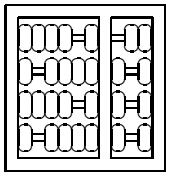

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Generating Test Suites for Timed Systems with Context Variables

A. Bonifácio *A. Moura*

Technical Report - IC-09-38 - Relatório Técnico

October - 2009 - Outubro

Generating Test Suites for Timed Systems with Context Variables

Adilson Luiz Bonifácio* Arnaldo Vieira Moura†

Abstract

Model-based testing has been widely used for testing critical and reactive systems. Some aspects of reactive systems are treated by extended models that captures the notion of data flow on the system as well as its interactions with the environment. Other aspects are captured by conventional timed models that allow for the continuous evolution of time variables. Devising formal methods to automatically generate test suites for systems that comprise both these has remained a challenge. In this work we use a new Timed Input/Output Context Automata (TIOCA) as a formal model for timed systems with context variables. We propose and prove the correctness of a new strategy to discretize TIOCA models, thus obtaining related grid automata. Grid automata allow us to automatically generate test cases based on test purposes, the latter being special TIOCA that specifically model those system properties to be tested. We also discuss how to extract test suites from the synchronous product of a TIOCA and an associated test purpose. Further, we show how to use test suites in order to automatically verify whether given implementations conform to the specification and, also, reflect the desired properties.

1 Introduction

The process of verifying and testing complex computational systems have been intensively investigated. Model-based testing has been one of the most promising techniques among those used to automatically generate test case suites for complex systems [22, 5, 17, 21]. In these formalisms, system requirements and functionalities are specified using mathematical models. Several such formalisms have been developed to automatically construct test case suites for pure timed systems with no context variables, as well as for systems were only context variables are present but no timing is considered. An approach to deal with both these aspects in the same model has remained a challenge.

In this work we treat complex systems that must obey time requirements, as well as must also satisfy data flow restrictions, the latter being specified by expressions involving context variables. In these cases, both the aspects of continuous time evolution and context variable

*Computing Institute, University of Campinas, adilson@ic.unicamp.br, Supported by CNPq grant 141978/2008-2

†Computing Institute, University of Campinas, arnaldo@ic.unicamp.br, Supported by CNPq grant 472504/2007-0

updates must be resolved within the same model. Traditionally, Timed Input/Output Automata (TIOA) have been used to express continuous time evolution [15, 13, 3]. A TIOA is a variant of the classical timed automata model [2, 1, 4]. In order to also capture data flow effects, we extend the TIOA model by introducing context variables. The new formal specification, in the form of Timed Input/Output Context Automata (TIOCA), allows for both continuous time evolution [8, 6] and data flow transformations [18, 23, 14, 19].

In order to extract test suites from these models we propose a new discretization mechanism using the notion of grid automata [7, 8]. We show how grid automata can be automatically obtained from the original TIOCA models. The discretization technique was inspired by ideas proposed in [3]. But, whereas in that work only time evolution was treated, now we address also context transformations. Notably, the proposed discretization allows for a much wider range of choices for the granularities of interest. Furthermore, the new formalism also makes precise the notion of timing and context boundaries. With that, it is then possible to precisely establish the relationship between the original system behavior and the corresponding grid automaton, when the former moves away from the established boundaries. More specifically, we show that TIOCA homomorphically simulate their corresponding grid automata, and vice-versa. This forms the basis that allows for the automatic test case generation for such systems. We also note that this new discretization method deviates from the traditional approach discussed in the open literature, which uses classical clock regions [2, 16].

In order to specify which system properties will be subjected to testing we use the notion of test purpose models [20, 12]. Given a specification and a test purpose model, their joint behavior is captured by their synchronous product. Given the product we can apply the discretization method to it, thus obtaining a corresponding grid automaton. Next, we can automatically extract test sequences from the resulting grid automaton using a simple traversal algorithm. Having extracted test sequences, we can easily obtain test cases that can be applied to implementation candidates. The implementation responds with output values sent back to the external environment. The latter are then combined with the original test cases, thus resulting in complete runs from which test conformance verdicts can be finally obtained [22].

We organize this work as follows. In Section 2, we define the notion of bounded values and bounded functions, and explore their basic properties. In Section 3, we define the new TIOCA model and some other important concepts. Section 4 presents the new discretization method. First we expose some basic concepts addressing context variables in Subsection 4.1, and then the grid construction and its properties are presented in Subsections 4.2 and 4.3. Section 5 discusses the process of generating test suites test purposes and synchronous products. It also shows how to apply test cases to obtain test verdicts. In Section 6 we briefly survey some related works. Finally, some concluding remarks appear in Section 7, together with some directions for future work.

2 Bounds and adjusted values

In this section, we define the notion of bounded values and bounded functions. The latter will be used to limit the excursions of timed automata up to a pre-defined boundary. We also introduce the notion of a set of variables and the corresponding set of rational conditions that can be derived from them. Such conditions will be used to specify both guards along transitions as well as state invariants. In order to keep the number of states of a timed model under control, we will use the notion of discretized automata. Although discretization is introduced in a later section, it will depend on the notion of adjusted values, which is also defined in this section. Finally, we also group in this section several properties involving conditions and bounded values that will be need in subsequent sections.

2.1 Fractional and integer parts

In what follows, the set of rationals, non-negative rationals and positive rationals will be denoted by $\mathbb{Q}$, $\mathbb{Q}_{\geq}$ and $\mathbb{Q}_{>}$, respectively. Also, given $t \in \mathbb{Q}_{\geq}$, we will denote the integral and fractional parts of t by $\lfloor t \rfloor$ and $\lceil t \rceil$, respectively. Hence, $t = \lfloor t \rfloor + \lceil t \rceil$ always holds.

We note the following simple facts that will be useful later.

Fact 1 *Let $x, y \in \mathbb{Q}_{\geq}$. If $x \geq y$ then $\lfloor x \rfloor \geq \lfloor y \rfloor$.*

Proof Assume $\lfloor x \rfloor < \lfloor y \rfloor$. Then $\lfloor x \rfloor \leq \lfloor y \rfloor - 1$. So $x < \lfloor x \rfloor + 1 \leq \lfloor y \rfloor - 1 + 1 = \lfloor y \rfloor \leq k$. Hence, $x < y$, contradicting the hypothesis. ∎

Fact 2 *Let $x, y \in \mathbb{Q}_{\geq}$ and let k a positive integer. If $x = y + k$ then $\lceil x \rceil = \lceil y \rceil$ and $\lfloor x \rfloor = \lfloor y \rfloor + k$.*

Proof We have that $x = y + k = (k + \lfloor y \rfloor) + \lceil y \rceil$. Since $k + \lfloor y \rfloor$ is an integer and $0 \leq \lceil y \rceil < 1$, we have $\lfloor x \rfloor = k + \lfloor y \rfloor$ and $\lceil x \rceil = \lceil y \rceil$. ∎

Fact 3 *Let $x, y, z \in \mathbb{Q}_{\geq}$. If $x = y + z$ then $\lceil x \rceil = \lceil \lceil y \rceil + \lceil z \rceil \rceil$ and $\lfloor x \rfloor = \lfloor y \rfloor + \lfloor z \rfloor + \lfloor \lceil y \rceil + \lceil z \rceil \rfloor$.*

Proof We have $x = (\lfloor y \rfloor + \lfloor z \rfloor) + \lceil y \rceil + \lceil z \rceil = (\lfloor y \rfloor + \lfloor z \rfloor + \lfloor \lceil y \rceil + \lceil z \rceil \rfloor) + \lceil \lceil y \rceil + \lceil z \rceil \rceil$. But $\lfloor y \rfloor + \lfloor z \rfloor + \lfloor \lceil y \rceil + \lceil z \rceil \rfloor$ is an integer and $0 \leq \lceil \lceil y \rceil + \lceil z \rceil \rceil < 1$. Then $\lfloor x \rfloor = \lfloor y \rfloor + \lfloor z \rfloor + \lfloor \lceil y \rceil + \lceil z \rceil \rfloor$ and $\lceil x \rceil = \lceil \lceil y \rceil + \lceil z \rceil \rceil$. ∎

Fact 4 *Let $x, y, z \in \mathbb{Q}_{\geq}$. If $\lceil x \rceil = \lceil y \rceil$ then $\lceil x + z \rceil = \lceil y + z \rceil$.*

Proof We have $\lceil x + z \rceil = \lceil \lceil x \rceil + \lceil z \rceil \rceil$, using Fact 3. Now, using the hypothesis we get $\lceil \lceil x \rceil + \lceil z \rceil \rceil = \lceil \lceil y \rceil + \lceil z \rceil \rceil$. But $\lceil \lceil y \rceil + \lceil z \rceil \rceil = \lceil y + z \rceil$, using Fact 3 again. ∎

2.2 Conditions and interpretations

Let C be a finite set of symbols, also called variables. A variable interpretation, or simply an interpretation, over C is a partial function from C into $\mathbb{Q}_{\geq}$. A total variable interpretation, or just a total interpretation, over C is a variable interpretation over C whose domain[1] is C.

Definition 5 *Let C be a set of variables. The set of all variable interpretations over C will be denoted by $[C \curvearrowright \mathbb{Q}_{\geq}]$. We denote the set of all total variable interpretations over C by $[C \to \mathbb{Q}_{\geq}]$.* ∎

Clearly, $[C \curvearrowright \mathbb{Q}_{\geq}] \subseteq [C \to \mathbb{Q}_{\geq}]$. When the intended set C is clear from the context, we may write simply interpretation, instead of variable interpretation over C.

Given a set of variables, we can form the set of all conditions over such variables.

Definition 6 *Let C be a set of variables. The set of all variable conditions, Φ_C, is comprised by all expressions δ that can be finitely generated using the rules*

$$\delta := \texttt{true} \mid c \leq \tau \mid \tau \leq c \mid \neg\delta \mid \delta_1 \wedge \delta_2,$$

where c is a variable and $\tau \in \mathbb{Q}_{\geq}$. ∎

We will take the usual liberties when writing variable conditions, *e.g.*, we may write $c \geq \tau$ for $\tau \leq v$, or $c < \tau$ instead of $\neg(\tau \leq c)$, or $\tau_1 \leq c \leq \tau_2$ for $(\tau_1 \leq c) \wedge (c \leq \tau_2)$.

Satisfiability is treated in the usual way.

Definition 7 *Let $\delta \in \Phi_C$ and let $\nu \in [C \curvearrowright \mathbb{Q}_{\geq}]$ be such that all variables occurring in δ are in $dom(\nu)$. Then we say that ν satisfies δ, denoted by $\nu \vDash \delta$, if δ evaluates to true when every variable c is replaced by $\nu(c)$ in δ and the value of the resulting propositional logic sentence is computed in the usual manner.* ∎

Note that when $\mathrm{dom}(\nu) = \emptyset$ we get $\Phi_C = \{\texttt{true}\}$, and so $\nu \vDash \texttt{true}$ always holds, for all interpretations $\nu \in [C \curvearrowright \mathbb{Q}_{\geq}]$.

We now define the operation of displacing an interpretation.

Definition 8 *Let $\nu \in [C \curvearrowright \mathbb{Q}_{\geq}]$ be an interpretation for C and let $\tau \in \mathbb{Q}_{\geq}$. The interpretation $\nu + \tau$ is defined as*

$$(\nu + \tau)(c) = \begin{cases} \nu(c) + \tau & \text{if } c \in dom(\nu) \\ \text{undefined} & \text{otherwise.} \end{cases}$$ ∎

That is, $\nu + \tau$ is given by just adding τ to all mappings under ν. Another important operation is interpretation overruling.

[1] $\mathrm{dom}(f)$ will denote the domain of a function f.

Definition 9 *Let ν, $\mu \in [C \rightharpoonup \mathbb{Q}_{\geq}]$. We define the interpretation $\nu \oplus \mu$ thus*

$$(\nu \oplus \mu)(c) = \begin{cases} \mu(c) & \text{if } c \in dom(\mu) \\ \nu(v) & \text{if } c \in (dom(\nu) - dom(\mu)) \\ undefined & \text{if } c \notin (dom(\nu) \cup dom(\mu)), \end{cases}$$

for all $c \in C$. ∎

That is, $\nu \oplus \mu$ assigns values first according to μ and, baring that, then it assigns values according to ν, if at all possible. Note that when ν or μ is total, then so is $\nu \oplus \mu$.

2.3 Bounded values and functions

Next, we turn to the notion of bounded values and bounded interpretations.

Definition 10 *Let $L \in \mathbb{Q}_{\geq}$ be a bound. Let $x \in \mathbb{Q}_{\geq}$ be a value, let A be a set and let $\alpha \in [A \rightharpoonup \mathbb{Q}_{\geq}]$ be a function. We define*

1. *The L-bounded x value, denoted x_L, is given by: (i) x, if $x \leq L$; or (ii) $\lfloor L \rfloor + \lceil x \rceil$, otherwise.*
2. *The L-bounded α function, denoted α_L, is obtained by letting $\alpha_L(a) = (\alpha(a))_L$, for all $a \in A$.* ∎

Note that we could have specified a different bound L_c for each variable c. In order to keep the notation uncluttered, however, we will consider a single bound L for all variables. It should be a simple matter to generalize any of the following results to the case when some variable bounds may be distinct.

Next, a few simple facts that will be used later.

Fact 11 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let $x \in \mathbb{Q}_{\geq}$ be a value. Then $\lceil x_L \rceil = \lceil x \rceil$ and $x_L \leq x$.*

Proof The equality is immediate from Definition 10. Now, if $x \leq L$, then $x_L = x$ and so $x_L \leq x$. When $x > L$, then $x_L = \lfloor L \rfloor + \lceil x \rceil$. From Fact 1, we get $\lfloor x \rfloor \geq \lfloor L \rfloor$. Then, $x_L \leq \lfloor x \rfloor + \lceil x \rceil = x$, and we are done. ∎

Fact 12 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let $x \in \mathbb{Q}_{\geq}$ be a value. If $x < \lfloor L \rfloor + 1$ then $x_L = x$.*

Proof If $x \leq L$ then we know that $x_L = x$. If $x > L$ then we have $\lfloor x \rfloor \geq \lfloor L \rfloor$, using Fact 1. From the hypothesis we get $\lfloor x \rfloor < \lfloor L \rfloor + 1$, and so $\lfloor x \rfloor \leq \lfloor L \rfloor$. Then, $\lfloor x \rfloor = \lfloor L \rfloor$ and we get $x_L = \lfloor L \rfloor + \lceil x \rceil = \lfloor x \rfloor + \lceil x \rceil = x$, as desired. ∎

Fact 13 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let $x \in \mathbb{Q}_{\geq}$ be a value. Then $x_L < \lfloor L \rfloor + 1$.*

Proof If $x < \lfloor L \rfloor + 1$ then $x_L = x$, using Fact 12. So, $x_L < \lfloor L \rfloor + 1$. If $x \geq \lfloor L \rfloor + 1$ then $x > L$, and we get $x_L = \lfloor L \rfloor + \lceil x \rceil$. Then, $x_L < \lfloor L \rfloor + 1$, since $\lceil x \rceil < 1$. ∎

Fact 14 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let $x \in \mathbb{Q}_{\geq}$ be a value. Then $(x_L)_L = x_L$.*

Proof From Fact 13, we know that $x_L < \lfloor L \rfloor + 1$. Now, using Fact 12 we get $(x_L)_L = x_L$. ∎

The next three propositions state that the L-bound of a sum of terms is the same as the L-bound of the sum of the L-bounds of the individual terms.

Proposition 15 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let x, $y \in \mathbb{Q}_{\geq}$. Then $(x + y)_L = (x + y_L)_L$.*

Proof There are two cases.

Case 1: $y < \lfloor L \rfloor + 1$. Then, by Fact 12, $y = y_L$ and we are done.

Case 2: $y \geq \lfloor L \rfloor + 1$. Then, $y_L = \lfloor L \rfloor + \lceil y \rceil$.

Also, $x + y \geq \lfloor L \rfloor + 1$ and so $(x + y)_L = \lfloor L \rfloor + \lceil x + y \rceil$. There are two subcases.

Case 2A: $x + y_L < \lfloor L \rfloor + 1$. Then, by Fact 12, $(x + y_L)_L = x + y_L = x + \lfloor L \rfloor + \lceil y \rceil$.

But $x + y_L < \lfloor L \rfloor + 1$ and so $x + \lfloor L \rfloor + \lceil y \rceil < \lfloor L \rfloor + 1$, and we have $x + \lceil y \rceil < 1$. Then $x = \lceil x \rceil$ and we get $\lceil x \rceil + \lceil y \rceil < 1$, and so $\lceil \lceil x \rceil + \lceil y \rceil \rceil = \lceil x \rceil + \lceil y \rceil$. Using Fact 3 we get $\lceil x + y \rceil = \lceil x \rceil + \lceil y \rceil = x + \lceil y \rceil$. This gives $(x + y)_L = (x + y_L)_L$, as desired.

Case 2B: $x + y_L \geq \lfloor L \rfloor + 1$. Then $(x + y_L)_L = \lfloor L \rfloor + \lceil x + y_L \rceil$.

But, by Fact 2, $\lceil x + y_L \rceil = \lceil x + \lfloor L \rfloor + \lceil y \rceil \rceil = \lceil x + \lceil y \rceil \rceil$. And, by Fact 2 again, $\lceil x + y \rceil = \lceil x + \lfloor y \rfloor + \lceil y \rceil \rceil = \lceil x + \lceil y \rceil \rceil$. Then $(x + y_L)_L = \lfloor L \rfloor + \lceil x + y \rceil = (x + y)_L$, as desired. ∎

Proposition 16 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let x, $y \in \mathbb{Q}_{\geq}$. Then $(x + y)_L = (x_L + y_L)_L$.*

Proof From Proposition 15 we get $(x + y)_L = (x + y_L)_L$. From Proposition 15 again, we get $(x + y_L)_L = (y_L + x)_L = ((y_L + x_L)_L)_L = ((x_L + y_L)_L)_L$. Now, from Fact 14, $((x_L + y_L)_L)_L = (x_L + y_L)_L$. Then, $(x + y)_L = (x_L + y_L)_L$. ∎

Proposition 17 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let x, $y \in \mathbb{Q}_{\geq}$. Let $x' = x$ or $x' = x_L$ and let $y' = y$ or $y' = y_L$. Then $(x + y)_L = (x' + y')_L$.*

Proof If $x' = x$ and $y' = y$ we are done. If $x' = x$ and $y' = y_L$ use Proposition 15. Similarly, if $x' = x_L$ and $y' = y$. Finally, if $x' = x_L$ and $y' = y_L$, use Proposition 16. ∎

We can now extend these results for larger sums.

Proposition 18 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let $x^i \in \mathbb{Q}_{\geq}$ be values, for $i = 1, \ldots, n$, with $n \geq 1$. Let $y^i = x^i$ or $y^i = (x^i)_L$, for $i = 1, \ldots, n$. Then*

$$\left(\sum_{i=1}^{n} x^i\right)_L = \left(\sum_{i=1}^{n} y^i\right)_L.$$

Proof We denote $(x^i)_L$ by x^i_L and proceed inductively on n.

When $n = 1$, if $y^1 = x^1$ we are done, and when $y^1 = x^1_L$ we use Fact 14.

Now, assume the result holds for some $n \geq 1$. Using Proposition 17, we get

$$\big(\sum_{i=1}^{n+1} x^i\big)_L = \big((\sum_{i=1}^{n} x^i) + x^{n+1}\big)_L = \big((\sum_{i=1}^{n} x^i)_L + y^{n+1}\big)_L.$$

Using the induction hypothesis and proposition 17, we get

$$\big(\sum_{i=1}^{n+1} x^i\big)_L = \big((\sum_{i=1}^{n} y^i)_L + y^{n+1}\big)_L = \big((\sum_{i=1}^{n} y^i) + y^{n+1}\big)_L = \big(\sum_{i=1}^{n+1} y^i\big)_L,$$

as desired. ∎

Proposition 19 *Let $L \in \mathbb{Q}_{\geq}$ be a bound. Let $W = \{w_1, \ldots, w_n\}$ be a set, with $n \geq 1$. Also let $k_i \geq 0$ be integer constants, $i = 1, \ldots, n$. Take $\alpha \in [W \to \mathbb{Q}_{\geq}]$. Then*

$$\Big[\sum_{i=1}^{n} k_i \alpha(w_i)\Big]_L = \Big[\sum_{i=1}^{n} k_i \alpha_L(w_i)\Big]_L.$$

Proof Using Proposition 18 we get $\Big[\sum_{i=1}^{n} k_i\alpha(w_i)\Big]_L = \Big[\sum_{i=1}^{n} \big(k_i\alpha(w_i)\big)_L\Big]_L$.

Now, $\Big[k_i\alpha(w_i)\Big]_L = \Big[\sum_{j=1}^{k_i} \alpha(w_i)\Big]_L$.

Hence, using Proposition 18 again, we obtain $\Big[k_i\alpha(w_i)\Big]_L = \Big[\sum_{j=1}^{k_i}(\alpha(w_i))_L\Big]_L = \Big[k_i\alpha_L(w_i)\Big]_L$.

Putting it together, we have $\Big[\sum_{i=1}^{n} k_i\alpha(w_i)\Big]_L = \Big[\sum_{i=1}^{n} \big(k_i\alpha_L(w_i)\big)_L\Big]_L$.

Finally, using Proposition 18 once more, we get $\Big[\sum_{i=1}^{n} k_i\alpha(w_i)\Big]_L = \Big[\sum_{i=1}^{n} k_i\alpha_L(w_i)\Big]_L$, as desired. ∎

The next two propositions examine the result of L-bounding displaced and overruled interpretations.

Proposition 20 *Let C be a set of variables and let $\nu \in [C \to \mathbb{Q}_{\geq}]$ be a variable interpretation, $\eta \in \mathbb{Q}_{\geq}$, and let L be a positive integer. Then $(\nu + \eta)_L = (\nu_L + \eta)_L$.*

Proof It suffices to show that $(\nu + \eta)_L(c) = (\nu_L + \eta)_L(c)$, for all $c \in C$. We know that, by definition, $(\nu + \eta)_L(c) = \big[(\nu + \eta)(c)\big]_L = \big[\nu(c) + \eta\big]_L$. Using Proposition 18, we now obtain $\big[\nu(c) + \eta\big]_L = \big[(\nu(c))_L + \eta\big]_L$. Using the definitions again, $\big[(\nu(c))_L + \eta\big]_L = \big[\nu_L(c) + \eta\big]_L = (\nu_L + \eta)_L(c)$, and the result follows. ∎

Proposition 21 *Let C be a set of variables and let $\nu \in [C \to \mathbb{Q}_{\geq}]$ and $\theta \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ be variable interpretations, and let L be a positive integer. Then $(\nu \oplus \theta)_L = (\nu_L \oplus \theta)_L$.*

Proof It suffices to show that $(\nu \oplus \theta)_L(c) = (\nu_L \oplus \theta)_L(c)$, for all $c \in C$. If $c \notin \mathrm{dom}(\theta)$, we have $(\nu \oplus \theta)_L(c) = \nu_L(c)$ and $(\nu_L \oplus \theta)_L(c) = (\nu_L)_L(c) = \nu_L(c)$, using Fact 14. If $c \in \mathrm{dom}(\theta)$, we have $(\nu \oplus \theta)_L(c) = \theta_L(c)$ and $(\nu_L \oplus \theta)_L(c) = \theta_L(c)$, completing the proof. ∎

Function addition is taken in its usual meaning. Hence, if $\alpha_1,\ \alpha_2 \in [C \to \mathbb{Q}_{\geq}]$, then $\alpha_1 + \alpha_2 \in [C \to \mathbb{Q}_{\geq}]$ is the function given by $(\alpha_1 + \alpha_2)(c) = \alpha_1(c) + \alpha_2(c)$ for all $c \in C$. The next lemma says that the satisfiability of variable conditions is oblivious to L-bounding any operand in function addition expressions.

Lemma 22 *Let C be a set of variables and let $\alpha_i,\ \beta_i \in [C \to \mathbb{Q}_{\geq}]$, $i = 1, 2$, be variable interpretations. Assume that $\beta_i = \alpha_i$ or $\beta_i = (\alpha_i)_L$ for all $i \in \{1, 2\}$. Further, let $I \in \Phi_C$ be a variable condition and let L be a positive integer greater than all constants occurring in I. Then $\alpha_1 + \alpha_2 \vDash I$ iff $\beta_1 + \beta_2 \vDash I$.*

Proof If I is `true` we are done. Assume now that I is not `true`.

From Fact 1 we know that $\beta_i(c) \leq \alpha_i(c)$, for all $c \in C$ and all $i \in \{1, 2\}$.

We treat first the four simple cases when I is $(c \leq \tau)$, $(c \geq \tau)$, $\neg(c \leq \tau)$ or $\neg(c \geq \tau)$.

Case 1: I is $(c \leq \tau)$, for some $c \in C$ and some $\tau \in \mathbb{Q}_{\geq}$.

If $\alpha_1 + \alpha_2 \vDash I$ then $(\alpha_1 + \alpha_2)(c) = \alpha_1(c) + \alpha_2(c) \leq \tau$. Then $\beta_1(c) + \beta_2(c) \leq \tau$ and so $\beta_1 + \beta_2 \vDash I$.

For the converse, assume $\beta_1(c) + \beta_2(c) \leq \tau$. From the hypothesis, $\beta_1(c) + \beta_2(c) \leq L$.

If $\alpha_1(c) > L$ then either (i) $\beta_1(c) = \alpha_1(c) > L$, or (ii) $\beta_1(c) = (\alpha_i(c))_L = \lfloor L \rfloor + \lceil \alpha_1(c) \rceil = L + \lceil \alpha_1(c) \rceil$, since L is an integer. In any case, we contradict $\beta_1(c) + \beta_2(c) \leq L$.

Similarly, we cannot have $\alpha_2(c) > L$. Hence, $\alpha_i(c) \leq L$ and we get $\beta_i(c) = \alpha_i(c)$, for $i = 1, 2$. Then, since $\beta_1(c) + \beta_2(c) \leq \tau$, we also get $\alpha_1(c) + \alpha_2(c) \leq \tau$, and so $\alpha_1 + \alpha_2 \vDash I$.

Case 2: I is $(c \geq \tau)$, for some $c \in C$ and some $\tau \in \mathbb{Q}_{\geq}$.

First, let $\beta_1 + \beta_2 \vDash I$. Then $\beta_1(c) + \beta_2(c) \geq \tau$ and so $\alpha_1(c) + \alpha_2(c) \geq \tau$, since $\alpha_i(c) \geq \beta_i(c)$, $i = 1, 2$. Then, $\alpha_1 + \alpha_2 \vDash I$.

For the converse, assume $\alpha_1(c) + \alpha_2(c) \geq \tau$. If $\alpha_1(c) \geq L + 1$, then $\beta_1(c) = \lfloor L \rfloor + \lceil \alpha_1(c) \rceil = L + \lceil \alpha_1(c) \rceil$, since L is an integer. Thus, $\beta_1(c) \geq \tau$, since $L > \tau$ from the hypothesis. Hence $\beta_1(c) + \beta_2(c) \geq \tau$ and so $\beta_1 + \beta_2 \vDash I$.

Similarly, when $\alpha_2(c) \geq L + 1$ the result also holds.

Now let $\alpha_i(c) < L + 1 = \lfloor L \rfloor + 1$ for $i = 1, 2$. From Fact 12, we get $\beta_i(c) = \alpha_i(c)$ and so $\beta_1(c) + \beta_2(c) \geq \tau$, and again $\beta_1 + \beta_2 \vDash I$.

Case 3: I is $\neg(c \leq \tau)$, for some $c \in C$ and some $\tau \in \mathbb{Q}_{\geq}$. Equivalently, we have $\alpha_1 + \alpha_2 \vDash (c > \tau)$. We proceed as in Case 2.

Case 4: I is $\neg(c \geq \tau)$, for some $c \in C$ and some $\tau \in \mathbb{Q}_{\geq}$. Equivalently, we have $\alpha_1 + \alpha_2 \vDash (c < \tau)$. We proceed as in Case 1.

Let $n \geq 0$ be the number of propositional connectives occurring in I. We proceed by induction on n.

BASIS: $n = 0$. The result holds by Cases 1 and 2.

INDUCTION STEP: assume the result holds for all I with at most n propositional connectives, where $n \geq 0$. Now, take some $I \in \Phi_C$ with $n+1$ propositional connectives. We have two cases:

Case I-1: I is $\delta_1 \wedge \delta_2$.

Since δ_i has at most n propositional connectives, $i = 1, 2$, from the induction hypothesis we get $\alpha_1 + \alpha_2 \vDash \delta_i$ iff $\beta_1 + \beta_2 \vDash \delta_i$, for $i = 1, 2$. Then, clearly, $\alpha_1 + \alpha_2 \vDash I$ iff $\beta_1 + \beta_2 \vDash I$.

Case I-2: I is $\neg\delta$.

Then δ has $n \geq 0$ propositional connectives. When $n = 0$, δ is $(c \leq \tau)$ or $(c \geq \tau)$, and the result follows by Cases 3 and 4, respectively.

Assume now that $n \geq 1$. We have two sub-cases:

I-2A: δ is $\neg\delta_1$ and δ_1 has $n-1$ propositional connectives. Then $\neg\delta$ is equivalent to $\neg\neg\delta_1$, that is, I is equivalent to δ_1. We can use the induction hypothesis and conclude that $\alpha_1 + \alpha_2 \vDash I$ iff $\beta_1 + \beta_2 \vDash I$.

I-2B: δ is $(\delta_1 \wedge \delta_2)$, that is, I is equivalent to $(\neg\delta_1) \vee (\neg\delta_2)$. Note that δ has n propositional connectives. Then δ_i has at most $(n-1)$ propositional connectives, that is, $\neg\delta_i$ has at most n propositional connectives, for $i = 1, 2$. Using the induction hypothesis we get $\alpha_1 + \alpha_2 \vDash \neg\delta_i$ iff $\beta_1 + \beta_2 \vDash \neg\delta_i$. Thus, $\alpha_1 + \alpha_2 \vDash (\neg\delta_1) \vee (\neg\delta_2)$ iff $\alpha_1 + \alpha_2 \vDash \neg\delta_i$ for some $i \in \{1, 2\}$ iff $\beta_1 + \beta_2 \vDash \neg\delta_i$ for some $i \in \{1, 2\}$ iff $\beta_1 + \beta_2 \vDash (\neg\delta_1) \vee (\neg\delta_2)$, completing the proof. ∎

A similar results holds when overruling.

Lemma 23 *Let C be a set of variables and let $\nu \in [C \to \mathbb{Q}_{\geq}]$ and $\theta \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ be variable interpretations and let $I \in \Phi_C$ be a variable condition. If $\nu \oplus \theta \vDash I$ then $\nu_L \oplus \theta \vDash I$, when L is a positive integer greater than all constants occurring in I.*

Proof When I is `true` we are done. Now, assume that I is not `true`.

Now we treat the four cases when I is $(c \leq \tau)$, $(c \geq \tau)$, $\neg(c \leq \tau)$ or $\neg(c \geq \tau)$, where $c \in C$ and $\tau \in \mathbb{Q}_{\geq}$. If $c \notin \mathrm{dom}(\theta)$, we get $(\nu \oplus \theta)(c) = \nu(c)$ and so $\nu \vDash I$. From Lemma 22, we get $\nu_L \vDash I$. Since $(\nu_L \oplus \theta)(c) = \nu_L(c)$ we conclude that $\nu_L \oplus \theta \vDash I$. If $c \in \mathrm{dom}(\theta)$, we get $(\nu \oplus \theta)(c) = \theta(c) = (\nu_L \oplus \theta)(c)$. Then, $\nu_L \oplus \theta \vDash I$.

Now, we proceed by induction on the number $n \geq 0$ of propositional connectives occurring in I.

BASIS: $n = 0$. The result follows by the first two cases discussed above.

INDUCTION STEP: assume the result holds for all I with at most n propositional connectives, where $n \geq 0$.

Now, take some $I \in \Phi_C$ with $n+1$ propositional connectives. We have two cases:

Case I-1: I is $\delta_1 \wedge \delta_2$.

We have $\nu \oplus \theta \vDash \delta_i$, $i = 1, 2$. But δ_i has $n-1$ propositional connectives, and the induction hypothesis gives $\nu_L \oplus \theta \vDash \delta_i$, for $i = 1, 2$. Hence, $\nu_L \oplus \theta \vDash I$.

Case I-2: I is $\neg\delta$.

Then δ has $n \geq 0$ propositional connectives. When $n = 0$, δ is $(c \leq \tau)$ or $(c \geq \tau)$, and the result holds by third and forth cases discussed above.

Assume now that $n \geq 1$. We have two sub-cases:

I-2A: δ is $\neg\delta_1$ and δ_1 has $n-1$ propositional connectives. Then $\neg\delta$ is equivalent to $\neg\neg\delta_1$, that is, to δ_1. Then, from the original hypothesis, $\nu \oplus \theta \vDash \delta_1$. By the induction hypothesis, $\nu_L \oplus \theta \vDash \delta_1$, that is $\nu_L \oplus \theta \vDash \neg\neg\delta_1$. Then $\nu_L \oplus \theta \vDash I$.

I-2B: δ is $(\delta_1 \wedge \delta_2)$, that is I is equivalent to $(\neg\delta_1) \vee (\neg\delta_2)$. Hence, $\nu \oplus \theta \vDash \neg\delta_1$, or $\nu \oplus \theta \vDash \neg\delta_2$. Note that δ has n propositional connectives. Then δ_i has at most $(n-1)$ propositional connectives, that is, $\neg\delta_i$ has at most n propositional connectives, for $i = 1, 2$. Using the induction hypothesis, we have $\nu_L \oplus \theta \vDash \neg\delta_1$ or $\nu_L \oplus \theta \vDash \neg\delta_2$. So, $\nu_L \oplus \theta \vDash \neg(\delta_1 \wedge \delta_2)$, that is $\nu_L \oplus \theta \vDash I$. ∎

2.4 Adjusted values

In order to discretize a timed model, we will need to choose a proper time granularity. Then, in the discrete model, any event will take place at instants that are integer multiples of the granularity, or at adjusted time values, as such time instants will be called.

In this section, we treat the notion of adjusted values and list some of its properties.

Definition 24 *Let $g \in \mathbb{Q}_{\geq}$. Then*

1. *A value $\ell \in \mathbb{Q}_{\geq}$ is* g-adjusted *iff ℓ is an integer multiple of g.*
2. *Let C be a set of variables. A variable condition $\delta \in \Phi_C$ is* g-adjusted *iff all constants occurring in δ are g-adjusted values.*
3. *Let C be a set of variables. A variable interpretation $\nu \in [C \curvearrowright \mathbb{Q}_{\geq}]$ is* g-adjusted *iff $\nu(c)$ is a g-adjusted value, for all $c \in dom(\nu)$.* ∎

When the value g can be inferred from the context, we may write adjusted instead of g-adjusted.

Two simple facts about adjusted values follow.

Proposition 25 *Let $g = 1/k$, with k a positive integer, and let $\ell \in \mathbb{Q}_{\geq}$. Then ℓ is a g-adjusted value iff both $\lfloor \ell \rfloor$ and $\lceil \ell \rceil$ are g-adjusted values.*

Proof Assume that $\lfloor \ell \rfloor = mg$ and $\lceil \ell \rceil = ng$, where m and n are non-negative integers. Then $\ell = (m+n)g$ and so ℓ is also a g-adjusted value. Conversely, assume that $\ell = mg$ for some non-negative integer m. Then $mg = (\lfloor \ell \rfloor g)/g + \lceil \ell \rceil$ and so $\lceil \ell \rceil = (m - \lfloor \ell \rfloor/g)g = (m - k\lfloor \ell \rfloor)g$. Since $(m - k\lfloor \ell \rfloor)$ is an integer, $\lceil \ell \rceil$ is also g-adjusted. Also, clearly, $\lfloor \ell \rfloor = \ell - \lceil \ell \rceil$ and so $\lfloor \ell \rfloor$ is g-adjusted since ℓ and $\lceil \ell \rceil$ are g-adjusted. ∎

Proposition 26 *Let $g = 1/k$, with k a positive integer, and let $L \in \mathbb{Q}_{\geq}$ be a g-adjusted value. Also, let C be a set of variables and let $\nu \in [C \curvearrowright \mathbb{Q}_{\geq}]$ be a g-adjusted variable interpretation. Then ν_L is also a g-adjusted interpretation.*

Proof Let $c \in C$. From Definition 10, we get $\nu_L(c) = \nu(c)$ or $\nu_L(c) = \lfloor L \rfloor + \lceil \nu(c) \rceil$. From Proposition 25, we know that $\lfloor L \rfloor$ and $\lceil \nu(c) \rceil$ are g-adjusted. So, clearly, $\nu_L(c)$ is g-adjusted for all $c \in C$, and the result follows. ∎

Another simple fact states that displacing g-adjusted interpretations, or overruling a g-adjusted interpretation, always results in a new interpretation that is also g-adjusted.

Proposition 27 *Let C be a set of variables, let ν, $\eta \in [C \curvearrowright \mathbb{Q}_{\geq}]$ be two g-adjusted interpretations, and let ℓ be a g-adjusted value. Then $\nu + \ell$ and $\nu \oplus \eta$ are also g-adjusted interpretations.*

Proof Assume that $\nu(c) \in \mathrm{dom}(\nu)$. Then, $(\nu + \ell)(c) = \nu(c) + \ell$, which is, clearly, a g-adjusted value. Thus, by Definition 24, $\nu + \ell$ is also g-adjusted.

Now, when $c \in \mathrm{dom}(\nu \oplus \eta)$ there are two cases. If $\eta(c) \in \mathrm{dom}(\eta)$, then $(\nu \oplus \eta)(c) = \eta(c)$ is g-adjusted. When $c \in (\mathrm{dom}(\nu) - \mathrm{dom}(\eta))$ then $(\nu \oplus \eta)(c) = \nu(c)$ is also g-adjusted. Clearly, $\nu \oplus \eta$ is a g-adjusted clock interpretation. ∎

Next, we show that any set of L-bounded interpretations is finite, provided that L is properly adjusted.

Lemma 28 *Let C be a set of variables, and let $g = 1/k$ with k a positive integer. Let $R \subseteq [C \curvearrowright \mathbb{Q}_{\geq}]$ be a set of g-adjusted interpretations, and let $L \in \mathbb{Q}_{\geq}$ be a g-adjusted value. Consider the L-bounded set $R_L = \{\nu_L \mid \nu \in R\}$. Then $|R_L| \leq (k\lfloor L \rfloor + k)^{|C|}$.*

Proof Consider some $\nu_L \in R_L$ and some $c \in C$. Let $t = \nu_L(c)$. By Proposition 26, t is always g-adjusted. Moreover, $t < \lfloor L \rfloor + 1$, by Fact 13. But $\lfloor L \rfloor = ng$, for some non-negative integer n, and so $\lfloor L \rfloor + 1 = ng + (1/g)g = (n+k)g$. Hence, t can have at most $n+k$ distinct g-adjusted values (counting from zero).

We conclude that any clock $c \in C$ can be mapped to at most $n + k$ distinct g-adjusted values, by ν_L bounded interpretations. Therefore, there are at most $(n + k)^{|C|}$ distinct ν_L interpretations. Since $n = \lfloor L \rfloor / g = k\lfloor L \rfloor$, the result follows. ∎

Had we used a distinct g-adjusted bound L_c for each variable c, the lemma would yield the bound $|R_L| \leq \prod_{c \in C} (k\lfloor L_c \rfloor + k)$.

3 TIOA with context variables

In this section we extend the TIOA model, now introducing both the notion of context variables and the notion of input and output parameters [18], thereby obtaining a Timed I/O Context Automata (TIOCA). Context variables play the role of common variables in ordinary programming. They can enter into expressions and can be assigned to. Input parameters will be used to read in values passed down by the environment. Similarly, output parameters can have their values computed and passed back to the environment.

3.1 An example

A TIOA that captures the behavior of a simple multimedia system is presented in [7]. The protocol operates by receiving image frames followed by their respective sound tracks, the latter being supposed to arrive within two time units after the preceding image frame. After the arrival of a sound track, the protocol must send an acknowledgment in no more than three time units after the reception of the image frame and no more than two time units after receiving the sound track. A reset message moves the system back to the initial state, so that it can await for the next image frame. Such a message must be issued no more than three time units after the image frame arrives and no more than two time units after the corresponding sound track arrives. When the input sound track does not follow the image frame within two time units, the system times out, issues an error message and goes back to the initial state.

We extended the multimedia protocol to capture the same behavior, but now allowing for output parameters and context variables, as depicted in Figure 1. The extended model

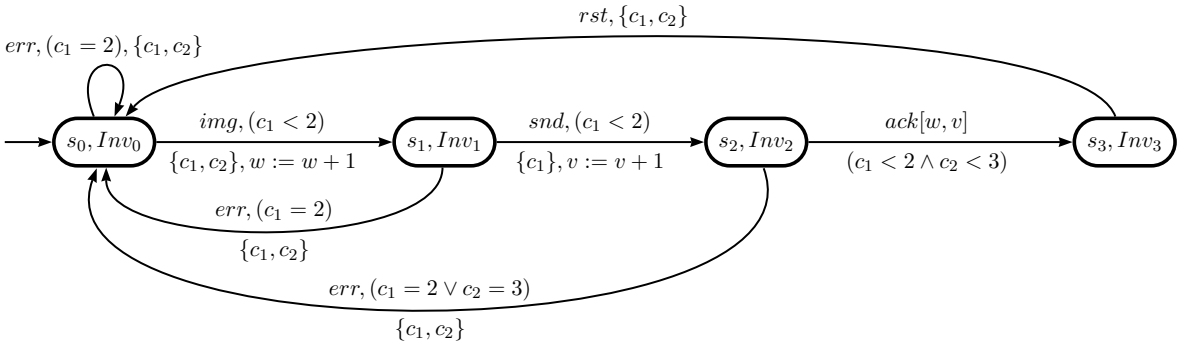

Figure 1: A TIOCA model for a multimedia protocol.

has four states, namely s_0, s_1, s_2 and s_3. The set of clocks is $C = \{c_1, c_2\}$, and the state invariants are shown in Table 1. The set of input actions is $X = \{img, snd, rst\}$, and the set of output actions is $Y = \{err, ack\}$. Recall that $\Sigma = X \cup Y$ is the set of action symbols. The intended meaning for the input symbols *img* and *snd* is the arrival of an image frame and of a sound track, respectively. The output symbol *err* signals an error, and the output symbol *ack* sends an acknowledgement signal back to the environment.

Inv_0	$c_1 \leq 2$	‖	Inv_1	$c_1 \leq 2$
Inv_2	$c_1 \leq 2 \wedge c_2 \leq 3$	‖	Inv_3	$c_1 \leq 2 \wedge c_2 \leq 3$

Table 1: State invariants

There are seven transitions, indicated by the arrows in the diagram. At each transition, we first indicate the corresponding action symbol. The action symbol may be followed by a list of guards, given within parentheses. The form of the guards pertinent to each type of transition will be detailed shortly. When there are no guards, the list is simply omitted. In the example, at the transition from s_1 to s_2 there is a clock guard in the form $(c_1 < 2)$. At the transition from s_3 to s_0 there are no guards. Next, after the list of guards, the set of clocks to be reset is given between braces. All resets move the corresponding clocks back to zero, with the other clocks remaining unchanged. If there are no clocks to reset, the clock reset list is omitted. In the example, at the transition from s_1 to s_2, only clock c_1 is reset to zero. There are no clock resets in the transition from s_2 to s_3. The last item at each transition indicates the corresponding context variable transformation. In the example, the set of context variables is $V = \{w, v\}$. As we can see from the diagram, at the transition from s_0 to s_1, the context variable transformation is the expression $w := w + 1$. The intended meaning of such an expression, besides the assignment, is that the other context variable, v, remains unchanged when this transition is taken. When all context variable transformations are omitted, as in the transition from s_2 to s_0, it should be understood that all variables remain unchanged when that transition is taken.

In this simple example, there are no input parameters. There are four transitions associated with some output action symbol in $Y = \{err, ack\}$. The transitions associated with err send no values back to the environment. In the transition on ack, from s_2 to s_3, the notation $ack[w, v]$ indicates that the current values of the variables w and v are passed back to the environment. In the sequel, we give more details about the use of input parameters and values passed back to the environment when output transitions are taken.

3.2 The extended TIOA

In this section we present a precise definition of the new extended TIOA model.

Consider a TIOA as defined in [3]. In addition, we will need a finite set R of input parameters, or parameters for short. Each input action symbol will have a particular set of parameters associated to it. We denote by $R_x \subseteq R$ the set of parameters associated with input action symbol x, for all $x \in X$. Each parameter will have a value from $\mathbb{Q}_{\geq}$ associated to it. From Definition 5 we obtain the sets of valuations over R_x, namely $[R_x \curvearrowright \mathbb{Q}_{\geq}]$ and $[R_x \to \mathbb{Q}_{\geq}]$. From Definitions 6 and 7, respectively, get the set of R_x conditions, Φ_x, and the notion of a valuation $\rho \in [R_x \curvearrowright \mathbb{Q}_{\geq}]$ satisfying a condition, $\delta \in \Phi_x$, written $\rho \vDash \delta$. Returning to the example at Section 3.1, we have $X = \{img, snd, rst\}$, $Y = \{err, ack\}$ and $R = \emptyset$.

Proceeding, we will also need a set V of context variables, all of which take values in $\mathbb{Q}_{\geq}$. Again, from Definitions 5, 6 and 7 we obtain the sets of context variable valuations

$[V \rightharpoonup \mathbb{Q}_{\geq}]$ and $[V \to \mathbb{Q}_{\geq}]$, together with the set of context variable conditions Φ_V and the notion of satisfiability $\lambda \vDash \delta$, for a context variable valuation λ and a context variable condition δ. In the example, the set of context variables is $V = \{w, v\}$.

Upon taking a discrete transition on an input action symbol x, the new model can also redefine the context variable values according to an expression involving the input parameter values associated to x and the current values of the context variables. In order to capture this effect, we associate with each discrete transition on an input action symbol x a map that takes a pair (ρ, λ), where $\rho \in [R_x \to \mathbb{Q}_{\geq}]$ and $\lambda \in [V \to \mathbb{Q}_{\geq}]$, and returns another context variable valuation in $[V \to \mathbb{Q}_{\geq}]$. We collect the set of all such mappings thus

$$F_x = \Big\{ \kappa \,|\, \kappa : [R_x \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}] \to [V \to \mathbb{Q}_{\geq}] \Big\}.$$

In the example, since there are no input parameters associated to img, we get $R_{img} = \emptyset$. Since $V = \{w, v\}$, we obtain

$$F_{img} = \Big\{ \kappa \,|\, \kappa : \{\emptyset\} \times [\{w, v\} \to \mathbb{Q}_{\geq}] \to [\{w, v\} \to \mathbb{Q}_{\geq}] \Big\}.$$

Similarly, $F_{snd} = F_{rst} = F_{img}$.

Putting it all together, we say that a discrete transition over an input action symbol x moves the machine from a state s to a state r, provided that certain guard conditions over clocks, input parameters and context variables are satisfied. Besides, upon moving from state s to state r, the machine can reset some clock values and redefine some context variable values. The new clock values are constants specified directly in the transition. The new context variable values are computed from the input parameter values and from the current values of the context variables. We can express these conditions by saying that a discrete transition over an input action symbol x is a member of the set

$$S \times \{x\} \times \Phi_C \times \Phi_x \times \Phi_V \times [C \rightharpoonup \mathbb{Q}_{\geq}] \times F_x \times S.$$

As an illustration, take the transition from s_0 to s_1, over the input action symbol img in Figure 1. The clock guard is $(c_1 < 2)$ and the input action symbol guard is `true`, which is not listed. The context variable guard is also not listed and so it is also taken as `true`. The set $\{c_1, c_2\}$ says that both clocks are reset to zero when the transition is taken. Finally, the new value of the context variable w is defined by the expression $w := w + 1$. More precisely, an appropriate mapping $\kappa \in F_{img}$ for this transition has the form

$$\kappa : \{\emptyset\} \times \big[\{w, v\} \to \mathbb{Q}_{\geq}\big] \to \big[\{w, v\} \to \mathbb{Q}_{\geq}\big],$$

where $\kappa(\emptyset, \lambda)$ is such that $\kappa(\emptyset, \lambda)(w) = \lambda(w) + 1$ and $\kappa(\emptyset, \lambda)(v) = \lambda(v)$.

We have an analogous situation with discrete transitions over output action symbols. The only difference is that, now, some context variables have their values returned back to the environment. In general, if the current context variable valuation is λ, the machine will output a partial valuation given by $\xi(\lambda)$, where ξ is a mapping specified in the transition. For that, we need a set of such mappings, thus

$$H_y^P = \big\{ \xi \,|\, \xi \in [V \to \mathbb{Q}_{\geq}] \to [V \rightharpoonup \mathbb{Q}_{\geq}] \big\},$$

where y is the output symbol. In the example, the transition from s_2 to s_3 specifies the output action symbol as $ack[w, v]$ indicating that the values of the context variables w and v will be returned unmodified to the environment. In this case, the output mapping ξ would be just the identity, that is, $\xi(\lambda) = \lambda$, for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$. In another transition, from s_1 to s_0, the output action symbol is written just as err, and we do not pass values to the environment. In this case, $\xi(\lambda) = \emptyset$, for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$. Also, when an output transition is taken, the values of the context variables may be modified, given their current values. We model this by specifying a mapping χ that takes a total context variable valuation and returns another total context variable valuation. That is, we specify a mapping from the set

$$H_y^T = \big\{\chi \,|\, \chi \in [V \to \mathbb{Q}_{\geq}] \to [V \to \mathbb{Q}_{\geq}]\big\},$$

where y is the output symbol. In the example, all output transitions do not modify the values of any context variable. So, in this case, we always have $\chi(\lambda) = \lambda$, for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$, at all output transitions.

Putting it all together, we say that on a discrete transition over an output action symbol y the machine moves from a state s to a state r, provided that certain clock and context variable conditions are satisfied. Upon taking the transition, clocks may be reset, context variable values are returned and may have their values adjusted. The clock reset values are constants specified directly at the transition. The values to be returned and the new context variable values are specified by choosing specific transformation mappings. That is, a discrete transition over an output action symbol y is a member of

$$S \times \{y\} \times \Phi_C \times \Phi_V \times [C \curvearrowright \mathbb{Q}_{\geq}] \times H_y^P \times H_y^T \times S.$$

We can now define the new timed I/O automaton.

Definition 29 *A Timed Input/Output Context Automaton (TIOCA) is given by a tuple* $(S, s_0, \Sigma, C, \nu_0, Inv, R, V, \lambda_0, T_X, T_Y)$*, where* S*,* $s_0 \in S$*,* $\Sigma = X \cup Y$*,* C*,* $\nu_0 \in [C \to \mathbb{Q}_{\geq}]$*, and* Inv *are as given in the TIOA definition [3, 15]. The set of parameters is* R*, the set of context variables is* V *and* $\lambda_0 \in [V \to \mathbb{Q}_{\geq}]$ *is the initial context variable valuation, where* $\lambda_0(v) = 0$ *for all* $v \in V$*. The set of transitions over input actions satisfies*

$$T_X \subseteq \bigcup_{x \in X} (S \times \{x\} \times \Phi_C \times \Phi_x \times \Phi_V \times [C \curvearrowright \mathbb{Q}_{\geq}] \times F_x \times S),$$

where F_x *is the set of context update function associated to* x*, given by*

$$F_x = \{\kappa \,|\, \kappa : [R_x \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}] \to [V \to \mathbb{Q}_{\geq}]\},$$

and $R_x \subseteq R$ *is the set of input parameters associated to* x*. Finally, the set of transitions over output action symbols satisfies*

$$T_Y \subseteq \bigcup_{y \in Y} (S \times \{y\} \times \Phi_C \times \Phi_V \times [C \curvearrowright \mathbb{Q}_{\geq}] \times H_y^P \times H_y^T \times S),$$

where

$$H_y^P = \{\xi \,|\, \xi \in [V \to \mathbb{Q}_{\geq}] \to [V \curvearrowright \mathbb{Q}_{\geq}]\} \quad \textit{and} \quad H_y^T = \big\{\chi \,|\, \chi \in [V \to \mathbb{Q}_{\geq}] \to [V \to \mathbb{Q}_{\geq}]\big\},$$

for all $y \in Y$*.* ∎

A transition in T_X is a tuple $(s, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r)$ saying that the machine can move from state s to state r over the input symbol x provided that the guards δ_1, δ_2, and δ_3, over clock variables, input parameters and context variables, respectively, are all enabled. Further, upon moving to state r, the mapping $\theta \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ indicates which clocks are reset and to which values. Finally, $\kappa \in F_x$ denotes the context variable update function, mapping input parameters valuations and context variable valuations into new context variable valuations. In the same vein, a transition in T_Y is a tuple $(s, y, \delta_1, \delta_2, \theta, \xi, \chi, r)$ specifying that the machine can move from state s to state r over the output action symbol y, provided that the guards δ_1 and δ_2, over clock variables and context variables, respectively, are enabled. Further, upon moving to state r the mapping $\theta \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ gives which clocks are reset and to which values. Lastly, $\xi \in H_y^P$ gives a partial function specifying values to be passed to the environment, and $\chi \in H_y^T$ returns the new values of the context variables, in the form of a total function.

Next, we want to specify successive moves of a TIOCA. For that, we need the notions of parameterized inputs and outputs.

Definition 30 *Let M be a TIOCA and let $\Upsilon \subseteq \Sigma$. We collect in Υ_M the set of all parameterized actions of M over Υ, that is,*

$$\Upsilon_M = \big\{(x, \rho) \,|\, x \in X \cap \Upsilon,\, \rho \in [R_x \to \mathbb{Q}_{\geq}]\big\} \bigcap \big\{(y, \rho) \,|\, y \in Y \cap \Upsilon,\, \rho \in [V \to \mathbb{Q}_{\geq}]\big\}.$$

∎

In particular, Ψ_X and Ψ_Y are the sets of parameterized inputs and outputs, respectively, of M.

A configuration for a TIOCA M is a triple (s, ν, λ), where $s \in S$ is a state, $\nu \in [C \to \mathbb{Q}_{\geq}]$ is a clock interpretation and $\lambda \in [V \to \mathbb{Q}_{\geq}]$ is a context variable valuation. The initial configuration is (s_0, ν_0, λ_0), where s_0 is the initial state of M, ν_0 is the initial clock interpretation of M and λ_0 is the initial context variable valuation of M. As before, let $\Gamma_M \subseteq S \times [C \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}]$ be the set of all configurations of M.

We can now define the semantics of a TIOCA.

Definition 31 *Let M be a TIOCA and let $\gamma_i = (s_i, \nu_i, \lambda_i) \in \Gamma_M$, $i = 1, 2$, be two configurations of M.*

1. *Let $\tau \in \mathbb{Q}_{\geq}$ be a time delay. Then there exists a* continuous move *from γ_1 to γ_2 over τ, denoted by $\gamma_1 \xrightarrow[\tau]{} \gamma_2$, if and only if: (i) $s_1 = s_2$; (ii) $\nu_2 = \nu_1 + \tau$; (iii) $\nu_1 + \eta \vDash Inv(s_1)$ for all η, $0 < \eta \leq \tau$; and (iv) $\lambda_2 = \lambda_1$. When τ is positive, we have a* non-trivial continuous move.

2. *Let $(x, \rho) \in X_M$ be a parameterized input. Then there exists a* discrete move *from γ_1 to γ_2 over (x, ρ), if and only if there exists a transition $(s_1, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, s_2) \in T_X$ such that: (i) $\nu_1 \vDash \delta_1$, $\rho \vDash \delta_2$ and $\lambda_1 \vDash \delta_3$; (ii) $\nu_2 = \nu_1 \oplus \theta$ and $\nu_2 \vDash Inv(s_2)$; and (iii) $\lambda_2 = \kappa(\rho, \lambda_1)$. We denote such a discrete move by $\gamma_1 \overset{(x,\rho)}{\to} \gamma_2$.*

3. *Let $(y, \mu) \in Y_M$ be a parameterized output. Then there exists a* discrete move *from γ_1 to γ_2 over (y, μ), if and only if there exists a transition $(s_1, y, \delta_1, \delta_2, \theta, \xi, \chi, s_2) \in T_Y$ such that: (i) $\nu_1 \vDash \delta_1$ and $\lambda_1 \vDash \delta_2$; (ii) $\nu_2 = \nu_1 \oplus \theta$ and $\nu_2 \vDash Inv(s_2)$; and (iii) $\mu = \xi(\lambda_1)$ and $\lambda_2 = \chi(\lambda_1)$. We denote such a discrete move by $\gamma_1 \overset{(y,\mu)}{\longrightarrow} \gamma_2$.* ∎

As usual, some of the decorations over and under the relation symbol $\longrightarrow$ may be dropped if they are clear from the context.

By concatenating parameterized actions and time delays we obtain parameterized words.

Definition 32 *Let M be a TIOCA and let $\Upsilon \subseteq \Sigma$. A parameterized timed I/O sequence, or simply parameterized timed word, for M over Υ is any sequence $\psi = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$, where $n \geq 0$ and each σ_i is either in Υ_M or is a time delay in $\mathbb{Q}_{\geq}$.* ∎

The set of all parameterized timed words for M for Υ will be denoted by $\Psi_{\Upsilon,M}$, with decorations omitted as appropriate. In particular, Ψ_X and Ψ_Y denote, respectively, the set of all parameterized input and parameterized output sequences for M. The empty parameterized timed word will be denoted by ε. The concatenation of parameterized timed words follows the standard definition for concatenation of timed words.

The movements of a TIOCA can now be specified.

Definition 33 *Let M be a TIOCA. The* movement relation *of M, $\vdash_M$, is a binary relation over $\Psi_M \times \Gamma_M$ such that $(\langle \sigma \rangle \cdot \psi, \gamma_1) \vdash_M (\psi, \gamma_2)$, with $\sigma \in (X_M \cup Y_M \cup \mathbb{Q}_{\geq})$ and $\psi \in \Psi_M$, if and only if either (i) $\sigma \in \mathbb{Q}_{\geq}$ and $\gamma_1 \underset{\sigma}{\rightarrow} \gamma_2$; or (ii) $\sigma \in (X_M \cup Y_M)$ with $\gamma_1 \overset{\sigma}{\rightarrow} \gamma_2$.* ∎

The k-th power of $\vdash_M$ will be indicated by $\overset{k}{\vdash}_M$, $k \geq 0$, and its reflexive transitive closure by $\overset{\star}{\vdash}_M$. Further, when $(\psi, \gamma) \vdash_M (\varepsilon, \varphi)$ we also write $\gamma \Vdash_M^{\psi} \varphi$, or $\gamma \Vdash_M \varphi$ when the particular parameterized timed sequence ψ is not relevant. When $\gamma \Vdash_M^{\psi} \varphi$ we say that we have a *run starting* at γ and *ending* at φ over ψ. Moreover, if γ is the initial configuration of the TIOCA, then we have an *execution ending* at φ. A configuration γ is *reachable* if there is an execution ending at γ. Also, a state s is *reachable* if there is a reachable configuration (s, ν, λ), for some clock interpretation ν and some context variable valuation λ.

4 TIOCA discretization

The same strategy used to discretize TIOA can be followed in order to obtain TIOCA discretizations. Clearly, variable values will have to be discretized too.

4.1 Context variable valuations

In order to discretize the context variable valuations and input action parameter valuations we impose an upper bound on their values.

We will represent by $K \in \mathbb{Q}_{\geq}$ the limiting boundary for context variable values. Recall Definition 10. Let $\lambda \in [V \rightharpoonup \mathbb{Q}_{\geq}]$. We will also refer to λ_K as the context variable valuation

λ bounded by K, or as a K-bounded context variable valuation. Note that we could have specified different K bounds for each context variable. However, in the sequel we will assume the same bound K for all context variable values in order to keep the notation under control. It is not hard to generalize the upcoming results when a possibly distinct parameter K_v is specified for each context variable v. Similarly, for each $\rho \in [R_x \rightharpoonup \mathbb{Q}_{\geq}]$, with $x \in X$, the corresponding K-bounded parameter valuation will be denoted by ρ_K.

We will assume that the mappings in H_y^P and in H_y^T are of a special linear form that we now make precise.

Definition 34 *Let V be a set of variables and let ξ be a mapping from $[V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$. We say that ξ is* fully linear *iff for all $v \in V$ there is an integer constant b, and a set of integer constants $\{a_w \,|\, w \in V\}$ such that, for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$ we have*

$$\xi(\lambda)(v) = \Big[\sum_{w \in V} a_w \lambda(w) \Big] + b,$$

whenever v is in the domain of $\xi(\lambda)$.

In a similar way, we will require that all mappings $\kappa \in F_x$ satisfy a bi-linearity condition.

Definition 35 *Let V be a set of variables and R a set of parameters. Let κ be a mapping from $[R \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$. We say that κ is* fully bi-linear *iff there are fully linear mappings ξ_1, from $[V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$, and ξ_2, from $[R \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$, such that $\kappa(\rho, \lambda) = \xi_1(\lambda) + \xi_2(\rho)$, for all $\rho \in [R \to \mathbb{Q}_{\geq}]$ and all $\lambda \in [V \to \mathbb{Q}_{\geq}]$.* ∎

Next we present some facts that will support proofs of the relationship between a TIOCA and the corresponding grid automaton.

Fact 36 *Let V be a set of variables and let K be a positive integer. Let ξ be a fully linear mapping from $[V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$. Then $\xi_K(\lambda) = \xi_K(\lambda_K)$, for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$.*

Proof It suffices to show that $\xi_K(\lambda)(v) = \xi_K(\lambda_K)(v)$, for all v in the domain of $\xi(\lambda)$.

Since ξ is fully linear, Definition 34 gives integer constants b and a_w, for all $w \in V$, such that

$$\xi_K(\lambda)(v) = (\xi(\lambda)(v))_K = \left[\sum_{w \in V} a_w \lambda(w) + b \right]_K = \left[\Big[\sum_{w \in V} a_w \lambda(w) \Big]_K + b \right]_K,$$

where we used Proposition 18. Now, using Proposition 19, we get

$$\left[\sum_{w \in V} a_w \lambda(w) \right]_K = \left[\sum_{w \in V} a_w \lambda_K(w) \right]_K.$$

Then using Proposition 18 again, we have

$$\xi_K(\lambda)(v) = \left[\Big[\sum_{w \in V} a_w \lambda_K(w) \Big]_K + b \right]_K = \left[\sum_{w \in V} a_w \lambda_K(w) + b \right]_K = (\xi(\lambda_K)(v))_K = \xi_K(\lambda_K)(v),$$

as desired. ∎

A similar result holds for bi-linear mappings.

Fact 37 *Let V be a set of variables, R a set of parameters and K a positive integer. Let κ be a fully bi-linear mapping from $[R \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$. Then $\kappa_K(\rho, \lambda) = \kappa_K(\rho_K, \lambda_K)$, for all $\rho \in [R \to \mathbb{Q}_{\geq}]$ and all $\lambda \in [V \to \mathbb{Q}_{\geq}]$.*

Proof Because κ is bi-linear, Definition 35 gives two mappings, namely, ξ from $[V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$, and χ from $[R \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$, such that $\kappa(\rho, \lambda) = \xi(\lambda) + \chi(\rho)$.

Take $\rho \in [R \to \mathbb{Q}_{\geq}]$ and $\lambda \in [V \to \mathbb{Q}_{\geq}]$. It suffices to show that $\kappa_K(\rho, \lambda)(v) = \kappa_K(\rho_K, \lambda_K)(v)$, for all v in the domain of $\kappa_K(\rho, \lambda)$.

Let v be in the domain of $\kappa(\rho, \lambda)$. Then $\kappa(\rho, \lambda)(v) = \xi(\lambda)(v) + \chi(\rho)(v)$. And, using Proposition 18, we have

$$\kappa_K(\rho, \lambda)(v) = \big[\kappa(\rho, \lambda)(v)\big]_K = \big[\chi(\rho)(v) + \xi(\lambda)(v)\big]_K = \Big[\big[\chi(\rho)(v)\big]_K + \big[\xi(\lambda)(v)\big]_K\Big]_K.$$

Now, using Fact 36, we may write $\big[\xi(\lambda)(v)\big]_K = \xi_K(\lambda)(v) = \xi_K(\lambda_K)(v)$. By the same reasoning, $\big[\chi(\rho)(v)\big]_K = \chi_K(\rho)(v) = \chi_K(\rho_K)(v)$. Then, $\kappa_K(\rho, \lambda)(v) = \big[\chi_K(\rho_K)(v) + \xi_K(\lambda_K)(v)\big]_K$ and, using Proposition 18 again,

$$\kappa_K(\rho, \lambda)(v) = \big[\chi(\rho_K)(v) + \xi(\lambda_K)(v)\big]_K = \big[\kappa(\rho_K, \lambda_K)(v)\big]_K = \kappa_K(\rho_K, \lambda_K)(v),$$

as desired. ∎

For the sake of shortening statements, we stipulate that the following assumption is in order from now on.

Assumption 38 *Let M be a TIOCA. All mappings in F_x are fully bi-linear mappings, for all $x \in X$. Also, all mappings in $H_y^P \cup H_y^T$ are fully linear mappings, for all $y \in Y$.* ∎

4.2 Grid automata and TIOCA

Before we can discuss the discretization process over TIOCA, we need to establish a grid unit for context variable values. Recall Definition 24.

Assumption 39 *From now on , we will assume a clock grid value in the form $g = 1/k$ and a context variable grid value in the form $h = 1/\ell$, with k and ℓ positive integers. Further, we will assume that L, $K \in \mathbb{Q}_{\geq}$ are the clock value boundary and the context variable value boundary, respectively.* ∎

From Proposition 26, it follows immediately that λ_K is h-adjusted when λ is h-adjusted.

By a reasoning entirely analogous to the proof of Lemma 28, we get that $(\ell\lfloor K \rfloor + \ell)^{|V|}$ is an upper bound on the size of any set of K-bounded context variable interpretations. Note that, had we used distinct h-adjusted bounds K_v for each context variable v, then $\prod_{v \in V} (\ell\lfloor K_v \rfloor + \ell)$ would be a proper upper bound. Similar results can be also derived for any set of input parameter valuations.

Now, we can say when a TIOCA is adjusted. We simply require that all constants occurring in the definition of the TIOCA be properly adjusted.

Definition 40 *Let $g = 1/k$ and $h = 1/\ell$, with k and ℓ positive integers. A TIOCA M is* $[g,h]$-adjusted *iff for all input transitions $(s, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r) \in T_X$ and all output transitions $(s, y, \delta_1, \delta_3, \theta, \xi, \chi, r) \in T_Y$ we have that δ_1 is a g-adjusted clock condition, δ_2 is a h-adjusted input parameter condition, δ_3 is a h-adjusted context variable condition and θ is a g-adjusted clock interpretation. Moreover, for all states $s \in S$, we require that $Inv(s)$ be a g-adjusted clock condition.* ∎

The next simple fact states that applying context update function over h-adjusted parameter valuation and h-adjusted context variable valuation, always results in a new context variable valuation that is also h-adjusted.

Proposition 41 *Let M be a $[g,h]$-adjusted TIOCA. Then,*

1. *$\xi(\lambda)$ is also a h-adjusted valuation, for any $\xi \in H_y^P \cup H_y^T$, $\lambda \in [V \to \mathbb{Q}_{\geq}]$ and $y \in Y$.*
2. *$\kappa(\rho, \lambda)$ is also a h-adjusted valuation, for any $\kappa \in F_x$, $\rho \in [R_x \to \mathbb{Q}_{\geq}]$, $\lambda \in [V \to \mathbb{Q}_{\geq}]$, $x \in X$.*

Proof Consider the first statement. Take $\lambda \in [V \to \mathbb{Q}_{\geq}]$ and $v \in V$, such that v is in the domain of $\xi(\lambda)$. It suffices to show that $\xi(\lambda)(v)$ is h-adjusted. By Assumption 38, we know that ξ is fully linear and so, by Definition 34, we can write $\xi(\lambda)(v) = \left[\sum_{w \in V} a_w \lambda(w)\right] + b$, where a_w and b are integer constants. Since λ is h-adjusted and a_w is an integer, we get that each term $a_w \lambda(w)$ is also h-adjusted. Then, clearly, the sum $\sum_{w \in V} a_w \lambda(w)$ is also h-adjusted. Finally, since b is an integer, and so is also h-adjusted, the final sum is h-adjusted too.

Now consider the second statement. Take $\rho \in [R_x \to \mathbb{Q}_{\geq}]$, for some $x \in X$, and $\lambda \in [V \to \mathbb{Q}_{\geq}]$. It suffices to show that $\kappa(\rho, \lambda)(v)$ is h-adjusted, for all v is in the domain of $\kappa(\rho, \lambda)$. Again, by Assumption 38 we know that κ is fully bi-linear. Then, from Definition 35 we get two fully linear mappings, ξ and χ, such that $\kappa(\rho, \lambda)(v) = \xi(\rho)(v) + \chi(\lambda)(v)$. By the reasoning just given above, we get that $\xi(\rho)(v)$ and $\chi(\lambda)(v)$ are both h-adjusted. Then, clearly, $\kappa(\rho, \lambda)(v)$ is also h-adjusted. ∎

A parameterized timed word is adjusted if all its time instants and all its parameter values are properly adjusted.

Definition 42 *Let Σ be an alphabet and R a set of parameters. A $[g,h]$-adjusted parameterized timed word over Σ and R is a parameterized timed word $\langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$ such that for all i, $1 \leq i \leq n$, if $\sigma_i \in \mathbb{Q}_{\geq}$ then σ_i is a g-adjusted value, and if $\sigma_i = (z_i, \rho_i)$ is a parameterized input or output, then ρ_i is a h-adjusted valuation.* ∎

The set of all $[g,h]$-adjusted parameterized timed words over Σ and R will be denoted by $\Psi_{[g,h],\Sigma,R}$. As before, we may drop subscripts if there is no reason for confusion.

A run over a $[g,h]$-adjusted timed word implies the $[g,h]$-reachability of the terminal configuration.

Definition 43 *Let M be a $[g,h]$-adjusted TIOCA. Also let $s \in S$, $\nu \in [C \to \mathbb{Q}_{\geq}]$ and $\lambda \in [V \to \mathbb{Q}_{\geq}]$. We say that the configuration (s,ν,λ) is $[g,h]$-reachable in M iff there is a $[g,h]$-adjusted parameterized timed word $\psi \in \Psi_{[g,h]}$, such that $(s_0,\nu_0,\lambda_0) \Vdash_M^{\psi} (s,\nu,\lambda)$.* ∎

Now we can define the grid automaton corresponding to a TIOCA.

Definition 44 *Let M be a $[g,h]$-adjusted TIOCA and let L be g-adjusted and K be h-adjusted values in $\mathbb{Q}_{\geq}$. Then the* $[L,K,g,h]$-TIOCA grid automaton*, or simply* $[L,K,g,h]$-TIOCA grid*, associated with M is the labelled transition system constructed by Algorithm 1.* ∎

Again, we may drop the qualifications L, K, g or h, when no confusion can arise.

The idea used in the algorithm is similar to that previously used in [3]. Except that, in this case, we have context variables and their interpretations and, further, transitions over input and output symbols have different behaviors.

Again, we show the result that Algorithm 1 always terminates. We also establish a bound on the number of states in the resulting labelled transition system.

Lemma 45 *Consider the procedure depicted as Algorithm 1. Then, it always halts with $|S_G| \leq |S| \times \big(k\lfloor L \rfloor + k\big)^{|C|} \times \big(\ell\lfloor K \rfloor + \ell\big)^{|V|}$.*

Proof Similar to the proof in [3]. First note that, by construction, Algorithm 1 only adds to RS elements in the form (r,η,μ), where s is a state, η is an L-bounded clock interpretation and μ is K-bounded context variable valuation. Moreover, by Proposition 27 and Proposition 26 we know that the interpretations constructed at lines 10, 22 and 28 named η are also g-adjusted. Also, the interpretation constructed at line 13 named μ is also h-adjusted. Hence, by Lemma 28, the number of distinct g-adjusted clock interpretations is bounded by $\big(k\lfloor L \rfloor + k\big)^{|C|}$. By the same reasoning, the number of h-adjusted variable valuations is at most $\big(\ell\lfloor K \rfloor + \ell\big)^{|V|}$. Since the number of states in M is $|S|$, the result follows. ∎

Again, had we specified a possibly different bound L_c for the value of each clock c in C and a possibly distinct bound K_v for the value of each context variable $v \in V$, the number of states in the grid automaton would be bounded by $|S| \times \prod_{c \in C} (k\lfloor L_c \rfloor + k) \times \prod_{v \in V} (\ell\lfloor K_v \rfloor + \ell)$. And the latter can be much smaller then $|S| \times \big(k\lfloor L \rfloor + k\big)^{|C|} \times \big(\ell\lfloor K \rfloor + \ell\big)^{|V|}$, if we take the safe values $L = \max_{c \in C}\{L_c\}$ and $K = \max_{v \in V}\{L_v\}$.

Now, we want to prove that a TIOCA and its corresponding grid display compatible behaviors, provided that the common input parameterized timed word is also $[g,h]$-adjusted.

A grid word in $\Sigma_G^{\star}$ induces a movement in the grid automaton in a usual way. We must remember, however, that a grid word carry the symbol g, as well as h-adjusted input parameter valuations (x,ρ) and h-adjusted output parameter valuations (y,λ).

Definition 46 *Let M_G be the grid automaton corresponding to a TIOCA M. The* movement relation *of M_G, $\vdash_G$, is a binary relation over $\Sigma_G^{\star} \times S_G$ given by $(\langle\sigma\rangle\psi, s) \vdash_G (\psi, r)$ if and only if there is a transition (s,σ,r) in M_G.* ∎

Input: L, $K \in \mathbb{Q}_{\geq}$; k, ℓ positive integers, $g = 1/k$, $h = 1/\ell$; $[g,h]$-adjusted TIOCA M.
Output: Grid $M_G = (S_G, s_G, \Sigma_G, T_G)$.
let $\Psi_z^K = \{(z, \rho_K) | (z,\rho) \in \Psi_{\{z\}}\}$, $z \in \Sigma$;
begin
  let $s_G = (s, \nu_0, \lambda_0)$; $T_G \leftarrow \emptyset$; $HS \leftarrow \emptyset$; $RS \leftarrow s_G$;
  while $RS \setminus HS \neq \emptyset$ **do**
    get a state (s,ν,λ) from $RS \setminus HS$; **move** (s,ν,λ) from RS to HS;
    foreach $(s, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r) \in T_X$ **do**
      if $\nu \vDash \delta_1$ *and* $\lambda \vDash \delta_3$ *and* $\nu \oplus \theta \vDash Inv(r)$ **then**
        let $\eta = (\nu \oplus \theta)_L$;
        foreach (x, ρ) **in** I_x^K **do**
          if $\rho \vDash \delta_2$ **then**
            let $\mu = \kappa_K(\rho, \lambda)$
            add the transition $((r,\nu,\lambda),(x,\rho),(r,\eta,\mu))$ to T_G;
            add the state (r,η,μ) to RS, if $(r,\eta,\mu) \notin HS$;
          end
        end
      end
    end
    foreach $(s, y, \delta_1, \delta_2, \theta, \xi, \chi, r) \in T_Y$ **do**
      if $\nu \vDash \delta_1$ *and* $\lambda \vDash \delta_2$ *and* $\nu \oplus \theta \vDash Inv(r)$ **then**
        let $\eta = (\nu \oplus \theta)_L; \rho = \xi_K(\lambda);\ \mu = \chi_K(\lambda)$;
        add the transition $((s,\nu,\lambda),(y,\rho),(r,\eta,\mu)$ to T_G;
        add the state (r,η,μ) to RS, if $(r,\eta,\lambda) \notin HS$;
      end
    end
    if $\nu + h \vDash Inv(s)$ *for all* $0 < h \leq g$ **then**
      let $\eta = (\nu + g)_L$;
      add the transition $((s,\nu,\lambda), g, (s,\eta,\lambda))$ to T_G;
      add the state (s,η,λ) to RS, if $(s,\eta,\lambda) \notin HS$;
    end
  end
  $S_G \leftarrow HS, \quad \Sigma_G \leftarrow \{g\} \cup \bigcup_{z \in \Sigma} I_z^K$;
  return;
end

Algorithm 1: Grid algorithm for TIOCA.

We will use the notation $\overset{k}{\underset{G}{\vdash}}$ and $\overset{\star}{\underset{G}{\vdash}}$ for the k-th power and for reflexive transitive closure of $\underset{G}{\vdash}$, respectively. Also, the simplified notation $\gamma \overset{\psi}{\underset{G}{\Vdash}} \rho$ may be used instead of

$(\psi, \gamma) \overset{\star}{\underset{G}{\vdash}} (\varepsilon, \rho)$. Further, we may write $\gamma \overset{}{\underset{G}{\Vdash}} \rho$ when the particular grid word is not relevant.

4.3 Relationship between a TIOCA and the corresponding grid

In this section we expose the relationship between the TIOCA model and its corresponding grid. We start with a technical result that will be useful later on.

Lemma 47 *Let M be a TIOCA and let M_G be the corresponding grid. Take $\nu \in [C \to \mathbb{Q}_{\geq}]$, $\lambda \in [V \to \mathbb{Q}_{\geq}]$ and $p \in S$. Let $(p, \nu_L, \lambda_K) \in S_G$, and assume that $\nu + \eta \vDash Inv(p)$ for all $0 < \eta \leq ig$, where $i \geq 0$. Then $(p, \nu_L, \lambda_K) \overset{g^i}{\underset{G}{\Vdash}} (p, \omega_L, \lambda_K)$ and $(p, \omega_L, \lambda_K) \in S_G$, with $\omega = \nu + ig$.*

Proof As in the proof of grid time movement in [3]. We write it for the sake of completeness. Define $\nu^j = \nu + jg$, for all j, $0 \leq j \leq i$. It suffices to prove that[2] $(p, \nu_L^j, \lambda_K) \in S_G$ and $(p, \nu_L, \lambda_K) \overset{g^j}{\underset{G}{\Vdash}} (p, \nu_L^j, \lambda_K)$, for all j, $0 \leq j \leq i$, since $\nu_L^i = (\nu + ig)_L = \omega_L$.
We proceed by induction on $j \geq 0$.

Basis: when $j = 0$ we get $g^j = \varepsilon$. Then, trivially, $(p, \nu_L, \lambda_K) \overset{g^j}{\underset{G}{\Vdash}} (p, \nu_L, \lambda_K)$. Also $\omega = \nu + 0g = \nu$ and so $\omega_L = \nu_L$. Then, $(p, \omega_L, \lambda_K) = (p, \nu_L, \lambda_K)$. Thus, $(p, \omega_L, \lambda_K) \in S_G$ and $(p, \nu_L, \lambda_K) \overset{g^j}{\underset{G}{\Vdash}} (p, \omega_L, \lambda_K)$, completing the basis.

Induction step: assume the result holds for some j, $0 \leq j < i$.

The induction hypothesis gives $(p, \nu_L, \lambda_K) \overset{g^j}{\underset{G}{\Vdash}} (p, \nu_L^j, \lambda_K)$ and $(p, \nu_L^j, \lambda_K) \in S_G$. Note that $(p, \nu_L^j, \lambda_K) \in S_G$ gives $(p, \nu_L^j, \lambda_K) \in HS$ at line 33 of Algorithm 1. Also, pairs are moved from RS into HS one at a time at line 7. Hence, at some iteration, (p, ν_L^j, λ_K) was chosen at line 7. We show that $\nu_L^j + \eta \vDash Inv(p)$, for all $0 < \eta \leq g$, so that line 27 of Algorithm 1 applies.

Let $0 < \eta \leq g$. We need $\nu_L^j + \eta \vDash Inv(p)$. We have $0 < jg + \eta \leq (j+1)g \leq ig$. From the hypothesis we get $\nu + (jg + \eta) \vDash Inv(p)$, that is $(\nu + jg) + \eta \vDash Inv(p)$, and so $\nu^j + \eta \vDash Inv(p)$. Using Lemma 22 we get $\nu_L^j + \eta \vDash Inv(p)$, as desired.

Thus, Algorithm 1, lines 28 – 30, will put $((p, \nu_L^j, \lambda_K), g, (p, \rho, \lambda_K))$ in T_G, where $\rho = (\nu_L^j + g)_L$. Therefore, we get $(p, \nu_L^j, \lambda_K) \overset{g}{\underset{G}{\Vdash}} (p, \rho_L, \lambda_K)$. Also, by line 30, (p, ρ_L, λ_K) will be added to RS if it is not already in HS. In any case, when the loop at line 6 terminates, we get (p, ρ_L, λ_K) in HS and, by line 33, $(p, \rho_L, \lambda_K) \in S_G$. Moreover, since $(p, \nu_L, \lambda_K) \overset{g^j}{\underset{G}{\Vdash}} (p, \nu_L^j, \lambda_K)$, we also get $(p, \nu_L, \lambda_K) \overset{g^{j+1}}{\underset{G}{\Vdash}} (p, \rho_L, \lambda_K)$. We extend the induction by showing that $\rho_L = \nu_L^{j+1}$. Since $\rho_L = ((\nu^j)_L + g)_L$, using Fact 20 we get $\rho_L = (\nu^j + g)_L = (\nu + jg + g)_L = (\nu + (j+1)g)_L = (\nu^{j+1})_L = \nu_L^{j+1}$, as desired. ∎

[2] Here, ν_L^j denotes $(\nu^j)_L$.

The following lemma states that all $[g,h]$-reachable configurations in a TIOCA are states in its corresponding grid.

Lemma 48 *M is a TIOCA and M_G its associated grid. Let L, K be positive integers greater than any constant occurring in M. If $(s,\nu,\lambda) \in S \times [C \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}]$ is $[g,h]$-reachable in M then $(s,\nu_L,\lambda_K) \in S_G$.*

Proof Using Definition 43, we know that $(s_0,\nu_0,\lambda_0) \Vdash_M^{\psi} (s,\nu,\lambda)$ for some $[g,h]$-adjusted parameterized timed word $\psi \in \Psi_{[g,h]}$. We proceed by induction on the length of ψ.

If $\psi = \varepsilon$ then $s_0 = s$, $\nu_0 = \nu$, and $\lambda_0 = \lambda$. Algorithm 1, line 4, puts (s,ν,λ) in RS. Now, the while loop at line 6, together with lines 7 and 7, will put (s_0,ν_0,λ_0) in HS. Then, by line 33, we get $(s,\nu,\lambda) \in S_G$.

Now, assume the result holds for any $[g,h]$-adjusted parameterized timed word ψ of length at most n, $n \geq 0$. Take $\varphi \in \Psi_{[g,h]}$ and $\sigma \in \Sigma \cup \mathbb{Q}_{\geq}$ with $\psi = \varphi \cdot \langle\sigma\rangle$, where φ has length n.

From $(s_0,\nu_0,\lambda_0) \Vdash_M^{\psi} (s,\nu,\lambda)$ we obtain $(s_0,\nu_0,\lambda_0) \Vdash_M^{\varphi} (r,\mu,\omega)$ and $(r,\mu,\omega) \Vdash_M^{\langle\sigma\rangle} (s,\nu,\lambda)$ for some $r \in S$, $\mu \in [C \to \mathbb{Q}_{\geq}]$ and $\omega \in [V \to \mathbb{Q}_{\geq}]$. Since ψ is $[g,h]$-adjusted, we have that σ and φ are $[g,h]$-adjusted. By the induction hypothesis, we get $(r,\mu_L,\omega_K) \in S_G$. Then, $(r,\mu_L,\omega_K) \in HS$ (line 33). Clearly, from line 4 of Algorithm 1, together with the while loop at line 6 and lines 7 and 7, we can conclude that (r,μ_L,ω_K) will be in RS. So, at some point, (r,μ_L,ω_K) will be chosen at line 7.

We have three cases:

Case 1: $\sigma = (x,\rho)$, for some $x \in X$ and some $\rho \in [R_x \to \mathbb{Q}_{\geq}]$.

Since $(r,\mu,\omega) \Vdash_M^{\langle\sigma\rangle} (s,\nu,\lambda)$ we must have in M a transition $(r,x,\delta_1,\delta_2,\delta_3,\theta,\kappa,s)$, for some $\delta_1 \in \Phi_C$, $\delta_2 \in \Phi_{R_x}$, $\delta_3 \in \Phi_V$, $\theta \in [C \curvearrowright \mathbb{Q}_{\geq}]$, and $\kappa \in F_x$, satisfying $\mu \vDash \delta_1$, $\rho \vDash \delta_2$, $\omega \vDash \delta_3$, $\lambda = \kappa(\rho,\omega)$, $\nu = \mu \oplus \theta$, and $\nu \vDash Inv(s)$.

From Lemma 23 and Lemma 22, we obtain $\mu_L \oplus \theta \vDash Inv(s)$ and $\mu_L \vDash \delta_1$, respectively. We also obtain $\omega_K \vDash \delta_3$ from Lemma 22. Since (r,μ_L,ω_K) will be chosen at line 7 of Algorithm 1, line 9 applies. Let $\alpha = (\mu_L \oplus \theta)_L$, as in line 10. Since $\rho \in [R_x \to \mathbb{Q}_{\geq}]$, we get $(x,\rho_K) \in I_x^K$, and so (x,ρ_K) will be chosen at line 11. Now, since $\rho \vDash \delta_2$, Lemma 22 gives $\rho_K \vDash \delta_2$, and we conclude that line 12 also applies. Let $\beta = \kappa_K(\rho_K,\omega_K)$, as in line 13. By line 15, the state (s,α,β) will be put into RS, if it is not already in HS. In any case, by the loop at line 6, we will get (s,α,β) in HS, and so, by line 33, we will have $(s,\alpha,\beta) \in S_G$. But, using Proposition 21, we have $\alpha = (\mu_L \oplus \theta)_L = (\mu \oplus \theta)_L$. Since $\nu = \mu \oplus \theta$, we obtain $\alpha = \nu_L$. Also, using Fact 37 we may write $\beta = \kappa_K(\rho_K,\omega_K) = \kappa_K(\rho,\omega)$. Since $\lambda = \kappa(\rho,\omega)$, we obtain $\beta = \lambda_K$, and so $(s,\nu_L,\lambda_K) \in S_G$, as desired.

Case 2: $\sigma = (y,\rho)$, for some $y \in Y$ and some $\rho \in [V \to \mathbb{Q}_{\geq}]$.

Since $(r,\mu,\omega) \Vdash_M^{\langle\sigma\rangle} (s,\nu,\lambda)$ we must have a transition $(r,\sigma,\delta_1,\delta_2,\theta,\xi,\chi,s)$ in T_y, for some $\delta_1 \in \Phi_C$, $\delta_2 \in \Phi_V$, $\theta \in [C \curvearrowright \mathbb{Q}_{\geq}]$, $\xi \in H_y^P$ and $\chi \in H_y^T$, satisfying $\mu \vDash \delta_1$, $\omega \vDash \delta_2$, $\lambda = \chi(\omega)$, $\rho = \xi(\lambda)$, and $\nu \vDash Inv(s)$, where $\nu = \mu \oplus \theta$.

From Lemma 23 we obtain $\mu_L \oplus \theta \vDash Inv(s)$. And from Lemma 22 we get $\mu_L \vDash \delta_1$ and $\omega_K \vDash \delta_2$. This enables line 21. Since (r, μ_L, ω_K) will be chosen at line 7, we let $\alpha = (\mu_L \oplus \theta)_L$ as in line 22, and $\beta = \chi_K(\lambda_K)$ as in line 22. Then, line 24 inserts the state (s, α, β) into RS, if it is not already in HS. In any case, by the loop at line 6, we will get (s, α, β) in HS, and so by line 33 we will have $(s, \alpha, \beta) \in S_G$. But, using Proposition 21, we have $\alpha = (\mu_L \oplus \theta)_L = (\mu \oplus \theta)_L$ and, since $\nu = \mu \oplus \theta$, we obtain $\alpha = \nu_L$. Also, $\lambda_K = \chi_K(\omega) = \chi_K(\omega_K) = \beta$, using Fact 36. Then, $(s, \nu_L, \lambda_K) \in S_G$, as desired.

Case 3: $\sigma \in \mathbb{Q}_{\geq}$.

Since σ is $[g,h]$-adjusted, we may write $\sigma = kg$, for some $k \geq 0$. Then, we obtain $(r, \mu, \omega) \overset{\langle kg \rangle}{\Vdash_M} (s, \nu, \lambda)$ where $s = r$, $\lambda = \omega$ and $\nu = \mu + kg$, with $\mu + \eta \vDash Inv(r)$, for all $0 < \eta \leq kg$. Since we already have $(r, \mu_L, \omega_K) \in S_G$, Lemma 47 gives $(r, \nu_L, \omega_K) \in S_G$. But $\lambda = \omega$, and so $(r, \nu_L, \lambda_K) \in S_G$, completing the proof. ∎

Now we show that all grid states correspond to reachable configurations in the corresponding TIOCA.

Lemma 49 *Let M_G be the grid corresponding to a TIOCA M. Let L, K be positive integers greater than any constant occurring in M. If $(s, \nu, \lambda) \in S_G$ then there is a $\mu \in [C \to \mathbb{Q}_{\geq}]$, $\omega \in [V \to \mathbb{Q}_{\geq}]$ and a $[g,h]$-adjusted parameterized timed word $\psi \in \Psi_{[g,h]}$ such that $(s_0, \nu_0, \lambda_0) \overset{\psi}{\Vdash_M} (s, \mu, \omega)$, with $\mu_L = \nu$ and $\omega_K = \lambda$.*

Proof From line 33 of Algorithm 1, we know that S_G is the set HS when the loop at line 6 terminates. From line 4, HS starts empty and elements are added to it one at a time and only at lines 15, 24, and 30. Hence, it suffices to show that the result holds for all triples (s, ν, λ) added to RS at these lines.

Let (r_i, μ_i, ω_i), $i \geq 0$, be the elements added to RS, in order. Clearly, $(r_0, \mu_0, \omega_0) = (s_0, \nu_0, \lambda_0)$ at line 4. Taking $\psi = \varepsilon$, the result is seen to hold for (r_0, μ_0, ω_0). Note also that $\nu_0 = (\nu_0)_L$, since $\nu_0(c) = 0$, for all $c \in C$, and also $\lambda_0 = (\lambda_0)_K$, since $\lambda_0(v) = 0$, for all $v \in V$.

Assume the result holds for (r_j, μ_j, ω_j), for all $0 \leq j < k$, for some $k \geq 1$. Consider (r_k, μ_k, ω_k). Since $k \geq 1$, (r_k, μ_k, ω_k) was added to RS at line 15, or at line 24, or at line 30. Hence, at that same iteration k some (r_j, μ_j, ω_j) with $j < k$ was chosen at line 7, with (r_k, μ_k, ω_k) subsequently added to RS also at iteration k. The induction hypothesis gives some $\psi \in \Psi_{[g,h]}$ such that $(s_0, \nu_0, \lambda_0) \overset{\psi}{\Vdash_M} (r_j, \mu, \omega)$ and $\mu_L = \mu_j$ and $\omega_K = \omega_j$. There are three cases.

Case 1: (r_k, μ_k, ω_k) was added to RS at line 15. Then, from line 8 we obtain a transition $(r_j, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r_k)$ in T_X with $\mu_j \vDash \delta_1$, $\omega_j \vDash \delta_3$ and $\mu_j \oplus \theta \vDash Inv(r_k)$. From lines 10 and 15, we get $\mu_k = (\mu_j \oplus \theta)_L$. From lines 11 and 12 we get a $\rho \in [R_x \to \mathbb{Q}_{\geq}]$ with $(x, \rho) \in I_x$ and $\rho_K \vDash \delta_2$. Then, lines 13 and 15 give $\omega_k = \kappa_K(\rho_K, \omega_j)$.

Since $\mu_L = \mu_j$ and $\omega_K = \omega_j$ we get $\mu_L \vDash \delta_1$ and $\omega_K \vDash \delta_3$. Together with $\rho_K \vDash \delta_2$, Lemma 22 gives $\mu \vDash \delta_1$, $\rho \vDash \delta_2$ and $\omega \vDash \delta_3$. Moreover, from $\mu_L = \mu_j$ and $\mu_j \oplus \theta \vDash Inv(r_k)$ we get $\mu_L \oplus \theta \vDash Inv(r_k)$. From Lemma 22 we may write $(\mu_L \oplus \theta)_L \vDash Inv(r_k)$, and then using Proposition 21 we may write $(\mu \oplus \theta)_L \vDash Inv(r_k)$. Using Lemma 22 again, we have $\mu \oplus \theta \vDash Inv(r_k)$.

Collecting, we have $(r_j, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r_k)$ in T_X, $\mu \vDash \delta_1$, $\rho \vDash \delta_2$, $\omega \vDash \delta_3$ and $\mu \oplus \theta \vDash Inv(r_k)$. Then we may write $(r_j, \mu, \omega) \overset{(x,\rho)}{\Vdash_M} (r_k, \mu \oplus \theta, \kappa(\rho, \omega))$. Therefore, composing we get $(s_0, \nu_0, \lambda_0) \overset{\psi\langle(x,\rho)\rangle}{\Vdash_M} (r_k, \mu \oplus \theta, \kappa(\rho, \omega))$.

We complete this case by showing $(\mu \oplus \theta)_L = \mu_k$ and $(\kappa(\rho, \omega))_K = \omega_k$. From Proposition 21, $(\mu \oplus \theta)_L = (\mu_L \oplus \theta)_L$. Since $\mu_L = \mu_j$ and $\mu_k = (\mu_j \oplus \theta)_L$, we get $(\mu \oplus \theta)_L = (\mu_j \oplus \theta)_L = \mu_k$, as desired. From Fact 37 $(\kappa(\rho, \omega))_K = \kappa_K(\rho, \omega) = \kappa_K(\rho_K, \omega_K)$. Since $\omega_K = \omega_j$ and $\omega_k = \kappa_K(\rho_K, \omega_j)$, we get $(\kappa(\rho, \omega))_K = \omega_k$, as also desired.

Case 2: (r_k, μ_k, ω_k) was added to RS at line 24. Then, from lines 20 and 21, we get a transition $(r_j, y, \delta_1, \delta_2, \theta, \xi, \chi, r_k)$ in T_Y with $\mu_j \vDash \delta_1$, $\omega_j \vDash \delta_2$ and $\mu_j \oplus \theta \vDash Inv(r_k)$. From lines 22, 22 and 24, $\mu_k = (\mu_j \oplus \theta)_L$, and $\omega_k = \chi_K(\omega_j)$.

Since $\mu_L = \mu_j$ and $\omega_K = \omega_j$, we get $\mu_L \vDash \delta_1$ and $\omega_K \vDash \delta_2$. Then, Lemma 22 yields $\mu \vDash \delta_1$ and $\omega \vDash \delta_2$. Also, from $\mu_L = \mu_j$ and $\mu_j \oplus \theta \vDash Inv(r_k)$, we get $\mu_L \oplus \theta \vDash Inv(r_k)$. From Lemma 22, we have $(\mu_L \oplus \theta)_L \vDash Inv(r_k)$. From Proposition 21 we may write $(\mu \oplus \theta)_L \vDash Inv(r_k)$ and so, from Lemma 22, we obtain $(\mu \oplus \theta) \vDash Inv(r_k)$.

Collecting, we have $(r_j, y, \delta_1, \delta_2, \theta, \xi, \chi, r_k)$ in T_Y, $\mu \vDash \delta_1$, $\omega \vDash \delta_2$, $(\mu \oplus \theta) \vDash Inv(r_k)$. Then $(r_j, \mu, \omega) \overset{(y,\rho)}{\Vdash_M} (r_k, \mu \oplus \theta, \chi(\omega))$. Therefore, $(s_0, \nu_0, \lambda_0) \overset{\psi\langle(y,\rho)\rangle}{\Vdash_M} (r_k, \mu \oplus \theta, \chi(\omega))$.

We complete this case by showing $(\mu \oplus \theta)_L = \mu_k$ and $\chi_K(\omega) = \omega_k$. From Proposition 21, $(\mu \oplus \theta)_L = (\mu_L \oplus \theta)_L$. Since $\mu_L = \mu_j$ and $\mu_k = (\mu_j \oplus \theta)_L$, we get $(\mu \oplus \theta)_L = (\mu_j \oplus \theta)_L = \mu_k$, as desired. Also, since $\omega_K = \omega_j$ and $\omega_k = \chi_K(\omega_j)$, we get $\omega_k = \chi_K(\omega_K) = \chi_K(\omega)$, where we used Fact 36. The argument for this case is complete.

Case 3: (r_k, μ_k, ω_k) was added to RS at line 30. Then from line 27 we obtain $\mu_j + \eta \vDash Inv(r_j)$, $0 < \eta \leq g$. Let $\mu_k = (\mu_j + g)_L$, as in line 28. Then, line 30 gives $\omega_k = \omega_j$ and $r_k = r_j$. Since $\mu_j = \mu_L$ we get $\mu_L + \eta \vDash Inv(r_j)$ and so $\mu + \eta \vDash Inv(r_j)$ by Lemma 22, for all $0 < \eta \leq g$. Also, since $\omega_j = \omega_K$ we get $\omega_K = \omega_k$. Then $(r_j, \mu, \omega) \overset{\langle g\rangle}{\Vdash_M} (r_j, \mu + g, \omega)$ and, since $r_j = r_k$ we get $(r_j, \mu, \omega) \overset{\langle g\rangle}{\Vdash_M} (r_k, \mu + g, \omega)$. Therefore, $(s_0, \nu_0, \lambda_0) \overset{\psi\cdot\langle g\rangle}{\Vdash_M} (r_k, \mu + g, \omega)$.

We complete this case by showing that $(\mu + g)_L = \mu_k$ and $\omega_K = \omega_k$. Since $\mu_k = (\mu_j + g)_L$ and $\mu_L = \mu_j$, we get $\mu_k = (\mu_L + g)_L$. Using Proposition 20, we conclude that $\mu_k = (\mu + g)_L$. Also, since we already have $\omega_K = \omega_j$ and $\omega_k = \omega_j$, we get $\omega_K = \omega_k$, as desired.

The induction is extended and we are done. ∎

The next definition maps adjusted parameterized timed words into grid words. Recall Definition 42.

Definition 50 *Let Σ be an alphabet and let $\Psi_{[g,h]}$ be the set of all $[g,h]$-adjusted parameterized timed words. Define the basic mappings: (i) every g-adjusted time value ig, with $i \geq 0$, is mapped to the grid word $g^i \in \Sigma_G^\star$; (ii) every parameterized input (x,ρ), with $x \in X$ and $\rho \in [R_x \to \mathbb{Q}_{\geq}]$, is mapped to the grid word $(x,\rho_K) \in \Sigma_G$; and (iii) every parameterized output (y,λ), with $y \in Y$ and $\lambda \in [V \to \mathbb{Q}_{\geq}]$, is mapped to the grid word $(y,\lambda_K) \in \Sigma_G$. Now define the morphism $f : \Psi_{[g,h]} \to \Sigma_G^\star$ in the natural way, by extending these basic mappings.* ∎

Next we show that a grid automaton can simulate the corresponding TIOCA over a $[g,h]$-adjusted parameterized timed word.

Theorem 51 *M is a TIOCA. Let $L, K \in \mathbb{Q}_{\geq}$ be positive integers greater than all constants occurring in M, and let M_G be the grid corresponding to M. Let $s,r \in S$, let $\nu,\omega \in [C \to \mathbb{Q}_{\geq}]$ and let $\lambda,\mu \in [V \to \mathbb{Q}_{\geq}]$ be such that (s,ν,λ) is $[g,h]$-reachable in M. If $(s,\nu,\lambda) \overset{\psi}{\Vdash}_M (r,\omega,\mu)$ then $(s,\nu_L,\lambda_K),(r,\omega_L,\mu_K) \in S_G$ and $(s,\nu_L,\lambda_K) \overset{f(\psi)}{\underset{G}{\Vdash}} (r,\omega_L,\mu_K)$.*

Proof Since (s,ν,λ) is $[g,h]$-reachable in M, we get a $[g,h]$-adjusted parameterized timed word ϕ such that $(s_0,\nu_0,\lambda_0) \overset{\phi}{\Vdash}_M (s,\nu,\lambda)$. Then $(s_0,\nu_0,\lambda_0) \overset{\phi\cdot\psi}{\Vdash}_M (r,\omega,\mu)$. Since $\phi \cdot \psi$ is also $[g,h]$-adjusted, (r,ω,μ) is also $[g,h]$-reachable in M. Thus, by Lemma 48, we get $(s,\nu_L,\lambda_K),(r,\omega_L,\mu_K) \in S_G$. It remains to show that $(s,\nu_L,\lambda_K) \overset{f(\psi)}{\underset{G}{\Vdash}} (r,\omega_L,\mu_K)$.

We proceed by induction on the length $n \geq 0$ of ψ, noting that $(s,\nu,\lambda) \overset{\psi}{\Vdash}_M (r,\omega,\mu)$.

Basis: when $n = 0$, we get $\psi = \varepsilon$ and so $s = r$, $\nu = \omega$ and $\lambda = \mu$. Thus $(s,\nu_L,\lambda_K) = (r,\omega_L,\mu_K)$ and we get $(s,\nu_L,\lambda_K) \overset{\varepsilon}{\underset{G}{\Vdash}} (r,\omega_L,\mu_K)$. Since $f(\psi) = \varepsilon$, the basis is complete.

Induction step: For all $x \in X$, we will denote the set of all $[g,h]$-adjusted parameterized timed inputs by $I_x = \{(x,\rho) | \rho \in [R_x \to \mathbb{Q}_{\geq}] \text{is } [g,h]\text{-adjusted}\}$, and will denote by $I_x^K = \{(x,\rho_K) | (x,\rho) \in I_x\}$ the set of all K-bounded such inputs in I_x. Similarly, we define I_y and I_y^K for all $y \in Y$.

Now, assume that the result holds for all $[g,h]$-adjusted parameterized timed words of length at most n. Take $\psi = \varphi\cdot\langle\sigma\rangle$, where φ has length n and $\langle\sigma\rangle$ has length one. Then, $(s,\nu,\lambda) \overset{\psi}{\Vdash}_M (r,\omega,\mu)$ gives $(s,\nu,\lambda) \overset{\varphi}{\Vdash}_M (p,\alpha,\beta)$ and $(p,\alpha,\beta) \overset{\langle\sigma\rangle}{\Vdash}_M (r,\omega,\mu)$, for some $\alpha \in [C \to \mathbb{Q}_{\geq}]$ and $\beta \in [V \to \mathbb{Q}_{\geq}]$. By the induction hypothesis, $(p,\alpha_L,\beta_K) \in S_G$ and $(s,\nu_L,\lambda_K) \overset{f(\varphi)}{\underset{G}{\Vdash}} (p,\alpha_L,\beta_K)$. Since, by definition, $f(\psi) = f(\varphi)\cdot f(\langle\sigma\rangle)$, it remains to show that $(p,\alpha_L,\beta_K) \overset{f(\langle\sigma\rangle)}{\underset{G}{\Vdash}} (r,\omega_L,\mu_K)$. Note that, since ψ is $[g,h]$-adjusted, then so is $\langle\sigma\rangle$. Then, we have three cases: when $\sigma \in I_x$ for some $x \in X$, when $\sigma \in I_y$ for some $y \in Y$, and when $\sigma \in \mathbb{Q}_{\geq}$.

CASE 1: $\sigma = (x, \rho) \in I_x$ for some $x \in X$.

Since $(p, \alpha, \beta) \overset{(x,\rho)}{\Vdash_M} (r, \omega, \mu)$, we must have a transition $(p, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r)$ in T_X with $\alpha \vDash \delta_1$, $\rho \vDash \delta_2$, $\beta \vDash \delta_3$, $\mu = \kappa(\rho, \beta)$, $\omega = \alpha \oplus \theta$, and $\omega \vDash Inv(r)$.

Recall that $(p, \alpha_L, \beta_K) \in S_G$. Hence, at some point, Algorithm 1 has chosen (p, α_L, β_K) at line 7. Moreover, using Lemma 22, from $\alpha \vDash \delta_1$ and $\beta \vDash \delta_3$ we get $\alpha_L \vDash \delta_1$ and $\beta_K \vDash \delta_3$. From $\alpha \oplus \theta \vDash Inv(r)$ and Lemma 23 we get $\alpha_L \oplus \theta \vDash Inv(r)$. Then, line 9 at Algorithm 1 is enabled. Let $\alpha' = (\alpha_L \oplus \theta)_L$, as in line 10. Also, clearly, $(x, \rho_K) \in I_x^K$ and, from $\rho \vDash \delta_2$ and Lemma 22, we get $\rho_K \vDash \delta_2$, showing that line 12 is also enabled. Let $\beta' = \kappa_K(\rho_K, \beta_K)$, as in line 13. Then line 14 adds $((p, \alpha_L, \beta_K), (x, \rho_K), (r, \alpha', \beta'))$ to T_G. Now, since $\sigma = (x, \rho)$, we get $f(\langle\sigma\rangle) = (x, \rho_K)$. Then, $(p, \alpha_L, \beta_K) \overset{f(\langle\sigma\rangle)}{\underset{G}{\Vdash}} (r, \alpha', \beta')$.

We complete this case by showing that $\alpha' = \omega_L$ and that $\beta' = \mu_K$. Since $\alpha' = (\alpha_L \oplus \theta)_L$, Proposition 21 gives $\alpha' = (\alpha \oplus \theta)_L$. Then $\alpha' = \omega_L$ since $\omega = \alpha \oplus \theta$. Also, since $\beta' = \kappa_K(\rho_K, \beta_K)$, Fact 37 gives $\beta' = \kappa_K(\rho, \beta)$. Then $\beta' = \mu_K$ since $\mu = \kappa(\rho, \beta)$.

CASE 2: $\sigma = (y, \rho) \in I_y$ for some $y \in Y$.

Since $(p, \alpha, \beta) \overset{(y,\rho)}{\Vdash_M} (r, \omega, \mu)$, we must have a transition $(p, y, \delta_1, \delta_2, \theta, \xi, \chi, r)$ in T_Y, with $\alpha \vDash \delta_1$, $\beta \vDash \delta_2$, $\mu = \chi(\beta)$, $\rho = \xi(\beta)$, $\omega = \alpha \oplus \theta$, and $\omega \vDash Inv(r)$.

Recall that $(p, \alpha_L, \beta_K) \in S_G$. Hence, at some point, Algorithm 1 has chosen (p, α_L, β_K) at line 7. From $\alpha \vDash \delta_1$ and $\beta \vDash \delta_2$, Lemma 22 gives $\alpha_L \vDash \delta_1$ and $\beta_K \vDash \delta_2$. From $\alpha \oplus \theta \vDash Inv(r)$ and Lemma 23 we get $\alpha_L \oplus \theta \vDash Inv(r)$. Then Algorithm 1, lines 22, 22 and 23, adds $((p, \alpha_L, \beta_K), (y, \rho_K), (r, \alpha', \beta'))$ to T_G, where $\alpha' = (\alpha_L \oplus \theta)_L$ and $\beta' = \chi_K(\beta_K)$. Since $\sigma = (y, \rho)$, we get $f(\langle\sigma\rangle) = (y, \rho_K)$. Hence, $(p, \alpha_L, \beta_K) \overset{f(\langle\sigma\rangle)}{\underset{G}{\Vdash}} (r, \alpha', \beta')$.

We complete this case by showing that $\alpha' = \omega_L$ and that $\beta' = \mu_K$. Since $\alpha' = (\alpha_L \oplus \theta)_L$, Fact 21 gives $\alpha' = (\alpha \oplus \theta)_L$. Then $\alpha' = \omega_L$ since $\omega = \alpha \oplus \theta$. Also, since $\beta' = \chi_K(\beta_K)$, Fact36 gives $\beta' = \chi_K(\beta)$ and, together with $\mu = \chi(\beta)$, we get $\beta' = \mu_K$.

CASE 3: $\sigma \in \mathbb{Q}_{\geq}$.

Since σ is $[g, h]$-adjusted, let $\sigma = ig$, for some $i \geq 0$. Moreover, since $(p, \alpha, \beta) \overset{\langle ig\rangle}{\Vdash_M} (r, \omega, \mu)$, we conclude that $p = r$, $\omega = \alpha + ig$, $\alpha + \eta \vDash Inv(r)$ for all $0 < \eta \leq ig$, and $\mu = \beta$. Hence, $\alpha + \eta \vDash Inv(p)$ for $0 < \eta \leq ig$. Since we already have $(p, \alpha_L, \beta_K) \in S_G$, Lemma 47 gives $(p, \alpha'_L, \beta_K) \in S_G$ and $(p, \alpha_L, \beta_K) \overset{g^i}{\underset{G}{\Vdash}} (p, \alpha'_L, \beta_K)$, with $\alpha' = \alpha + ig$. So, $\alpha' = \omega$ and, since we already have $\mu = \beta$, we get $(p, \alpha'_L, \beta_K) = (r, \omega_L, \mu_K)$. Since $f(\langle\sigma\rangle) = g^i$, we can write $(p, \alpha_L, \beta_K) \overset{f(\langle\sigma\rangle)}{\underset{G}{\Vdash}} (r, \omega_L, \mu_K)$, as desired.

The induction is extended, completing the proof. ∎

The next theorem shows that a TIOCA imitates the corresponding grid automaton.

Theorem 52 *M_G is the grid corresponding to a TIOCA M where $L, K \in \mathbb{Q}_{\geq}$ are positive integers greater than all constants occurring in M. Let $(s, \widehat{\omega}, \widehat{\mu}), (r, \omega, \mu) \in S_G$ with $(s, \widehat{\omega}, \widehat{\mu}) \overset{\psi}{\Vdash}_G (r, \omega, \mu)$, for some $\psi \in \Sigma_G^\star$. Then there are some $\alpha, \widehat{\alpha} \in [C \to \mathbb{Q}_{\geq}]$, some $\beta, \widehat{\beta} \in [V \to \mathbb{Q}_{\geq}]$ and some $\widehat{\psi} \in \Psi_{[g,h]}$, such that $(s, \widehat{\alpha}, \widehat{\beta}) \overset{\widehat{\psi}}{\vdash}_M (r, \alpha, \beta)$, with $\widehat{\omega} = \widehat{\alpha}_L$, $\omega = \alpha_L$, $\widehat{\mu} = \widehat{\beta}_K$, $\mu = \beta_K$ and $f(\widehat{\psi}) = \psi$.*

Proof We proceed by induction on the length $n \geq 0$ of ψ.

Basis: when $n = 0$, we get $\psi = \varepsilon$, $s = r$, $\widehat{\omega} = \omega$ and $\widehat{\mu} = \mu$. Let $\widehat{\psi} = \varepsilon$. Since $(s, \widehat{\omega}, \widehat{\mu}) \in S_G$, Lemma 49, gives $\widehat{\alpha} \in [C \to \mathbb{Q}_{\geq}]$, with $\widehat{\alpha}_L = \widehat{\omega}$, and $\widehat{\beta} \in [V \to \mathbb{Q}_{\geq}]$, with $\widehat{\beta}_K = \widehat{\mu}$.

Take $\alpha = \widehat{\alpha}$ and $\beta = \widehat{\beta}$. Then $(s, \widehat{\alpha}, \widehat{\beta}) \overset{\varepsilon}{\vdash}_M (r, \alpha, \beta)$, and so $(s, \widehat{\alpha}, \widehat{\beta}) \overset{\widehat{\psi}}{\vdash}_M (r, \alpha, \beta)$, with $f(\widehat{\psi}) = \psi$. Moreover, $\omega = \widehat{\omega} = \widehat{\alpha}_L = \alpha_L$, and $\mu = \widehat{\mu} = \widehat{\beta}_K = \beta_K$, completing the basis.

Induction step: For all $x \in X$, we will denote the set of all $[g, h]$-adjusted parameterized timed inputs by $I_x = \{(x, \rho) | \rho \in [R_x \to \mathbb{Q}_{\geq}]$is $[g, h]$-adjusted$\}$, and will denote by $I_x^K = \{(x, \rho_K) | (x, \rho) \in I_x\}$ the set of all K-bounded such inputs. Similarly, we define I_y and I_y^K for all $y \in Y$.

Now, assume that the result holds for all grid words of length at most n. Take $\psi = \varphi \cdot \sigma \in \Sigma_G^\star$, where φ has length n and $\sigma \in \Sigma_G$. Then, $(s, \widehat{\omega}, \widehat{\mu}) \overset{\psi}{\Vdash}_G (r, \omega, \mu)$ gives $(s, \widehat{\omega}, \widehat{\mu}) \overset{\varphi}{\Vdash}_G (p, \omega', \mu')$ and $(p, \omega', \mu') \overset{\sigma}{\Vdash}_G (r, \omega, \mu)$, with $(p, \omega', \mu') \in S_G$. By the induction hypothesis we get some $\widehat{\alpha}, \alpha' \in [C \to \mathbb{Q}_{\geq}]$, some $\widehat{\beta}, \beta' \in [V \to \mathbb{Q}_{\geq}]$ and some $\varphi' \in \Psi_{g,h}$, with $(s, \widehat{\alpha}, \widehat{\beta}) \overset{\varphi'}{\vdash}_M (p, \alpha', \beta')$, $\widehat{\alpha}_L = \widehat{\omega}$, $\alpha'_L = \omega'$, $\widehat{\beta}_K = \widehat{\mu}$, $\beta'_K = \mu'$ and $f(\varphi') = \varphi$.

We extend the induction by showing that $(p, \alpha', \beta') \overset{\langle \sigma' \rangle}{\vdash}_M (r, \alpha, \beta)$, for some $\alpha \in [C \to \mathbb{Q}_{\geq}]$ some $\beta \in [V \to \mathbb{Q}_{\geq}]$ and some parameterized word σ', with $\alpha_L = \omega$, $\beta_K = \mu$ and $f(\langle \sigma' \rangle) = \sigma$. Note that then we will have $(p, \widehat{\alpha}, \widehat{\beta}) \overset{\varphi' \langle \sigma' \rangle}{\vdash}_M (r, \alpha, \beta)$, with $f(\varphi') f(\langle \sigma' \rangle) = \varphi\sigma = \psi$, as desired.

Note that, since $\alpha'_L = \omega'$ and $\beta'_K = \mu'$, we also have $(p, \alpha'_L, \beta'_K) \overset{\sigma}{\Vdash}_G (r, \omega, \mu)$. There are three cases: when $\sigma \in I_x^K$ for some $x \in X$, when $\sigma \in I_y^K$ for some $y \in Y$, and when $\sigma = g$.

Case 1: $\sigma = (x, \rho) \in I_x^K$, for some $x \in X$.

Then $(p, \alpha'_L, \beta'_K) \overset{(x,\rho)}{\Vdash}_G (r, \omega, \mu)$, and we must have in T_G a transition in the form $((p, \alpha'_L, \beta'_K), (x, \rho), (r, \omega, \mu))$. From Algorithm 1, this can only happen if such a transition was added to T_G at line 14. Hence, from lines 8 to 15, we must have

a transition $(p, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r)$ in T_X. Moreover, from line 9 we get $\alpha'_L \vDash \delta_1$, $\beta'_K \vDash \delta_3$ and $\alpha'_L \oplus \theta \vDash Inv(r)$. Also, from line 12 we get $\rho \vDash \delta_2$. Finally, from lines 10 and 13, we may write, respectively, $\omega = (\alpha'_L \oplus \theta)_L$ and $\mu = \kappa_K(\rho, \beta'_K)$.

Recall that we want $(p, \alpha', \beta') \overset{\langle\sigma'\rangle}{\Vdash_M} (r, \alpha, \beta)$, with $\alpha_L = \omega$, $\beta_K = \mu$ and $f(\langle\sigma'\rangle) = \sigma$. Let $\sigma' = (x, \rho) = \sigma$. Since $(x, \rho) \in I_x^K$ we know that $\rho = \rho'_K$, for some $\rho' \in [V \to \mathbb{Q}_{\geq}]$. Then, by Proposition 14, we get $\rho_K = (\rho'_K)_K = \rho'_K = \rho$. Thus, $f(\langle\sigma'\rangle) = (x, \rho_K) = (x, \rho) = \sigma$. It remains to show that $(p, \alpha', \beta') \overset{(x,\rho)}{\Vdash_M} (r, \alpha, \beta)$, with $\alpha_L = \omega$ and $\beta_K = \mu$.

Since we already have $\alpha'_L \vDash \delta_1$ and $\beta'_K \vDash \delta_3$, Lemma 22 yields $\alpha' \vDash \delta_1$ and $\beta' \vDash \delta_3$. Also, from $\alpha'_L \oplus \theta \vDash Inv(r)$ and Lemma 22, we get $(\alpha'_L \oplus \theta)_L \vDash Inv(r)$, and using Proposition 21 we have $(\alpha' \oplus \theta)_L \vDash Inv(r)$, and so $\alpha' \oplus \theta \vDash Inv(r)$ by Lemma 22 again. Since we know that the transition $(p, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r)$ is in T_X, and that $\rho \vDash \delta_2$, we can write $(p, \alpha', \beta') \overset{(x,\rho)}{\Vdash_M} (r, \alpha' \oplus \theta, \kappa(\rho, \beta'))$. Let $\alpha = \alpha' \oplus \theta$. Then $\alpha_L = (\alpha' \oplus \theta)_L$ and, using Proposition 21, we get $\alpha_L = (\alpha'_L \oplus \theta)_L = \omega$, as desired. Now let $\beta = \kappa(\rho, \beta')$. Then, $\beta_K = \kappa_K(\rho, \beta')$. By Fact 37 we get $\beta_K = \kappa_K(\rho_K, \beta'_K)$. Since we already have $\rho_K = \rho$, we get $\beta_K = \kappa_K(\rho, \beta'_K) = \mu$, concluding this case.

Case 2: $\sigma = (y, \rho) \in I_y^K$, for some $y \in Y$.

Then $(p, \alpha'_L, \beta'_K) \overset{(y,\rho)}{\underset{G}{\Vdash}} (r, \omega, \mu)$, and we must have in T_G a transition in the form $((p, \alpha'_L, \beta'_K), (y, \rho), (r, \omega, \mu))$. From Algorithm 1, this can only happen if such a transition was added to T_G at line 23. Hence, from lines 20 and 21, we must have a transition $(p, y, \delta_1, \delta_2, \theta, \xi, \chi, r)$ in T_Y, with $\alpha'_L \vDash \delta_1$, $\beta'_K \vDash \delta_2$ and $\alpha'_L \oplus \theta \vDash Inv(r)$. Further, from lines 22 to 24, we know that $\omega = (\alpha'_L \oplus \theta)_L$, $\mu = \chi_K(\beta'_K)$ and $\rho = \xi_K(\beta'_K)$.

Recall that we want $(p, \alpha', \beta') \overset{\langle\sigma'\rangle}{\Vdash_M} (r, \alpha, \beta)$, with $\alpha_L = \omega$, $\beta_K = \mu$ and $f(\langle\sigma'\rangle) = \sigma$.

Since we already have $\alpha'_L \vDash \delta_1$ and $\beta'_K \vDash \delta_2$, Lemma 22 yields $\alpha' \vDash \delta_1$ and $\beta' \vDash \delta_2$. Also, from $\alpha'_L \oplus \theta \vDash Inv(r)$ and Lemma 22, we get $(\alpha'_L \oplus \theta)_L \vDash Inv(r)$, and using Proposition 21 we have $(\alpha' \oplus \theta)_L \vDash Inv(r)$, and so $\alpha' \oplus \theta \vDash Inv(r)$ by Lemma 22 again. We know that the transition $(p, y, \delta_1, \delta_2, \theta, \xi, \chi, r)$ is in T_Y. We can then write $(p, \alpha', \beta') \overset{\langle(y,\rho')\rangle}{\Vdash_M} (r, \alpha, \beta)$, with $\alpha = \alpha' \oplus \theta$, $\beta = \chi(\beta')$ and $\rho' = \xi(\beta')$. Let $\sigma' = (y, \rho')$. It remains to show that $f(\langle\sigma'\rangle) = \sigma$, $\alpha_L = \omega$ and $\beta_K = \mu$.

We have $f(\langle\sigma'\rangle) = f(\langle(y, \rho')\rangle = (y, \rho'_K)$. Since $\rho' = \xi(\beta')$, using Fact 36, we obtain $\rho'_K = \xi_K(\beta') = \xi_K(\beta'_K) = \rho$. Then, $f(\langle\sigma'\rangle) = (y, \rho) = \sigma$. Next, $\alpha_L = (\alpha' \oplus \theta)_L = (\alpha'_L \oplus \theta)_L = \omega$, using Proposition 21. Also, $\beta_K = \chi_K(\beta') = \chi_K(\beta'_K)$, using Fact 36. We then get $\beta_K = \mu$, completing this case.

Case 3: $\sigma = g$.

Since $(p, \alpha'_L, \beta'_L) \overset{g}{\underset{G}{\Vdash}} (r, \omega, \mu)$, we must have a transition $((p, \alpha'_L, \beta'_K), g, (r, \omega, \mu))$

in T_G. From Algorithm 1, lines 27 – 30, we have $p = r$, $\omega = (\alpha'_L + g)_L$, $\alpha'_L + \eta \vDash Inv(p)$ for all $0 < \eta \leq g$, and $\mu = \beta'_K$.

We need $(p, \alpha'_l, \beta'_L) \overset{\langle\sigma'\rangle}{\Vdash_M} (r, \alpha, \beta)$, with $\alpha_L = \omega$, $\beta_K = \mu$ and $f(\langle\sigma'\rangle) = \sigma$.

We already have $p = r$. Fix any η, $0 < \eta \leq g$. From $\alpha'_L + \eta \vDash Inv(p)$ and Lemma 22 we get $(\alpha'_L + \eta)_L \vDash Inv(p)$. From Proposition 20 it follows that $(\alpha' + \eta)_L \vDash Inv(p)$, and from Lemma 22 again, $\alpha' + \eta \vDash Inv(p)$. We may conclude that $\alpha' + \eta \vDash Inv(p)$ for all $0 < \eta \leq g$. We can then write $(p, \alpha'_l, \beta'_L) \overset{\langle\sigma'\rangle}{\Vdash_M} (r, \alpha, \beta)$ where $\sigma' = g$, $\alpha = \alpha' \oplus \theta$ and $\beta = \beta'$.

Then, $f(\langle\sigma'\rangle) = f(\langle g\rangle) = g = \sigma$. Also, using Proposition 20, $\alpha_L = (\alpha' \oplus \theta)_L = (\alpha'_L \oplus \theta)_L = \omega$. Finally, $\beta_K = \beta'_K = \mu$, completing this case.

The induction is extended and the proof is complete. ∎

5 Generating test cases for TIOCA

Now we show how to generate test cases by using test purposes and TIOCA. First we define acyclic TIOCA, which can be used for modeling faults or desired properties.

Definition 53 *A TIOCA is acyclic iff its subjacent directed graph, defined by taking states as nodes and transitions as edges, is acyclic.* ∎

Test purposes are acyclic TIOCA equipped with fail and desired states.

Definition 54 *A TIOCA test purpose is an acyclic TIOCA with two special sets of states: a set $F \subseteq S$, of fail states, and a set $D \subseteq S$, of desired states, with $F \cap D = \emptyset$.* ∎

We also need the notion of synchronous product over TIOCA. We construct the synchronous product algorithmically.

Definition 55 *Let M_1 and M_2 be two TIOCA, with $C_1 \cap C_2 = \emptyset$ and $V_1 \cap V_2 = \emptyset$. The synchronous product of M_1 and M_2 is the TIOCA constructed by Algorithm 2.* ∎

Algorithm 2 constructs the synchronous product by first pairing the initial state of both participating TIOCA in order to create the initial state of the product TIOCA. Then, the product transitions are constructed by an exhaustive search for new transitions and new states. Finally, we want a state (s_1, s_2) to be a fail state in the product when s_2 is a fail state in the purpose model. Similarly for the desired states.

Definition 56 *Let M_1 be a TIOCA and M_2 be a TIOCA test purpose. In the product automaton of Definition 55, a state (s_1, s_2) is a desired or fail state iff s_2 is a desired or fail state, respectively, in M_2.* ∎

Input: TIOCA M_1 and M_2 with $C_1 \cap C_2 = \emptyset$ and $V_1 \cap V_2 = \emptyset$.
Output: The synchronous product M_P.
begin
  $C_P \leftarrow C_1 \cup C_2$; $R_P \leftarrow R_1 \cup R_2$; $V_P \leftarrow V_1 \cup V_2$; $\Sigma_P \leftarrow \Sigma_1 \cup \Sigma_2$;
  $RS \leftarrow s_0^P = (s_0^1, s_0^2)$; $Inv_P(s_0^P) \leftarrow Inv_1(s_0^1) \wedge Inv_2(s_0^2)$; $T_P \leftarrow \emptyset$, $HS \leftarrow \emptyset$;
  while $RS \setminus HS \neq \emptyset$ **do**
    choose $s = (s_1, s_2)$ from $RS \setminus HS$;
    move s from RS to HS;
    foreach $a \in X$ **do**
      if $(s_i, a, \delta_i^1, \delta_i^2, \delta_i^3, \theta_i, \kappa_i, s_{i+2}) \in T_i^X$, $i = 1, 2$ **then**
        add (s_3, s_4) to RS; **let** $Inv_P(s_3, s_4) \leftarrow Inv_1(s_3) \wedge Inv_2(s_4)$;
        add $((s_1, s_2), a, \delta_1^1 \wedge \delta_2^1, \delta_1^2 \wedge \delta_2^2, \delta_1^3 \wedge \delta_2^3, \theta_1 \oplus \theta_2, \kappa_1 \oplus \kappa_2, (s_3, s_4))$ to T_P^X;
      end
      if $(s_i, a, \delta_i^1, \delta_i^2, \delta_i^3, \theta_i, \kappa_i, s_{i+2}) \in T_i^X$ *for some* $s_{i+2} \in S$, *and*
      $(s_j, a, \delta_j^1, \delta_j^2, \delta_j^3, \theta_j, \kappa_j, s_{j+2}) \notin T_j^X$ *for all* $s_{j+2} \in S$, *with* $i \neq j$, $i, j \in \{1, 2\}$
      then
        if $i = 1$ **then** $(p, q) = (s_3, s_2)$ **else** $(p, q) = (s_1, s_4)$;
        add (p, q) to RS, **let** $Inv_P(p, q) \leftarrow Inv_1(p) \wedge Inv_2(q)$,
        add $((s_1, s_2), a, \delta_i^1, \delta_i^2, \delta_i^3, \theta_i, \kappa_i, (p, q))$ to T_P^X;
      end
    end
    foreach $a \in Y$ **do**
      if $(s_i, a, \delta_i^1, \delta_i^2, \theta_i, \xi_i, \chi, s_{i+2}) \in T_i^Y$, $i = 1, 2$ **then**
        add (s_3, s_4) to RS; **let** $Inv_P(s_3, s_4) \leftarrow Inv_1(s_3) \wedge Inv_2(s_4)$,
        add $((s_1, s_2), a, \delta_1^1 \wedge \delta_2^1, \delta_1^2 \wedge \delta_2^2, \theta_1 \oplus \theta_2, \xi_1 \oplus \xi_2, \chi_1 \oplus \chi_2, (s_3, s_4))$ to T_P^Y;
      end
      if $(s_i, a, \delta_i^1, \delta_i^2, \theta_i, \xi_i, \chi_i, s_{i+2}) \in T_i^Y$ *for some* $s_{i+2} \in S$, *and*
      $(s_j, a, \delta_j^1, \delta_j^2, \theta_j, \xi_j, \chi_j, s_{j+2}) \notin T_j^Y$ *for all* $s_{j+2} \in S$, *with* $i \neq j$, $i, j \in \{1, 2\}$,
      then
        if $i = 1$ **then** $(p, q) = (s_3, s_2)$ **else** $(p, q) = (s_1, s_4)$;
        add (p, q) to RS, **let** $Inv_P(p, q) \leftarrow Inv_1(p) \wedge Inv_2(q)$,
        add $((s_1, s_2), a, \delta_i^1, \delta_i^2, \theta_i, \xi_i, \chi_i, (p, q))$ to T_P^Y;
      end
    end
  end
  $S_P \leftarrow HS$;
end

Algorithm 2: Synchronous product for TIOCA.

5.1 TIOCA test case generation and the grid automaton

In order to generate a grid automaton, we need to define the product of a specification and a test purpose, when given by their respective TIOCA.

The grid automaton is obtained from the resulting product following the methods presented in Section 4. In Figure 2 we illustrate a framework to obtain both the product and also the grid automaton.

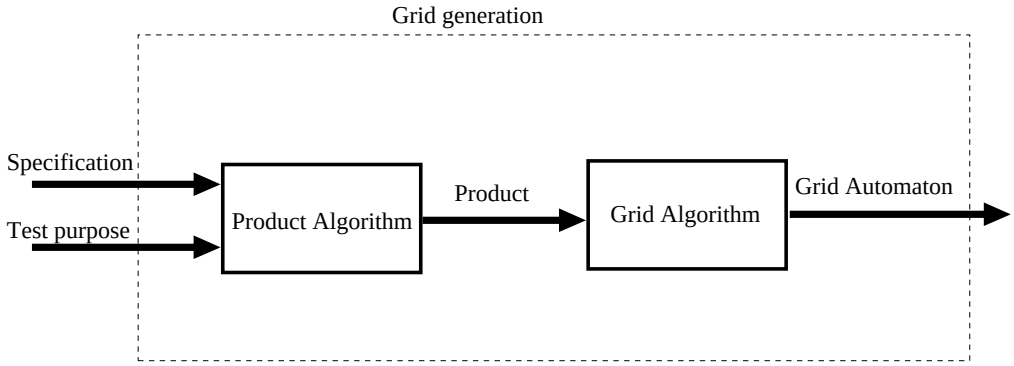

Figure 2: The framework to generate the grid automaton.

After the grid construction, test sequences are extracted by traversing the grid. The traversal operation starts at the initial state of the grid and searches down until it finds fail or desired states, depending on the nature of the test. Upon finding one such state, the corresponding test sequence is output. A recursive traversal procedure is depicted in Algorithm 3. Clearly, the set of all test sequences is generated by a call in the form $TiocaTestGeneration(s_0, \epsilon)$, where s_0 is the initial state of the grid and ϵ is the empty string.

One can then construct $[g, h]$-adjusted timed words from all grid words extracted from

```
1  TiocaTestGeneration(INPUT: state s of a TIOCA grid M_G; OUTPUT:
   Parameterized timed test sequence PTTS;)
2  begin
3  |  if s is a leaf then
4  |  |  Write PTTS;
5  |  else
6  |  |  foreach neighbor, r, of s reached over a transition on σ do
7  |  |  |  PTTS ← {σ} · PTTS;
8  |  |  |  TiocaTestGeneration(r,PTTS);
9  |  |  end
10 |  end
11 end
```

Algorithm 3: The traversal algorithm for TIOCA.

the grid automaton using the mapping given at Definition 50. Note that delay transitions in the grid represent continuous evolutions and parameterized timed inputs and outputs represent context changes in the original specification, up to the chosen boundary.

When the test purpose models desired behaviors of a system, the verification process

issues a "pass" verdict only when the implementation respects the specification and it satisfies the test purpose, for all sequences in the test suite. If, on the other hand, the testing if based on a purpose automaton with fail states, then the verification process issues a positive verdict when the implementation satisfies the specification and also reaches a faulty state, for any of the sequences in the test suite.

5.2 Test cases and observable behavior over TIOCA

In order to drive a TIOCA, we need to supply it with a sequence of parameterized timed input symbols, and in order to extract the behavior of a run, we need to project a parameterized timed word onto a subset of parameterized timed output actions.

Definition 57 *Let Σ be an alphabet and let $\Upsilon \subseteq \Sigma$. Let $\psi = \langle \sigma_1, \ldots, \sigma_n \rangle$, $n \geq 0$, be a parameterized timed word over Σ. The projection of ψ over Υ, denoted by $\psi{\downarrow_\Upsilon}$, is the parameterized timed word over Υ given by the concatenation $\psi{\downarrow_\Upsilon} = \phi_1 \cdot \phi_2 \cdot \ldots \cdot \phi_n$, where*

$$\phi_i = \begin{cases} \langle \sigma_i \rangle & \text{if } \sigma_i \in \Psi_\Upsilon, \\ \varepsilon & \text{otherwise.} \end{cases}$$ ∎

That is, the projection extracts only the parameterized actions in the subset of interest, together with timing values.

Parameterized test cases are parameterized timed words where only input action symbols can occur.

Definition 58 *A* parameterized test sequence *for a TIOCA M is a parameterized timed word over its input actions, that is, an element of Ψ_X.* ∎

Figure 3 illustrates a framework for extracting parameterized timed test cases.

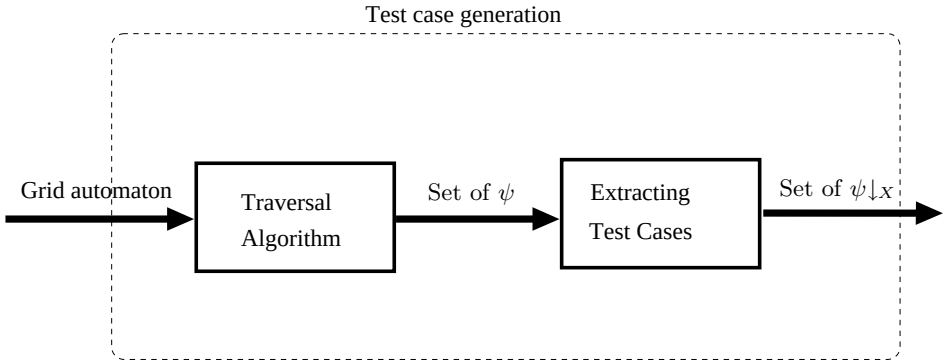

Figure 3: The framework to extract test cases.

Given a parameterized test sequence ψ and given a start configuration γ, we can discover all parameterized timed words that have ψ as a projection over X_M and that produce a run starting at γ.

Definition 59 *Let M be a TIOCA and let $\psi \in \Psi_X$ and $\gamma \in \Gamma_M$. A* support *for ψ and γ is a parameterized timed word $\rho \in \Psi_M$ such that $\psi = \rho{\downarrow_X}$ and $\gamma \overset{\rho}{\Vdash} \mu$, for some $\mu \in \Gamma_M$.* ∎

That is, a support for an input parameterized timed word ψ and a configuration γ is a timed word ρ that drives the TIOCA from γ to some configuration μ, and such that the projection of ρ over the set of parameterized input action symbols X_M is precisely ψ. The set of all supports for ψ and γ will be denoted by $\Lambda_{\psi,\gamma}$. When γ is the initial configuration of M, we also write Λ_ψ. Clearly, any $\rho \in \Lambda_\psi$ is a run of M.

We can now define observables associated with a parameterized timed test sequence and a start configuration.

Definition 60 *Let M be a TIOA and let $\psi \in \Psi_X$, $\gamma \in \Gamma_M$. The (*observable*)* behavior *of M from γ over ψ is the set $\{\rho{\downarrow_X} \mid \rho \in \Lambda_{\psi,\gamma}\}$.* ∎

We will denote by $\mathcal{O}_{\psi,\gamma}$ the set of observable behaviors of M from γ over ψ. When γ is the initial configuration of M we may simply write $\mathcal{O}_\psi$.

5.3 Applying timed test cases

An implementation behavior is investigated by performing experiments over it [22]. Such experiments consist of applying stimuli to the implementation and observing its responses.

We obtain a set of parameterized timed test sequences by traversing the corresponding grid automaton which, in turn, was obtained from the product between the specification and the test purpose. Such parameterized timed test sequences are sequences of parameterized input and output actions, as well as time delays. Parameterized timed test cases then are extracted from the parameterized timed test sequences, by projecting such sequences on the parameterized inputs.

A parameterized test execution is obtained by applying a parameterized timed test case to an implementation and observing the respective responses. The output responses of the implementation under test are then combined with the respective parameterized timed test cases. By this process, we obtain parameterized test executions, also called *runs*, which combine parameterized input actions and time delays from the parameterized timed test cases, together with time delays and observed parameterized outputs from the implementation.

Suppose that a parameterized test case is submitted to an implementation under test. Then, usually, a time delay must pass before the next parameterized input action occurs, or before the next parameterized output action is observed. Note that the time delay preceding a parameterized output symbol is not under the control of the observer, and so we cannot know in advance the exact instants when parameterized outputs will occur. In any case, when collecting a test run, the time delay actually observed before a parameterized output symbol occurrence is adjoined to the run being constructed, followed by the corresponding parameterized output symbol. When the next action is a parameterized input action the time delay specified on the test case is under control of the observer. In this case, this

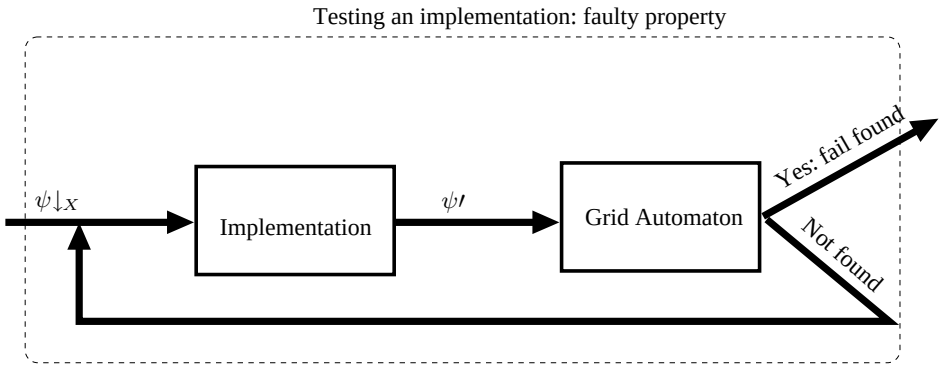

Figure 4: The framework to test implementations: faulty property.

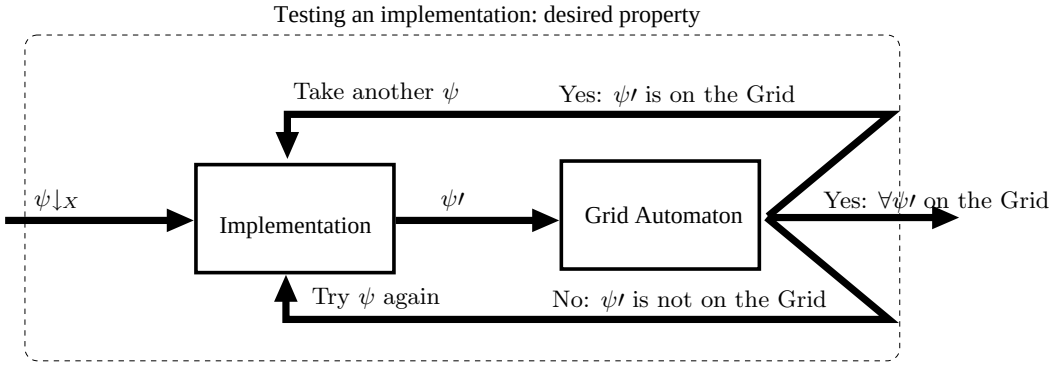

Figure 5: The framework to test implementations: desired property.

time delay, followed by the parameterized input symbol, is also adjoined to the run being collected.

Using this approach, a set of runs can be collected by applying parameterized timed test cases over the candidate implementation. After that we apply such parameterized test executions again to the corresponding grid automaton. We can, then, verify if runs obtained from the implementation candidate are also runs over the corresponding grid automaton. When a run from the implementation is also a run over the grid, we can say that both models are in conformance, with respect to the given test case. Figures 4 and 5 illustrate these processes.

When a faulty property is specified by the test purpose model, the product automaton will contain special faulty states. So, the parameterized timed test cases extracted from the grid will be sequences of parameterized input actions and time delays that lead to faulty states. We then collect runs over the implementation using these parameterized timed test cases. By applying such runs back to the original grid, if any of them reaches a faulty state, we can announce a positive test verdict, that is, a fault was confirmed over the implementation. If all collected runs, when applied to the grid, do not reach faulty states,

then the test is deemed inconclusive.

On the other hand, when the test purpose models a desired property, we must make sure that all runs, when applied to the grid, terminate at desired states. In this case, the test is positive, otherwise it is inconclusive.

Note that, in this approach, we test implementations by resubmitting runs back to the grid automaton. In a first alternative, we can assume that the grid automaton is fully constructed, and is kept in memory. Then we are able to efficiently search down forward for special states and, also, we can effectively submit a run to the grid. As a second alternative, avoiding the full grid construction, we can use the grid construction algorithm to resubmit collected runs to the grid. When dealing with fault properties, avoiding the full grid construction can be a reasonable alternative, since we only need one conformance hit.

6 Related works

Many works have discussed formal methods to automatically generate test cases suites for complex systems. Depending on specific characteristics of the target systems, earlier studies discuss either techniques for timed systems or methods applicable to reactive systems with context variables, but in both cases only one of these approaches is treated in isolation. In this section we briefly describe some such works.

En-Nouaary and Dssouli discuss a timed test case generation method in [8]. This method is based on TIOA specifications and test purposes. It also uses the notion of synchronous product. However, their discretization technique is based on the classical notion of clock regions, thus imposing a strict relationship between the number of clock variables present in the models and the granularity that must be chosen in order to obtain the corresponding grid automaton. Instead of constructing a grid automaton directly, the notion of a region graph is first used to represent a possibly infinite transition system. Then, by a process of sampling, a grid automaton is derived from the transition system. Test sequences are then extracted from the grid automaton. But the authors do not detail how one could apply the timed test cases that are extracted from the grid automaton. Further, a TIOA models continuous time evolution by means of clock variables only. Context variables are not allowed in a TIOA model, since it is not fit to capture the notion of context transformation.

Fouchal [11] proposes another test case generation method based on TIOA. Test purposes are used as in [8] in order to capture specific system properties. In Fouchal's proposal, region graphs are also used, and are likewise sampled in order to obtain grid automata. Algorithms are used in an exhaustive test generation process, but no guarantees of correctness are offered. Dense time has but a superficial treatment, making it difficult to realize how precise are the timed test sequences that are obtained, specially when the original models are combined with test purposes. Further, also, the models do not deal with context variables.

In [10], Fouchal and co-workers present a test execution strategy similar to the one discussed in this work. Although both strategies are related, our work deals properly with dense time in order to capture timed properties, whereas in [10] a notion of timed elements is used to imitate continuous time evolution, in a process that offers no guarantees of accuracy about the obtained test suites. Once again, context transformations and facilities

for returning values to the environment are not considered.

Similar approaches appeared in [9, 7, 6], none of them dealing with context transformations. All of them also use the classical notion of clock regions in order to obtain grid automata from which test case sequences can be extracted. But the large number of clock variables present in typical models often lead to huge grid automata, due to the exponential number of clock regions and the need to enforce the relationship between the number of clocks and the chosen granularity. In contrast, our approach allows for an ample range of choices of appropriate granularity values, thus leading to more controllable grid automata and to more manageable test suites.

Petrenko and co-workers [18] offer a different approach. In this work the aim is to verify whether a test sequence yields the same outputs when driving the specification and a suspicious configuration. Suspicious configurations are obtained with the aid of expertise from system testers. Although test purposes for modeling properties also need the expertise from system testers, in our work we can model both fail and desired system behavior, allowing for more general testing capabilities. Further, no treatment of continuous time evolution is provided in [18]. In our work, by contrast, we can handle timed systems with context variables.

Jeannet and co-workers [14] use the ioSTS formalism as specification models. In this work, an enumeration approach of data values is used in order to avoid the state space explosion problem. But, since a discretization method is not used, it is problematic to capture continuous time evolution in an appropriate manner in these models, thus making it difficult to test timed properties. Moreover, the method can incur in high costs when calculating constraints using approximations when searching for test sequences. Further, again, the formal model and the proposed method do not deal simultaneously with timed requirements and context transformations.

7 Concluding Remarks

Methods and techniques for automatically generating test cases for critical and reactive systems have been proposed, many of which are based on formal methods. Among those, some deal with continuous time evolution, others allow some form of data flow. But a single formalism capable of treating both these issues simultaneously has not been so far formally developed.

In this work we propose a method to automatically generate and apply test suites for timed systems with context variables. The basic formal model here used is the Timed Input/Output Context Automaton (TIOCA), a generalization of the earlier Timed Input/Output Automaton (TIOA). Then, a general way of discretizing TIOCA was discussed and proven correct. This discretization technique avoids the classical notion of clock regions, and allows for an ample range of granularity values that can be chosen in the discretization process. We demonstrate that the grid automaton thus obtained was capable of homomorphically simulating the original timed context system, and vice-versa. This provided for the development of automatic methods for generating test suites for systems that exhibit both continuous time evolution as well as context transformations.

In order to model specific system properties that are to be put under test over candidate implementations, we use the notion of test purpose models. Together with the notion of synchronous TIOCA product, the discretization algorithm is able to generate grid automata that reflect both the behavior of the original timed context system, as well as properties specified by the test purpose. By automating the extraction of test cases from this grid automaton, the desired test suites can be extracted. We provide detailed proofs of correctness for all properties and results of interest.

For future works, we suggest a formal development of the notion of conformance testing, perhaps using supports and observables, as stated in the text. Further, a more efficient process of extracting test suites could be explored, namely, one that constructs the grid automaton on-the-fly, while extracting test sequences. The grid algorithm could also be used implicitly when testing the complete runs, after those are obtained by combining the test cases and the observable behaviors of the implementation candidates. As another suggestion, one could investigate how to automatically factor out common subwords from test cases, thus obtaining shorter test suites.

References

[1] R. Alur. Timed automata. In *CAV'99*, number 1633 in LNCS, 1999.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] Adilson Luiz Bonifácio and Arnaldo Vieira Moura. A New Timed Discretization Method for Automatic Test Generation for Timed Systems. Technical Report IC-09-31, Institute of Computing, University of Campinas, September 2009.

[4] Rachel Cardell-Oliver. Conformance tests for real-time systems with timed automata specifications. *Formal Aspects of Computing*, 12(5):350–371, 2000.

[5] Steven J. Cunning and Jerzy W. Rozenblit. Automating test generation for discrete event oriented embedded system s. *J. Intell. Robotics Syst.*, 41(2-3):87–112, 2005.

[6] A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi. Timed test cases generation based on state characterization technique. In *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, page 220, Washington, DC, USA, 1998. IEEE Computer Society.

[7] Abdeslam En-Nouaary. A scalable method for testing real-time systems. *Software Quality Control*, 16(1):3–22, 2008.

[8] Abdeslam En-Nouaary and Rachida Dssouli. A guided method for testing timed input output automata. In *TestCom*, pages 211–225, 2003.

[9] Abdeslam En-Nouaary, Rachida Dssouli, and Ferhat Khendek. Timed wp-method: Testing real-time systems. *IEEE Trans. Softw. Eng.*, 28(11):1023–1038, 2002.

[10] H. Fouchal, E. Petitjean, and S. Salva. Testing timed systems with timed purposes. In *RTCSA '00: Proceedings of the Seventh International Conference on Real-Time Systems and Applications*, page 166, Washington, DC, USA, 2000. IEEE Computer Society.

[11] Hacène Fouchal. Conformance testing techniques for timed systems. In *SOFSEM*, pages 1–19, 2002.

[12] A. Gargantini. Conformance testing. In M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science*, pages 87–111. Springer-Verlag, 2005.

[13] R. Gawlick, R. Segala, J. F. Sogaard-Andersen, and N. A. Lynch. Liveness in timed and untimed systems. In *Automata, Languages and Programming*, pages 166–177, 1994.

[14] Bertrand Jeannet, Thierry Jéron, and Vlad Rusu. Model-based test selection for infinite-state reactive systems. In *FMCO*, pages 47–69, 2006.

[15] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. *The Theory of Timed I/O Automata (Synthesis Lectures in Computer Science)*. Morgan & Claypool Publishers, 2006.

[16] K. G. Larsen and W. Yi. Time abstracted bisimulation: Implicit specifications and decidability. *j-LECT-NOTES-COMP-SCI*, 802:160–176, 1994.

[17] Brian Nielsen and Arne Skou. Test generation for time critical systems: Tool and case study. *ecrts*, 00:0155, 2001.

[18] Alexandre Petrenko, Sergiy Boroday, and Roland Groz. Confirming configurations in efsm testing. *IEEE Trans. Softw. Eng.*, 30(1):29–42, 2004.

[19] Alexandre Petrenko and Nina Yevtushenko. Testing from partial deterministic fsm specifications. *IEEE Trans. Comput.*, 54(9):1154–1165, 2005.

[20] Jan Tretmans. Test generation with inputs, outputs, and quiescence. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, volume 1055 of *Lecture Notes in Computer Science*, pages 127–146. Springer, 1996.

[21] Jan Tretmans. Testing concurrent systems: A formal approach. In J.C.M Baeten and S. Mauw, editors, *CONCUR '99: Proceedings of the 10th International Conference on Co ncurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 46–65, London, UK, 1999. Springer-Verlag.

[22] Jan Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing*, pages 1–38, 2008.

[23] Chang-Jia Wang and Ming T. Liu. Generating test cases for efsm with given fault models. In *INFOCOM*, pages 774–781, 1993.