

On the Symbolic Verification of Timed Systems*

Jesper Møller Jakob Lichtenberg Henrik Reif Andersen Henrik Hulgaard

Technical Report IT-TR-1999-024
Department of Information Technology, Building 344
Technical University of Denmark
DK-2800 Lyngby, Denmark
E-mail: {jmr, jali, hra, henrik}@it.dtu.dk

Abstract

This paper describes how to analyze a timed system symbolically. That is, given a symbolic representation of a set of (timed) states (as an expression), we describe how to determine an expression that represents the set of states that can be reached either by firing a discrete transition or by advancing time. These operations are used to determine the set of reachable states symbolically. We also show how to symbolically determine the set of states that can reach a given set of states (i.e., a backwards step), thus making it possible to verify TCTL-formulae symbolically. The analysis is fully symbolic in the sense that both the discrete and the continuous part of the state space are represented symbolically. Furthermore, both the synchronous and asynchronous concurrent composition of timed systems can be performed symbolically.

The symbolic representations are given as formulae expressed in a simple first-order logic over difference constraints containing only the Boolean operators and existential quantification. Together with a recently developed data structure for efficient manipulations of the logic, the symbolic representation provides the potential of drastically increasing the size of timed systems that can be verified in practice.

Keywords: verification, real-time systems, symbolic model checking, timed automata, TCTL, difference constraint systems.

1. Introduction

Model checking [14] is today used extensively for formal verification of finite state systems such as digital circuits and embedded software. The success of the technique is

primarily due to the use of a *symbolic* representation of sets of states and relations between states as predicates over Boolean variables (using for instance BDDs [9]). By representing, for example, the set of reachable states as a predicate instead of explicitly enumerating the elements of the set, it is possible to verify systems with a very large number of states [12]. However, these symbolic methods do not easily generalize to models that contain continuous variables ranging over non-countable domains like for example real-time systems where time is modeled using continuous variables and the behavior of a system is specified using constraints on these variables. One problem is how to succinctly represent the usually infinite number of states of such systems; another problem is how to perform the basic verification operations (resetting clocks, advancing the time of clocks, etc.) symbolically on this representation in order to compute the reachable state space or to verify a temporal property of the system.

In this paper we show that a simple first-order propositional logic, called *difference constraint expressions*, over inequalities of the form $x - y \leq d$ is sufficient to perform a fully symbolic verification of real-time systems. Thus, both the representation of states and the basic verification operations are expressed entirely within this logic. Sets of states of a timed system are represented as propositions in the logic, and the operations needed to determine the set of next- or previous-states are performed using logical connectives and existential quantification. Formally, a difference constraint expression $\psi \in \Psi$ is generated by the grammar

$$\psi ::= x - y \leq d \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \exists x.\psi, \quad (1)$$

where x, y are real-valued variables and $d \in \mathbb{R}$ is a constant. An important observation is that it is not possible to directly express constraints of the form $x \leq d$ in the logic. Instead, such constraints are expressed as $x - z \leq d$, where z is a designated variable used to represent “zero.” The z -variable plays a central rôle when performing model checking of a

*Supported by a grant from the Danish Technical Research Council.

timed system since it allows us to advance time using a single existential quantification.

In this paper we focus on how to construct the set of reachable states symbolically using difference constraint expressions. The paper is organized as follows: In Section 2, we introduce a simple model of timed systems and sketch how it can represent timed automata. Section 3 shows how to determine the set of reachable states of such timed system symbolically. Section 4 describes how to perform a synchronous parallel composition symbolically (i.e., without explicitly constructing the product automaton). Section 5 introduces a data structure for representing difference constraint expressions efficiently. Section 6 presents an application of the symbolic verification of timed systems and finally Section 7 summarizes the contributions.

1.1. Related Work

Model checking of timed systems (timed automata in particular; see [26] for a survey) has been extensively studied and a number of tools exist for verifying such systems. One approach is based on making the dense domains discrete by assuming that timers only can take integer or rational values. Such a discretization makes it possible to use BDDs for representing both the discrete states and the associated timing information [3, 8, 10, 13]. However, this way of representing dense domains is often inefficient; the BDD representation is very sensitive to the granularity of the discretization and to the size of the delay ranges.

The unit-cube approach [2] models time as dense but represents the timing information using a finite number of equivalence classes. Again, the number of timed states is dependent on the size of the delay ranges and easily becomes unmanageable. Several more recent timing analysis methods use difference bound matrices (DBMs) [15] for representing the timing information [7, 17, 21, 25]: a set of DBMs representing the possible timer configurations is associated with each discrete state of the system. One can see the use of DBMs as expanding formulae of the form (1) into disjunctive normal form and representing each conjunction of difference constraints using a difference bound matrix. Although DBMs provide a compact representation of a convex set of clock configurations, there are several serious problems with the approaches based on DBMs: first, the number of DBMs for representing the timing information associated with a given state can become very large; second, there is no sharing or reuse of DBMs among the different discrete states; and third, each discrete state is represented explicitly, thus these approaches are limited by the number of reachable states of the system (the well-known state explosion problem). Several attempts have been made to remedy these shortcomings, for example by using partial order methods [6, 22, 23] or by using approximate meth-

ods [4, 5, 24]. Although these approaches do address the first problem, they are still susceptible to the last two problems since each state is represented explicitly.

Henzinger et al. [16] addresses the third problem by suggesting a symbolic approach to model checking of timed automata. There are a number of differences between this approach and ours: First, we show that the simple first-order logic (1) with only one type of clock constraints ($x - y \leq d$) is sufficient for representing the set of states of a timed system. Second, we show how to perform all operations needed in symbolic model checking within this logic. Third, we show how to perform the core operation of advancing time by introducing a designated variable z and using existential quantification. And finally, we show how to perform the synchronous parallel composition of timed automata symbolically.

2. Modeling Timed Systems

Timed guarded commands are a simple notation for modeling systems with time. The notation is sufficient for encoding popular notations for systems with time such as timed automata [2] and timed Petri nets [7].

2.1. Timed Guarded Commands

A *timed guarded command program* P is a tuple (B, C, T, I) , where B is a set of Boolean variables, C is a set of continuous variables called *clocks*, T is a set of *timed guarded commands*, and I is a *state invariant*. A timed guarded command $t \in T$ has the form

$$g \rightarrow \vec{v} := \vec{d},$$

where g is a guard and $\vec{v} := \vec{d}$ is a multi-assignment of n constant values $\vec{d} \in (\mathbb{B} \cup \mathbb{R})^n$ to Boolean variables and clocks $\vec{v} \in (B \cup C)^n$. Guards and state invariants are expressions $\phi \in \Phi$ constructed from the following grammar:

$$\begin{aligned} \phi ::= & \mathbf{0} \mid \mathbf{1} \mid x \sim d \mid x - y \sim d \mid b \\ & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists b.\phi \mid \exists x.\phi, \end{aligned} \quad (2)$$

where $x, y \in C$ are clocks, $b \in B$ is a Boolean variable, $d \in \mathbb{R}$ is a constant, and \sim is a relational operator from $\{\leq, <, =, \neq, >, \geq\}$. The symbols $\mathbf{0}$ and $\mathbf{1}$ denote false and true, respectively, and the symbols \neg (negation), \wedge (conjunction) and \exists (existential quantification) have their usual meaning. Notice that an existential quantification of a variable x can always be eliminated by keeping all implicit constraints induced by x , for example, $\exists x.((z - x < 1) \wedge (x - y \leq 0))$ is equivalent to $z - y < 1$.

Example 1 An example of a guarded command program is $P = (\{l_1, l_2\}, \{x, y\}, \{t_1, t_2\}, I)$, where the guarded commands are

$$\begin{aligned} t_1 &: l_1 \wedge (1 \leq x \leq 3) \rightarrow l_1, l_2 := 0, 1 \\ t_2 &: l_1 \wedge (7 \leq x \leq 9) \rightarrow l_1, l_2, y := 0, 1, 0, \end{aligned}$$

and the state invariant is

$$I = (l_1 \rightarrow (x \leq 9)) \wedge (l_2 \rightarrow (x \neq 5)).$$

□

2.2. Semantics of Timed Guarded Commands

A state of the program $P = (B, C, T, I)$ is an interpretation (i.e., a value assignment) of the Boolean variables and the clocks. For each Boolean variable $b \in B$, $s(b) \in \mathbb{B}$ denotes the interpretation of b in the state s , and for each clock $x \in C$, $s(x) \in \mathbb{R}$ denotes the interpretation of x in the state s . We use the notation $s[x := y + d]$ to denote the state s' equivalent to s except that $s'(x) = s(x) + d$.

A state (and sets of states) can be represented by an expression $\phi \in \Phi$ of the form (2). The state s satisfies an expression ϕ , written $s \models \phi$, if ϕ evaluates to true in the state s , and we write $\llbracket \phi \rrbracket$ for the set of states that satisfy ϕ .

The semantics of a timed guarded command program $P = (B, C, T, I)$ is a transition system (S, \rightarrow) , where S is the set of states of the program, and \rightarrow is the transition relation. In each state, the program can either execute a command $t \in T$ if its guard is true (a discrete transition) or let time pass δ time units (a timed transition). Executing a command changes the value of some or all of the variables, and letting time pass uniformly increases the values of all clocks by δ . We use the notation $s \xrightarrow{t} s'$ for a discrete transition from the state s to s' obtained by executing the command t , and the notation $s \xrightarrow{\delta} s'$ for a timed transition obtained by increasing all clocks by δ . The discrete transition \xrightarrow{t} for a timed command $t \in T$ of form $g \rightarrow \vec{v} := \vec{d}$ is defined by the following rule:

$$\frac{s \models g \quad s[\vec{v} := \vec{d}] \models I}{s \xrightarrow{t} s[\vec{v} := \vec{d}]} . \quad (3)$$

The timed transition $\xrightarrow{\delta}$ for advancing all clocks by δ is defined by the following rule:

$$\frac{\delta \geq 0 \quad \forall \delta'. 0 \leq \delta' \leq \delta : s[\vec{c} := \vec{c} + \delta'] \models I}{s \xrightarrow{\delta} s[\vec{c} := \vec{c} + \delta]} , \quad (4)$$

where $\delta, \delta' \in \mathbb{R}$, \vec{c} denotes a vector of all clocks in C , and $\vec{c} + \delta$ denotes the vector where δ is added to the clocks in \vec{c} .

Example 2 Consider the timed guarded command program P from Example 1 and let s be a state satisfying $l_2 \wedge (x <$

$5)$. There are infinitely many timed transitions from s in the transition system for P , but none of these timed transitions leads to a state where $x \geq 5$ because the state invariant $x \neq 5$ needs to hold continuously¹. □

2.3. Reachability

Given a transition system (S, \rightarrow) for a timed guarded command program $P = (B, C, T, I)$ and a set of states $S \subseteq \mathcal{S}$, $Next_{\text{discrete}}(S, t)$ denotes the set of states reachable from S by executing a timed guarded command t in T :

$$Next_{\text{discrete}}(S, t) = \{s' : s \in S \wedge s \xrightarrow{t} s'\} . \quad (5)$$

Similarly, the set of states reachable from S by advancing time by δ is given by:

$$Next_{\text{timed}}(S, \delta) = \{s' : s \in S \wedge s \xrightarrow{\delta} s'\} . \quad (6)$$

The set of states reachable from S by executing any timed guarded command in T is given by:

$$Next_{\text{discrete}}(S) = \bigcup_{t \in T} Next_{\text{discrete}}(S, t) . \quad (7)$$

The set of states reachable from S by advancing time by an arbitrary amount is given by:

$$Next_{\text{timed}}(S) = \bigcup_{\delta \in \mathbb{R}} Next_{\text{timed}}(S, \delta) . \quad (8)$$

The set of states reachable from S , denoted $Reachable(S)$, is defined as the least fixed point of the function $F(U) = S \cup Next(U)$, where

$$Next(U) = Next_{\text{discrete}}(U) \cup Next_{\text{timed}}(U) . \quad (9)$$

2.4. Encoding Timed Automata

Timed guarded command programs can be used to model popular notations for timed systems such as timed automata [2]. A timed automaton over a set of clocks consists of a set of events, a set of locations, and a set of timed transitions. Each location is associated with a location invariant over the clocks, and each timed transition from location l to location l' is labeled with an event a and has a guard g over the clocks. Furthermore, each of the timed transitions has a set of clocks $\{\vec{c}\}$ to be reset when the timed transition is fired:

$$l \xrightarrow[\{\vec{c}\}]{a, g} l' .$$

A timed automaton is encoded as a timed guarded command program. Each location can be encoded as a Boolean

¹This is an example of a program with a *time-blocked* state [1].

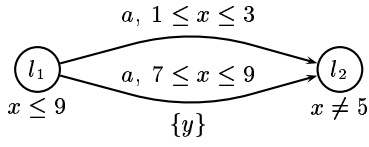


Figure 1. The timed automaton in Example 3.

variable². In a shared variable model as ours, the presence of an event from an alphabet Σ can be modeled by a global *event variable* e taking on any of the values in Σ . This variable can for instance be encoded using a logarithmic number of Boolean variables. Each timed transition corresponds to a timed guarded command:

$$l \wedge e_a \wedge g \rightarrow l, l', \vec{c} := \mathbf{0}, \mathbf{1}, \vec{0}.$$

The guard of the command is the guard of the timed transition conjoined with the source location of the timed transition and a condition e_a requiring the event variable e to have the value $a \in \Sigma$. The multi-assignment assigns $\mathbf{0}$ to the source location and $\mathbf{1}$ to the destination location of the timed transitions and resets the relevant clocks.

Example 3 Figure 1 shows an automaton over the clocks $\{x, y\}$ with two locations and two timed transitions. Encoding this automaton as a program P yields the timed guarded command program from Example 1 when ignoring the event a . \square

3. Analyzing Timed Guarded Commands

To verify properties of a timed guarded command program $P = (B, C, T, I)$, we symbolically analyze the corresponding transition system $(\mathcal{S}, \rightarrow)$. That is, given a set of states represented by a formula ψ , we determine a formula that represents the set of states that can be reached by executing a timed guarded commands according to the inference rule (3) or by advancing time according to the inference rule (4). This formula can be obtained by manipulations entirely within a simplified version of the logic (2) called *difference constraint expressions*.

3.1. Difference Constraint Expressions

A difference constraint expression $\psi \in \Psi$ is generated by the following grammar³:

$$\psi ::= x - y \leq d \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \exists x.\psi \mid b \mid \exists b.\psi. \quad (10)$$

Any expression ϕ generated by the grammar (2) can be represented by a difference constraint expression ϕ_z obtained

²This is sometimes referred to as a “one-hot” encoding of the locations.

³For convenience, we include the Boolean variables in this version of the grammar; in Section 5, we show how Boolean variables are eliminated by encoding them as difference constraints.

by introducing a new variable z (denoting “zero”) and performing the following two steps: First, replace each occurrence of a constraint of the form $x \sim d$ in ϕ with the difference constraint $x - z \sim d$. Second, express each difference constraints of the form $x - y \sim d$ in terms of the relational operator \leq using the identities:

$$\begin{aligned} x - y < d &= \neg(y - x \leq -d) \\ x - y = d &= (x - y \leq d) \wedge (y - x \leq -d) \\ x - y > d &= \neg(x - y \leq d) \\ x - y \geq d &= y - x \leq -d. \end{aligned}$$

We use $\llbracket \psi \rrbracket_z$ as a shorthand for $\llbracket \exists z.(\psi \wedge z = 0) \rrbracket$, that is, $\llbracket \psi \rrbracket_z$ is the set of states that satisfy ψ when z is equal to 0. It is easy to prove that $\llbracket \phi \rrbracket = \llbracket \phi_z \rrbracket_z$, for any $\phi \in \Phi$.

We define two useful operators on difference constraint expressions: replacement and assignment. *Replacement* syntactically substitutes all occurrences of a variable x by a variable y plus a constant d in an expression ψ , denoted by $\psi[y + d/x]$. If x and y are different variables, the replacement $\psi[y + d/x]$ can be expressed in the grammar (10) as $\exists x.(\psi \wedge (x - y = d))$. Otherwise, $\psi[x + d/x]$ is defined as $\psi[t/x][x + d/t]$, where t is a variable different from x and not occurring in ψ .

Assignment gives a variable x the value of a variable y plus a constant d , denoted by $\psi[x := y + d]$. If x and y are different variables, the assignment $\psi[x := y + d]$ can be expressed in the grammar (10) as $(\exists x.\psi) \wedge (x - y = d)$. Otherwise, the assignment $\psi[x := x + d]$ is defined as $\psi[x - d/x]$ (which might seem counter-intuitive). Assignment and replacement of Boolean variables are defined in the standard way.

Eliminating the constraints of the form $x \sim d$ from the grammar in (2) makes it possible to add δ to all clocks simultaneously by “moving” the common reference-point z :

$$\llbracket \psi[\vec{c} := \vec{c} + \delta] \rrbracket = \llbracket \psi[z := z - \delta] \rrbracket. \quad (11)$$

Furthermore, as will be shown in the following, the set of states reachable by advancing time by any value δ can be computed by an existential quantification of z .

3.2. Next States Functions

Given an expression ψ of the form (10) representing a set of states $\llbracket \psi \rrbracket_z \subseteq \mathcal{S}$, we symbolically determine an expression representing the set of states reachable from $\llbracket \psi \rrbracket_z$.

The set of states reachable by firing the timed guarded command t from any state in $\llbracket \psi \rrbracket_z$ is determined by the function $\text{NEXT}_{\text{discrete}}(\psi, t)$. The function restricts ψ to the subset where the guard g holds, performs the assignment of the constants \vec{d} to the variables \vec{v} , and restricts the resulting

set to the subset where the state invariant I holds:

$$\text{NEXT}_{\text{discrete}}(\psi, g \rightarrow \vec{v} := \vec{d}) = (\psi \wedge g_z)[\vec{v} := \vec{d}] \wedge I_z, \quad (12)$$

where the assignment $[\vec{v} := \vec{d}]$ is a shorthand for $c_i := z + d_i$ for each of the clocks c_i in \vec{v} and $b_i := d_i$ for each of Boolean variables in \vec{v} .

The set of states that can be reached from the set $\llbracket \psi \rrbracket_z$ by firing any discrete transition is given by:

$$\text{NEXT}_{\text{discrete}}(\psi) = \bigvee_{t \in T} \text{NEXT}_{\text{discrete}}(\psi, t). \quad (13)$$

The z -variable plays a central rôle when determining the set of states that can be reached from $\llbracket \psi \rrbracket_z$ by firing a timed transition. The function $\text{NEXT}_{\text{timed}}(\psi, \delta)$ symbolically determines the set of states that can be reached from $\llbracket \psi \rrbracket_z$ by advancing time by some amount $\delta \in \mathbb{R}$:

$$\text{NEXT}_{\text{timed}}(\psi, \delta) = \exists z'. (\psi \wedge (z - z' = \delta) \wedge P_{\text{next}})[z/z'], \quad (14)$$

where

$$P_{\text{next}} = (z' \leq z) \wedge \forall z''. ((z' \leq z'' \leq z) \rightarrow I_z[z''/z]). \quad (15)$$

The predicate P_{next} expresses whether time can be advanced by $\delta = z - z'$, that is, δ must be non-negative and the state invariant must hold continuously between z' and z .

Example 4 The predicate P_{next} for the state invariant $x \neq 5$ is:

$$P_{\text{next}} = (z' \leq z) \wedge \forall z''. ((z' \leq z'' \leq z) \rightarrow (x - z'' \neq 5)),$$

which is equivalent to:

$$P_{\text{next}} = (z' \leq z) \wedge \left((x < z' + 5) \vee ((x > z' + 5) \wedge (x > z + 5)) \right).$$

□

We fire all timed transitions symbolically from the set of states $\llbracket \psi \rrbracket_z$ by

$$\begin{aligned} \text{NEXT}_{\text{timed}}(\psi) &= \bigvee_{\delta \in \mathbb{R}} \text{NEXT}_{\text{timed}}(\psi, \delta) \\ &= \exists z'. (\psi \wedge P_{\text{next}})[z/z'], \end{aligned} \quad (16)$$

where P_{next} is defined in (15). The correctness of the three next state functions is given by the following theorem which is proved in Appendix A.

Theorem 1 (Correctness of NEXT-functions) For any expression $\psi \in \Psi$, any timed guarded command $t \in T$, and any $\delta \in \mathbb{R}$ the following holds:

$$\llbracket \text{NEXT}_{\text{discrete}}(\psi, t) \rrbracket_z = \text{Next}_{\text{discrete}}(\llbracket \psi \rrbracket_z, t) \quad (17)$$

$$\llbracket \text{NEXT}_{\text{timed}}(\psi, \delta) \rrbracket_z = \text{Next}_{\text{timed}}(\llbracket \psi \rrbracket_z, \delta) \quad (18)$$

$$\llbracket \text{NEXT}_{\text{timed}}(\psi) \rrbracket_z = \text{Next}_{\text{timed}}(\llbracket \psi \rrbracket_z). \quad (19)$$

Example 5 Consider a set of states satisfying $\phi = ((1 \leq x \leq 3) \vee (7 \leq x \leq 9))$. If the state invariant is $x \neq 5$ (and thus P_{next} is as given in Example 4), the set of states obtained by firing a timed transition from $\llbracket \psi \rrbracket_z$ is given by:

$$\begin{aligned} \text{NEXT}_{\text{timed}}(\phi_z) &= (\exists z'. (\phi_z \wedge P_{\text{next}}))[z/z'] \\ &= ((1 \leq x - z' < 5) \vee (7 \leq x - z'))[z/z'] \\ &= (1 \leq x - z < 5) \vee (7 \leq x - z). \end{aligned}$$

□

A timed guarded command $t \in T$ is called *urgent* if it is required to fire instantaneously whenever the guard becomes true. Modifying the semantics to handle urgent commands is straightforward if the guard does not contain clocks: Given a set $T' \subseteq T$ of urgent timed guarded commands, we let URG denote the predicate:

$$\text{URG} = \bigvee_{g \rightarrow \vec{v} := \vec{d} \in T'} g_z.$$

Consider a state $s \in \llbracket \psi \rrbracket_z$. We can only fire a timed transition $s \xrightarrow{\delta} s'$ if there are no urgent transitions enabled in s . Thus, when determining the set of states reachable by firing a timed transition from a state in $\llbracket \psi \rrbracket_z$, we first restrict ψ to only include states in which there are no enabled urgent commands and then advance time for this subset only: $\text{NEXT}_{\text{timed}}(\psi \wedge \neg \text{URG})$.

The symbolic manipulations defined in (13) and (16) form the basis for constructing the set of reachable states symbolically. Let $\text{NEXT}(\psi)$ be a function which determines the set of states which can be reached by firing either a discrete or a timed transition from a state in $\llbracket \psi \rrbracket_z$:

$$\text{NEXT}(\psi) = \text{NEXT}_{\text{discrete}}(\psi) \vee \text{NEXT}_{\text{timed}}(\psi).$$

The set of states reachable from $\llbracket \psi \rrbracket_z$, denoted $\text{REACHABLE}(\psi)$, is the least fixed point of the function $F(U) = \psi \vee \text{NEXT}(U)$, which can be determined using a standard fixed-point iteration. Detecting that a fixed point has been reached is done by checking that two successive approximations ψ_i and ψ_{i+1} are semantically equivalent (i.e., that $\psi_i \leftrightarrow \psi_{i+1}$ is a tautology). If the transition system contains a cycle with both a timed transition and a discrete transition where one of the clocks is not reset, then determination of the fixed point will potentially not terminate.

Example 6 Consider the program from Example 1. The set of states reachable from $\phi = l_1 \wedge \neg l_2 \wedge (x = y = 0)$ is $\llbracket \text{REACHABLE}(\phi_z) \rrbracket_z$, where:

$$\begin{aligned} \text{REACHABLE}(\phi_z) = & \\ & (l_1 \wedge \neg l_2 \wedge x = y \wedge x - z \leq 9) \vee \\ & (\neg l_1 \wedge l_2 \wedge 1 \leq x - z < 5 \wedge x = y) \vee \\ & (\neg l_1 \wedge l_2 \wedge 7 \leq x - z \wedge 7 \leq x - y \leq 9). \end{aligned}$$

Figure 2 shows the relation between x and y at location l_2 . \square

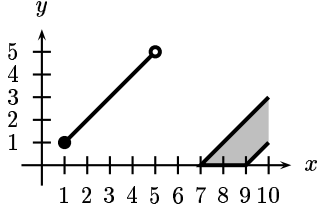


Figure 2. The relationship between clocks x and y at location l_2 .

3.3. Previous State Functions

To perform symbolic model checking, for example of a TCTL formula, the set of states that can reach a given set $\llbracket \psi \rrbracket_z$ needs to be determined. The set of states that can reach $\llbracket \psi \rrbracket_z$ by firing a timed guarded command $g \rightarrow \vec{v} := \vec{d}$ is given by

$$\text{PREV}_{\text{discrete}}(\psi) = \bigvee_{g \rightarrow \vec{v} := \vec{d} \in T} (\exists \vec{v}. \psi \wedge \vec{v} = \vec{d}) \wedge g_z \wedge I_z.$$

The set of states that can reach $\llbracket \psi \rrbracket_z$ by firing a timed transition is determined analogously to the forward case:

$$\text{PREV}_{\text{timed}}(\psi) = \exists z'. (\psi \wedge P_{\text{prev}})[z/z'],$$

where

$$P_{\text{prev}} = (z \leq z') \wedge \forall z''. ((z \leq z'' \leq z') \rightarrow I_z[z''/z]).$$

The set of states that can reach a state in $\llbracket \psi \rrbracket_z$ by firing a single transition is:

$$\text{PREV}(\psi) = \text{PREV}_{\text{discrete}}(\psi) \vee \text{PREV}_{\text{timed}}(\psi).$$

Thus, we can construct the set of states that can reach a state satisfying ψ as the fixed-point of the function $B(U) = \psi \vee \text{PREV}(U)$. Moreover, PREV can be used to perform symbolic model checking of TCTL [16]. TCTL is a timed version of CTL [14] obtained by extending the logic with an auxiliary set of clocks called *specification clocks*. These clocks do not appear in the model and are used to express

timing bounds on the temporal operators. The atomic predicates of TCTL are difference constraints over the clocks from the model and the specification clocks. Semantically, the specification clocks become part of the state, they proceed synchronously with the other clocks but are not changed by the model. A specification clock u can be bound and reset by a *reset quantifier* $u.\psi$.

Symbolically, we can find the set of states satisfying a given TCTL formula ψ by a backward computation using a fixed-point iteration for the temporal operators. For instance, the set of states satisfying the formula $\psi_1 EU \psi_2$ is computed symbolically as the least fixed point of the function

$$B(U) = \psi_2 \vee (\psi_1 \wedge \text{PREV}(U)).$$

The set of states satisfying $u.\psi$ is computed symbolically as $\exists u. \psi \wedge (u - z = 0)$, i.e., the reset quantifier corresponds to restricting the value of u to zero and then remove it by existential quantification. The atomic predicates and the Boolean connectives correspond precisely to the corresponding difference constraint expressions.

3.4. Next State Relation

In the previous sections, we have determined the set of states using a constrained image approach. To compose systems *synchronously*, as used in for instance timed automata, the timed guarded commands are encoded as a relation R over “present-state” variables $V = B \cup C \cup \{z\}$ and the “next-state” variables $V' = \{v' : v \in V\}$. The relation for the timed guarded command program P is the disjunction of the transition relation of the individual guarded commands:

$$R_{\text{discrete}} = \bigvee_{g \rightarrow \vec{v} := \vec{d} \in T} g_z \wedge (\vec{v}' = \vec{d}) \wedge \text{unch}(V \setminus \{\vec{v}'\}),$$

where $\text{unch}(X) = \bigwedge_{v \in X} v = v'$. The relation for advancing time is

$$R_{\text{timed}} = P_{\text{next}} \wedge \text{unch}(V \setminus \{z\}).$$

Using the relation

$$R = R_{\text{discrete}} \vee R_{\text{timed}}, \quad (20)$$

the function $\text{NEXT}(\psi)$ needed in a forwards fixed-point iteration becomes

$$\text{NEXT}(\psi) = (\exists \vec{v}. \psi \wedge R)[\vec{v}'/\vec{v}],$$

where \vec{v} is a vector containing all variables in V . Similarly, assuming that ψ is expressed over \vec{v}' , the backwards one-step operation is given by

$$\text{PREV}(\psi) = (\exists \vec{v}'. \psi \wedge R)[\vec{v}'/\vec{v}].$$

4. Symbolic Concurrent Compositions

In the previous sections we have shown how to symbolically analyze a timed guarded command program. The underlying semantics is *asynchronous*, i.e., the total transition relation is given as a predicate constructed as a disjunction. However, as we will show in this section, the use of difference constraint expressions as a basis for the symbolic verification of timed systems is not restricted to an asynchronous semantics but generalizes directly to synchronous compositions as it is used in for instance timed automata.

For this purpose, we extend the timed guarded command programs with a *synchronous composition operator* $*$. A synchronous system of timed guarded command programs P_j is written as:

$$P_1 * P_2 * \dots * P_m.$$

If R_j denotes the asynchronous relational semantics of the timed guarded command program P_j as given by (20), then the relational semantics of a synchronous system $P_1 * P_2 * \dots * P_m$ is $\bigwedge_{j=1}^m R_j$. Although simple, this composition is in fact quite powerful. Not only will it have the effect that the discrete transitions are synchronized but also that time proceeds at the same rate in each component. To see an example of this, we show how to symbolically model the parallel composition of timed automata as defined by Alur and Dill [2].

Assume that a set of timed automata A_j , $1 \leq j \leq m$, with alphabets Σ_j are represented by timed guarded command programs P_j using the event variable e to represent the current event from the alphabet. These automata must synchronize on common events. One way to obtain the parallel composition $A_1 \parallel A_2 \parallel \dots \parallel A_m$ of the timed automata is to make the alphabets equal for the automata by adding idling transitions and then construct the standard product automaton with respect to the events, properly combining the guards (by conjunction), the resetting of clocks (by set union), and the changes of location (implemented as multi-assignments). Using synchronous guarded command programs, this can be obtained in two steps. First, the automata are extended to a common alphabet $\Sigma = \bigcup_{j=1}^m \Sigma_j$ by adding to P_j , for all $a \in \Sigma \setminus \Sigma_j$, the idling guarded command $(e_a \rightarrow :=)$ where the guard e_a expresses that the event variable e has the value a . Denote by \tilde{P}_j the resulting timed guarded command program. Assuming that the location variables of the individual automata are disjoint, the full system is then given by:

$$\tilde{P}_1 * \dots * \tilde{P}_m * P_e,$$

where P_e is the guarded command program with no extra Boolean variables or clocks, with the trivially true state in-

variant, and with the transitions:

$$\begin{aligned} 1 &\rightarrow e := a_1 \\ &\vdots \\ 1 &\rightarrow e := a_k \end{aligned}$$

These transitions have the effect of non-deterministically choosing the next event.⁴ If \tilde{R}_j is the relation expressing \tilde{P}_j , $1 \leq j \leq m$, and R_e is the relation expressing P_e , then the resulting transition relation for a composition of the automata is

$$R = \bigwedge_{j=1}^m \tilde{R}_j \wedge R_e.$$

The next-state operator is as usual:

$$\text{NEXT}(\psi) = (\exists \vec{v}. \psi \wedge R)[\vec{v}/\vec{v}'],$$

where \vec{v} includes all the Boolean and clock variables, as well as the variables for encoding the event variable.⁵

Using difference constraint expressions, we get the benefit that well-known and very useful tricks from the work on ROBDDs, such as early variable quantification and partitioned representation of the transition relation [11] is immediately applicable, as well as the compositional techniques developed in [18]. Due to the power of difference constraint expressions, the composition is not limited to the composition of Alur and Dill, but other compositions, for instance, as used in Kronos [25] could be expressed symbolically as well.

Thus, we have shown how the parallel composition of a set of timed automata can be analyzed fully symbolically, i.e., both symbolically with respect to the parallel composition *and* with respect to the representation of sets of clock valuations and discrete states.

5. Difference Decision Diagrams

Difference decision diagrams (DDD)s are a new data structure [20] for representing difference constraint expressions using a decision diagram in a manner similar to the BDD representation of Boolean expressions. DDDs can be ordered and reduced making it possible to check for tautology and satisfiability in constant time. Furthermore, the operations for constructing and manipulating DDDs according to

⁴In the actual computation of a fixed-point the role of P_e can be replaced by simply existentially quantifying out the variables encoding e (see for instance [18]).

⁵As suggested in section 2.4 this can be done with Boolean variables, or by using a single continuous non-clock variable e with its own zero-variable z_e and encoding the values as intervals, for instance, $a_1 = [0; 1]$, $a_2 = [1; 2]$, \dots , $a_k = [k - 1; k]$, i.e., the expression e_{a_i} is $i - 1 \leq e - z_e < i$.

the syntactic constructions of (10) are easily defined recursively on the DDD data structure, thus making it simple to specify and implement algorithms for these operations.

Given a formula $\psi \in \Psi$, we encode each Boolean variable $b_i \in B$ in ψ as a difference constraint $x_i - x'_i < 0$, where $x_i, x'_i \in C$ are clocks only used in the encoding of b_i . With this encoding of Boolean variables the grammar for difference constraint expressions (10) can be simplified to

$$\psi ::= x - y \leq d \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \exists x.\psi. \quad (21)$$

Similar to how BDDs represent the meaning of Boolean formulae implicitly, we represent the meaning $\llbracket \psi \rrbracket$ of a difference constraint expression ψ of the form (21) using a decision diagram in which the vertices contain difference constraints. Formally, a DDD is a directed acyclic graph (V, E) with two terminals $\mathbf{0}$ and $\mathbf{1}$ and a set of non-terminal vertices. Each non-terminal vertex corresponds to the if-then-else operator $\alpha \rightarrow \psi_1, \psi_0$, defined as $(\alpha \wedge \psi_1) \vee (\neg\alpha \wedge \psi_0)$, where the test expression α is a difference constraint and the high-branch ψ_1 and low-branch ψ_0 are other DDD vertices. Each vertex v in a DDD denotes a difference constraint expression ψ^v given by:

$$\psi^v = \alpha(v) \rightarrow \psi^{high(v)}, \psi^{low(v)}, \quad (22)$$

where $\alpha(v)$ is the difference constraint of v , and $high(v)$ and $low(v)$ are the high- and low-branches, respectively.

As an example of a DDD consider the following expression ψ over $x, y, z \in \mathbb{R}$:

$$\psi = 1 \leq x - z \leq 3 \wedge (y - z \geq 2 \vee y - x \geq 0). \quad (23)$$

Figure 3 shows ψ as an (x, y) -plot for $z = 0$ and the corresponding DDD.

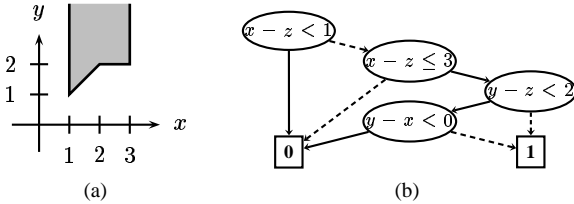


Figure 3. The expression ψ in (23) as (a) an (x, y) -plot for $z = 0$, and (b) a difference decision diagram.

Similar to ROBDDs, DDDs can be ordered and reduced. There are three levels of reduction: local reduction, path reduction, and full reduction. *Local reduction* corresponds to the reduction requirements for ROBDDs (i.e., no duplicate vertices and no redundant tests), but unlike ROBDDs, local reduction does not make DDDs a canonical representation of difference constraint expressions. *Path reduction* requires that all paths are feasible (i.e., the conjunction of difference constraints along a path has a solution), and this property makes DDDs *semi-canonical*. In

a semi-canonical representation, there is exactly one DDD for a tautology (the terminal $\mathbf{1}$) and exactly one DDD for an unsatisfiable expression (the terminal $\mathbf{0}$). Thus, with semi-canonical DDDs it is straightforward to test for tautology, satisfiability, and equivalence (after using APPLY with a bimplication).

The function $\text{APPLY}(op, u, v)$ is used to combine two ordered, locally reduced DDDs rooted at u and v with a Boolean operator op , e.g., the negation and conjunction operations in (21). APPLY is a generalization of the version used for ROBDDs and has running time $O(|u||v|)$, where $|\cdot|$ denotes the number of vertices in a DDD.

Existential quantification of a variable x in a DDD rooted at u consists of removing all vertices reachable from u containing x , but keeping all implicit constraints induced by x among the other variables. The algorithm implementing the existential quantification computes the modified and additional constraints in polynomial time, yet its worst-case running time is exponential since the resulting DDD must be ordered.

6. An Application

We demonstrate the applicability and efficacy of the symbolic approach by analyzing a timed version of Milner's scheduler [19]. Milner's scheduler consists of N cyclers, connected in a ring, that cooperate on controlling N tasks. With each cycler i we associate three Boolean variables c_i , h_i , and t_i as well as two clocks H_i and T_i . Each cycler i has two guarded commands and the task is modeled by a third guarded command:

$$\begin{aligned} c_i \wedge \neg t_i &\rightarrow H_i, T_i, t_i, c_i, h_i := 0, 0, \mathbf{1}, \mathbf{0}, \mathbf{1} \\ h_i \wedge (H_i^l \leq H_i \leq H_i^u) &\rightarrow c_{(i \bmod N)+1}, h_i := \mathbf{1}, \mathbf{0} \\ t_i \wedge (T_i^l \leq T_i \leq T_i^u) &\rightarrow t_i := \mathbf{0} \end{aligned}$$

The first guarded command is urgent, i.e., we use the urgency predicate:

$$\text{URG} = \bigvee_{1 \leq i \leq N} c_i \wedge \neg t_i.$$

The state invariant I is given by:

$$I = \bigwedge_{1 \leq i \leq N} (h_i \Rightarrow H_i \leq H_i^u) \wedge (t_i \Rightarrow T_i \leq T_i^u).$$

Table 1 shows the results of computing the set of reachable states for increasing N using a forward fixed-point iteration with front sets using the DDDs to represent the difference constraint expressions.

7. Conclusion

We have shown how difference constraint expressions can be used to fully symbolically represent and verify concur-

Table 1. The results of the experiments as carried out on a standard PC with a limit of 500,000 vertices (20MB of memory). The experiments were made with two different versions of the upper and lower bounds for the timers. For the first series of experiments (columns 2–4) we have chosen the delays to be $(T_i^l, T_i^u, H_i^l, H_i^u) = (80, 100, 25, 200)$ implying that up to six of the $2N$ timers are concurrently active. For the second series of experiments (columns 5–7) we changed the bounds to $(80 + i, 125 + i, 14 + i, 200 + i)$ increasing the number of active concurrent timers to a maximum of nine. The t -columns show the CPU time (in seconds) needed to find the reachable state space. The v -columns show the number of DDD vertices used for representing the reachable state space, and the b -columns show how many vertices are used to represent the Boolean part of the state space.

N	t	v	b	t	v	b
1	<1	19	5	<1	19	5
2	<1	96	19	<1	123	9
4	<1	645	58	1	2093	44
8	<1	1956	163	52	29972	217
16	1	5508	427	31	36978	557
32	4	18372	955	38	65803	953
64	22	67140	2011	92	146590	1569
128	162	256836	4123	384	362362	2442

rent timed systems. A key idea is to avoid representing absolute constraints. Instead these constraints are expressed relative to a special variable z , which allows, for instance, the synchronous advancing of clocks to be performed by a single existential quantification.

As with BDDs, the effect of applying the transitions and advancing the time can be expressed by either a constrained image computation or by a relation between pre- and post-variables. Both representations can be used for symbolic verification and the relational representation furthermore allows concurrent synchronous composition to be expressed symbolically. The similarities with BDDs allow the powerful techniques developed for BDDs, to be immediately applied to difference constraint expressions.

Our results show that an efficient implementation of difference constraint expressions is highly desirable and we propose an implementation using difference decision diagrams. Difference decision diagrams (DDD) possess the same abilities as BDDs of providing a compact representation while admitting an efficient tautology check. Just as BDDs provide an implementation of quantified Boolean logic, which allows the symbolic verification of discrete systems [12], DDDs provides an efficient implementation of difference constraint expressions, which allows the symbolic verification of timed systems.

Continuing extending the power of the underlying Boolean logic, the difference constraints could be replaced by the more powerful linear inequalities yielding Presburger formulae. An efficient representation of Presburger formulae would therefore, along the lines of this paper, immediately provide a symbolic verification of a guarded command language with Boolean combinations of linear inequalities as guards and linear expressions in assignments (including the extensions to automata and concurrent compositions).

References

- [1] M. Abadi and L. Lamport. An old-fashioned recipe for real-time. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *REX Workshop: "Real Time: Theory in Practice"*, *REX Workshop, Mook, The Netherlands, 1991*, volume 600 of *LNCS*, pages 1–27. Springer-Verlag, 1992.
- [2] R. Alur and D. Dill. The theory of timed automata. In *Real-Time: Theory in Practice*, number 600 in *LNCS*, pages 28–73. Springer, 1991.
- [3] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of timed automata. In *Int. Workshop on Hybrid and Real-Time System*, Mar. 1997.
- [4] F. Balarin. Approximate reachability analysis of timed automata. In *Proc. Real-Time Systems Symposium*, pages 52–61. IEEE, 1996.
- [5] W. Belluomini and C. J. Myers. Efficient timing analysis algorithms for timed state space exploration. In *Proc. Int. Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1997.
- [6] W. Belluomini and C. J. Myers. Verification of timed systems using POSETs. In *Proc. Computer Aided Verification (CAV)*, June 1998.
- [7] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.
- [8] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In *CAV'97*, *LNCS 1254*, pages 179–190, 1997.
- [9] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.
- [10] J. R. Burch. *Trace Algebra for Automatic Verification of Real-Time Concurrent Systems*. PhD thesis, Carnegie Mellon, Aug. 1992.
- [11] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In *Proc. 1991 Int. Conf. on VLSI*, Aug. 1991.
- [12] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439. IEEE Computer Society Press, 1990.
- [13] S. V. Campos, E. Clarke, W. Marrero, M. Minea, and H. Hiraiishi. Computing quantitative characteristics of finite-state real-time systems. In *Real-Time Systems Symposium*, pages 266–70. IEEE, Dec. 1994.

- [14] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.
- [15] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, LNCS 407. Springer, 1989.
- [16] T. A. Henzinger, Z. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
- [17] K. G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Proc. of the 10th Int. Conference on Fundamentals of Computation Theory*, number 965 in LNCS, pages 62–88, Aug. 1995.
- [18] J. Lind-Nielsen, H. R. Andersen, G. Behrmann, H. Hulgaaard, K. Kristoffersen, and K. G. Larsen. Verification of large state/event systems using compositionality and dependency analysis. In *Tools and Algorithms for the Construction and Analysis of Systems TACAS*, Lecture Notes in Computer Science. Springer-Verlag, 1998.
- [19] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- [20] J. Møller and J. Lichtenberg. Difference decision diagrams. Master’s thesis, Department of Information Technology, Technical University of Denmark, Building 344, DK-2800 Lyngby, Denmark, Aug. 1998.
- [21] T. G. Rokicki. *Representing and Modeling Digital Circuits*. PhD thesis, Stanford University, 1993.
- [22] T. G. Rokicki and C. J. Myers. Automatic verification of timed circuits. In D. L. Dill, editor, *Computer Aided Verification (CAV)*, number 818 in LNCS, pages 468–480, 1994.
- [23] E. Verilind, G. de Jong, and B. Lin. Efficient partial enumeration for timing analysis of asynchronous systems. In *Proc. ACM/IEEE Design Automation Conference*, 1996.
- [24] H. Wong-Toi and D. Dill. Approximations for verifying timing properties. In *Theories and Experiences for Real-Time Systems Development*. World Scientific Publishing, 1994.
- [25] S. Yovine. Kronos: A verification tool for real-time systems. *Springer Int. Journal of Software Tools for Technology Transfer*, 1(1/2), Oct. 1997.
- [26] S. Yovine. Model checking timed automata. In *Embedded Systems*, LNCS. Springer-Verlag, 1998.

A. Proof of Theorem 1

Proof of (17)

Using the definition (5):

$$Next_{\text{discrete}}(\llbracket \psi \rrbracket_z, t) = \{s' : s \in \llbracket \psi \rrbracket_z \wedge s \xrightarrow{t} s'\}.$$

From the inference rule (3), it follows that $s' = s[\vec{v} := \vec{d}]$, $s \models g$, and $s[\vec{v} := \vec{d}] \models I$. Thus,

$$\begin{aligned} Next_{\text{discrete}}(\llbracket \psi \rrbracket_z, t) &= \{s[\vec{v} := \vec{d}] : s \in \llbracket \psi \wedge g_z \rrbracket_z \wedge \\ &\quad s[\vec{v} := \vec{d}] \in \llbracket I_z \rrbracket_z\} \\ &= \llbracket (\psi \wedge g_z)[\vec{v} := \vec{d}] \wedge I_z \rrbracket_z \\ &= \llbracket NEXT_{\text{discrete}}(\psi, t) \rrbracket_z. \end{aligned}$$

Proof of (18)

Using the definition (6):

$$Next_{\text{timed}}(\llbracket \psi \rrbracket_z, \delta) = \{s' : s \in \llbracket \psi \rrbracket_z \wedge s \xrightarrow{\delta} s'\}. \quad (24)$$

From the inference rule (4), it follows that $s' = s[\vec{c} := \vec{c} + \delta]$, $s \in \llbracket \psi \rrbracket_z$, $\delta \geq 0$, and $\forall \delta'. 0 \leq \delta' \leq \delta : s[\vec{c} := \vec{c} + \delta'] \models I$. We have that

$$\begin{aligned} s[\vec{c} := \vec{c} + \delta'] \models I &\Leftrightarrow s[\vec{c} := \vec{c} + \delta'] \in \llbracket I_z \rrbracket_z \\ &\Leftrightarrow s \in \llbracket I_z[\vec{c} := \vec{c} - \delta'] \rrbracket_z \\ &\Leftrightarrow s \in \llbracket I_z[z := z + \delta'] \rrbracket_z \\ &\Leftrightarrow s \in \llbracket I_z[z - \delta'/z] \rrbracket_z. \end{aligned}$$

The second to last step follows since $\llbracket \psi[\vec{c} := \vec{c} + \delta] \rrbracket = \llbracket \psi[z := z - \delta] \rrbracket$ and the last step follows from the definition of assignment, i.e., $\llbracket \psi[x := x + d] \rrbracket = \llbracket \psi[x - d/x] \rrbracket$.

We now introduce two new variables, z' and z'' , defined to be $z' = z - \delta$ and $z'' = z - \delta'$, respectively. Then (24) can be written as

$$\begin{aligned} Next_{\text{timed}}(\llbracket \psi \rrbracket_z, \delta) &= \{s' : s \in \llbracket \psi \rrbracket_z \wedge \\ &\quad \exists z'. ((\delta = z - z') \wedge (z - z' \geq 0) \wedge \\ &\quad \forall z''. z' \leq z'' \leq z : s \in \llbracket I_z[z''/z] \rrbracket_z)\}. \end{aligned}$$

Observing that the condition on the invariant now can be expressed within the logic (10), we have that

$$\begin{aligned} Next_{\text{timed}}(\llbracket \psi \rrbracket_z, \delta) &= \{s[\vec{c} := \vec{c} + \delta] : \\ &\quad s \in \llbracket \psi \wedge \exists z'. ((\delta = z - z') \wedge (z - z' \geq 0) \wedge \\ &\quad \forall z''. (z' \leq z'' \leq z) \rightarrow I_z[z''/z]) \rrbracket_z\}. \end{aligned}$$

Using that $\{s[\vec{c} := \vec{c} + \delta] : s \in \llbracket \psi \rrbracket_z\}$ is identical to $\llbracket \psi[\vec{c} := \vec{c} + \delta] \rrbracket_z$, we obtain

$$\begin{aligned} Next_{\text{timed}}(\llbracket \psi \rrbracket_z, \delta) &= \\ &\quad \llbracket \exists z'. (\psi \wedge (\delta = z - z') \wedge P_{\text{next}})[\vec{c} := \vec{c} + \delta] \rrbracket_z. \end{aligned}$$

Since $\llbracket \phi[\vec{c} := \vec{c} + \delta] \rrbracket = \llbracket \phi[z := z - \delta] \rrbracket = \llbracket \phi[z + \delta/z] \rrbracket$, it follows that

$$\begin{aligned} Next_{\text{timed}}(\llbracket \psi \rrbracket_z, \delta) &= \\ &\quad \llbracket \exists z'. (\psi \wedge (\delta = z - z') \wedge P_{\text{next}})[z + \delta/z] \rrbracket_z. \end{aligned}$$

Using the definition of replacement, this can be written as

$$Next_{\text{timed}}(\llbracket \psi \rrbracket_z, \delta) = \llbracket (\psi \wedge P_{\text{next}})[z - \delta/z'][z + \delta/z] \rrbracket_z.$$

Observe that for any predicate $Q(z, z')$, $\llbracket Q(z, z')[z - \delta/z'][z + \delta/z] \rrbracket = \llbracket Q(z, z')[z' + \delta/z][z/z'] \rrbracket$. Using this with $Q(z, z') = \psi \wedge P_{\text{next}}$, we get:

$$\begin{aligned} Next_{\text{timed}}(\llbracket \psi \rrbracket_z, \delta) &= \llbracket (\psi \wedge P_{\text{next}})[z' + \delta/z][z/z'] \rrbracket_z \\ &= \llbracket \exists z'. (\psi \wedge P_{\text{next}} \wedge (z - z' = \delta))[z/z'] \rrbracket_z \\ &= \llbracket NEXT_{\text{timed}}(\psi, \delta) \rrbracket_z. \end{aligned}$$

Proof of (19)

Using the definition (8):

$$Next_{\text{timed}}(\llbracket \psi \rrbracket_z) = \bigcup_{\delta \in \mathbb{R}} Next_{\text{timed}}(\llbracket \psi \rrbracket_z, \delta).$$

Using (18), it follows that

$$Next_{\text{timed}}(\llbracket \psi \rrbracket_z) = \llbracket \bigvee_{\delta \in \mathbb{R}} \exists z'. (\psi \wedge (z - z' = \delta) \wedge P_{\text{next}}) [z/z'] \rrbracket_z.$$

Since existential quantification distributes over disjunction,

$$\begin{aligned} Next_{\text{timed}}(\llbracket \psi \rrbracket_z) &= \llbracket \exists z'. (\psi \wedge P_{\text{next}} \wedge \\ &\quad \bigvee_{\delta \in \mathbb{R}} (z - z' = \delta)) [z/z'] \rrbracket_z \\ &= \llbracket \exists z'. (\psi \wedge P_{\text{next}}) [z/z'] \rrbracket_z \\ &= \llbracket NEXT_{\text{timed}}(\psi) \rrbracket_z. \end{aligned}$$