On Synthesizing Controllers from Bounded-Response Properties

Oded Maler[1], Dejan Nickovic[1], and Amir Pnueli[2,3]

[1] Verimag, 2 Av. de Vignate, 38610 Gières, France
{Dejan.Nickovic,Oded.Maler}@imag.fr
[2] Weizmann Institute of Science, Rehovot 76100, Israel
[3] New York University, 251 Mercer St. New York, NY 10012, USA
Amir.Pnueli@cs.nyu.edu

Abstract. In this paper we propose a complete chain for synthesizing controllers from high-level specifications. From real-time properties expressed in the logic MTL we generate, under bounded-variability assumptions, *deterministic* timed automata to which we apply safety synthesis algorithms to derive a controller that satisfies the properties by construction. Some preliminary experimental results are reported.

1 Introduction

The problem of synthesizing controllers automatically from high-level specifications has been posed by Church [Chu63] and solved theoretically by Büchi and Landweber [BL69, TB73]. Although the topic has been subject to further, more modern, investigations, synthesis has not enjoyed the passage from theory to practice as did the similar and simpler problem of verification, mostly due to the practical complexity of the proposed algorithms. Recently some improvements have been made for untimed [PPS06, PP06] and timed [CDF+05] systems, that led to the synthesis of some non trivial controllers. This work is a further step in this direction which attempts to give a general feasible solution for the following problem:

Given a bounded-response temporal property φ defined over two distinct action alphabets A and B (encoded using mutually-disjoint sets of propositional variables), build a finite-state transducer (controller) from A^ω to B^ω such that all of its behaviors satisfy φ at all positions.

The controller in question is realized by an automaton that observes what the environment does (some $a \in A$), changes its state accordingly and outputs some $b \in B$. The whole situation can be viewed as a two-player zero-sum game between the controller and its environment where one seeks a winning strategy for the controller (see [M07] for a unified game-theoretic model). Unlike other approaches, for example those used in the control of discrete event systems [RW89] or our previous work [MPS95, AMP95], we do not start with a given "plant" or "arena" in a form of a transition system and an acceptance/winning condition expressed in terms of its states. Our starting point, like in [PR89], is a temporal logic formula which specifies constraints on the behaviors of the players as well as desired properties of their interaction. Hence the first step in the

W. Damm and H. Hermanns (Eds.): CAV 2007, LNCS 4590, pp. 95–107, 2007.

synthesis procedure is to derive the automaton *from the formula* and then apply synthesis algorithms to this automaton.

A major difficulty in such a procedure stems for the fact that synthesis algorithms are more naturally defined over *input-deterministic* automata, or, to be more precise, over automata where each non-deterministic choice can be *unambiguously* attributed to one of the two players. In such automata each joint choice of the two players induces only one transition from every state.[1] In contrast, the commonly-used procedures for translating temporal logic formulae go through non-deterministic automata whose determinization leads to automata of prohibitively-large size. Another obstacle toward the efficient realization of synthesis algorithms is the fact that the acceptance conditions in the generated automata require a complicated fixed-point computation in order to find the winning states and strategies.

In this work we avoid some of these problems by restricting our attention to *bounded-response* properties which are known to be equivalent to safety properties. These properties represent a large part of what users are interested in (especially in hard real-time systems) and lead to automata with simpler acceptance conditions (just avoid bad states) and hence to a simpler synthesis procedure. Concerning the limited scope of bounded-response properties compared to more general *liveness* properties, we can make the following comments. Liveness properties typically specify something that should "eventually" happen without specifying an upper bound on the time to elapse between now and that eventuality. Obviously, liveness properties can be viewed as an abstraction of the real specification which requires not only that some response is eventually forthcoming (which is often useless by itself), but also provides an *upper bound* on the maximal delay on the arrival of the response. In some cases, the use of such abstractions may be justified on various grounds. However, we hope to convince the reader that, in many other cases, the synthesis from bounded-response properties is very relevant and preferable and can be carried out efficiently for non-trivial problems. For such cases, why settle for an abstraction when you can work directly with the precise specification?

The main contribution of this paper is an efficient machinery that allows one to synthesize controllers automatically from specifications expressed using the real-time temporal logic MTL [Koy90], often interpreted of the time domain $\mathbb{R}_+$. Our first contribution is a transformation of such formulae, under *bounded variability assumptions* to *deterministic* timed automata. This detrminization is of particular interest as it is based on transforming the formula into a *past* formula and then applying the ideas presented in [MNP05]. The obtained automaton is then interpreted as a timed game automaton [MPS95, AMP95] to which we apply a synthesis algorithm to derive the controller.

The rest of the paper is organized as follows: Section 2 presents the syntax and semantics of the bounded-response fragment of MTL. Section 3 shows how to translate future bounded MTL formulae into past formulae and deterministic timed automata. Section 4 reports some preliminary experiments in synthesizing an arbiter from its specifications, while Section 5 mentions ongoing and future efforts to improve the performance.

[1] A notable exception is the case where the controller has limited observability and thus, after observing a sequence of adversary actions it may find itself in one of several states and its chosen action should be good with respect to *all* these states. In this case, the nondeterminism plays in favor of the adversary.

2 Signals and Their Bounded Temporal Logic

Timed behaviors can be described using either *time-event sequences* consisting of instantaneous events separated by time durations or discrete-valued *signals* which are functions from time to some discrete domain. In this work we use Boolean signals as the semantic domain, but the extension of the results to time-event sequences (which are equivalent to the timed traces of [AD94]) need not be a difficult exercise.

Let the time domain $\mathbb{T}$ be the set $\mathbb{R}_{\geq 0}$ of non-negative real numbers and let $\mathcal{B} = \{0, 1\}$. An n-dimensional Boolean signal ξ is a partial function $\xi : \mathbb{T} \to \mathbb{B}^n$ whose domain of definition is an interval $I = [0, r)$, $r \in \mathbb{N} \cup \{\infty\}$. We say that the length of the signal is r and denote this fact by $|\xi| = r$ and let $\xi[t]$ stand for the value of the signal at time t. We use $t \oplus [a, b]$ to denote $[t + a, t + b)$, that is, the Minkowski sum of $\{t\}$ and $[a, b]$, and $t \ominus [a, b] = [t - b, t - a) \cap \mathbb{T}$ for the inverse operation with saturation at zero. In the sequel we will restrict our attention to well-behaving signals whose variability is bounded.

Definition 1 (Bounded Variability). *A signal ξ is of (Δ, k)-bounded variability if for every interval of the form $[t, t + \Delta]$ the number of changes in the value of ξ is at most k. A bounded-variability signal is a signal for which such $\Delta > 0$ and finite k exist.*

Proposition 1 (Preservation of Bounded Variability). *Let ξ_1 and ξ_2 be two infinite bounded variability signals characterized, respectively, by (Δ, k_1) and (Δ, k_2), and let $\xi = \xi_1$ op ξ_2 be a signal obtained by applying the Boolean operation op to ξ_1 and ξ_2. Then, ξ is of $(\Delta, k_1 + k_2)$-bounded variability.*

This fact, which follows from the observation that for ξ to switch at time t, at least one of ξ_1 and ξ_2 should switch, implies that if we assume bounded variability of the propositional signals, we will also have bounded variability for the signals that indicate the truth values of subformulae. Hence we can build the automaton corresponding to the formula in an inductive and compositional manner based on the temporal testers introduced in [KP05] for discrete time and extended in [MNP05, MNP06] for dense time. In this construction bounded variability will be guaranteed at all levels.

We define the logic MTL-B as a bounded-horizon variant of the real-time temporal logic MTL [Koy90], such that *all* future temporal modalities are restricted to intervals of the form $[a, b]$ with $0 \leq a \leq b$ and $a, b \in \mathbb{N}$, but allow the unbounded past operator $\mathcal{S}$ (*since*) which is not really unbounded. Note that unlike MITL [AFH96], we allow "punctual" modalities with $a = b$ and in this case we will use a as a shorthand for $[a, a]$. Another deviation from MTL is the introduction of an additional past operator *precedes* ($\mathcal{P}$) which is roughly the bounded *until* operator from the point of view of the *end* of the relevant segment of the signal. This operator is *not* proposed for user-friendliness purposes, but rather to facilitate the translation from future to past. The basic formulae of MTL-B are defined by the grammar

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 \mid \varphi_2 \mathcal{S}_{[a,b]} \varphi_1 \mid \varphi_2 \mathcal{S} \varphi_1 \mid \varphi_1 \mathcal{P}_{[a,b]} \varphi_2$$

where p belongs to a set $P = \{p_1, \ldots, p_n\}$ of propositions corresponding naturally to the coordinates of the n-dimensional Boolean signal considered. The *future fragment*

of MTL-B uses only the $\mathcal{U}_{[a,b]}$ modality while the *past fragment* uses only the $\mathcal{S}_{[a,b]}$, $\mathcal{S}$ and $\mathcal{P}_{[a,b]}$ modalities. The satisfaction relation $(\xi, t) \models \varphi$, indicating that signal ξ satisfies φ at position t, is defined inductively below. We use $p[t]$ to denote the projection of $\xi[t]$ on the dimension that corresponds to variable p.

$$
\begin{aligned}
(\xi,t) &\models p && \leftrightarrow p[t] = \mathrm{T}\\
(\xi,t) &\models \neg\varphi && \leftrightarrow (\xi,t) \not\models \varphi\\
(\xi,t) &\models \varphi_1 \vee \varphi_2 && \leftrightarrow (\xi,t) \models \varphi_1 \text{ or } (\xi,t) \models \varphi_2\\
(\xi,t) &\models \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 && \leftrightarrow \exists\, t' \in t \oplus [a,b]\ (\xi,t') \models \varphi_2 \text{ and}\\
& && \quad \forall t'' \in [t,t'], (s,t'') \models \varphi_1\\
(\xi,t) &\models \varphi_2 \mathcal{S}_{[a,b]} \varphi_1 && \leftrightarrow \exists t' \in t \ominus [a,b]\ (\xi,t') \models \varphi_1 \text{ and}\\
& && \quad \forall t'' \in [t',t], (\xi,t'') \models \varphi_1\\
(\xi,t) &\models \varphi_2 \mathcal{S} \varphi_1 && \leftrightarrow \exists t' \in [0,t]\ (\xi,t') \models \varphi_1 \text{ and}\\
& && \quad \forall t'' \in (t',t], (\xi,t'') \models \varphi_1\\
(\xi,t) &\models \varphi_1 \mathcal{P}_{[a,b]} \varphi_2 && \leftrightarrow \exists t' \in t \ominus [0,b-a]\ (\xi,t') \models \varphi_2 \text{ and}\\
& && \quad \forall t'' \in [t'-b,t']\ (\xi,t'') \models \varphi_1
\end{aligned}
$$

It is important to note the difference between the future and the past operators (see Figure 1): the *until* operator points from time t toward the future, while the *since* and *precedes* operators point from t backwards. On the other hand, the *until* and *precedes* operators differ from the *since* operators as they speak on the interval *before* the event that should be observed within a bounded time interval, while the latter refers to the interval immediately *after* its occurrence.

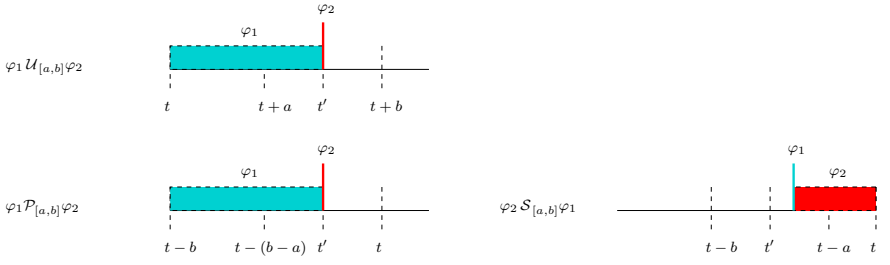

Fig. 1. The semantic definitions of *until*, *precedes* and *since*

From basic MTL-B operators one can derive other standard Boolean and temporal operators, in particular the time-constrained *sometime in the past*, *always in the past*, *eventually in the future* and *always in the future* operators whose semantics is defined as

$$
\begin{aligned}
(\xi,t) &\models \diamondminus_{[a,b]} \varphi \leftrightarrow \exists t' \in t \ominus [a,b]\ (\xi,t') \models \varphi\\
(\xi,t) &\models \boxminus_{[a,b]} \varphi \leftrightarrow \forall t' \in t \ominus [a,b]\ (\xi,t') \models \varphi\\
(\xi,t) &\models \diamondsuit_{[a,b]} \varphi \leftrightarrow \exists t' \in t \oplus [a,b]\ (s,t') \models \varphi\\
(\xi,t) &\models \Box_{[a,b]} \varphi \leftrightarrow \forall t' \in t \oplus [a,b]\ (\xi,t') \models \varphi
\end{aligned}
$$

Note that our definition of the semantics of the timed *until* and *since* operators differs slightly from their conventional definition since it requires a time instant t' where *both*

$(\xi, t') \models \varphi_2$ and $(\xi, t') \models \varphi_1$. For the untimed *since* operator we retain the standard semantics.

Each future MTL-B formula φ admits a number $D(\varphi)$ which indicates its *temporal depth*. Roughly speaking, to determine the satisfaction of φ by a signal ξ from any position t, it suffices to observe the value of ξ in the interval $[t, t + D(\varphi)]$. This property is evident from the semantics of the (bounded) temporal operators and admits the following recursive definition:

$$\begin{array}{ll} D(p) & = 0 \\ D(\neg\varphi) & = D(\varphi) \\ D(\varphi_1 \vee \varphi_2) & = \max\{D(\varphi_1), D(\varphi_2)\} \\ D(\varphi_1 \mathcal{U}_{[a,b]} \varphi_2) & = b + \max\{D(\varphi_1), D(\varphi_2)\} \end{array}$$

Note that D is a syntax-dependent *upper bound* on the actual depth: the satisfiability of a formula φ may be determined according to segments of ξ shorted than $D(\varphi)$. For example, $D(\Box_{[a,b]}\, \mathrm{T}) = b$, but the formula requires no part of ξ for its satisfiability to be determined. At the end of the day we are interested in properties of the form $\Box\, \varphi$ where φ is any (future, past or mixed) MTL-B formula. These properties are interpreted over infinite-duration signals and require that all segments of ξ of length $D(\varphi)$ satisfy φ.

3 From MTL-B to Deterministic Timed Automata

In [MP04, MNP05] we have studied the relation between real-time temporal logics and deterministic timed automata. It turns out that the non-determinism associated with real-time logics has two rather *independent* sources described below.

- *Acausality*: the semantics of future temporal logics is acausal in the sense that the satisfiability of a formula at position t may depend on the value of the sequence or signal at time $t' > t$. If the automaton has to output this value at time t, it has no choice but to "guess" at time t and abort later at time t' the computations that correspond to wrong predictions (see more detailed explanation in [MNP06]). This bounded non determinism is harmless and in the untimed case, that is, for LTL, it can be determinized away. We conjecture that such a detrminization procedure exists also for the timed case, but so far none has been reported. This problem does not exist for *past* temporal logic whose semantics is causal and hence it translates naturally into deterministic automata.
- *Unbounded variability*: when there is no bound on the variability of input signals, the automaton needs to remember the occurrence times of an unbounded number of events and use an unbounded number of clocks. All the pathological examples concerning non-determinizability and non-closure under complementation for timed automata [AD94] are based on this phenomenon.

In [MNP05] we have shown that the determinism of past MITL, compared to the non-determinism of future MITL, is a result of a syntactic accident which somehow imposes bounded variability (or indifference to small fluctuations) for the former but not the latter. The punctual version, past MTL, remains non deterministic (and of infinite memory) because the operator $\diamondsuit_a$ realizes an ideal delay element which requires unbounded memory.

The approach taken in this work in order to get rid of both sources of non determinism is the following: we use full MTL, that is, allow punctual modalities, but assume that we are dealing with signals of (Δ, k)-bounded variability, hence we can dispense with the severe form of non determinism.[2] We then transform future MTL-B formulae to past MTL-B formula which, under the bounded variability assumption, can be translated to deterministic timed automata. This part of the result is an extension of what we have shown in [MNP05] for the (non-punctual) *since* operator.

The key idea of the transformation is to change the time direction from future to past and hence eliminate the "predictive" aspect of the semantics. We will present an operator Π which takes as an argument a future formula φ and a displacement d, and transforms it to an "equivalent" past formula ψ such that φ is satisfied by a signal from position t iff ψ is satisfied by the same signal from $t + d$.

Definition 2 (Pastify Operator). *The operator Π on future* MTL-B *formulae φ and a displacement $d \geq D(\varphi)$ is defined recursively as:*

$$\begin{aligned}
&\Pi(p, d) = \diamondminus_d p \\
&\Pi(\neg\varphi, d) = \neg\Pi(\varphi, d) \\
&\Pi(\varphi_1 \vee \varphi_2, d) = \Pi(\varphi_1, d) \vee \Pi(\varphi_2, d) \\
&\Pi(\varphi_1 \mathcal{U}_{[a,b]} \varphi_2, d) = \Pi(\varphi_1, d-b) \mathcal{P}_{[a,b]} \Pi(\varphi_2, d-b)
\end{aligned}$$

Note that according the this definition $\Pi(\diamondsuit_{[a,b]}\, \varphi, d) = \diamondminus_{[0,b-a]}\, \Pi(\varphi, d-b)$.

Proposition 2 (Relation between φ and $\Pi(\varphi, d)$). *Let φ be a bounded future formula and let $\psi = \Pi(\varphi, d)$ with $d \geq D(\varphi)$. Then for every ξ and $t \geq 0$ we have:*

$$(\xi, t) \models \varphi \text{ iff } (\xi, t + d) \models \psi \tag{1}$$

Proof: We proceed by induction on the structure of the formula. The base case, the atomic propositions, satisfy (1) trivially. Proceeding to the inductive case, we show that if (1) holds for formulae with complexity (nesting of operators) m, it holds as well for formulae of complexity $m + 1$. For Boolean operators this is straightforward. Assume now that φ_1 and φ_2 satisfy (1) and we will show that so does $\varphi = \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$. Note that by definition, if $D(\varphi) = d$ then $D(\varphi_1) \leq d - b$ and $D(\varphi_2) \leq d - b$. Let $\psi_1 = \Pi(\varphi_1, d - b)$ and $\psi_1 = \Pi(\varphi_1, d - b)$. The fact the $(\xi, t) \models \varphi$ amounts to

$$\exists t' \in t \oplus [a, b]\ (\xi, t') \models \varphi_2 \wedge \forall t'' \in [0, t']\ (\xi, t'') \models \varphi_1.$$

According to the inductive hypothesis we have that for such t' and t''

$$(\xi, t' + d - b) \models \psi_2 \text{ and } (\xi, t'' + d - b) \models \psi_1.$$

By letting $r' = t' + d - b$ and $r'' = t'' + d - b$ and substituting the constraints on t' and t'' we obtain

$$\exists r' \in t + d \ominus [0, b - a]\ (\xi, r) \models \psi_2 \wedge \forall r'' \in [t + d - b, r]\ (\xi, r'') \models \psi_1,$$

[2] It is worth noting that the procedure of [T02] for subset construction on-the-fly, that is, determinization with respect to a *given* (and hence of bounded variability) input, works due to the same reasons.

which is exactly the definition of $(\xi, t+d) \models \psi_1 \mathcal{P}_{[a,b]} \psi_2$.

For the other direction assume $(\xi, t+d) \models \psi_1 \mathcal{P}_{[a,b]} \psi_2$ which means that

$$\exists r' \in t + d \ominus [0,(b-a)]\ (\xi, r') \models \psi_2 \wedge \forall r'' \in [t+d-b, r'](\xi, r'') \models \psi_1.$$

By the inductive hypothesis such r' and r'' satisfy

$$(\xi, r' - (d-b)) \models \varphi_1 \ \text{ and } \ (\xi, r'' - (d-b)) \models \varphi_1.$$

Letting $t' = r' - (d-b)$ and $t'' = r'' - (d-b)$ and substituting the constraints on r' and r'' we obtain

$$\exists t' \in t \oplus [a,b]\ (\xi, t') \models \varphi_2 \wedge \forall t'' \in [t, t']\ (\xi, t'') \models \varphi_1$$

which means that $(\xi, t) \models \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$. ▟

Corollary 1 (Equisatifaction of $\Box\,\varphi$ and $\Box\,\psi$). *An infinite signal ξ satisfies $\Box\,\varphi$ iff it satisfies $\Box\,\psi$ where $\psi = \Pi(\varphi, D(\varphi))$.*

We now proceed to the construction of a deterministic timed automaton accepting exactly signals satisfying a past MTL-B formula ψ under a bounded-variability assumption. The construction, inspired by [KP05], is compositional in the sense that it yields a network of deterministic signal transducers (testers), each corresponding to a subformula of ψ. The output of every tester for ψ' at time t equals to the satisfaction of ψ' from t. A more formal description of this framework can be found in [MNP05, MNP06]. We first present a generic automaton, the *event recorder* which was first introduced in [MNP05] for the purpose of showing that the operator $\diamondsuit_{[a,b]}$ admits a deterministic timed automaton.

The automaton depicted in Figure 2 accepts signals satisfying $\diamondsuit_{[a,b]}\,\varphi$ by simply memorizing at any time instant t the value of φ in the past temporal window $[t-b, t]$. Assuming that φ is of bounded variability and cannot change more than $2m$ times in an interval of length b, the states of the automaton, $\{0, 01, \ldots, (01)^m 0\}$, correspond to the qualitative form of the value of φ in that time interval. Each clock x_i (respectively, y_i) measures the time elapsed since the i^{th} rising (respectively, falling) of φ in the temporal window. When φ first becomes true, automaton moves from 0 to 01 and resets x_1. When φ becomes false it moves to 010 while resetting y_1 and so on. When clock $y_1 > b$, the first 01-episode of φ becomes irrelevant for the satisfaction of $\diamondsuit_{[a,b]}\,\varphi$ and can be forgotten. This is achieved by the "vertical" transitions which are accompanied by "shifting" the clocks values, that is, applying the operations $x_i := x_{i+1}$ and $y_i := y_{i+1}$ for all i. This allows us to use only a finite number of clocks.

The following proposition, first observed in [MN04], simplifies the construction of the automaton. It follows from the fact that if a bounded-variability signal is true at two close points, it has to be true throughout the interval between them.

Proposition 3. *If p is a signal of $(a,1)$-bounded variability then*

- $(\xi, t) \models p \mathcal{U}_{[a,b]} q$ iff $(\xi, t) \models p \wedge \diamondsuit_{[a,b]}(p \wedge q)$
- $(\xi, t) \models p \mathcal{P}_{[a,b]} q$ iff $(\xi, t) \models \diamondsuit_b\, p \wedge \diamondsuit_{[0,b-a]}(p \wedge q)$

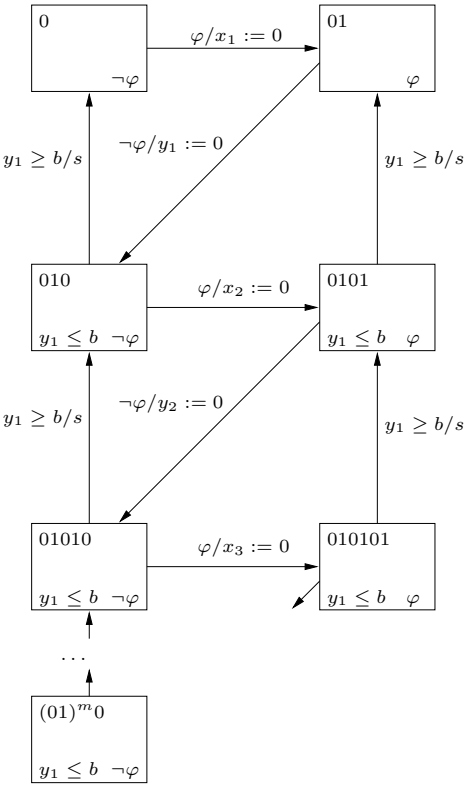

Fig. 2. An event recorder, an automaton which has φ as input and $\diamondsuit_{[a,b]}\,\varphi$ as output. The input labels and staying conditions are written on the bottom of each state. Transitions are decorated by the input labels of the target states and by clock resets. The clock shift operator is denoted by the symbol s. The automaton outputs 1 whenever $x_1 \geq a$.

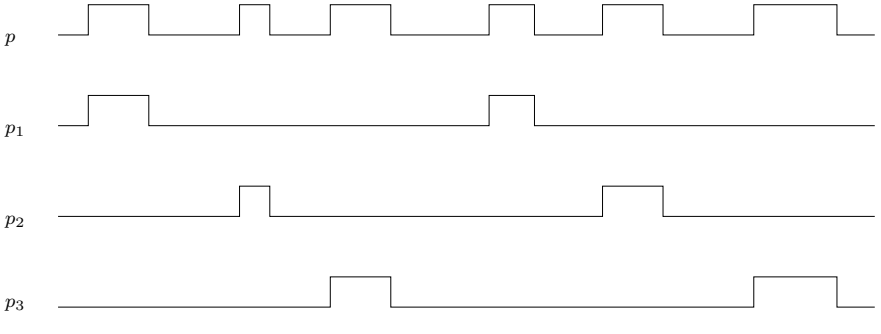

Fig. 3. Splitting p into $p_1 \vee p_2 \vee p_3$

Hence for a signal p satisfying this property, the automaton for $\mathcal{P}_{[a,b]}$ can be constructed from the event recorder by means of simple Boolean composition. Suppose now that p is of (a,k)-bounded variability with $k > 1$. We can decompose it into k signals $p_1, \ldots, p_k$ such that $p = p_1 \vee p_2 \cdots p_k$, $p_i \wedge p_j$ is always false for every $i \neq j$ and each p_i is of $(a,1)$-bounded variability. This is achieved by letting p_i rise and fall only on the j^{th} rising and falling of p, where $j = i \mod k$, as is illustrated, for $k = 3$, in Figure 3. It is not hard to see that for such p_i's we have

$$(\xi, t) \models p\mathcal{U}_{[a,b]}q \text{ iff } (\xi, t) \models \bigvee_{i=1}^{k} p_i \mathcal{U}_{[a,b]} q$$

and

$$(\xi, t) \models p\mathcal{P}_{[a,b]}q \text{ iff } (\xi, t) \models \bigvee_{i=1}^{k} p_i \mathcal{P}_{[a,b]} q.$$

The splitting of p can be done trivially using an automaton realizing a counter modulo k.

Corollary 2 (MTL-B to Deterministic Timed Automata). *Any* MITL-B *formulae can be transformed, under bounded-variability assumptions, into equivalent deterministic timed automata.*

4 Application to Synthesis

4.1 Discrete and Dense-Time Tools

What remains to be done is to transform the automaton into a timed game automaton by distinguishing controllable and uncontrollable actions and applying the synthesis algorithm. There are currently several choices for timed synthesis tools divided into two major families depending one whether discrete or dense time tools are used.[3]

- *Discrete time*: the logic and the automata are interpreted over the time domain $\mathbb{N}$. A major advantage of this approach is that the automaton becomes finite state and can be subject to symbolic verification and synthesis using BDDs, which is very useful when the discrete state space is large. On the other hand, the sensitivity of discrete time analysis to the size of the constants is much higher and will lead to explosion when they are large. Discrete-time synthesis of scheduler for fairly-large systems has been reported in [KY03].
- *Dense time*: here we have the opposite problem, namely there is a compact symbolic representation of subsets of the clock space, but the discrete states are enumerated. Several implementations of synthesis algorithms based on [MPS95] exist. One is the tool `SynthKro` included in the standard distribution of Kronos and described in [AT02], which works by standard fixpoint computation. Another alternative, which restricts the algorithm to work only on the reachable part of the

[3] Contrary to commonly-held beliefs, the important point of timed automata is not the density of time but the *symbolic* treatment of timing constraints using addition and inequalities rather than state enumeration.

state space is the tool `FlySynth` which refines the reachability graph of the game automaton according to the time-abstract bisimulation relation [TY01] yielding a finite quotient to which *untimed* synthesis algorithms can be applied [TA99]. Finally, the tool `Uppaal-Tiga` improves upon these ideas by combining forward and backward search, resulting in the most "on-the-fly" algorithm for timed synthesis [CDF+05] and probably the most effective existing tool for timed synthesis.

We have conducted our first experiments in discrete time using a synthesis algorithm implemented on top of the tool TLV, while working on the implementation of an improved dense time algorithm combining ideas from [TY01] and [CDF+05].

4.2 Example: Deriving an Arbiter

To demonstrate our approach we present a bounded-future specification of an *arbiter* module whose architectural layout is shown in Figure 4-(a). The arbiter is expected to allocate a single resource among n clients. The clients post their *requests* for the resource on the input ports $r_1, \ldots, r_n$ and receive notification of their *grants* on the arbiter's output ports $g_1, \ldots, g_n$. The protocol of communication between each client and the arbiter follows the cyclic behavior described in Figure 4-(b,c).

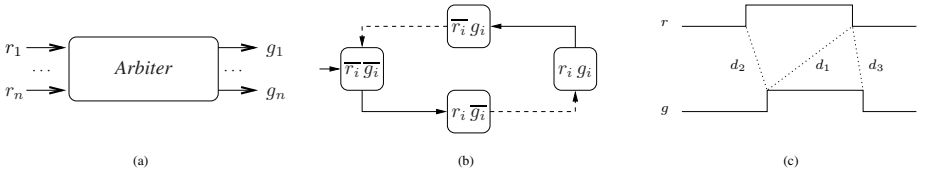

Fig. 4. (a) The architecture of an Arbiter; (b) The communication protocol between the arbiter and client i. Uncontrollable actions of the client (environment) are drawn as solid arrows, while controllable actions which are performed by the arbiter (controller) drawn as dashed arrows; (c) A typical interaction between the arbiter and a client.

In the initial state both r_i and g_i are low (0). Then, the client acts first by setting r_i to high (1) indicating a request to access the shared resource. Next, it is the turn of the arbiter to respond by raising the *grant* signal g_i to high. Sometimes later, the client terminates and indicates its readiness to relinquish the resource by lowering r_i. The arbiter acknowledges the release of the resource by lowering down the grant signal g_i.

We structure the specification into subformulae I^E, I^C, S^E, S^C, L^E and L^C denoting, respectively, the initial condition, safety component, and (bounded) liveness components of the environment (client) and the controller (arbiter). They are given by

$$
\begin{array}{l}
I^E : \bigwedge_i \overline{r_i} \\
I^C : \bigwedge_i \overline{g_i} \\
S^E : \bigwedge_i r_i \implies r_i \mathcal{S}(\overline{r_i} \wedge \overline{g_i})) \quad \wedge \quad \bigwedge_i (\overline{r_i} \implies \overline{r_i}\, \mathcal{B}(r_i \wedge g_i)) \\
S^C : \bigwedge_i (g_i \implies g_i \mathcal{S}(r_i \wedge \overline{g_i})) \quad \wedge \quad \bigwedge_i (\overline{g_i} \implies \overline{g_i}\, \mathcal{B}(\overline{r_i} \wedge g_i))
\end{array}
$$

$$L^E : \bigwedge_i (g_i \implies \Diamond_{[0,d_1]} \overline{r_i})$$
$$L^C : \bigwedge_i (r_i \implies \Diamond_{[0,d_2]} g_i) \wedge \bigwedge_i (\overline{r_i} \implies \Diamond_{[0,d_3]} \overline{g_i})$$

The initial-condition requirements I^E and I^C state that initially all variables are low. The safety requirements S^E and S^C ensure that the environment and arbiter conform to the protocol as described in Figure 4-(b). In the untimed case, this is usually specified using the next-time operator $\bigcirc$ but in dense time specify these properties using the the untimed past $\mathcal{S}$ and $\mathcal{B}$ operators.Thus, the requirement $(r_i \implies r_i \mathcal{S}(\overline{r_i} \wedge \overline{g_i}))$ states that if r_i is currently high, it must have been continuously high since a preceding state in which both r_i and g_i were low. The reader can verify that the combination of the safety properties enforces the protocol.

The (bounded) liveness property $g_i \implies \Diamond_{[0,d_1]} \overline{r_i}$ requires that if g_i holds then within b time units, client C_i should release the resource by lowering r_i. The property $(r_i \implies \Diamond_{[0,d_2]} g_i)$ specifies quality of service by saying that every client gets the resource at most d_2 time after requesting it. Finally, property $\overline{r_i} \implies \Diamond_{[0,d_3]} \overline{g_i}$ requires that the arbiter senses the release of the resource within d_3 time and considers it available for further allocations. Note that the required response delays for the various properties employ different time constants. This is essential, because the specification is realizable only if d_2, the time bound on raising g, is at least $n(d_1 + d_3)$. This reflects the "worst-case" situation that all clients request the resource at about the same time, and the arbiter has to service each of them in turn, until it gets to the last one.

The various components are combined into a single MTL-B formula by transforming them to past formulae and requiring that the controller does not violate its requirements as long as the environment does not violate hers:

$$(I^E \implies I^C) \quad \wedge \quad \Box(\boxminus(\Pi(S^E) \wedge \Pi(L^E)) \implies (\Pi(S^C) \wedge \Pi(L^C))) \quad (2)$$

Below we report some preliminary experiments in automatic synthesis of the arbiter. Table 1 shows the results of applying the procedure to Equation (2) with $d_3 = 1$ and d_1 (the upper bound on the execution time of the client) varying between 2 and 4. The N column indicates the number of clients, the *Size* column indicate the number of BDD nodes in the symbolic representation of the transition relation of the synthesized automaton and *Time* indicates the running time (in seconds) of the synthesis procedure. As one can see, there is a natural exponential growth in N and also in d_2 as expected using discrete time.

Table 1. Results for $d_1 = 2, 3, 4$

N	d_1	d_2	Size	Time	d_1	d_2	Size	Time	d_1	d_2	Size	Time
2	2	4	466	0.00	3	5	654	0.01	4	6	946	0.02
3	2	8	1382	0.14	3	10	2432	0.34	4	12	4166	0.51
4	2	12	4323	0.63	3	15	7402	1.12	4	18	16469	2.33
5	2	16	13505	1.93	3	20	26801	4.77	4	24	50674	10.50
6	2	20	43366	8.16	3	25	84027	22.55	4	30	168944	64.38
7	2	24	138937	44.38	3	30	297524	204.56	4	36	700126	1897.56

5 Conclusions and Future Work

We have made an important step toward making synthesis a usable technology by suggesting MTL-B as a suitable formalism that can handle a variety of bounded response properties encountered in the development of real-time systems. We have provided a novel translation form real-time temporal logic to deterministic timed automata via transformation to past formulae and using the reasonable bounded-variability assumption. We have demonstrated the viability of this approach by deriving a non-trivial arbiter from specifications.

In the future we intend to focus on efficient symbolic algorithms in the spirit of [CDF+05] and conduct further experiments in order to characterize the relative merits of discrete and dense-time algorithms. We also intend to apply the synthesis algorithm to more complex specifications of real-time scheduling problems.

References

[AT02] Altisen, K., Tripakis, S.: Tools for Controller Synthesis of Timed Systems. In: RT-TOOLS'02 (2002)

[AD94] Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126, 183–235 (1994)

[AFH96] Alur, R., Feder, T., Henzinger, T.A.: The Benefits of Relaxing Punctuality (first published in PODC'91). Journal of the ACM 43, 116–146 (1996)

[A04] Asarin, E.: Challenges in Timed Languages. Bulletin of EATCS 83 (2004)

[ACM02] Asarin, E., Caspi, P., Maler, O.: Timed Regular Expressions. The Journal of the ACM 49, 172–206 (2002)

[AMP95] Asarin, E., Maler, O., Pnueli, A.: Symbolic Controller Synthesis for Discrete and Timed Systems. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) Hybrid Systems II. LNCS, vol. 999, pp. 1–20. Springer, Heidelberg (1995)

[BL69] Buchi, J.R., Landweber, L.H.: Solving Sequential Conditions by Finite-state Operators. Trans. of the AMS 138, 295–311 (1969)

[CDF+05] Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005)

[Chu63] Church, A.: Logic, Arithmetic and Automata. In: Proc. of the Int. Cong. of Mathematicians 1962, pp. 23–35 (1963)

[KP05] Kesten, Y., Pnueli, A.: A Compositional Approach to CTL* Verification. Theoretical Computer Science 331, 397–428 (2005)

[KY03] Kloukinas, C., Yovine, S.: Synthesis of Safe, QoS Extendible, Application Specific Schedulers for Heterogeneous Real-Time Systems. In: ECRTS'03, pp. 287–294 (2003)

[Koy90] Koymans, R.: Specifying Real-time Properties with Metric Temporal Logic. Real-time Systems 2, 255–299 (1990)

[M07] Maler, O.: On Optimal and Reasonable Control in the Presence of Adversaries. In: Annual Reviews in Control (2007)

[MN04] Maler, O., Nickovic, D.: Monitoring Temporal Properties of Continuous Signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004)

[MNP05] Maler, O., Nickovic, D., Pnueli, A.: Real Time Temporal Logic: Past, Present, Future. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 2–16. Springer, Heidelberg (2005)

[MNP06] Maler, O., Nickovic, D., Pnueli, A.: From MITL to Timed Automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 274–289. Springer, Heidelberg (2006)

[MP04] Maler, O., Pnueli, A.: On Recognizable Timed Languages. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 348–362. Springer, Heidelberg (2004)

[MPS95] Maler, O., Pnueli, A., Sifakis, J.: On the Synthesis of Discrete Controllers for Timed Systems. In: Mayr, E.W., Puech, C. (eds.) STACS 95. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)

[PPS06] Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of Reactive(1) Designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2005)

[PP06] Piterman, N., Pnueli, A.: Faster Solutions of Rabin and Streett Games. In: LICS'06, pp. 275–284 (2006)

[PR89] Pnueli, A., Rosner, R.: On the Synthesis of a Reactive Module. In: POPL'89, pp. 179–190 (1989)

[RW89] Ramadge, P.J., Wonham, W.M.: The Control of Discrete Event Systems. Proc. of the IEEE 77, 81–98 (1989)

[TB73] Trakhtenbrot, B.A., Barzdin, Y.M.: Finite Automata: Behavior and Synthesis. North-Holland, Amsterdam (1973)

[T02] Tripakis, S.: Fault Diagnosis for Timed Automata. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 205–224. Springer, Heidelberg (2002)

[TA99] Tripakis, S., Altisen, K.: On-the-Fly Controller Synthesis for Discrete and Timed Systems, FM'99 1999. In: Woodcock, J.C.P., Davies, J., Wing, J.M. (eds.) FM 1999. LNCS, vol. 1709, Springer, Heidelberg (1999)

[TY01] Tripakis, S., Yovine, S.: Analysis of Timed Systems using Time-abstracting Bisimulations. Formal Methods in System Design 18, 25–68 (2001)