the procedures for the two logics. This is the case, for instance, for the untimed logics FIL and PTL($\mathcal{S}, \mathcal{U}$), where such a "combined" decision procedure follows from purely automata-theoretic methods (see some related comments in Part I, Section 5).

However, MITL when extended with the (untimed) past operator $\mathcal{S}$ appears to be more expressive. For instance, the RTFIL formula

$$[\rightarrow a \mid \rightarrow a, \rightarrow b, \rightarrow c)\mathsf{len}(2.0, 4.2]$$

can be expressed, modulo the interval constructibility condition, which we ignore for the sake of simplicity, in this extended version of MITL, which we call MITL($\mathcal{S}$), by

$$\neg a\mathcal{U}\left(\begin{array}{l} a \\ \neg b \\ \neg b\mathcal{U}\left(\begin{array}{l} b \\ \neg c \\ \neg c\mathcal{U}c \end{array}\right) \\ \mathsf{true}\mathcal{U}_{(2.0,4.2]}\left(\begin{array}{l} c \\ \neg c\mathcal{S}\left(\begin{array}{l} b \\ \neg b\mathcal{S}\left(\begin{array}{l} a \\ \neg a\mathcal{S}\mathsf{true} \end{array}\right) \end{array}\right) \end{array}\right) \end{array}\right)$$

where, as in [1], we have assumed the operator $\mathcal{S}$ and $\mathcal{U}$ to be strict in both arguments. It is an open question whether MITL($\mathcal{S}$) can express an arbitrary property expressible in RTFIL. Methods such as those used by Kutty et al. [19] may be helpful in trying to answer the question. We believe, however, that even if the answer were to be in the affirmative, this might involve a severe succinctness penalty, especially in the case of RTFIL formulae with timing constraints nested deeply within interval modalities.

[15] In each case, the introduction of auxiliary predicates mitigates the problem. Note also that the TPTL [6], with "freeze" quantification, can express the RTFIL property given earlier. Unfortunately, TPTL is undecidable when interpreted over a dense time domain.

A related, although somewhat academic, question concerns the complexity of the validity problems for real time versions of GIL of Part I. In other words, what happens to the validity problem for RTFIL when we extend it with backwards searches, obtaining the real time counterpart RTGIL of GIL? By the results of Part I, it is clearly at least non-elementary – but, is it even decidable?

6. Conclusion

We have presented a real time interval logic RTFIL which conservatively extends the timeless logic FIL. The logic extends FIL in a natural way to allow real time specification, without sacrificing decidability. We have presented a formal semantics for the logic and have given a decision procedure for it. That RTFIL involves an additional exponential factor proportional to the number of clocks and the constants appearing in the specification should come as no surprise to those familiar with other dense-time logics.

A prototype RTFIL theorem-prover based on a tableau-theoretic analogue of the decision procedure given in this paper has been implemented and used to verify some simple real time systems [21, 22, 26]. However, many opportunities remain for improving the system and making it more practical as a real life verification system. Apart from the use of efficient data structures, such as binary decision diagrams for state encoding, efficient heuristics, such as those used in [8], will need to be used in order to reduce the space requirements for the verification. Since our procedure is automata-theoretic, it can directly benefit from any advances in verification technology based on ω-automata and their real time extensions. The tools are available by anonymous ftp from alpha.ece.ucsb.edu in the directory/pub/RTGIL.

There is also a need for a proof calculus for the logic in the style of the natural deduction calculi that are now gaining popularity in many applications. The success or failure of an "expensive" logic such as RTFIL would depend crucially upon whether one is able to obtain a clean proof system. We consider our decision procedure an important first step in this direction. For instance, our reduction and transition rules can be seen as a form of "rewrite rules" for a tableau proof system. The incorporation of timers in a formal manner into such tableaux, however, presents non-trivial difficulties. One approach might be to use time variables with such operations as resetting, assignment, comparison and difference, to simulate the role of timers. However, such an approach is probably far too low level to be useful. On the other hand, some appropriate mixture of automated inference within such a proof system, along with user assistance at crucial points, would be more practical.

Finally from a more theoretical standpoint, there are interesting expressiveness questions regarding RTFIL and other decidable real time logics based on a dense model of time. The apparent duality between our approach and that of MITL, as outlined in the previous section, clearly merits further study. Another interesting direction involves identifying a natural decidable fragment of *parametric* RTFIL, in the sense of [7].

Acknowledgements

We thank Rajeev Alur for his comments on the conference version of this paper. We are also grateful to the anonymous referees for their critical reading, and for pointing out several errors in the first version.

Appendix A: Proof of Theorem 3.10

The proof of Theorem 3.10 makes heavy use of the following fact.

Fact 1. *Let X_1 and X_2 be finitely variable and right continuous functions from R to finite subsets of a set S. Let $P(b_1,\ldots,b_n)$ be a boolean function of n variables $b_1,\ldots,b_n$, and let $x_1,\ldots,x_n$ be elements of S. Then the functions*[16]

1. *$X:R \rightarrow 2^S$ defined by $X(t) = X_1(t) \cup X_2(t)$,*
2. *$B:R \rightarrow \{true, false\}$ defined by $B(t) = P[b_i \leftarrow x_i]_i$, where $x_i \in X_1(t)$*

are also finitely variable and right continuous.

Proof of Theorem 3.10. Let $\mathcal{M}$ be an admissible model. Then $\text{dom}\mathcal{M}^f = \text{dom}\,\mathcal{M}$. Moreover, since $\mathsf{scl}(f)$ is finite for any formula f, clearly $\mathcal{M}^f$ is image finite. It remains to prove that $\mathcal{M}^f$ is finitely variable and right continuous. The proof is by induction on the inclusion order induced by the subformula closure.

For the first of two base cases, we note that

$$\mathcal{M}^p(t) = \begin{cases} \{\mathsf{true}, p\} & \text{if } p \in \mathcal{M}(t) \\ \{\mathsf{true}, \neg p\} & \text{otherwise} \end{cases}$$

Finite variability and right continuity of $\mathcal{M}^p$ then follows easily from that of $\mathcal{M}$ for any $p \in \mathcal{P}$.

For the remaining base case, we note that sup $\text{dom}\,\mathcal{M} \neq t$ for $t \in \text{dom}\,\mathcal{M}$ so that, for any $t \in \text{dom}(\mathcal{M}), d \in Q$,

$$\mathcal{M}^{\mathsf{len}(0,d]}(t) = \begin{cases} \{\mathsf{true}, \mathsf{len}(0,d]\} & \text{if sup dom}\,\mathcal{M} - t \leqslant d \\ \{\mathsf{true}, \neg\mathsf{len}(0,d]\} & \text{otherwise} \end{cases}$$

Thus there is at most one right-continuous change in the valuation of $\mathcal{M}^{\mathsf{len}(0,d]}$ over $\text{dom}\,\mathcal{M}^{\mathsf{len}(0,d]}$.

For the induction step, we consider two sample cases. The remaining cases are similar.

Case 1. Consider $\mathcal{M}^{f_1 \wedge f_2}$. We have

$$\mathcal{M}^{f_1 \wedge f_2}(t) = \mathcal{M}^{f_1}(t) \cup \mathcal{M}^{f_2}(t) \cup X(t)$$

[16] The abbreviation $P[x_i \leftarrow y_i]_i$ denotes simultaneous substitution of y_i for x_i, for every i.

where

$$X(t) = \begin{cases} \{f_1 \wedge f_2\} & \text{if } f_1 \in \mathcal{M}^{f_1}(t) \text{ and } f_2 \in \mathcal{M}^{f_2}(t) \\ \{\neg(f_1 \wedge f_2)\} & \text{otherwise} \end{cases}$$

Clearly, X is finitely variable and right continuous by the second caluse of Fact 1, since $\mathcal{M}^{f_1}$ and $\mathcal{M}^{f_2}$ are. By the first clause of Fact 1, so is $\mathcal{M}^{f_1 \wedge f_2}$.

Case 2. Consider now the case of $\mathcal{M}^f$ with $f = [\rightarrow a, \theta_1 | \rightarrow b, \theta_2)f'$.

From the definitions of extension and subformula closure, we have

$$\mathcal{M}^f(t) = \bigcup_{i=1}^{4} \mathcal{M}^{f_i}(t) \cup \mathcal{M}^a(t) \cup \mathcal{M}^b(t) \cup X(t)$$

with

$$X(t) = \begin{cases} \{f\} & \text{if } \begin{cases} f_1 \in \mathcal{M}^{f_1}(t) \text{ or} \\ f_2 \in \mathcal{M}^{f_2}(t) \text{ or} \\ a \in \mathcal{M}^a(t) \text{ and } f_3 \in \mathcal{M}^{f_3}(t) \text{ or} \\ b \in \mathcal{M}^b(t) \text{ and } f_4 \in \mathcal{M}^{f_4}(t) \text{ or} \\ B(t) \end{cases} \\ \{\neg f\} & \text{otherwise} \end{cases}$$

where

$$f_1 = [\rightarrow a, \theta_1 | \rightarrow)\mathbf{false}$$
$$f_2 = [\rightarrow b, \theta_2 | \rightarrow)\mathbf{false}$$
$$f_3 = [\theta_1 | \rightarrow b, \theta_2)f'$$
$$f_4 = [\rightarrow a, \theta_1 | \theta_2)f'$$

and $B(t)$ is a boolean condition defined by

$$B(t) = \begin{cases} \exists t' > t \begin{pmatrix} a \in \mathcal{M}^a(t') \wedge f_3 \in \mathcal{M}^{f_3}(t') \wedge \\ \forall t''(t \leqslant t'' < t' \Rightarrow \neg b \in \mathcal{M}^b(t'') \wedge \neg a \in \mathcal{M}^a(t'')) \end{pmatrix} \\ \text{or} \\ \exists t' > t \begin{pmatrix} b \in \mathcal{M}^b(t') \wedge f_4 \in \mathcal{M}^{f_4}(t') \wedge \\ \forall t''(t \leqslant t'' < t' \Rightarrow \neg b \in \mathcal{M}^b(t'') \wedge \neg a \in \mathcal{M}^a(t'')) \end{pmatrix} \end{cases}$$

We now show that $B(t)$ is itself right continuous and finitely variable. By the induction hypothesis each of the functions $\mathcal{M}^a, \mathcal{M}^b, \mathcal{M}^{f_3}$ and $\mathcal{M}^{f_4}$ is right continuous and finitely variable. Consider now an arbitrary point $t \in \operatorname{dom} \mathcal{M}$. We have the following possibilities. Either $a \in \mathcal{M}^a(t)$ or $b \in \mathcal{M}^b(t)$ or neither. In the first two cases $B(t)$ is false, and continues to be false at least up to (but possibly not including) the least t' where neither $a \in \mathcal{M}^a(t')$ nor $b \in \mathcal{M}^b(t')$. Consider therefore the third case, for which $\neg a \in \mathcal{M}^a(t)$ and $\neg b \in \mathcal{M}^b(t)$. Now we have two cases depending on whether there is any point $t' > t$ where either $a \in \mathcal{M}^a(t')$ or $b \in \mathcal{M}^a(t')$.

- Assume not. Then clearly B continues to be false for all $t' \geqslant t$.
- In the alternative case, let $t' > t$ be the least point such that either $a \in \mathscr{M}^a(t')$ or $b \in \mathscr{M}^b(t')$. Then B is false on $[t, t')$ if

$$\neg((a \in \mathscr{M}^a(t') \wedge f_3 \in \mathscr{M}^{f_3}(t')) \vee (b \in \mathscr{M}^b(t') \wedge f_4 \in \mathscr{M}_{f_4}(t'))),$$

  and otherwise B is true on $[t, t')$

This establishes the right continuity of B.

Let $D_{\mathscr{M}^a}$ represent the set of points at which $\mathscr{M}^a$ has a (left) discontinuity, and similarly $D_{\mathscr{M}^b}$ for $\mathscr{M}^b$. For a subset S of R and $t \in R$, let $S \downarrow t = \{s \in S \mid s \leqslant t\}$. The finite variability condition for $\mathscr{M}^a$ is then equivalent to saying that $D_{\mathscr{M}^a} \downarrow t$ is finite for any $t \in R$. By the induction hypothesis $\mathscr{M}^a$ and $\mathscr{M}^b$ are finitely variable, so each of $D_{\mathscr{M}^a}$ and $D_{\mathscr{M}^b}$ has this property and, therefore, so also does $D_{\mathscr{M}^a} \cup D_{\mathscr{M}^b}$, and a fortiori any subset of $D_{\mathscr{M}^a} \cup D_{\mathscr{M}^b}$. As our argument above for the right continuity of B clearly shows, B is constant between any two consecutive points (in the usual ordering) in $D_{\mathscr{M}^a} \cup D_{\mathscr{M}^b}$. Therefore, $D_B \subseteq D_{\mathscr{M}^a} \cup D_{\mathscr{M}^b}$, giving finite variability for B.

Now, using Fact 1, we obtain right continuity and finitely variability, first for X, and then for $\mathscr{M}^f$. □

Note that the proof of right continuity, above, is equivalent to a proof of Theorem 2.5.

Appendix B: Proof of Theorem 4.1

As we stated earlier, our proof is by reduction to the halting problem for two-counter Minsky machines. A two-counter machine M has two counters C_1 and C_2. Assume that it is started on an input tape with a sequence of n instructions $\langle P_1, \ldots, P_n \rangle$. An instruction specifies that one of the two counters must be incremented or decremented (by one), or the head must move to another instruction conditional on one of the counters being zero; following a non-jump instruction, the head must non-deterministically proceed to one of two specified instructions. A configuration of M is a triple $\langle i, c_1, c_2 \rangle$, where $i \in [n]$ is the tape cell containing the instruction P_i that the head is reading, and c_1, c_2, are the values of the counters C_1 and C_2. We assume, without loss of generality, that the machine accepts if it reaches the last instruction, P_n.

Proof of Theorem 4.1. We encode a computation of M as a sequence of configurations along the time axis as follows. To encode the configuration $\langle i, c_1, c_2 \rangle$, we let the predicate P_i be true in the first half (of duration 1) of the "cell" of duration 2, representing the configuration, and false throughout the remaining duration of the cell. During this remaining half (of duration 1) of the cell, we let C_1 and C_2 oscillate precisely c_1 and c_2 times, respectively, before the start of the next configuration. However, at the end of the cell, we require all propositions to become quiescent, so that there is a non-zero duration during which all propositions are false. The next time that one of the P_i's (instruction predicates) becomes true, represents the start of the next configuration. Crucial to this is the ability to copy an entire prefix of a configuration into the next, so that the moves of M are simulated appropriately.

In the encoding that follows, we shall let x range over the set [2], and i, k, k_1, k_2 over $[n]$. Recall, for the sequel, the following definitional abbreviations from Section 2.2 of Part I. As before, we also use vertical juxtaposition as an abbreviation for conjunction.

$$\begin{aligned}\diamond a &\overset{\text{def}}{=} \neg[\rightarrow a| \rightarrow)\textsf{false}\\ \Box a &\overset{\text{def}}{=} \neg\diamond\neg a\\ b\mathcal{U}a &\overset{\text{def}}{=} \diamond a \wedge [\rightarrow \neg b \vee a| \rightarrow)a\end{aligned}$$

We first define well-behavedness of P_i's, $\mathbf{well}_p$, i.e. instructions are active only during the first half of a cycle, and never in the second half:

$$\mathbf{well}_\mathrm{p} \overset{\text{def}}{=} \begin{pmatrix} [-| \rightarrow +1)\bigwedge_i(P_i \Rightarrow \Box P_i) \\ [\rightarrow +1| \rightarrow +2)\Box\bigwedge_i \neg P_i \\ [\rightarrow +2| \rightarrow)\bigvee_i P_i \end{pmatrix}$$

Similarly, we define by $\mathbf{well}_c$ the well-behavedness of the counters, i.e. counters are active only during the second half and never during the first half, and counter pulses are coterminally false within any configuration:

$$\mathbf{well}_c \overset{\text{def}}{=} \begin{pmatrix} [-| \rightarrow +1)\Box\bigwedge_x \neg C_x \\ [\rightarrow +1| \rightarrow)[-| \rightarrow \bigvee_i P_i)\diamond\Box\bigwedge_x \neg C_x \end{pmatrix}$$

In the following set of definitions, **one-ins** encodes the fact that precisely one instruction is being read at any instant; ZERO(x), $x \in [2]$ that the value of counter x is zero. The predicate READ(k), $k \in [n]$, when true at the beginning of a new configuration, indicates that in the next configuration the instruction P_k is being read.

$$\textbf{one-ins} \overset{\text{def}}{=} \Box\bigwedge_i \left(P_i \Rightarrow \neg\bigvee_{j\neq i} P_j\right)$$

$$\textsc{zero}(x) \overset{\text{def}}{=} [-| \rightarrow +2)\Box\neg C_x$$

$$\textsc{read}(k) \overset{\text{def}}{=} [\rightarrow +2| \rightarrow)P_k$$

The following predicates are useful for encoding the changes in value of the counters, and testing for zero. The predicate NOMORE(x), $x \in [2]$, indicates when true that there are no more pulses of C_x until the beginning of the next configuration. The predicate LAST(x) when true indicates that this is the last pulse of C_x within this configuration. The predicate ONEMORE(x) indicates that there is precisely one more "complete" pulse of C_x in this configuration before the start of the next configuration.

$$\textsc{nomore} \overset{\text{def}}{=} \begin{pmatrix} \bigwedge\limits_i \neg P_i \\ \neg C_x \\ [-| \rightarrow \bigvee\limits_i P_i)\Box\neg C_x \end{pmatrix}$$

$$\textsc{onemore} \overset{\text{def}}{=} \left(\begin{matrix} \neg C_x \\ [-| \rightarrow \textsc{nomore}(x)) \begin{pmatrix} \Diamond C_x \\ [\rightarrow C_x, \rightarrow \neg C_x| \rightarrow)\Box\neg C_x \end{pmatrix} \end{matrix} \right)$$

$$\textsc{last}(C_x) \overset{\text{def}}{=} \begin{pmatrix} C_x \\ |-| \rightarrow \textsc{nomore}(x))\Box C_x \end{pmatrix}$$

Making use of the above abbreviations, we now define $\textsc{no-decr}(x)$, $x \in [2]$, to mean that counter x does not decrease from the current configuration to the next. Similarly, $\textsc{incr}(x)$ indicates that counter x is incremented by one from this configuration to the next, and $\textsc{decr}(x)$ indicates that it is incremented from the current configuration to the next.

$$\textsc{no-decr}(x) \overset{\text{def}}{=} (C_x \equiv [\rightarrow +2| \rightarrow)C_x \mathcal{U} \textsc{nomore}(C_x))$$

$$\textsc{incr}(x) \overset{\text{def}}{=} \begin{pmatrix} \textsc{no-decr}(x) \\ [\rightarrow \textsc{nomore}(C_x), \rightarrow +2| \rightarrow)\textsc{onemore}(x) \end{pmatrix}$$

$$\textsc{decr}(x) \overset{\text{def}}{=} \begin{pmatrix} (C_x \equiv [\rightarrow +2| \rightarrow)C_x)\mathcal{U}\textsc{last}(C_x) \\ [\rightarrow \textsc{last}(C_x), \rightarrow +2| \rightarrow)\textsc{nomore}(C_x) \end{pmatrix}$$

For the purpose of encoding a move on a specific instruction, we assume that corresponding to each instruction P_i the following predicates are available (essentially this is the "input" to our encoding procedure):

$\textsc{incr}_x(P_i) \overset{\text{def}}{=}$ increment counter x

$\textsc{decr}_x(P_i) \overset{\text{def}}{=}$ decrement counter x

$\textsc{test}_{x,k}(P_i) \overset{\text{def}}{=}$ if counter x is zero, go to P_k

$\textsc{jmp}_{k_1,k_2}(P_i) \overset{\text{def}}{=}$ non-deterministically go toP_{k_1}orP_{k_2}

Thus, there are $n^2 + 2n + 4$ predicates on each instruction P_i, a total of $O(n^3)$ propositions. Let us define an auxiliary predicate $\textsc{nd-jmp}(i)$ as

$$\textsc{nd-jmp}(i) \overset{\text{def}}{=} \bigwedge_{k_1,k_2} (\textsc{jmp}_{k_1,k_2}(P_i) \Rightarrow (\textsc{read}(k_1) \vee \textsc{read}(k_2)))$$

Using the above, the well-behavedness of the moves is encoded as follows: **move** ensures that the correct action is taken on an instruction, and **exec** ensures that all the moves are well-behaved, i.e. that the encoding is indeed an execution of M.

$$\mathbf{move} \overset{\mathrm{def}}{=} \bigwedge_i \left(P_i \Rightarrow \left(\begin{array}{c} \bigwedge_x \textsc{incr}_x(P_i) \Rightarrow \bigwedge \begin{pmatrix} \textsc{incr}(x) \\ \textsc{nd-jmp}(i) \end{pmatrix} \\ \bigwedge_x \textsc{decr}_x(P_i) \Rightarrow \bigwedge \begin{pmatrix} \neg\textsc{zero}(x) \Rightarrow \textsc{decr}(x) \\ \textsc{nd-jmp}(i) \end{pmatrix} \\ \bigwedge_{x,k} \textsc{test}_{x,k}(P_i) \Rightarrow \bigwedge \begin{pmatrix} \textsc{zero}(x) \Rightarrow \textsc{read}(k) \\ \neg\textsc{zero}(x) \Rightarrow \textsc{nd-jmp}(i) \end{pmatrix} \end{array} \right) \right)$$

$$\mathbf{exec} \overset{\mathrm{def}}{=} \begin{pmatrix} \textbf{one-ins} \\ \mathbf{well_p} \wedge \mathbf{well_c} \wedge \mathbf{move} \\ \Box(\neg \bigvee_i P_i \Rightarrow [\rightarrow \bigvee_i P_i | \rightarrow) \mathbf{well_p} \wedge \mathbf{well_c} \wedge \mathbf{move} \end{pmatrix}$$

The encoding of the halting problem for the given M is now straightforward, assuming as stated before that the machine accepts iff it reaches the last instruction P_n:

$$P_1 \wedge \bigwedge_i \textsc{zero}(x) \wedge \mathbf{exec} \wedge \Diamond P_n \qquad \Box$$

In fact, by simply replacing eventual acceptance $\Diamond P_n$ above, by recurrent acceptance $\Box\Diamond P_n$, we obtain an encoding of the recurrence problem for two-counter machines. It is shown in [1] that this problem is Σ_1^1-hard. It follows that the validity problem for RTFIL+ is at least Π_1^1-hard, so there is no finitary axiomatization for RTFIL+.

References

[1] R. Alur, Techniques for automatic verification of real time systems, Ph.D. Thesis, Dept. of Comput. Sci., Stanford University, 1991.

[2] R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs and H. Wong-Toi, An implementation of three algorithms for timing verification based on automata emptiness, in: *Proc. 13th IEEE Real Time Syst. Symp.* (1992) 157–166.

[3] R. Alur and D. Dill, Automata for modeling real time systems, in: *Proc. 17th Int. Colloq. Automata, Languages and Programming*, Lecture notes in Computer Science, Vol. 443 (Springer, Berlin, 1990) 322–335.

[4] R. Alur, T. Feder and T. Henzinger, The benefits of relaxing punctuality, in: *Proc. 10th ACM Principles of Dist. Comput.* (1991) 139–152.

[5] R. Alur, L. Fix and T.A. Henzinger, A determinizable calss of timed automata, in: *Proc. 6th Workshop Comput. Aided Verif.*, 818 (Springer, Berlin, 1994) 1–13.

[6] R. Alur and T. Henzinger, A really temporal logic, in: *Proc. 30th IEEE Found. Comput. Sci.* (1989) 164–169.

[7] R. Alur, T.A. Henzinger and M.Y. Vardi, Parametric real time reasoning, in: *Proc. 25th ACM Symp. Theory of Comput.* (1993) 592–601.

[8] R. Alur, A. Itai, R. Kurshan and M. Yannakakis, Timing verification by successive approximation, in: *Proc. 4th Workshop Comput. Aided Verif.* Lecture notes in Computer Science, Vol. 663 (Springer, Berlin, 1992) 137–150.

[9] R.J.R. Back, Refinement calculus, Part II: Parallel and reactive programs, in: *Proc REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, 430 (Springer, Berlin, 1989) 67– 93.

[10] H. Barringer, R. Kuiper and A. Pnueli, A really abstract concurrent codel and its temporal logic, in: *Proc. 18th ACM Principles Prog. Lang.* (1986) 173–183.

[11] J.R. Büchi, On a decision method in restricted second-order arithmetic, in: *Logic, Methodology and Philosophy of Science, Proc. 1960 Congress* (Stanford University Press, Palo Alto, CA, 1960) 1–11.
[12] Z. Chaochen, M.R. Hansen and P. Sestoft, Decidability and undecidability results for the duration calculus, in: *Proc. 10th Symp. Theoret. Aspects Comput. Sci.*, 665 (Springer, Berlin, 1993) 58–68.
[13] Z. Chaochen, C.A.R. Hoare and A.P. Ravn, A calculus of durations, in: *Inform Processing Lett.* **40** (1991) 269–276.
[14] D. Dill, Timing assumptions and verification of finite-state concurrent systems, in: *Proc. Int. Workshop Automatic Verification Methods for Finite-State Systems*, Lecture notes in Computer Science, Vol. 407 (Springer, Berlin, 1989) 196–212.
[15] J. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computations* (Addison-Wesley, Reading, MA, 1st ed., 1979).
[16] R. Koymans, (Real) time: a philosophical perspective, in: *Proc. REX Workshop Real-Time: Theory in Practice*, Lecture notes in Computer Science, Vol. 600 (Springer, Berlin, 1991) 353–370
[17] R. Koymans, J. Vytopil and W.-P. de Roever, Real-time programming and asynchronous message-passing, in: *Proc. 2nd ACM Principles of Dist. Comput.* (1983) 187–197.
[18] R.P. Kurshan, Analysis of discrete event coordination, in: *Proc. REX Workshop Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, 430 (Springer, Berlin, 1989) 414–453.
[19] G. Kutty, L.E. Moser, P.M. Melliar-Smith, Y.S. Ramakrishna and L.K. Dillon, Axiomatizations of interval logics, *Fundam Inform* **XXIV** (1995) 313–331.
[20] P.M. Melliar-Smith, Extending interval logic to real times systems, in: *Proc. Conf. Temporal Logic in Specification*, Lecture notes in Computer Science, Vol. 398 (Springer, Berlin, 1987) 224–242.
[21] P.M. Melliar-Smith, L.E. Moser, Y.S. Ramakrishna, G. Kutty and L.K. Dillon, A system for automated deduction in graphical interval logic, in: *Proc. 1st Int. Conf. Temporal Logic*, Lecture Notes in Artificial Intelligence, Vol. 827 (subseries of Lecture Notes in Computer Science) (Springer, Berlin, 1994)).
[22] L.E. Moser, Y.S. Ramakrishna, G. Kutty, P.M. Melliar- Smith and L.K. Dillon, A graphical environment for design of concurrent real time systems,submitted.
[23] K.T. Narayana and A. Aaby, Specification of real time systems in real time temporal interval logic, in: *Proc. 9th IEEE Real Time Syst. Symp.* (1988) 86–95.
[24] Y.S. Ramakrishna, Interval logics for temporal specification and verification, Ph.D. Thesis, Dept. of Elec. and Comput. Eng., University of California, Santa Barbara, 1993.
[25] Y.S. Ramakrishna, L.K. Dillon, L.E. Moser, P.M. Melliar-Smith and G. Kutty, A real time interval logic and its decision procedure, in: *Proc. 13th Found. Softw. Tech. & Theoret. Comput. Sci.*, Lecture notes in Computer Science, Vol. 761 (Springer, Berlin, 1993) 173–192.
[26] Y.S. Ramakrishna, P.M. Melliar-Smith, L.E. Moser, L.K. Dillon and G. Kutty, Really visual temporal reasoning, in: *Proc. 14th IEEE Real Time Syst. Symp.* (1993) 262–273.
[27] Y.S. Ramakrishna, P.M. Melliar-Smith, L.E. Moser, L.K. Dillon and G. Kutty, Interval logics and their decision procedures, Part I: An interval logic, *Theoret. Comput. Sci.* **166** (1996) 1–47.
[28] R.R. Razouk and M.M. Gorlick, A real time interval logic for reasoning about executions of real time programs, in: *Proc. 3rd Symp. Testing, Analysis and Verif., SIGSOFT Softw. Eng. Notes*, Vol. 114 (1989) 10–19.
[29] F.B. Schneider, B. Bloom and K. Marzullo, Putting time into proof outlines, in: *Proc. REX Workshop Real Time: Theory in Practice*, Lecture notes in Computer Science, Vol. 600 (Springer, Berlin, 1991) 618–639.
[30] R.L. Schwartz, P.M. Melliar-Smith and F. Vogt, An interval logic for higher-level temporal reasoning, in: *Proc. 2nd ACM Principles of Dist. Comput.* (1983) 173–186.
[31] P. Wolper, The tableau method for temporal logic: an overview, *Logique et Analyse*, Novelle Série 28^e Année, Vol. 110–111 (1985) 119–136.