Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 255 (2009) 103–118
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

SAT-based Verification for Timed Component Connectors

Stephanie Kemper[1]

Centrum Wiskunde & Informatica, Amsterdam, The Netherlands, s.kemper@cwi.nl

Abstract

Component-based software construction relies on suitable models underlying components, and in particular the coordinators which orchestrate component behaviour. Verifying correctness and safety of such systems amounts to model checking the underlying system model, where model checking techniques not only need to be correct but—since system sizes increase—also scalable and efficient.
In this paper, we present a SAT-based approach for bounded model checking of Timed Constraint Automata. We present an embedding of bounded model checking into propositional logic with linear arithmetic, which overcomes the state explosion problem to deal with large systems by defining a product that is linear in the size of the system. To further improve model checking performance, we show how to embed our approach into an extension of counterexample guided abstraction refinement with Craig interpolants.

Keywords: Timed Constraint Automata, Abstraction Refinement, Model Checking, SAT, Component-based Software Engineering

1 Introduction

Component-based software engineering amounts to constructing large systems by composing individual components. The correctness and safety of these concurrent systems depend on actions that happen at the *right time*, i.e., before or after a certain deadline, or within a certain time interval. As components are often available as black boxes only, timed coordination has to be done by the time-aware component connectors. Timed Constraint Automata [4] (TCA) have been originally defined as a semantical model for the coordination language $\mathcal{R}$eo [3]. They offer a powerful stand-alone coordination mechanism for implementing coordinating connectors in networks of timed components exchanging data through multiple channels.

The computational complexity introduced by the infinite state space of these *real-time systems* leads to severe limitations in scalability even within very well-established model checkers like Uppaal (http://www.uppaal.com). Aside from the

[1] Part of this research has been funded by the Dutch BSIK/BRICKS project.

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2009.10.027

omnipresent state explosion problem [9] already present in finite state model checking, current model checking techniques for real-time systems are still limited in the number of concurrent quantitative temporal observations (measured by clocks). A particularly dramatic cause of the state explosion problem is the exponential blow-up obtained by forming the cross product for parallel composition of TCA. To avoid this, we define a linear-size parallel composition for the logical representation of TCA. Typically, only a reduced part of the full parallel composition has to be expanded from our representation during satisfiability checking (SAT solving).

Very sophisticated and well-optimised techniques (e.g., [17]) guide high-end SAT solvers to explore only a comparably narrow fragment around the part of the state space relevant for the particular safety property. We build upon this development by choosing a linear arithmetic/propositional encoding, a philosophy that has successfully proven its great potential in finite state systems [8]. With this basis, we exploit the particularities of transition systems induced by TCA using abstraction refinement to deal with the challenges of infinite states.

Timed Constraint Automata.

TCA are a combination of constraint automata [5] (CA) and timed automata (TA) with location invariants [2,1], offering a powerful coordination mechanism to model coordinators in dynamically reconfiguring networks of a static number of components. Rather than having direct static connections between the components, communication is orchestrated by connectors "in the middle", which impose a certain communication pattern—for example delays—on associated components. Moreover, the coordinator can dynamically reconfigure the network (connections), by sending data values received from an input component to different output components (and vice versa). Still, from the component's perspective, communication happens via the same connection in all cases. The major conceptual difference to other timed models like TA is that a positive amount of time is required to elapse before every visible data flow. This reflects the idea that actions which happen at the same time are truly atomic and thus collapse to a single transition.

Abstraction Refinement.

Abstraction refinement [9,12] is a promising direction of research to overcome the challenges of the state explosion problem and infinite state model checking, while preserving correctness of verification results. Abstraction techniques over-approximate system behaviour by removing constraints that are considered irrelevant for verifying a particular specification. If the abstract system is safe (no error state is reachable) then, by conservative over-approximation, so is the original.

Based on the representation of TCA in propositional logic with linear arithmetic, the iterative abstraction refinement loop consists of the following steps: applying the abstraction function to the representation, we automatically produce a simpler *abstract* version of it. After *unfolding* the resulting transition formula k times, a *satisfiability check* solves the bounded reachability question in the abstract system. Depending on the outcome, the system has either been proven safe (error state is

unreachable) within bound k, or needs to be analysed with respect to an abstract counterexample (*concretised*), again using SAT solving. If the abstract counterexample has a counterpart in the non-abstracted system, then the real-time system is unsafe. Otherwise, the counterexample is *spurious* and results from an inappropriate choice of abstraction. Analysing the counterexample (with Craig interpolants derived by the SAT solver, e.g. FOCI (based on [16]) or MathSAT [15]) then helps to *refine* the abstraction and start over until the system is proven safe (within bound k) or unsafe.

Implementation

We have extended the CA editor in the Eclipse Coordination Tools [10] to support generation and editing of TCA (including various syntactical checks, e.g. well-formedness of clock constraints). Within this platform, we have implemented the translation of TCA to propositional formulas with linear arithmetic constraints (front end), as described in this paper. Further, we have implemented the generation of input files for the MathSAT solver (back end), allowing us to analyse the underlying TCA in detail. Having split the formula generation in two parts, it is very easy to switch to another solver, by just exchanging the back end. This implementation is scheduled to be part of the next release of the Eclipse Coordination Tools.

Organisation of the Paper.

In the next section, we discuss some related work. After introducing TCA and bounded model checking (BMC) in Section 3, we present a faithful representation of TCA in propositional logic with linear arithmetic for BMC in Section 4, and give some soundness results. In Section 5, we introduce a uniform abstraction, extend the algebraic perspective on soundness from Section 4 to correspondence results about abstraction, and briefly recall how to exploit spurious counterexamples for refining abstractions. Section 6 concludes the paper and discusses some future work.

2 Related Work

Blechmann and Baier [7] present a purely propositional symbolic encoding of CA, tailored for finite-state bisimulation checking using ordered binary decision diagrams. While it is not clear how to integrate timing information into their approach (and how to handle the induced infinite state space), our approach in addition is specifically fitted for abstraction refinement.

Jhala and McMillan [13] present an abstraction refinement approach for predicate abstraction. Using interpolants, they generate refinements which take into account specific characteristics of the property. A limitation, however, is the fact that their approach relies on an appropriate choice of predicates for predicate abstraction. Our approach can be considered as a quick (hence, scalable) approximation of predicate abstraction, where predicate discovery is evident by exploiting the nature of TCA.

The abstraction refinement framework presented by Clarke *et al.* [9] works with Kripke structures originating from finite state programs. In contrast, our approach deals with the challenges of infinite state model checking, as introduced by the notion of real-time clocks. Further, we directly use a formula representation which is tailored for SAT-based bounded model checking.

In this paper, we build on the SAT-based approach for TA presented by Kemper and Platzer [14], but take into account the special transition characteristics of TCA (namely that a positive amount of time has to elapse before every visible transition), and include data and data constraints. We define an abstraction function that is simple, yet powerful, and is able to preserve more information in the abstract case than the corresponding abstraction function in [14], which reduces the number of spurious counterexamples.

The model checker Vereofy (`http://www.vereofy.de`) provides tools for model checking (untimed) CA, but to the best of our knowledge, the framework presented in this work is the first approach for model checking TCA.

3 Timed Constraint Automata

In this section, we introduce the standard notations for TCA [4] in the dense time domain $\texttt{Time}=\mathbb{R}_{\geq 0}$, and for BMC [8], and we present our running example.

3.1 Syntax

In the sequel, let $\mathcal{N}$ be a finite, nonempty set of ports, through which TCA exchange data values, and let $\mathcal{D}ata$ be a finite data domain of possible data values which can be sent or received via ports. For simplicity of representation, we assume $\mathcal{D}ata$ contains a special element $\perp$ representing "no data".

Definition 3.1 [Data Constraint, Clock Constraint] A *data assignment* $\delta{\in}DA(\mathcal{N})$ *over (data domain* $\mathcal{D}ata$ *and) port set* $\mathcal{N}$ is a mapping $\delta{:}\mathcal{N}{\to}\mathcal{D}ata$, assigning to each port $A{\in}\mathcal{N}$ the data value which is (currently) pending at A.[2] We use the shorthand notation d_A for the value $\delta(A)$. A *clock valuation* $\nu{\in}\mathcal{V}(\mathcal{X})$ *over a finite set of clocks* $\mathcal{X}$ is a mapping $\nu{:}\mathcal{X}{\to}\texttt{Time}$ assigning to each clock $x{\in}\mathcal{X}$ an element from the time domain `Time`, its current value.

Data constraints $dc{\in}DC(\mathcal{N})$ *over (data domain* $\mathcal{D}ata$ *and) port set* $\mathcal{N}$, and *clock constraints* $cc{\in}CC(\mathcal{X})$ *over* $\mathcal{X}$ are defined as follows:

$$dc ::= \texttt{true} \mid d_A{=}d \mid dc_1 \wedge dc_2 \mid \neg dc, \text{with } A{\in}\mathcal{N} \text{ and } d{\in}\mathcal{D}ata$$
$$cc ::= \texttt{true} \mid x{\sim}n \mid cc_1 \wedge cc_2, \text{with } x{\in}\mathcal{X}, n{\in}\mathbb{N} \text{ and } \sim \in\{<,\leq,=,\geq,>\}.$$

We use the symbol $\models$ for the standard satisfaction relation on data and clock constraints. To ensure that clock constraints hold among subsequent steps, we assume them to be *convex*, i.e., they do not contain $\vee, \neg$ [1]. This property is used

[2] If no data is pending at port A, $\delta(A)$ evaluates to the special value "no data".

for efficient representation. Non-convex clock constraints however can be simulated by splitting locations (for invariants) respectively transitions (for guards).

Definition 3.2 [Timed Constraint Automaton] A *TCA (over data domain $\mathcal{D}ata$)* is a tuple $\mathcal{T}=(S,\mathcal{X},\mathcal{N},E,s_0,I)$, with S a finite set of locations, $s_0\in S$ the initial location, $\mathcal{X}$ a finite set of clocks, $\mathcal{N}$ a finite set of ports, $I:S\to\mathcal{CC}(\mathcal{X})$ a function assigning a clock constraint (*location invariant*) to every location, and the finite set of transitions $E\subseteq S\times 2^{\mathcal{N}}\times\mathcal{DC}(\mathcal{N})\times\mathcal{CC}(\mathcal{X})\times 2^{\mathcal{X}}\times S$. For a transition $t=(s,N,dc,cc,X,s')\in E$, we require $dc\in\mathcal{DC}(N)$ (*data guard of* t) and $cc\in\mathcal{CC}(N)$ (*clock guard of* t), and both to be satisfiable. X is called *clock set of* t, and N is called *port set of* t; if $N=\emptyset$, transition t is called *invisible*, otherwise, it is called *visible*.

The idea of visible and invisible transitions is that the latter do not represent observable data flow (as no ports are involved), but just serve for internal synchronisation purposes, for example by resetting clocks. The former correspond to observable behaviour, namely data flow through all ports contained in the port set of the transition.

Remark 3.3 [Notation of TCA] If not stated otherwise, below we shall always assume the constituents of a TCA $\mathcal{T}$ to be denoted as $\mathcal{T}=(S,\mathcal{X},\mathcal{N},E,s_0,I)$.

Within a system of TCA, two automata synchronise if the port sets of the involved transitions coincide on common ports. This gives rise to the following definition.

Definition 3.4 [TCA Product] Let $\mathcal{T}_i=(S_i,\mathcal{X}_i,\mathcal{N}_i,E_i,s_{0,i},I_i)$, $i=1,2$, be TCA, with $\mathcal{X}_1\cap\mathcal{X}_2=\emptyset$ and $S_1\cap S_2=\emptyset$ (can be achieved by renaming). The *product of* $\mathcal{T}_1$ *and* $\mathcal{T}_2$, denoted $\mathcal{T}_1\bowtie\mathcal{T}_2$, is a new TCA $\mathcal{T}_1\bowtie\mathcal{T}_2=(S_1\times S_2,\mathcal{X}_1\cup\mathcal{X}_2,\mathcal{N}_1\cup\mathcal{N}_2,E,(s_{0,1},s_{0,2}),I)$, with $I:S_1\times S_2\to\mathcal{CC}(\mathcal{N}_1\cup\mathcal{N}_2)$ such that $I(s_1,s_2)=I_1(s_1)\wedge I_2(s_2)$, and E is defined by

$$\frac{\begin{array}{c}(s_1,N_1,dc_1,cc_1,X_1,s_1')\in E_1\\(s_2,N_2,dc_2,cc_2,X_2,s_2')\in E_2\\N_1\cap\mathcal{N}_2=N_2\cap\mathcal{N}_1,N_1\neq\emptyset,N_2\neq\emptyset,dc_1\wedge dc_2\neq\texttt{false}\end{array}}{(\langle s_1,s_2\rangle,N_1\cup N_2,dc_1\wedge dc_2,cc_1\wedge cc_2,X_1\cup X_2,\langle s_1',s_2'\rangle)\in E}\quad(1)$$

$$\frac{(s_1,N_1,dc_1,cc_1,X_1,s_1')\in E_1,N_1\cap\mathcal{N}_2=\emptyset,s_2\in S_2}{(\langle s_1,s_2\rangle,N_1,dc_1,cc_1,X_1,\langle s_1',s_2\rangle)\in E}\quad(2)$$

and the symmetric rule of the latter.

Rule (1) captures the synchronisation of visible transitions: the nonempty port sets have to coincide on common ports, i.e. data flows through the same set of shared ports on both transitions. The case where $N_1\cap\mathcal{N}_2=N_2\cap\mathcal{N}_1=\emptyset$ (i.e., the set of shared ports is empty) represents a system step where each automaton performs a *local* visible transition. Rule (2) describes the execution of a local transition (visible or invisible) in one automaton, while the other automaton remains in its current location. Note that in case this local transition is preceded by a time delay, the other automaton actually performs a delay transition.

3.2 Semantics

In TCA, a positive amount of time has to elapse before every visible transition, while invisible transitions may be instantaneous. The underlying idea is that all actions which happen at the same time atomically collapse to a single transition. Other timed models, like e.g. TA, allow to execute a sequence of visible transitions without delays in between, such that a sequential order is imposed on actions which conceptually happen at the same time. Such behaviour is ruled out in the semantics of TCA, which is defined as the set of runs of the associated labelled transition system (LTS) $\mathcal{S}_{\mathcal{T}}$ [4].

Configurations $\langle s, \nu \rangle$ of $\mathcal{S}_{\mathcal{T}}$ consist of a location s and a clock valuation ν, such that ν satisfies the invariant $I(s)$ of s. A delayed transition $\langle s, \nu \rangle \xrightarrow{N,\delta,t} \langle s', \nu{+}t[X{:=}0] \rangle$ in $\mathcal{S}_{\mathcal{T}}$, with $t{>}0$, results from a transition (s, N, dc, cc, X, s') in $\mathcal{T}$. It increases all clocks by the same amount of time t (delay), such that the guard cc is satisfied afterwards, data flows through all ports in N, while the data assignment δ satisfies the data constraint dc, and all clocks in X are reset to zero. In addition, an invisible transition $(s, \emptyset, \texttt{true}, cc, X, s')$ in $\mathcal{T}$ gives rise to an instantaneous transition $\langle s, \nu \rangle \xrightarrow{\emptyset,\emptyset,0} \langle s', \nu[X{:=}0] \rangle$ in $\mathcal{S}_{\mathcal{T}}$, where ν satisfies the guard cc, and all clocks in X are set to zero. A *run of $\mathcal{S}_{\mathcal{T}}$ starting in configuration q*, denoted $\mathbf{q}$, is a sequence of transitions $\mathbf{q}{=}q \xrightarrow{N,\delta,t} q_1 \xrightarrow{N',\delta',t'} \ldots$ which is either time divergent (i.e. infinite, and $t{+}t'{+}\ldots{=}\infty$) or finite and ends in a *terminal configuration* $\langle s, \nu \rangle$ (i.e. without outgoing transitions, allowing for infinite passage of time: $\forall t{>}0{:}\nu{+}t \models I(s)$). The *trace semantics of $\mathcal{T}$* is given by the set $Run_{\mathcal{T}}$ of initial runs (i.e., starting in the initial configuration) of $\mathcal{S}_{\mathcal{T}}$. With $Run_{\mathcal{T},k}$, we denote the set of finite prefixes of elements of $Run_{\mathcal{T}}$ of (at most) length k.

3.3 Example

An example for a TCA network with dynamic reconfiguration is shown in Figure 1. We assume $\mathcal{D}ata{=}\{1, 2\}$ (thus, actually $\mathcal{D}ata{=}\{1, 2, \perp\}$, and we omit constraints equal to `true` as well as empty sets on transitions.

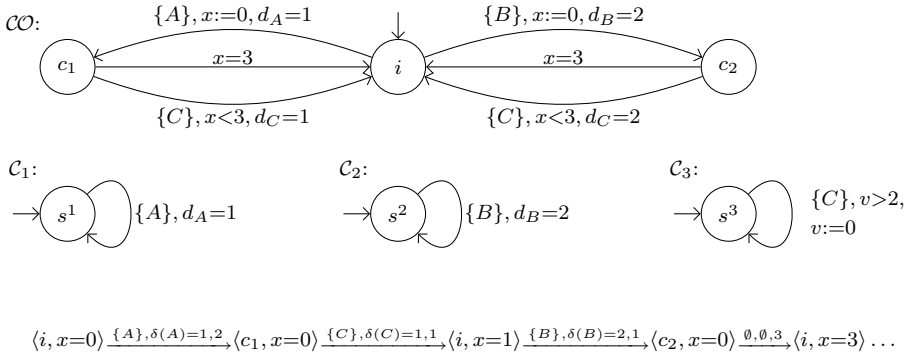

Fig. 1. Dynamic Reconfiguration of a Network

The automata represent a network of three simple components ($\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$, in the middle), together with a coordinator ($\mathcal{CO}$, at the top) which orchestrates data flow between the components. The general idea of the coordinator is to repeatedly receive input from either component $\mathcal{C}_1$ (through port A) or $\mathcal{C}_2$ (through port B), and to send the received data value to component $\mathcal{C}_3$ (through port C), which delays for at least 2 time units between the receiving of subsequent data items. Further, the connector only accepts data value 1 from $\mathcal{C}_1$, and data value 2 from $\mathcal{C}_2$. The reconfiguration of connecting either $\mathcal{C}_1$ or $\mathcal{C}_2$ to $\mathcal{C}_3$ is done purely by the coordinator; from the perspective of $\mathcal{C}_3$, nothing changes, since data always arrives through the same port C. In addition, the coordinator has a timeout constraint: if the received data item is not accepted by $\mathcal{C}_3$ within 3 time units, it is lost, which is represented by the invisible transitions with guard $x{=}3$. An example run of the coordinator is shown at the bottom of Figure 1.

3.4 Bounded Model Checking

Bounded model checking (BMC) has turned out to be amongst the most promising approaches for verification of safety properties [8]. The principle is to examine prefix fragments of the transition system, and successively increase the exploration bound until it reaches (a computable indicator of) the diameter of the system—in which case the system has been proven safe—or an unsafe run has been discovered.

Definition 3.5 [Bounded safety] Let $\mathcal{T}$ be a TCA, let $s{\in}S$ be an error location. $\mathcal{T}$ is *safe with respect to s within bound k*, denoted by $\mathcal{T}{\models_k}\neg\exists\Diamond s$, if there is no run in $Run_{\mathcal{T},k}$ ending in s. Otherwise, $\mathcal{T}$ is called *unsafe with respect to s*.

The lifting of $\neg\exists\Diamond s$ to reason about configurations rather than locations is straightforward. On the basis of these reachability properties, other bounded LTL specifications can be verified as well, using the encoding in [6].

4 Representation of Timed Constraint Automata

In this section, we construct a formula $\varphi(\mathcal{T})$ in propositional logic with linear arithmetic that represents the behaviour of a TCA $\mathcal{T}$ (given by the runs of $\mathcal{S}_\mathcal{T}$, cf. Section 3.2), by defining transition characteristics from step `t-1` to step `t`. For BMC, we unfold $\varphi(\mathcal{T})$ k times (for `k` steps), which yields a formula $\varphi(\mathcal{T})_k$ representing all (prefixes of) runs of $\mathcal{S}_\mathcal{T}$ for `k` steps. This formula, together with a representation of the safety property, is unsatisfiable iff $\mathcal{T}$ is safe within bound k.

4.1 Basic Components

The possible behaviour of a TCA depends on the values of its constituents (clocks, locations, data pending at ports), and changes over time. Therefore, we "parameterise" the variables representing these constituents by the step t they are evaluated in, and we call this *localisation*: the localisation ψ_t of a formula ψ is obtained by

adding index t to all variable symbols occurring in ψ. Thus, if ψ is of vocabulary x, s, d, ψ_t is of vocabulary $\mathtt{x_t}, \mathtt{s_t}, \mathtt{d_t}$ instead. In particular, we use:

Locations For every location $s \in S$, the Boolean variable $\mathtt{s_t}$ represents whether the TCA is in location s in step $\mathtt{t}$.

Data Values, Ports The injective mapping $\Delta: \mathcal{D}ata \to \mathbb{N}$ assigns a natural number $\mathtt{t^i}$ to each element d_i of $\mathcal{D}ata$, with $1 \leq \mathtt{t^i} \leq |\mathcal{D}ata|$, and $\mathtt{t}^{|\mathcal{D}\mathtt{ata}|} \stackrel{\text{def}}{=} \mathtt{t}^{\perp}$ representing $\perp$. For every port $A \in \mathcal{N}$, the Boolean *activity variable* $\mathtt{A_t}$ *of* A represents whether data flows through A in step t, and the natural *data variable* $\mathtt{DA_t}$ *of* A represent which data occurs at A in step $\mathtt{t}$ (in case of no data flow, $\mathtt{DA_t}$ evaluates to $\mathtt{t}^{\perp}$).

Data Constraints For a data constraint $d_A = d_i$, with $\Delta(d_i) = \mathtt{t^i}$, the formula $\mathtt{A_t} \wedge (\mathtt{DA_t} = \mathtt{t^i})$ evaluates to $\mathtt{true}$ iff $\delta(A) = d_i$ in step $\mathtt{t}$.

Clocks For every clock $x \in \mathcal{X}$, the rational variable $\mathtt{x_t}$ (*clock reference*) represents the absolute point in time where x was last reset prior to step $\mathtt{t}$. An additional rational variable $\mathtt{z_t}$ (*absolute time reference*) represents the absolute amount of time that has passed until step $\mathtt{t}$. The clock value of x at step $\mathtt{t}$ is thus obtained by $\mathtt{z_t - x_t}$. Note that linear arithmetic is equisatisfiable for rational and real variables [14].

Clock Constraints For a clock constraint $cc = x \sim n$ (cf. Definition 3.1), the formula $\mathtt{z_t - x_t} \sim n$, denoted $\mathtt{cc_t}$, evaluates to $\mathtt{true}$ iff cc holds in step $\mathtt{t}$, and the formula $\mathtt{z_t - x_{t\text{-}1}} \sim n$, denoted $\mathtt{cc_{t\Delta}}$ and called *inter-step representation*, evaluates to $\mathtt{true}$ iff cc holds in step $\mathtt{t}$ and x has not been reset since step $\mathtt{t\text{-}1}$.

The representation of other constraints is straightforward, by using conjunctions (and negations, in case of data constraints) of the aforementioned representations.

The inter-step representation is needed for correct representation of delayed transitions in $\mathcal{S}_{\mathcal{T}}$, cf. Section 3.2: the invariant of the target location s' is evaluated under the valuation $\nu + t[X:=0]$, that means *after* the time delay and *after* the execution of the transition. In contrast, the invariant of the source location s and the clock guard of the transition are evaluated under the valuation $\nu + t$, that means *after* the passage of time, but *before* the execution of the transition. The inter-step representation is used to access the clock value at this particular point in time "in the middle" of the execution step.

4.2 Transition Relation

The representation of the transition relation needs to take care of the special behaviour of TCA, namely, that every visible transition has to be preceded by a positive time delay, whereas invisible transitions may be instantaneous. It constrains the possible valuations of variables representing the configuration at subsequent step $\mathtt{t}$ depending on those at step $\mathtt{t\text{-}1}$. Conceptually, the delay is represented by evolving from $\mathtt{t\text{-}1}$ to $\mathtt{t}$, while the (instantaneous) location change takes place at $\mathtt{t}$.

Definition 4.1 [Timed Constraint Automaton Representation] Let $\mathcal{T}$ be a TCA, let $e = (s, N, dc, cc, X, s')$ and $e' = (s, \emptyset, \mathtt{true}, cc, X, s')$ be a visible respectively invisible transition in $\mathcal{T}$. The *formula representation* $\varphi(\mathcal{T})$ *of the transition relation of* $\mathcal{T}$

is defined in (10) in Figure 2.

$$\varphi^i(\mathcal{T}) = \bar{\mathtt{s}}_0 \wedge \mathtt{I}(\bar{\mathtt{s}})_0 \wedge \bigwedge_{s\in S, s\neq\bar{s}} \neg \mathtt{s}_0 \wedge \bigwedge_{A\in\mathcal{N}} (\neg \mathtt{A}_0 \wedge (\mathtt{DA}_0 = \mathtt{t}^\perp)) \wedge \bigwedge_{x\in X} (\mathtt{x}_0 = 0) \wedge (\mathtt{z}_0 = 0) \tag{3}$$

$$\varphi^v(e) = \mathtt{s}_{t\text{-}1} \wedge \bigwedge_{A\in N} \mathtt{A}_t \wedge \bigwedge_{A\notin N} \neg \mathtt{A}_t \wedge \mathtt{dc}_t \wedge \bigwedge_{x\in X} (\mathtt{x}_t = \mathtt{z}_t) \wedge \bigwedge_{x\notin X} (\mathtt{x}_t = \mathtt{x}_{t\text{-}1}) \wedge (\mathtt{z}_{t\text{-}1} < \mathtt{z}_t) \wedge \mathtt{cc}_{t\Delta} \wedge \mathtt{I}(\mathtt{s})_{t\Delta} \wedge \mathtt{s'}_t \tag{4}$$

$$\varphi^\tau(e) = \mathtt{s}_{t\text{-}1} \wedge \bigwedge_{A\in\mathcal{N}} \neg \mathtt{A}_t \wedge \bigwedge_{x\in X} (\mathtt{x}_t = \mathtt{z}_t) \wedge \bigwedge_{x\notin X} (\mathtt{x}_t = \mathtt{x}_{t\text{-}1}) \wedge (\mathtt{z}_{t\text{-}1} \leq \mathtt{z}_t) \wedge \mathtt{cc}_{t\Delta} \wedge \mathtt{I}(\mathtt{s})_{t\Delta} \wedge \mathtt{s'}_t \tag{5}$$

$$\varphi^E(\mathcal{T}) = \bigvee_{e\in E, N\neq\emptyset} \varphi^v(e) \vee \bigvee_{e\in E, N=\emptyset} \varphi^\tau(e) \tag{6}$$

$$\varphi^S(\mathcal{T}) = \bigwedge_{s\in S} (\neg \mathtt{s}_t \vee \mathtt{I}(\mathtt{s})_t) \wedge \bigwedge_{s'\in S, s\prec s'} \neg(\mathtt{s}_t \wedge \mathtt{s}'_t) \tag{7}$$

$$\varphi^D(\mathcal{T}) = \bigwedge_{A\in\mathcal{N}} (\neg \mathtt{A}_t \Leftrightarrow \mathtt{DA}_t = \mathtt{t}^\perp) \wedge \bigwedge_{A\in\mathcal{N}} ((\mathtt{DA}_t \geq \mathtt{t}^1) \wedge (\mathtt{DA}_t \leq \mathtt{t}^\perp)) \tag{8}$$

$$\varphi^d(s) = \mathtt{s}_{t\text{-}1} \wedge \mathtt{s}_t \wedge (\mathtt{z}_{t\text{-}1} \leq \mathtt{z}_t) \wedge \bigwedge_{A\in\mathcal{N}} \neg A \wedge \bigwedge_{x\in\mathcal{X}} (\mathtt{x}_t = \mathtt{x}_{t\text{-}1}) \tag{9}$$

$$\varphi(\mathcal{T}) = \varphi^i(\mathcal{T}) \wedge \varphi^E(\mathcal{T}) \wedge \varphi^S(\mathcal{T}) \wedge \varphi^D(\mathcal{T}) \tag{10}$$

$$\varphi(\mathcal{T})_k = \bigwedge_{1\leq j\leq k} \varphi(\mathcal{T})_{j/t} \tag{11}$$

$$\varphi(\mathcal{T}_1 \bowtie \mathcal{T}_2) = \varphi(\mathcal{T}_1) \wedge \varphi(\mathcal{T}_2) \tag{12}$$

$$\varphi(\mathcal{T}_1 \bowtie \mathcal{T}_2)_k = \bigwedge_{1\leq j\leq k} \varphi(\mathcal{T}_1 \bowtie \mathcal{T}_2)_{j/t} \tag{13}$$

Fig. 2. Transition relation representation

The automaton starts in its initial location $\bar{s}$ (3) in step $\mathtt{0}$,[3] the invariant of which has to be satisfied, all clocks are set to zero, and data must not flow through any port. Before executing a visible transition (4) in step $\mathtt{t}$, $\mathcal{T}$ is in location s. After the elapse of a positive amount of time ($\mathtt{z}_{t\text{-}1} < \mathtt{z}_t$), after which the invariant $\mathtt{I}(\mathtt{s})_{t\Delta}$ of s and the clock guard $\mathtt{cc}_{t\Delta}$ of the transition hold, $\mathcal{T}$ switches to location s'. All clocks referenced in the clock set X are set to the actual point in time, while the values of the other clocks do not change. Data flows through all ports A contained in the port set N, while the other ports are inactive, and the data constraint $\mathtt{dc}_t$ is satisfied. Due to convexity, the invariant needs to be checked at the end of the time delay only, as it inductively holds at the beginning (3), (7). The execution of an invisible transition (5) is similar, except that the amount of time elapsed may be zero, and data must not flow through any port. The disjunction of all visible and invisible transitions expresses nondeterministic transition choice (6).

In any step, the current location is unique, and its invariant holds (7) ($\prec$ denotes an arbitrary but fixed order on the location set S). For ports without data flow, the pending data value has to be the special value "no data", and only values from the domain $\mathcal{D}ata$ may be pending at the ports (8).

4.3 Unfolding for Bounded Model Checking

In order to represent the reachability problem of BMC for a TCA $\mathcal{T}$ in logic, the formula representation $\varphi(\mathcal{T})$ (10) is unfolded, i.e., instantiated for all steps up to bound k. The resulting formula $\varphi(\mathcal{T})_k$ is called *k-unfolding of $\mathcal{T}$*, and is defined in (11), where $\psi_{j/t}$ denotes the localisation of ψ, with index t replaced by j.

[3] To avoid confusion with localisation indices, in (3) we denote the initial location as $\bar{s}$ rather than s_0, so its representation is $\bar{\mathtt{s}}_0$ rather than the odd-looking $(\mathtt{s}_0)_0$.

Intuitively, a satisfying interpretation (model) of $\varphi(\mathcal{T})_k$ corresponds to a run of $\mathcal{S}_\mathcal{T}$ of length k, i.e., to one possible behaviour of $\mathcal{T}$ for the first k steps. Checking the reachability of an error location s amounts to conjoining $\varphi(\mathcal{T})_k$ with the representation $\rho \stackrel{\text{def}}{=} \mathtt{s}_0 \vee \mathtt{s}_1 \vee \ldots \vee \mathtt{s}_k$ of the reachability property, such that $\mathcal{T} \models_k \neg\exists\Diamond s$ holds iff the conjunction $\varphi(\mathcal{T})_k \wedge \rho$ is unsatisfiable. Lifting ρ to reason about configurations or even execution sequences is straightforward. For example, an LTL property $s \rightarrow \bigcirc s'$ can be represented as $\rho = (\mathtt{s}_0 \wedge \mathtt{s'}_1) \vee (\mathtt{s}_1 \wedge \mathtt{s'}_2) \vee \ldots (\mathtt{s}_{k-1} \wedge \mathtt{s'}_k)$.

4.4 Example

Consider again the TCA $\mathcal{C}_3$ in Figure 1. With $\mathcal{D}ata = \{1, 2, \perp\}$, and Δ such that $\Delta(1)=1$, $\Delta(2)=2$, and $\Delta(\perp)=3$, the representation of $\mathcal{C}_3$ according to Definition 4.1 is shown in Figure 3 (we omit constraints equal to `true`).

$\varphi^i(\mathcal{C}_3) = \mathtt{s}^3{}_0 \wedge \neg \mathtt{C}_0 \wedge (\mathtt{DC}_0 = 3) \wedge (\mathtt{v}_0 = 0) \wedge (\mathtt{z}_0 = 0)$

$\varphi^E(\mathcal{C}_3) = \mathtt{s}^3{}_{t-1} \wedge \mathtt{C}_t \wedge (\mathtt{v}_t = \mathtt{z}_t) \wedge (\mathtt{z}_{t-1} < \mathtt{z}_t) \wedge (\mathtt{z}_t - \mathtt{v}_{t-1} > 2) \wedge \mathtt{s}^3{}_t$

$\varphi^D(\mathcal{C}_3) = (\neg \mathtt{s}^3{}_t \Leftrightarrow (\mathtt{DC}_t = 3)) \wedge (\mathtt{DC}_t \geq 1) \wedge (\mathtt{DC}_t \leq 3)$

$\varphi(\mathcal{C}_3) = \varphi^i(\mathcal{C}_3) \wedge \varphi^E(\mathcal{C}_3) \wedge \varphi^D(\mathcal{C}_3)$

Fig. 3. TCA Representation Example

4.5 Product of Timed Constraint Automata

The cross product of TCA, as defined in Definition 3.4, is exponential in the worst case, which is a severe limitation to the size of systems that can be verified. We define a logical representation of systems of TCA which is *linear* in the number of automata. The basic idea is to retain the representations of the individual automata, and model check them "in parallel". We require variables representing common ports to have the same name in both representations, such that constraints involving these ports are automatically satisfied simultaneously in both representation.

To model single local transitions, as described by (1) in Definition 3.4, we introduce explicit delay transitions (cf. Section 3.1): the representation of a delay transition $\varphi^d(s)$ in location s is defined in (9). Note that these delay transitions are in accordance with Definition 3.2, as they correspond to invisible loops of the form $(s, \emptyset, \mathtt{true}, \mathtt{true}, \emptyset, s)$. Therefore, in particular, (9) permits zero-delays. For two TCA $\mathcal{T}_1$ and $\mathcal{T}_2$, with $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$ and $S_1 \cap S_2 = \emptyset$ (can be achieved by renaming), the *representation of* $\mathcal{T}_1 \bowtie \mathcal{T}_2$, denoted $\varphi(\mathcal{T}_1 \bowtie \mathcal{T}_2)$, is given in (12), where (6) is understood to be the disjunction of (4), (5) and (9). The *k-unfolding of the product* is defined in the same way as for individual automata, it is shown in (13).

Note that the existence of such a linear product is not immediately clear, but in fact is a result of our design decision of explicitly mentioning all ports on each transition (cf. (4), (5) and (9)). This decision—though seeming unnecessary at first glance—together with the assumption that common ports have the same name, ensures that transitions in different TCA may only be executed in parallel if they fulfil the conditions described in Definition 3.4. In this way, we do not need to mention all possible synchronisations explicitly, and thus avoid the exponential blow-up.

Theorem 4.2 (Correctness of representation) *The formula representation of TCA, as defined in Definition 4.1, is correct, that means it exhibits the same behaviour as $\mathcal{T}$.*

We have proven this by showing that every model of $\varphi(\mathcal{T})_k$ corresponds to a run of length k of $\mathcal{T}$, and vice versa. The results directly carry over to the product representation. For a detailed discussion and proof, we refer to the extended version of this paper, available at www.cwi.nl/~kemper.

4.6 Discussion

Using propositional formulas as intermediate representation ("front end"), we may fall back on the abstraction refinement framework of [14] ("back end"), and, more importantly, we can take advantage of existing high-performance SAT solving technologies. Our representation is specifically tailored for SAT solving: in addition to providing conjunctive normal form (CNF) whenever possible, (7) and (8) are binary clauses, while (3) even consists of unit clauses. With respect to speed of verification, binary clauses are very efficient: the 2-SAT problem is polynomial. Though—due to the disjunctive nature of transition choices—(6) is not in CNF, it can easily be transformed to short CNF (see e.g. [11]) when introducing new symbols.

The restriction to convex clock constraints does not reduce the expressiveness of our model (cf. Section 3.1), but on the contrary significantly simplifies the representation formulas, since clock constraints need to be checked at the beginning and at the end of a time delay only, rather than at all intermediate points (cf. (7)). We further simplify verification by defining a product representation which is linear in the number of automata (12). In this way, we also avoid the exponential state space blow-up when forming the cross product.

Though communication is often regarded as a one-to-one relation, our representation is already suited for general n-ary communication: by having ports carrying the same name in more than two automata, our approach naturally generalises to one-to-many or even many-to-many communication models.

5 Abstraction

In this section, we show how to adapt the abstraction technique of *abstraction by merging omission (MO)* [14] to our representation. MO is a simple and fast but nevertheless powerful abstraction technique specifically tailored to work on logical formulas. The removal of constraints considered irrelevant to the particular safety property yields an over-approximation.

5.1 Abstraction by Merging Omission

The basic idea of MO is to reduce the system complexity by decreasing the number of symbols in $\varphi(\mathcal{T})$, while retaining as much information about the transition characteristics as possible (the abstract formula is weaker than $\varphi(\mathcal{T})$, though). It is defined for formulas in negation normal form (NNF), to which $\varphi(\mathcal{T})$ can be easily

transformed. MO uniformly works on the different syntactical categories: it merges location and port variables, by mapping them to the same image according to a *map of merging*, and it removes rational variables and arithmetic constraints according to a *set of omission*.

Definition 5.1 [Abstraction by merging omission] Let $\mathcal{T}$ be a TCA, let $\varphi(\mathcal{T})$ be in NNF. Let $\mathtt{S}$, $\mathtt{X}$, $\mathtt{N_A}$ and $\mathtt{N_{DA}}$ be the variable sets representing locations, clocks, port activity variables and port data variables, respectively, all without indices, let $\vartheta{:}\mathtt{N_{DA}}\to\mathtt{N_A}$ be a mapping such that $v\in\mathtt{N_{DA}}$ and $\vartheta(v)=v'\in\mathtt{N_A}$ are data and activity variable of the same port. Let $\mathcal{AS} \subseteq \mathtt{X}\cup\mathcal{CC}(\mathtt{X})\cup\mathcal{DC}(\mathtt{N_{DA}})$ be a set not containing compound formulas, let $\gamma{:}\mathtt{S}\cup\mathtt{N_A}\to\mathtt{S}'\dot\cup\mathtt{N_A}'$ be a mapping to some fresh sets of propositional variables $\mathtt{S}'$ and $\mathtt{N_A}'$.

The *abstraction by merging omission* of $\varphi(\mathcal{T})$ with respect to $\mathcal{AS}$ and γ is defined by applying transformation α, as depicted in Figure 4.

$$\alpha(L) = \begin{cases} L & L \text{ neg., } cont(L)\cap(\mathtt{S}\cup\mathtt{N_A})\neq\emptyset, \gamma(cont(L))=id \quad (14a)\\ L & L \text{ neg., } cont(L)\cap(\mathcal{AS}\cup\mathtt{S}\cup\mathtt{N_A})=\emptyset, \\ & \quad \forall v\in cont(L)\cap\mathtt{N_{DA}}{:}\gamma(\vartheta(v))=id \quad (14b)\\ \gamma(L) & L \text{ pos., } cont(L)\cap\mathcal{AS}=\emptyset, \\ & \quad \forall v\in cont(L)\cap\mathtt{N_{DA}}{:}\gamma(\vartheta(v))=id \quad (14c)\\ \mathtt{true} & \text{otherwise} \quad (14d)\end{cases}$$

$$\alpha(F\wedge G) = \alpha(F)\wedge\alpha(G)$$
$$\alpha(F\vee G) = \alpha(F)\vee\alpha(G)$$

Here, F and G are formulas in NNF, L a literal, $cont(L)$ the set of atomic formulas and variables occurring in L. We understand γ to be applied recursively to elements $v\in cont(L)$ if $v\in S\cup\mathtt{N_A}$: for example, $\gamma(\neg\mathtt{s}) = \neg\gamma(\mathtt{s})$, $\gamma((\mathtt{z}-\mathtt{x})\sim c) = ((\mathtt{z}-\mathtt{x})\sim c)$, and $\gamma(\mathtt{DA}=\mathtt{t}) = (\mathtt{DA}=\mathtt{t})$.

Fig. 4. Abstraction by merging omission

MO uniformly captures abstraction on all syntactic categories contained in $\varphi(\mathcal{T})$: negative propositional variables not meant to be abstracted are kept unchanged (14a), the map γ is applied to positive propositional variables only ((14c), γ is the identity for symbols not meant to be abstracted). Clock constraints not contained in $\mathcal{AS}$ are retained unchanged, for both positive (14c) and negative (14b) constraints. Data constraints are treated in a similar way as clock constraints. However, to guarantee α yields an over-approximation, we may retain only those data constraints that reason about ports not merged by γ, ensured by the constraint $\forall v\in cont(L)\cap\mathtt{N_{DA}}{:}\gamma(\vartheta(v))=id$ in (14b) and (14c). In all other cases, α maps the literal to $\mathtt{true}$ (14d), in this way performing a quick variant of existential abstraction [9], while exploiting the structural relationships of clocks and TCA. This gives rise to the following lemma.

Lemma 5.2 (Abstraction by weakening) *The abstraction MO yields a conservative approximation, that means $\alpha(F)$ is weaker than F in the sense that the implication $F\to\alpha(F)$ is* valid *(true in all models).*

Lifting α to the presence of localisations is straightforward: γ and $\mathcal{AS}$ are understood oblivious to indices in the NNF of $\varphi(\mathcal{T})$, such that indices directly carry over

to $\varphi(\mathcal{T})_k$ unchanged (defining different abstractions for different steps is possible using the same definition of α but we consider it to be less useful). Note that α is homomorphic with respect to $\{\wedge, \vee\}$, which proves the equality of $\alpha(\varphi(\mathcal{T})_k)$ and $\alpha(\varphi(\mathcal{T}))_k$ (except for speed of computing the abstraction, where $\alpha(\varphi(\mathcal{T}))_k$ is superior).

The major difference between our abstraction function and the one presented in [14] is the fact that we do *not* in general map negative propositional variables to `true` (14a). Such abstraction is of course necessary for symbols meant to be abstracted, as $\neg$`s` does not allow to conclude $\neg$`u` in case of a merge γ(`r`)=γ(`s`)=`u`. However, in case γ(`s`)=`s`, i.e. when `s` is kept unchanged, it is safe to keep $\neg$`s` in the abstract formula $\alpha(\varphi(\mathcal{T}))$. Furthermore, with respect to meaningful results, we even consider it necessary to retain $\neg$`s`, as otherwise the abstraction becomes too coarse: mapping all negative propositional symbols to `true`, the abstraction effectively deletes the consistency constraint $\varphi^S(\mathcal{T})$ on locations (7) as well as part of the consistency constraint $\varphi^D(\mathcal{T})$ on data values (8), such that the rest becomes meaningless. In particular, the existence of a TCA $\tilde{\mathcal{T}}$, as claimed in the correctness proof of the abstraction, cannot be guaranteed any more. See the extended version of this paper, available at www.cwi.nl/~kemper, for further details.

5.2 Example

Consider the example in Section 4.4. To abstract from timing information, we choose $\mathcal{AS}$=$\{v\}$, and γ=*id*. The resulting formulas of applying α with respect to γ and $\mathcal{AS}$ to the formulas in Figure 3 are shown in Figure 5.

$\alpha(\varphi^i(\mathcal{C}^3)) = \mathtt{s}^3{}_0 \wedge \neg\mathtt{C}_0 \wedge (\mathtt{DC}_0{=}3) \wedge (\mathtt{z}_0{=}0)$
$\alpha(\varphi^D(\mathcal{C}^3)) = \varphi^D(\mathcal{C}^3)$

$\alpha(\varphi^E(\mathcal{C}^3)) = \mathtt{s}^3{}_{t\text{-}1} \wedge \mathtt{C}_t \wedge (\mathtt{z}_{t\text{-}1}{<}\mathtt{z}_t) \wedge \mathtt{s}^3{}_t$
$\alpha(\varphi(\mathcal{C}^3)) = \alpha(\varphi^i(\mathcal{C}^3)) \wedge \alpha(\varphi^E(\mathcal{C}^3)) \wedge \alpha(\varphi^D(\mathcal{C}^3))$

Fig. 5. TCA Abstraction Example

Theorem 5.3 (Correctness of abstraction) *The abstraction α yields a correct over-approximation on runs.*

This result is already captured by Lemma 5.2. Here, we have proven an even stronger correctness result, by showing the existence of a homomorphism between concrete and abstract sets of runs. For a detailed discussion and proof, we again refer to the extended version of this paper, available at www.cwi.nl/~kemper.

5.3 Abstraction Refinement

In this section, we give a brief overview of our abstraction refinement methodology. The general abstraction refinement paradigm [9] consists of three steps: (1) generate the initial abstraction, (2) model check the abstract system, and, if required, (3) refine the abstraction.

Generate the initial abstraction If there is no additional knowledge about the system, the initial abstraction simply removes all symbols in $CC(\mathtt{X}) \cup DC(\mathtt{N_{DA}})$ from $\varphi(\mathcal{T})$, and merges all symbols in $\mathtt{S}$ to a single one (we refer to [9] for improved techniques), thereby collapsing to a single trivial location (accordingly for ports). Yet, the next refinement iterations will quickly discover more relevant parameters.

Model checking the abstract system If $\alpha(\varphi(\mathcal{T}))_k$, together with a representation ρ of the safety property (cf. Section 4.3), is unsatisfiable, the system is safe within bound k (cf. Definition 3.5, Theorem 4.2 and Theorem 5.3). Otherwise, the counterexample needs to be *concretised*, which amounts to checking $\varphi(\mathcal{T})_k \wedge \rho$, in conjunction with the variable valuations π representing the abstract counterexample, and concretising constraints of the form $\mathtt{u} \rightarrow \mathtt{s} \vee \mathtt{r}$ for all locations and ports s and r with $\gamma(\mathtt{s}) = \gamma(\mathtt{r}) = \mathtt{u}$. This check can be done very quickly, since the single abstract counterexample is highly restrictive. If the conjunction is satisfiable, a counterexample to the property is found. Otherwise, the counterexample is spurious, and the abstraction needs to be refined.

Refining the abstraction To identify ill-abstracted parameters, we stratify the formulas $\varphi(\mathcal{T})_k$, ρ and π (i.e., align them along their unfolding depth k), and derive a sequence of Craig interpolants (e.g. [16]),[4] one for every bisection into prefix and suffix. By definition, both the prefix of the first interpolant $G \equiv \mathtt{false}$ and the suffix of the last interpolant $\tilde{G} \equiv \mathtt{true}$ are unsatisfiable, and, for P being the set of symbols subject to abstraction, at least one of the symbols in $IA \stackrel{def}{=} cont(G) \cap P$ has been wrongly abstracted.

The difficulty—in particular in automatic abstraction refinement—is then to define heuristics describing the application of the two refinement strategies (a) refine a symbol from IA, and (b) rule out the subtrace represented by the common parts of the prefix of G and the suffix of $\tilde{G}$. The former quickly collapses to the concrete system if applied too frequently, while the latter cannot yield results as long as essential parameters are inadequately abstracted. Thus, it is necessary to define heuristics that strike a suitable balance between (a) and (b).

The *fully automatic* heuristic presented in [14] (together with its optimisations) is a compromise between the drawbacks of the two alternatives: after refining a parameter (a), a fixed number of traces (fractions of the unfolding depth k have turned out to be most promising) is ruled out (b) before refining the next symbol according to (a).

5.4 Discussion

We do not have to distinguish between abstraction of different constituents of TCA, since α works uniformly depending just on the different syntactical categories (propositional, natural, real variables), which happen to represent different concepts of TCA. Yet, in contrast to [14], our abstraction function does not remove negative propositional variables from the formula in case the map of merging γ is

[4] A Craig interpolant for an inconsistent pair of formulas (A, B) is a formula C that is implied by *prefix* A, inconsistent with *suffix* B and contains only common symbols of A and B; it is thus an over-approximation of A and an under-approximation of $\neg B$.

the identity for these. This speeds up the verification process, since we preserve a bigger part of the formula structure of $\varphi(\mathcal{T})$, which not only provides more meaningful results, but therefore also results in less cycles in the abstraction refinement loop.

Proving a strong correctness result for the abstraction permits to conclude the existence of a corresponding effective abstraction technique on TCA, which produces the abstract automaton $\tilde{\mathcal{T}}$. Yet, the formalisation of the direct construction will be much less uniform than what has been presented here.

As a second major result of the strong correctness proof, we get that every abstraction satisfying Lemma 5.2 is already proven correct in our framework. The existence of the abstract TCA $\tilde{\mathcal{T}}$, however, is not a general consequence, but a particular result of our strong correctness. This makes α a very powerful and universal technique, yet it remains efficient due to its purely syntactical definition.

6 Conclusion and Future Work

In this paper, we have presented a SAT-based approach for bounded model checking of TCA. We have defined an embedding of bounded model checking for systems of TCA into propositional logic with linear arithmetic, and introduced a uniform logic-based abstraction for clocks, locations, port names and data values. This logical representation directly benefits from state-of-the-art SAT solving techniques, and allows a linear-size representation of parallel composition. We expect the structural relationships underlying the abstraction to provide the basis for a framework to generalise our work to other scenarios.

Besides this, future work includes performance comparisons when using a logarithmic encoding for locations and ports (though automatic abstraction is more involved in that case), and the application and comparison of both variants on case studies. After having defined an *abstraction* that is tailored towards TCA in this paper, naturally the next step is to define tailor-made *refinement heuristics* for TCA, by exploiting the algebraical and logical principles underlying them. As a first step, we plan to add isomorphy inference reasoning to strategy (b) (cf. Section 5.3).

We believe our framework provides means to better understand the functioning of TCA, $\mathcal{R}$eo coordinators and $\mathcal{R}$eo networks [3] (for which TCA serve as formal model). To further improve this, we plan to integrate a back translation from formulas to TCA into the Eclipse Coordination Tools, such that e.g. the result of abstraction can be viewed as a TCA in the editor. Our framework further facilitates verification of these connectors, for example whether an implementation meets its specification. We intend to use the framework within a testing environment of $\mathcal{R}$eo networks, which will enables us to perform black and white box testing, for example check the feasibility of a certain interaction behaviour.

References

[1] Rajeev Alur. Timed automata. In N. Halbwachs and D. Peled, editors, *CAV*, volume 1633 of *LNCS*, pages 8–22. Springer, 1999.

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

[3] Farhad Arbab. $\mathcal{R}$eo: a channel-based coordination model for component composition. *Mathematical. Structures in Comp. Sci.*, 14(3):329–366, 2004.

[4] Farhad Arbab, Christel Baier, Frank S. de Boer, and Jan J. M. M. Rutten. Models and temporal logics for timed component connectors. In *SEFM*, pages 198–207. IEEE Computer Society, 2004.

[5] Farhad Arbab, Christel Baier, Jan J. M. M. Rutten, and Marjan Sirjani. Modeling component connectors in $\mathcal{R}$eo by constraint automata (extended abstract). *Electr. Notes Theor. Comput. Sci.*, 97:25–46, 2004.

[6] Gilles Audemard, Alessandro Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded model checking for timed systems. In Doron Peled and Moshe Y. Vardi, editors, *International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, volume 2529 of *LNCS*, pages 243–259. Springer, November 2002.

[7] Tobias Blechmann and Christel Baier. Checking equivalence for $\mathcal{R}$eo networks. *Electr. Notes Theor. Comput. Sci.*, 215:209–226, 2008.

[8] Edmund M. Clarke, Armin Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[9] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

[10] Eclipse Coordination Tools. http://reo.project.cwi.nl/.

[11] Reiner Hähnle. Short CNF in finitely-valued logics. In Henryk Jan Komorowski and Zbigniew W. Ras, editors, *ISMIS*, volume 689 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 1993.

[12] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In Neil D. Jones and Xavier Leroy, editors, *POPL*, pages 232–244. ACM, 2004.

[13] Ranjit Jhala and Kenneth L. McMillan. Interpolant-based transition relation approximation. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2005.

[14] Stephanie Kemper and André Platzer. SAT-based abstraction refinement for real-time systems. *Electr. Notes Theor. Comput. Sci.*, 182:107–122, 2007.

[15] The MathSAT 4 SMT solver. http://mathsat4.disi.unitn.it/index.html.

[16] Kenneth L. McMillan. An interpolating theorem prover. In Kurt Jensen and Andreas Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2004.

[17] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, pages 530–535. ACM, 2001.