Component-based Abstraction Refinement for Timed Controller Synthesis

Hans-Jörg Peter
Reactive Systems Group
Universität des Saarlandes
Saarbrücken, Germany
peter@cs.uni-sb.de

Robert Mattmüller
Foundations of Artificial Intelligence Group
Albert-Ludwigs-Universität Freiburg
Freiburg, Germany
mattmuel@informatik.uni-freiburg.de

Abstract

We present a novel technique for synthesizing controllers for distributed real-time environments with safety requirements. Our approach is an abstraction refinement extension to the on-the-fly algorithm by Cassez et al. from 2005 [7]. Based on partial compositions of some environment components, each refinement cycle constructs a sound abstraction that can be used to obtain under- and over-approximations of all valid controller implementations. This enables (1) early termination if an implementation does not exist in the over-approximation, or, if one does exist in the under-approximation, and (2) pruning of irrelevant moves in subsequent refinement cycles. In our refinement loop, the precision of the abstractions incrementally increases and converges to all specification-critical components.

We implemented our approach in a prototype synthesis tool and evaluated it on an industrial benchmark. In comparison with the timed game solver UPPAAL-TIGA*, our technique outperforms the nonincremental approach by an order of magnitude.*

1 Introduction

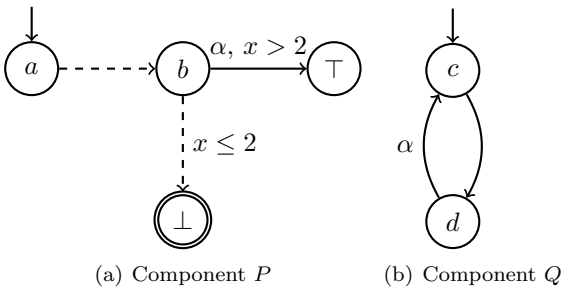
Establishing the correctness of real-time systems is a main research goal in computer science. Model checking and synthesis are two heavily studied formal approaches to automatically achieve this goal. While model checking decides whether a system satisfies a logical property, synthesis derives implementations from formal specifications that are correct by construction.

The reactive nature of embedded devices requires a computational model that considers *open* rather than *closed* systems. In contrast to the latter one, open systems exhibit an input/output interface for interacting with each other. The controller synthesis problem asks whether there exists an implementation of an open system *controller* interacting with an open system *environment* such that a given requirement is always satisfied. Timed controller synthesis, where the environment is given as a distributed real-time system, is an interesting extension because it is both practically relevant (see, e.g., [4, 19, 8]) and theoretically tractable (see [22, 3, 12, 15]).

The increasing distributedness in the design of embedded devices calls for a dense time model and gives rise to the state space explosion problem. Synthesizing a real-time controller for a complex distributed environment is an inherently difficult task because of the underlying dense time domain and the exponentially growing discrete control structure. The main goal of our approach is to reduce the complexity of a global synthesis problem that considers all components to a sequence of simpler local synthesis problems that consider only some of the components.

The solution to each local synthesis problem yields a sound approximation of all valid global controller implementations which, in turn, enables early termination: if an implementation does not exist in an over-approximation, or, if one does exist in an under-approximation. Every sequence of local synthesis problems converges to the global problem that considers all specification-critical components.

Even though in the worst case, our approach, overall, considers more components than the nonincremental synthesis problem (there might be a quadratic blow-up), we provide pruning rules that can be used to significantly reduce the workload of the local problems. Indeed, our experimental results show that even in those cases where the global controller must be synthesized, the amortized costs of our approach are less



(a) Component P (b) Component Q

Figure 1. Network of two reactive real-time components P and Q which synchronize on action α.

than the work of synthesizing a global controller in the first place.

Figure 1 shows two reactive real-time components P and Q. Environment and controller transitions are denoted by dashed and solid lines, respectively. Component P comprises the locations a, b, $\top$ (representing the safe location), and $\bot$ (representing the error location). Component Q comprises the locations c and d. Both components synchronize on action α: if the clock x is greater than 2, then the controller can execute the transition from b to $\top$ in P if and only if he also executes d to c in Q. Now, the controller synthesis problem is to decide whether there is an execution of controller transitions such that the environment never reaches the error location $\bot$. If the environment is quick enough (i.e., as long as $x \leq 2$), she can enforce $\bot$ independently of all controller transitions. The environment could even succeed if we give more power to the controller by allowing the execution of α in P without requiring that Q is in d. Hence, for this example it suffices to consider P alone and not the composition with Q to decide the controller synthesis problem.

Related Work. Automatic program synthesis has a long tradition in computer science that goes back to Church [9], Büchi, and Landweber [6] in the 1960s. In the 1990s, Asarin, Maler, Pnueli, and Sifakis gave a game-theoretic formulation of the controller synthesis problem [22, 3] in the timed automata framework by Alur and Dill [2]. A first on-the-fly method for solving timed games was proposed by Tripakis and Altisen which, however, requires an expensive preprocessing step [23, 1]. As a remedy to this problem, Cassez et al. proposed a fully on-the-fly timed game solving algorithm that combines the backward construction for computing the winning states with a forward reachability analysis [7, 4]. Our work is a direct extension of this approach which furthermore exploits the compositional nature of the system specification.

Abstraction refinement approaches were heavily studied in the model checking domain. A well-known technique called *counter example guided abstraction refinement* [10] starts with a simple initial abstraction of the global system and, in a refinement loop, incrementally concretizes relevant parts of the abstraction until a sound model checking result can be established. Hereby, the refinement is guided by abstract (i.e., potentially spurious) error traces. Henzinger et al. adapted this idea to controller synthesis [17] where abstractions are defined over the game states, counter examples are abstract strategies, and refinement corresponds to the splitting of abstract game states. De Alfaro and Roy introduced three-valued abstractions [13] where the refinement is guided by the difference between an under- and over-approximation of the game states. In contrast to these two approaches that only consider abstractions over game structures, our approach obtains abstractions from partial compositions of timed game automata [22, 3]. Because of the syntactic nature of the composition operator of that model [2], we also require a notion of modality in the syntax. We therefore extend timed game automata by distinguishing between may- and must-transitions in the style of [20, 18].

Our component-based abstraction refinement idea resembles other incremental composition techniques, e.g., for minimizing finite state systems [16]. Dräger et al. investigate using abstractions from partial compositions as a basis for approximating error distances for directed model checking [14].

Contribution. In this paper, we present a novel technique for synthesizing controllers for real-time environments with safety requirements. The contributions of the paper are the following.

- We propose a new technique that reduces the complex problem of synthesizing a global controller for a distributed real-time environment to a sequence of simpler local synthesis problems.
- To that end, we introduce a new composition-based notion of modality in networks of timed game automata enabling a sound over- and under-approximation of all valid global controller implementations.
- We give an automatic abstraction refinement algorithm that incrementally concretizes these approximations converging to the precise set of all valid controller implementations.

- We provide empirical evidence of the effectiveness of our approach by comparing it with the leading timed controller synthesis tool UPPAAL-TIGA.

The rest of this paper is structured as follows. Section 2 recalls the game-theoretic fundamentals of timed controller synthesis. Section 3 describes how under- and over-approximations can be obtained from partial compositions. Based on that, Section 4 proposes an abstraction refinement algorithm that computes a chain of approximations with increasing precision. In Section 5, we report on experimental results with a prototype implementation of our approach and the tool UPPAAL-TIGA. We conclude with an outlook in Section 6.

2 Controller Synthesis as a Game

Synthesizing a controller means finding a winning strategy in a two-player, turn-based, zero-sum game, where a *controller* plays against a hostile *environment*. We say that the environment is the exists-player (player $\exists$) since her goal is to eventually reach some error state on some computation path. Dually, we say that the controller is the forall-player (player $\forall$) since his goal is to always avoid any error state on all computation paths.

2.1 Timed Games

Timed Game Automata. We use timed game automata to model reactive environment components. A *timed game automaton* (TGA) [2, 22, 3] A is a tuple $(L, I, \Sigma, \Delta, \chi, E)$, where L is a finite set of locations, $I \subseteq L$ is a set of initial locations, Σ is a finite set of actions that is partitioned into a set of controller actions $\Sigma_\forall$ and environment actions $\Sigma_\exists$, $\Delta \subseteq (L \times \Sigma \times \mathcal{C}(\chi) \times 2^\chi \times L)$ is the set of transitions, χ is a finite set of real valued clocks, and $E \subseteq L$ is a (possibly empty) set of error locations.

The clock constraints $\varphi \in \mathcal{C}(\chi)$ are of the form

$$\varphi = \top \mid x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2,$$

where x is a clock in χ and c is a constant in $\mathbb{N}_0$. A *clock valuation* $\vec{t} : \chi \to \mathbb{R}_{\geq 0}$ assigns a nonnegative value to each clock and can also be represented by a $|\chi|$-dimensional vector $\vec{t} \in \mathbb{R}^\chi_{\geq 0}$. We use $\mathcal{R} = 2^{\mathbb{R}^\chi_{\geq 0}}$ to denote the set of all clock valuations. We write $\vec{t}[\lambda := 0]$, $\lambda \subseteq \chi$, for clock resets and $\vec{t} + d$, $d \in \mathbb{R}_{\geq 0}$, for uniform time elapse.

Communicating components synchronize on shared actions. Intuitively, components synchronize on an action a iff they emit a at the same time. Formally, the composition is defined as a binary associative operator $\|$ [2]. Two given TGA $A_1 = (L_1, I_1, \Sigma_1, \Delta_1, \chi_1, E_1)$ and $A_2 = (L_2, I_2, \Sigma_2, \Delta_2, \chi_2, E_2)$ with disjoint clock sets $\chi_1 \cap \chi_2 = \emptyset$ can be *composed* to a new TGA $A = A_1 \| A_2$ such that $A = (L_1 \times L_2, I_1 \times I_2, \Sigma_1 \cup \Sigma_2, \Delta, \chi_1 \cup \chi_2, E)$, where $E = E_1 \times L_2 \cup L_1 \times E_2$ and the set of transitions Δ contains

- for $a \in \Sigma_1 \cap \Sigma_2$: $\langle (l_1, l_2), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (l_1', l_2') \rangle$ if $\langle l_1, a, \varphi_1, \lambda_1, l_1' \rangle \in \Delta_1$ and $\langle l_2, a, \varphi_2, \lambda_2, l_2' \rangle \in \Delta_2$,
- for $a \in \Sigma_1 \setminus \Sigma_2$, $l_2 \in L_2$: $\langle (l_1, l_2), a, \varphi_1, \lambda_1, (l_1', l_2) \rangle$ if $\langle l_1, a, \varphi_1, \lambda_1, l_1' \rangle \in \Delta_1$, and
- for $a \in \Sigma_2 \setminus \Sigma_1$, $l_1 \in L_1$: $\langle (l_1, l_2), a, \varphi_2, \lambda_2, (l_1, l_2') \rangle$ if $\langle l_2, a, \varphi_2, \lambda_2, l_2' \rangle \in \Delta_2$.

A set of communicating components, represented as TGA, form a *component network* (or just *network*) $\mathcal{N} = \{C_1, \ldots, C_n\}$. We say that $N = C_1 \| \ldots \| C_n$ is $\mathcal{N}$*'s TGA*. A network $\mathcal{M} = \{C_1, \ldots, C_m\}$ is a *subnetwork* of $\mathcal{N}$ iff $\mathcal{M} \subseteq \mathcal{N}$. In this case we say that $\mathcal{E} = \mathcal{N} \setminus \mathcal{M}$ is the *neighborhood* of $\mathcal{M}$ w.r.t. $\mathcal{N}$. When we refer to the subnetwork $\mathcal{N}_E$ of $\mathcal{N}$, we mean all components whose set of error locations is not empty.

Timed Game Structures. The semantics of timed game automata is defined in terms of *timed game structures*. A timed game structure (TGS) G is a tuple $(S, S_0, \Gamma_\forall, \Gamma_\exists)$ where S is an infinite set of states, $S_0 \subseteq S$ are the initial states, and $\Gamma_p \subseteq S \times (\Sigma_p \cup \mathbb{R}_{\geq 0}) \times S$, $p \in \{\forall, \exists\}$, are the timed moves of the players.

A timed game automaton $(L, I, \Sigma, \Delta, \chi, E)$ induces a timed game structure $G = (L \times \mathcal{R}, I \times \{\vec{0}\}, \Gamma_\forall, \Gamma_\exists)$, where

$$\begin{aligned}\Gamma_p = \big\{ (s, d, s') \mid \exists d \geq 0 : s' = s + d \big\} \cup \\ \big\{ ((l, \vec{t}), a, (l', \vec{t}')) \mid \exists \langle l, a, \varphi, \lambda, l' \rangle \in \Delta : \\ a \in \Sigma_p \wedge \vec{t} \models \varphi \wedge \vec{t}' = \vec{t}[\lambda := 0] \big\},\end{aligned}$$

for $p \in \{\forall, \exists\}$. Note that for a game state $s = (l, \vec{t}) \in S$ and a delay $d \in \mathbb{R}_{\geq 0}$, we write $s + d$ for $(l, \vec{t} + d)$.

Projections. Let $\mathcal{M} \subseteq \mathcal{M}' \subseteq \mathcal{N}$, w.l.o.g. $\mathcal{M} = \{C_1, \ldots, C_k\}$, $\mathcal{M}' = \mathcal{M} \cup \{C_{k+1}, \ldots, C_m\}$, and let $M = (L, I, \Sigma, \Delta, \chi, E)$ and $M' = (L', I', \Sigma', \Delta', \chi', E')$ be the TGA of $\mathcal{M}$ and of $\mathcal{M}'$, respectively, and let $G = (L \times \mathcal{R}, I \times \{\vec{0}\}, \Gamma_\forall, \Gamma_\exists)$ and $G' = (L' \times \mathcal{R}', I' \times \{\vec{0}\}, \Gamma'_\forall, \Gamma'_\exists)$, $S = L \times \mathcal{R}$, $S' = L' \times \mathcal{R}'$, be the TGS of M and M'. We define *projections* of locations, constraints, clock resets, clock valuations, sets of states, and sets of transitions as follows:

$$\begin{aligned}l' \downarrow_\mathcal{M} &:= (l_1', \ldots, l_k') \\ &\quad \text{for } l' = (l_1', \ldots, l_k', l_{k+1}', \ldots, l_m') \in L'\end{aligned}$$

$$\varphi'\downarrow_{\mathcal{M}} := \begin{cases} \varphi_1'\downarrow_{\mathcal{M}} \wedge \varphi_2'\downarrow_{\mathcal{M}} & \text{if } \varphi' = \varphi_1' \wedge \varphi_2' \\ \varphi' & \text{if } \varphi' = x \bowtie c \wedge x \in \chi \\ \top & \text{if } \varphi' = x \bowtie c \wedge x \notin \chi \\ \top & \text{if } \varphi' = \top \end{cases}$$
$$\text{for } \varphi' \in \mathcal{C}(\chi')$$
$$\lambda'\downarrow_{\mathcal{M}} := \lambda' \cap \chi \text{ for } \lambda' \subseteq \chi'$$
$$\vec{t}'\downarrow_{\mathcal{M}} := (x_1, \ldots, x_k)$$
$$\text{for } \vec{t}' = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_m) \in \mathbb{R}_{\geq 0}^{\chi'}$$
$$s'\downarrow_{\mathcal{M}} := (l'\downarrow_{\mathcal{M}}, \vec{t}'\downarrow_{\mathcal{M}}) \text{ for } s' = (l', \vec{t}') \in S'$$
$$\Delta''\downarrow_{\mathcal{M}} := \{\langle l_1, a, \varphi, \lambda, l_2\rangle \in \Delta \mid \langle l_1', a', \varphi', \lambda', l_2'\rangle \in \Delta'' \wedge l_1 = l_1'\downarrow_{\mathcal{M}} \wedge a = a' \wedge \varphi \equiv \varphi'\downarrow_{\mathcal{M}} \wedge \lambda = \lambda'\downarrow_{\mathcal{M}} \wedge l_2 = l_2'\downarrow_{\mathcal{M}}\} \text{ for } \Delta'' \subseteq \Delta'$$
$$\Gamma''\downarrow_{\mathcal{M}} := \{(s_1, u, s_2) \in S \times (\Sigma_p \cup \mathbb{R}_{\geq 0}) \times S \mid (s_1', u', s_2') \in \Gamma'' \wedge s_1 = s_1'\downarrow_{\mathcal{M}} \wedge u = u' \wedge s_2' = s_2'\downarrow_{\mathcal{M}}\} \text{ for } \Gamma'' \subseteq \Gamma_p', p \in \{\forall, \exists\}$$
$$\lceil S''\downarrow_{\mathcal{M}} \rceil := \{s \in S \mid \exists s' \in S'' : s = s'\downarrow_{\mathcal{M}}\} \text{ for } S'' \subseteq S'$$
$$\lfloor S''\downarrow_{\mathcal{M}} \rfloor := \{s \in S \mid (\exists s' \in S'' : s = s'\downarrow_{\mathcal{M}}) \wedge (\forall s' \in S' : s = s'\downarrow_{\mathcal{M}} \Rightarrow s' \in S'')\} \text{ for } S'' \subseteq S'$$

Projected set inclusion and *set subsumption* are two relations over sets of states. For two sets of states S and S' we define

$$S \sqsubseteq_{\mathcal{M}} S' \text{ iff } S \subseteq \lfloor S'\downarrow_{\mathcal{M}} \rfloor \text{ and}$$
$$S \sqsupseteq_{\mathcal{M}} S' \text{ iff } S \supseteq \lceil S'\downarrow_{\mathcal{M}} \rceil.$$

Strategies and Outcomes. A strategy is a function that determines a particular player's decisions during the course of a game. In general, a strategy is defined as a mapping from a history of events to a concrete game move. However, in the timed controller synthesis setting with safety requirements under complete information, it is sufficient to consider state-based (or memoryless) strategies [22, 3, 12]. Formally, a *memoryless strategy* for player p is a partial function $f_p : S \to \Sigma_p \cup \{\bot\}$ that maps a game state either to an (active) action move of p or to $\bot$ representing a (passive) wait move.

The notion of an *outcome* defines the combination of two strategies $f_\exists$ and $f_\forall$. Formally, the set $\mathsf{Outcome}(f_\exists, f_\forall) \subseteq S$ comprises those states that are eventually visited when player $\exists$ sticks to $f_\exists$ and player $\forall$ sticks to $f_\forall$.

Reachability Games. Let $G = (S, S_0, \Gamma_\forall, \Gamma_\exists)$ be a timed game structure and $K \subseteq S$ a set of goal states, then (G, K) represents a *reachability game*. Player $\exists$ wins (G, K) if and only if she can enforce a visit to K. More formally, player $\exists$ wins iff

$$\exists f_\exists \, \forall f_\forall : \mathsf{Outcome}(f_\exists, f_\forall) \cap K \neq \emptyset.$$

In the following, w.l.o.g.[1], we will only consider reachability games for player $\exists$.

2.2 Solving Timed Games

Solving a reachability game (G, K) means computing the set of states from which player $\exists$ has a strategy to enforce an outcome that eventually visits a state in K. Before we come to the actual solving algorithm, we formalize the notion of controllability.

Definition 2.1 *For a timed game structure $G = (S, S_0, \Gamma_\forall, \Gamma_\exists)$, the* timed enforceable predecessor operator *[22, 3] $\pi : 2^S \to 2^S$ for player $\exists$ is defined as*

$$\pi(X) = \pi^a(X) \cup \pi^t(X),$$

where π^a is the active *timed enforceable predecessor operator*

$$\pi^a(X) = \{s \in S \mid \exists (s, u, s') \in \Gamma_\exists : u \in \Sigma_\exists \wedge s' \in X\}$$

and π^t is the passive *timed enforceable predecessor operator*

$$\pi^t(X) = \{s \in S \mid \exists d \geq 0 : s + d \in X \wedge \forall 0 \leq d' < d : \forall (s + d', c, s') \in \Gamma_\forall : c \in \Sigma_\forall \Rightarrow s' \in X\}.$$

Note that with this definition of π we constitute an *asymmetric game semantics*, i.e., we prioritize the environment in situations where both players can execute an active move. We believe that this is a more natural view in the sense that one is always interested in controller implementations that consider a worst-case behavior of the environment. It was shown in [22, 3] and later in [7] that π can be effectively computed using finite abstractions (such as clock regions or zones) of the infinite timed state space.

We define those states from which player $\exists$ has a winning strategy to eventually enforce K as the $\exists$-*attractor of K*. Thus, a valid controller implementation must avoid any state from the $\exists$-attractor of the error states. For a reachability game (G, K), the computation of the $\exists$-attractor $\mathsf{Attr}(G, K)$ is carried out by iteratively applying π as a least fixed point construction on K [22, 3, 12]. Algorithm 1 gives a definition in pseudo code.

[1] Reachability objectives for the controller or safety objectives for the environment are dual forms of the case considered here.

Algorithm 1 Fixed point algorithm for computing the $\exists$-attractor of a set of goal states K.

$\mathsf{A}_0 := K$
$n := 0$
repeat
  $n := n + 1$
  $\mathsf{A}_n := \mathsf{A}_{n-1} \cup \pi(\mathsf{A}_{n-1})$
until $\mathsf{A}_n = \mathsf{A}_{n-1}$
$\mathsf{Attr}(G, K) := \mathsf{A}_n$

Player $\exists$ wins (G, K) if and only if $S_0 \cap \mathsf{Attr}(G, K) \neq \emptyset$. There exists a valid controller implementation if and only if player $\exists$ does not win. In this case, a winning strategy for the controller (and therefore also a controller implementation) can be directly derived from $S \setminus \mathsf{Attr}(G, K)$.

A real-time environment, given as a network of timed game automata $\mathcal{N} = \{C_1, \ldots, C_n\}$, induces a reachability game $\mathsf{Game}(\mathcal{N}) = (G, K)$ such that G is the game structure induced by $\mathcal{N}$'s TGA $(L, I, \Sigma, \Delta, \chi, E)$ and $K = E \times \mathcal{R}$ is the set of $\mathcal{N}$'s error states.

Theorem 2.1 *[22, 3] The controller synthesis problem for reactive real-time environments, given as networks of timed game automata with safety requirements, is decidable.*

The following lemma states that if we add a component C to a network such that adding C restricts the moves of player $\forall$ and does not restrict the moves of player $\exists$, then the winning possibilities of player $\exists$ are independent of C.

Lemma 2.2 *For a network $\mathcal{M}$ and a component $P \notin \mathcal{M}$, let $G = (S, S_0, \Gamma_\forall, \Gamma_\exists)$ be $\mathcal{M}$'s TGS, $G' = (S', S_0', \Gamma_\forall', \Gamma_\exists')$ be $\mathcal{M} \cup \{C\}$'s TGS, K' be $\mathcal{M} \cup \{C\}$'s error states, and $K = \lfloor K' \downarrow_{\mathcal{M}} \rfloor$. If C's and $\mathcal{M}$'s $\exists$-actions are disjoint, i.e., $\Sigma_\exists^{\mathcal{M}} \cap \Sigma_\exists^C = \emptyset$, and $\Gamma_\forall' \downarrow_{\mathcal{M}} \subseteq \Gamma_\forall$, then*

$$\mathsf{Attr}(G, K) \sqsubseteq_{\mathcal{M}} \mathsf{Attr}(G', K').$$

Proof: Let A_i and A_i' be the attractor sets after the i^{th} iteration of Algorithm 1 of (G, K) and (G', K'), respectively. We show $\mathsf{A}_i \subseteq \lfloor \mathsf{A}_i' \downarrow_{\mathcal{M}} \rfloor$, for any $i \in \mathbb{N}_0$, by induction over i, i.e., over the construction of Attr. Let L' be set of C's locations and $\mathcal{R}'$ the set of all clock valuations of the clocks in C that are not in $\mathcal{M}$.
For the base case, if $(l_1, \vec{t}_1) \in K$ is a goal state, then, by definition of error and goal states, $((l_1, l_2), (\vec{t}_1, \vec{t}_2)) \in K'$ is also a goal state, for any $l_2 \in L'$ and $\vec{t}_2 \in \mathcal{R}'$. Hence, $\mathsf{A}_0 = K = \lfloor K' \downarrow_{\mathcal{M}} \rfloor = \lfloor \mathsf{A}_0' \downarrow_{\mathcal{M}} \rfloor$.
For the induction step $i \mapsto i + 1$ we show that $\mathsf{A}_i \subseteq \lfloor \mathsf{A}_i' \downarrow_{\mathcal{M}} \rfloor$ implies $\mathsf{A}_{i+1} \subseteq \lfloor \mathsf{A}_{i+1}' \downarrow_{\mathcal{M}} \rfloor$. Assume $\forall (l_1, \vec{t}_1) \in \mathsf{A}_i : \forall (l_2, \vec{t}_2) \in L' \times \mathcal{R}' : ((l_1, l_2), (\vec{t}_1, \vec{t}_2)) \in \mathsf{A}_i'$ and let $(l_1, \vec{t}_1) \in \mathsf{A}_{i+1} = \mathsf{A}_i \cup \pi(\mathsf{A}_i) = \mathsf{A}_i \cup \pi^a(\mathsf{A}_i) \cup \pi^t(\mathsf{A}_i)$ be arbitrarily chosen. We show that $\forall (l_2, \vec{t}_2) \in L' \times \mathcal{R}' : ((l_1, l_2), (\vec{t}_1, \vec{t}_2)) \in \mathsf{A}_{i+1}'$ by distinguishing three cases:
(1) If $(l_1, \vec{t}_1) \in \mathsf{A}_i$, then, by i.h., $\forall (l_2, \vec{t}_2) \in L' \times \mathcal{R}' : ((l_1, l_2), (\vec{t}_1, \vec{t}_2)) \in \mathsf{A}_i' \subseteq \mathsf{A}_{i+1}'$.
(2) If $(l_1, \vec{t}_1) \in \pi^a(\mathsf{A}_i)$ is an active enforceable predecessor, then $\exists ((l_1, \vec{t}_1), u, (l_1', \vec{t}_1')) \in \Gamma_\exists : (l_1', \vec{t}_1') \in \mathsf{A}_i \wedge u \in \Sigma_\exists$. From $\Sigma_\exists^{\mathcal{M}} \cap \Sigma_\exists^C = \emptyset$ follows that if there is a move $((l_1, \vec{t}_1), u, (l_1', \vec{t}_1')) \in \Gamma_\exists$, $u \in \Sigma_\exists^{\mathcal{M}}$, then there is also a move $(((l_1, l_2), (\vec{t}_1, \vec{t}_2)), u, ((l_1', l_2), (\vec{t}_1', \vec{t}_2))) \in \Gamma_\exists'$, for all $l_2 \in L'$ and all $\vec{t}_2 \in \mathcal{R}'$. We apply this fact to each state in A_i', for which we know, by i.h., $\forall (l_1, \vec{t}_1) \in \mathsf{A}_i : \forall (l_2, \vec{t}_2) \in L' \times \mathcal{R}' : ((l_1, l_2), (\vec{t}_1, \vec{t}_2)) \in \mathsf{A}_i'$, to finally obtain $((l_1, l_2), (\vec{t}_1, \vec{t}_2)) \in \pi^a(\mathsf{A}_i') \subseteq \pi(\mathsf{A}_i') \subseteq \mathsf{A}_{i+1}'$.
(3) If $(l_1, \vec{t}_1) \in \pi^t(\mathsf{A}_i)$ is a passive enforceable predecessor, then $\exists d : (l_1, \vec{t}_1 + d) \in \mathsf{A}_i \wedge \forall d' < d : \forall ((l_1, \vec{t}_1 + d'), c, (l_1', \vec{t}_1')) \in \Gamma_\forall : c \in \Sigma_\forall \Rightarrow (l_1', \vec{t}_1') \in \mathsf{A}_i$. From $\Gamma_\forall' \downarrow_{\mathcal{M}} \subseteq \Gamma_\forall$ follows that if there was no spoiling $\forall$-move $((l_1, \vec{t}_1), c, (l_1', \vec{t}_1')) \in \Gamma_\forall : c \in \Sigma_\forall$, then there is also no spoiling $\forall$-move $(((l_1, l_2), (\vec{t}_1, \vec{t}_2)), c, ((l_1', l_2'), (\vec{t}_1', \vec{t}_2'))) \in \Gamma_\forall'$, for all $(l_2, \vec{t}_2), (l_2', \vec{t}_2') \in L' \times \mathcal{R}'$. We apply this fact to each state in A_i', for which we know, by i.h., $\forall (l_1', \vec{t}_1') \in \mathsf{A}_i : \forall (l_2, \vec{t}_2) \in L' \times \mathcal{R}' : ((l_1', l_2), (\vec{t}_1, \vec{t}_2)) \in \mathsf{A}_i'$, to finally obtain $((l_1, l_2), (\vec{t}_1, \vec{t}_2)) \in \pi^t(\mathsf{A}_i') \subseteq \pi(\mathsf{A}_i') \subseteq \mathsf{A}_{i+1}'$. □
Intuitively, once a state of a subnetwork $\mathcal{M} \subseteq \mathcal{M}'$ is winning, it remains winning in the context of any non-influencing neighborhood $\mathcal{M}' \setminus \mathcal{M}$.

Dually, the following lemma states that if we add a component C to a network such that adding C restricts the moves of player $\exists$ and does not restrict the moves of player $\forall$, then the winning possibilities of player $\forall$ are independent of C.

Lemma 2.3 *For a network $\mathcal{M}$ and a component $P \notin \mathcal{M}$, let $G = (S, S_0, \Gamma_\forall, \Gamma_\exists)$ be $\mathcal{M}$'s TGS, $G' = (S', S_0', \Gamma_\forall', \Gamma_\exists')$ be $\mathcal{M} \cup \{C\}$'s TGS, K' be $\mathcal{M} \cup \{C\}$'s error states, and $K = \lceil K' \downarrow_{\mathcal{M}} \rceil$. If C's and $\mathcal{M}$'s $\forall$-actions are disjoint, i.e., $\Sigma_\forall^{\mathcal{M}} \cap \Sigma_\forall^C = \emptyset$, and $\Gamma_\exists' \downarrow_{\mathcal{M}} \subseteq \Gamma_\exists$, then*

$$\mathsf{Attr}(G, K) \sqsupseteq_{\mathcal{M}} \mathsf{Attr}(G', K').$$

The proof is analogous to the one from Lemma 2.2.

From a practical point of view, solving a timed game by only computing the attractor set has the huge disadvantage that potentially many forward unreachable (and therefore irrelevant) states are considered in the backward construction. As a remedy to this problem, the authors of [7] proposed an efficient *on-the-fly* timed game solving algorithm that extends the classic pure backward approach by combining it with a forward

reachability analysis, ensuring that only forward reachable states are considered. Here, the attractor set construction alternates between forward (post) steps that explore new reachable states and backward (pre) steps that back-propagate winning information. Thereby, all pre-steps are intersected with the so-far known reachable state space. In our approach, we can adapt this on-the-fly algorithm for solving the local sub-games.

3 Abstract Timed Games

In this section, we describe how to obtain sound abstractions of the global game, that considers all components, from local games, that consider only some components. The winning possibilities of a particular player depend on all (her and her opponent's) moves in the game. The set of winning states of player p decreases if we either (1) remove some moves that are controllable by p or (2) add additional moves that are controllable by her opponent. Using this idea, by privileging (strictly) one of the players, we can systematically under- or over-approximate the attractor set.

In Section 3.1, we define how a partial composition induces an under- and over-approximation of the moves of the players which, in turn, are used in Section 3.2 to obtain sound attractor set approximations.

3.1 Modal Timed Game Automata

In the style of [20, 18], we introduce an extension to the classic definition of timed game automata which allows us to distinguish between transitions that might be available and transitions that are surely available representing under- and over-approximations of the moves of the players in the game.

Definition 3.1 *A* modal timed game automaton *(MTGA) is a tuple* $M = (L, I, \Sigma, \Delta^{may}, \Delta^{must}, \chi, E)$ *with* $\Delta^{must} \subseteq \Delta^{may}$ *and* $(L, I, \Sigma, \Delta^{may}, \chi, E)$ *and* $(L, I, \Sigma, \Delta^{must}, \chi, E)$ *are timed game automata, where* Δ^{may} *is the set of transitions that might be available (may-transitions) and* Δ^{must} *is the set of transitions that are surely available (must-transitions).*

For short, we write $M^{may} = (L, I, \Sigma, \Delta^{may}, \chi, E)$ and $M^{must} = (L, I, \Sigma, \Delta^{must}, \chi, E)$ to refer to M's *may* and *must TGA*, respectively.

The notion of networks extends from TGA to MTGA in a straightforward manner. The modal composition operator, however, slightly extends the non-modal definition.

Let $M_1 = (L_1, I_1, \Sigma_1, \Delta_1^{may}, \Delta_1^{must}, \chi_1, E_1)$ and $M_2 = (L_2, I_2, \Sigma_2, \Delta_2^{may}, \Delta_2^{must}, \chi_2, E_2)$ be two MTGA with disjoint clock sets $\chi_1 \cap \chi_2 = \emptyset$. Furthermore, let $\mathcal{E}$ be the neighborhood of M_1 and M_2 that contains the remaining components which represent potential synchronization partners. Then, M_1 and M_2 can be *composed* to a new MTGA $M = (L, I, \Sigma, \Delta^{may}, \Delta^{must}, \chi, E)$ denoted as $M = M_1 \| _{\mathcal{E}} M_2$ such that $M^{may} = M_1^{may} \| M_2^{may}$ and the set of must-transitions Δ^{must} is the largest subset of Δ^{may} that only contains transitions with actions not emitted by any component from $\mathcal{E}$:

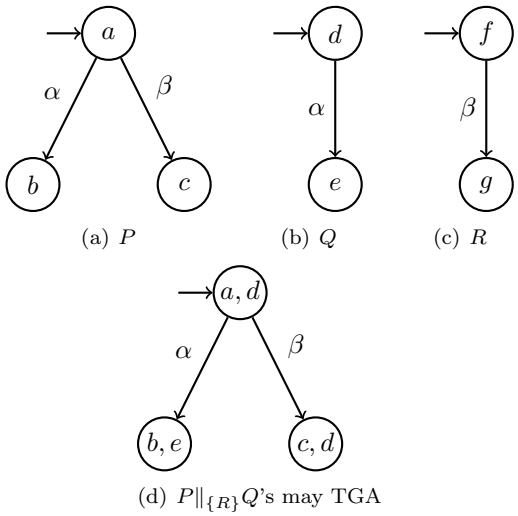
$$\Delta^{must} = \left\{ \langle l, a, \varphi, \lambda, l' \rangle \in \Delta^{may} \mid \forall C \in \mathcal{E} : a \notin \Sigma^C \right\}$$

where Σ^C stands for the set of component C's actions.

The must-transitions of a parallel composition of two components comprise those transitions that are surely contained in any further parallel composition (since the necessary global synchronization criterion is already satisfied). The may-transitions of a parallel composition of two components that are not must can disappear in a further composition (since it is yet unclear whether the necessary global synchronization criterion will be completely satisfied). The following lemma states this observation more formally and follows directly from the definitions of Δ^{must} and Δ^{may}.

Lemma 3.1 *For a subnetwork* $\mathcal{M}$ *and a component* $C \notin \mathcal{M}$*, if* M *is* $\mathcal{M}$*'s MTGA and* M' *is* $\mathcal{M} \cup \{C\}$*'s MTGA, then* $\Delta_M^{must} \subseteq \Delta_{M'}^{must}\downarrow_{\mathcal{M}}$ *and* $\Delta_{M'}^{may}\downarrow_{\mathcal{M}} \subseteq \Delta_M^{may}$.

Informally, Lemma 3.1 states that we increase the precision when we add new components to a subnetwork $\mathcal{M}$ in the sense that $\mathcal{M}$'s MTGA's may-transitions decrease and $\mathcal{M}$'s MTGA's must-transitions increase.



(a) P (b) Q (c) R

(d) $P\|_{\{R\}}Q$'s may TGA

Figure 2. Network $\{P, Q, R\}$.

Figure 2 shows a network $\mathcal{N} = \{P, Q, R\}$. The may TGA of the composition of P and Q is shown in Figure 2(d). Here, α is also a must-transition since R does not emit α.

3.2 Attractor Set Approximation

When we compose a component network $\mathcal{M}$ which is part of a larger network $\mathcal{N}$, then $\mathcal{M}$'s MTGA $M = (L, I, \Sigma, \Delta^{may}, \Delta^{must}, \chi, E)$, w.r.t. the neighborhood $\mathcal{E} = \mathcal{N} \setminus \mathcal{M}$, represents an *abstraction* of $\mathcal{N}$ in the sense that we can use M's modal moves to approximate the attractor sets of the players.

More precisely, we obtain an *under-approximation* of the $\exists$-attractor if (1) player $\forall$ chooses his moves from Δ^{may} and (2) player $\exists$ chooses her moves from Δ^{must}. Dually, we obtain an *over-approximation* of the $\exists$-attractor if (1) $\mathcal{M}$ contains all error components of $\mathcal{N}$, (2) player $\forall$ chooses his moves from Δ^{must}, and (3) player $\exists$ chooses her moves from Δ^{may}.

In the following, let

- $(S, S_0, \Gamma_\forall^{may}, \Gamma_\exists^{may})$ be M^{may}'s TGS and
- $(S, S_0, \Gamma_\forall^{must}, \Gamma_\exists^{must})$ be M^{must}'s TGS.

Formally, $\mathcal{M}$ induces

1. a *weakened reachability game* $\mathsf{WGame}_\mathcal{N}(\mathcal{M}) = ((S, S_0, \Gamma_\forall^{may}, \Gamma_\exists^{must}), E \times \mathcal{R})$;
2. a *strengthened reachability game* $\mathsf{SGame}_\mathcal{N}(\mathcal{M}) = ((S, S_0, \Gamma_\forall^{must}, \Gamma_\exists^{may}), E \times \mathcal{R})$.

Lemma 3.2 *For a network $\mathcal{N}$ and a subnetwork $\mathcal{M} \subseteq \mathcal{N}$, if $C \in \mathcal{N} \setminus \mathcal{M}$, then*

$$\mathsf{Attr}(\mathsf{WGame}_\mathcal{N}(\mathcal{M})) \sqsubseteq_\mathcal{M} \mathsf{Attr}(\mathsf{WGame}_\mathcal{N}(\mathcal{M} \cup \{C\})).$$

Proof: Let $(G, K) = \mathsf{WGame}_\mathcal{N}(\mathcal{M})$ and $(G', K') = \mathsf{WGame}_\mathcal{N}(\mathcal{M} \cup \{C\})$. From the definition of $\|$ follows $K \sqsubseteq_\mathcal{M} K'$. From the definitions of WGame and must-moves we obtain $\Sigma_\exists^\mathcal{M} \cap \Sigma_\exists^C = \emptyset$, and according to Lemma 3.1 we deduce $\Gamma'_\forall{\downarrow_\mathcal{M}} \subseteq \Gamma_\forall$. Hence, from Lemma 2.2 it immediately follows that $\mathsf{Attr}(G, K) \sqsubseteq_\mathcal{M} \mathsf{Attr}(G', K')$. □

Lemma 3.3 *For a network $\mathcal{N}$ and a subnetwork $\mathcal{M} \subseteq \mathcal{N}$, if $C \in \mathcal{N} \setminus \mathcal{M}$, then*

$$\mathsf{Attr}(\mathsf{SGame}_\mathcal{N}(\mathcal{M})) \sqsupseteq_\mathcal{M} \mathsf{Attr}(\mathsf{SGame}_\mathcal{N}(\mathcal{M} \cup \{C\})).$$

The proof is analogous to the one of Lemma 3.2.

Lemma 3.4 *For a network $\mathcal{N}$,*

$$\begin{aligned} &\mathsf{Attr}(\mathsf{SGame}_\mathcal{N}(\mathcal{N})) \\ = \ &\mathsf{Attr}(\mathsf{WGame}_\mathcal{N}(\mathcal{N})) \\ = \ &\mathsf{Attr}(\mathsf{Game}(\mathcal{N})). \end{aligned}$$

Proof: By definition, the neighborhood of the global network is empty: $\mathcal{E} = \mathcal{N} \setminus \mathcal{N} = \emptyset$. Hence, from the definition of MTGA in Section 3.1, we can conclude that the game structures induced by $\mathsf{SGame}_\mathcal{N}(\mathcal{N})$, $\mathsf{WGame}_\mathcal{N}(\mathcal{N})$, and $\mathsf{Game}(\mathcal{N})$ coincide since $\mathcal{N}$'s MTGA coincides with $\mathcal{N}$'s TGA. □

4 Abstraction Refinement

In this section, we describe the abstraction refinement loop of our approach that computes a chain of attractor set approximations with increasing precision.

4.1 Refinement Loop

For an environment given as the global component network $\mathcal{N} = \{C_1, \ldots, C_m\}$, during the course of the refinement process, we consider chains of subnetworks $\mathcal{M}_i$ of the form

$$\mathcal{M}_0 \subsetneq \mathcal{M}_1 \subsetneq \ldots \subsetneq \mathcal{M}_n = \mathcal{N}.$$

On each refinement cycle i, $0 \leq i \leq n$, we obtain

- an under-approximation $\lfloor \mathsf{A}_i \rfloor = \mathsf{Attr}(\mathsf{WGame}_\mathcal{N}(\mathcal{M}_i))$ and
- an over-approximation $\lceil \mathsf{A}_i \rceil = \mathsf{Attr}(\mathsf{SGame}_\mathcal{N}(\mathcal{M}_i))$

of the global $\exists$-attractor $\mathsf{A} = \mathsf{Attr}(\mathsf{Game}(\mathcal{N}))$ as described in Section 3. According to Lemmas 3.2, 3.3, and 3.4, these attractor set chains are of the form

$$\begin{aligned} &\lfloor \mathsf{A}_0 \rfloor \sqsubseteq_{\mathcal{M}_0} \lfloor \mathsf{A}_1 \rfloor \sqsubseteq_{\mathcal{M}_1} \ldots \sqsubseteq_{\mathcal{M}_{n-1}} \lfloor \mathsf{A}_n \rfloor = \mathsf{A} \quad \text{and} \\ &\lceil \mathsf{A}_0 \rceil \sqsupseteq_{\mathcal{M}_0} \lceil \mathsf{A}_1 \rceil \sqsupseteq_{\mathcal{M}_1} \ldots \sqsupseteq_{\mathcal{M}_{n-1}} \lceil \mathsf{A}_n \rceil = \mathsf{A}. \end{aligned}$$

Initially, we select $\mathcal{M}_0 = \mathcal{N}_E$. Note that for only obtaining under-approximations, $\mathcal{M}_0 \subseteq \mathcal{N}_E$ suffices. In refinement cycle i, we extend $\mathcal{M}_i$ by other components from $\mathcal{N} \setminus \mathcal{M}_i$. Hereby, the selection of the components is guided by a refinement heuristics that takes abstract (i.e., potentially spurious) strategies into account. While spurious $\exists$-strategies are used to shrink $\lceil \mathsf{A}_i \rceil$, spurious $\forall$-strategies are used to enlarge $\lfloor \mathsf{A}_i \rfloor$.

The loop terminates if either

1. player $\exists$ can win $\mathsf{WGame}_\mathcal{N}(\mathcal{M}_i)$ (i.e., there surely does not exist a controller), or
2. player $\exists$ cannot win $\mathsf{SGame}_\mathcal{N}(\mathcal{M}_i)$ (i.e., there surely exists a controller).

Indeed, these criteria suffice for soundness:

Theorem 4.1 *The following two statements hold for any subnetwork $\mathcal{M}_i$:*

1. *If $\lfloor S_0{\downarrow_{\mathcal{M}_i}} \rfloor \cap \lfloor \mathsf{A}_i \rfloor \neq \emptyset$, then $S_0 \cap \mathsf{A} \neq \emptyset$.*
2. *If $\lceil S_0{\downarrow_{\mathcal{M}_i}} \rceil \cap \lceil \mathsf{A}_i \rceil = \emptyset$, then $S_0 \cap \mathsf{A} = \emptyset$.*

Proof: Case 1. Let $\lfloor S_0\downarrow_{\mathcal{M}_i}\rfloor \cap \lfloor \mathsf{A}_i\rfloor \neq \emptyset$. Since $\lfloor \mathsf{A}_i\rfloor \sqsubseteq_{\mathcal{M}_i} \lfloor \mathsf{A}_n\rfloor = \mathsf{A}$ by Lemmas 3.2 and 3.4, by definition we get $\lfloor \mathsf{A}_i\rfloor \subseteq \lfloor \mathsf{A}\downarrow_{\mathcal{M}_i}\rfloor$. Thus, $\lfloor S_0\downarrow_{\mathcal{M}_i}\rfloor \cap \lfloor \mathsf{A}\downarrow_{\mathcal{M}_i}\rfloor \neq \emptyset$, say $s \in \lfloor S_0\downarrow_{\mathcal{M}_i}\rfloor \cap \lfloor \mathsf{A}\downarrow_{\mathcal{M}_i}\rfloor$. Since $s \in \lfloor S_0\downarrow_{\mathcal{M}_i}\rfloor$, there is a $t \in S_0$ such that $t\downarrow_{\mathcal{M}_i} = s$. Since $s \in \lfloor \mathsf{A}\downarrow_{\mathcal{M}_i}\rfloor$, for all s' with $s'\downarrow_{\mathcal{M}_i} = s$, also $s' \in \mathsf{A}$ has to hold. In particular, $t \in \mathsf{A}$, hence $t \in S_0 \cap \mathsf{A} \neq \emptyset$.
Case 2. Assume for contradiction that $s \in S_0 \cap \mathsf{A}$. Since $s \in S_0$, by definition we have $s\downarrow_{\mathcal{M}_i} \in \lceil S_0\downarrow_{\mathcal{M}_i}\rceil$. Since $s \in \mathsf{A}$ and because, by Lemmas 3.3 and 3.4, $\lceil \mathsf{A}_i\rceil \sqsupseteq_{\mathcal{M}_i} \lceil \mathsf{A}_n\rceil = \mathsf{A}$ (i.e., by definition, $\lceil \mathsf{A}_i\rceil \supseteq \lceil \mathsf{A}\downarrow_{\mathcal{M}_i}\rceil$), we get $s\downarrow_{\mathcal{M}_i} \in \lceil \mathsf{A}\downarrow_{\mathcal{M}_i}\rceil \subseteq \lceil \mathsf{A}_i\rceil$. Thus, $s\downarrow_{\mathcal{M}_i} \in \lceil S_0\downarrow_{\mathcal{M}_i}\rceil \cap \lceil \mathsf{A}_i\rceil$, i.e., $\lceil S_0\downarrow_{\mathcal{M}_i}\rceil \cap \lceil \mathsf{A}_i\rceil \neq \emptyset$, a contradiction. □

The following theorem guarantees termination of the refinement loop and convergence with the global reachability game.

Theorem 4.2 *The component-based abstraction refinement loop terminates and converges to the global reachability game.*

Proof: Since there are only finitely many components, $\mathcal{M}_n$ eventually converges to $\mathcal{N}$, for a finite $n \in \mathbb{N}_0$, and we end up with the global reachability game. From Lemma 3.4 it follows that the attractor sets $\lceil \mathsf{A}_n\rceil = \mathsf{SGame}_{\mathcal{M}_n}(\mathcal{N})$, $\lfloor \mathsf{A}_n\rfloor = \mathsf{WGame}_{\mathcal{M}_n}(\mathcal{N})$, and $\mathsf{A} = \mathsf{Game}(\mathcal{M}_n)$ coincide. Then, it is easy to see that exactly one the two termination criteria must hold. □

4.2 Component Selection

The speed of the convergence of the attractor set approximations heavily depends on the selection of the components for refinement. In this section, we specify basic criteria for the selection of candidate transitions to refine on.

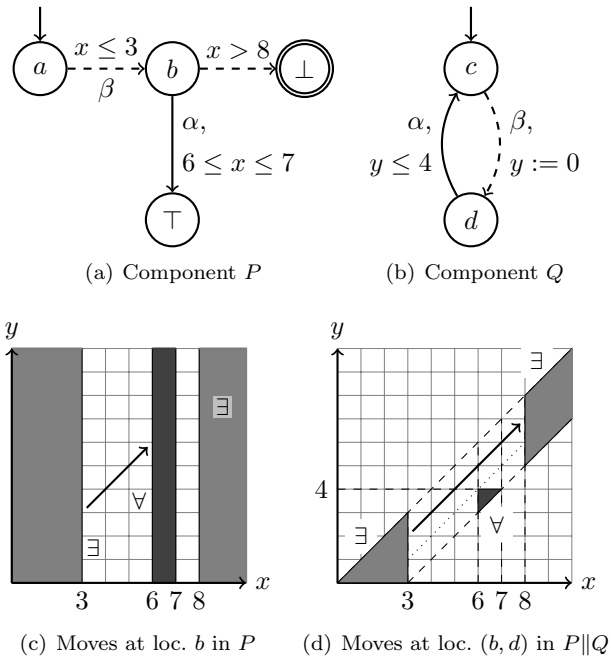
In the following, let $\mathcal{M}_i \subseteq \mathcal{N}$ be the subnetwork of the i^{th} refinement cycle, and let $\lfloor \mathsf{A}_i\rfloor$ and $\lceil \mathsf{A}_i\rceil$ be the corresponding under- and over-approximations of the global $\exists$-attractor A, respectively. For two game states $s = (l, \vec{t})$ and $s' = (l', \vec{t'})$, which are connected by a $\exists$-move $(s, u, s') \in \Gamma_\exists$, $u \in \Sigma_\exists \cup \mathbb{R}_{\geq 0}$,

- in order to *enlarge* $\lfloor \mathsf{A}_i\rfloor$,
  we select $s \notin \lfloor \mathsf{A}_i\rfloor$ and $s' \in \lfloor \mathsf{A}_i\rfloor$;
- in order to *shrink* $\lceil \mathsf{A}_i\rceil$,
  we select $s \in \lceil \mathsf{A}_i\rceil$ and $s' \in \lceil \mathsf{A}_i\rceil$.

Now, we select a pure may-transition $\langle l, a, \varphi, \lambda, l''\rangle \in \Delta^{may} \setminus \Delta^{must}$ such that

- if $l = l'$ and $\exists d > 0 : \vec{t'} = \vec{t} + d$, then $a \in \Sigma_\forall$;
- if $l \neq l'$, then $a \in \Sigma_\exists$.

Once a candidate transition with action a is chosen, we select a component $C \in \mathcal{N} \setminus \mathcal{M}_i$ with $a \in \Sigma^C$ and refine the subnetwork such that $\mathcal{M}_{i+1} = \mathcal{M}_i \cup \{C\}$.



(a) Component P (b) Component Q

(c) Moves at loc. b in P (d) Moves at loc. (b, d) in $P \| Q$

Figure 3. Enlargement of the under-approximation of the $\exists$-attractor by concretizing transition α which is controlled by player $\forall$.

Figure 3 shows an example refinement on transition α. If we consider component P alone (as abstraction of the network $\{P, Q\}$), the controller is spuriously too powerful by allowing him to take transition α whenever P is in location b and $6 \leq x \leq 7$. Only after the refinement (i.e., the composition with component Q) the environment gets a chance to circumvent α to enforce location $\bot$.

4.3 Pruning Irrelevant Moves

In the construction of the attractor set approximations $\lfloor \mathsf{A}_i\rfloor$ and $\lceil \mathsf{A}_i\rceil$, $i > 0$, we can use the intermediate results of the preceding refinement cycle $\lfloor \mathsf{A}_{i-1}\rfloor$ and $\lceil \mathsf{A}_{i-1}\rceil$ for pruning of irrelevant moves. We identify a move $(s, a, s') \in \Gamma_\forall \cup \Gamma_\exists$ as *irrelevant* if either

- $a \in \Sigma_\forall$ and $s'\downarrow_{\mathcal{M}_{i-1}} \in \lfloor \mathsf{A}_{i-1}\rfloor$ or
- $a \in \Sigma_\exists$ and $s'\downarrow_{\mathcal{M}_{i-1}} \notin \lceil \mathsf{A}_{i-1}\rceil$.

Irrelevant moves do not need to be considered in any attractor set construction because they lead to game states that are already known to be winning for the opponent.

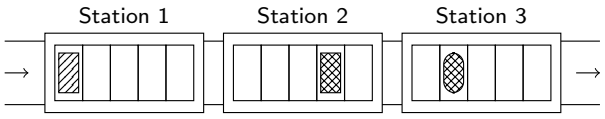


Figure 4. Gear Production Stack (GPS) with three stations and five sub-processing units each.

5 Benchmarks

We implemented our approach as a prototype tool and evaluated it on a benchmark based on a case study from our industrial partners BPS-IT Solutions and META-LEVEL Software AG. We compared the results with those produced by the timed game solver UPPAAL-TIGA [4]. In our prototype, the actual local attractor set construction is implemented according to the forward/backward approach suggested in [7]. All low-level clock zone operations were implemented using the UPPAAL-DBM library [5].

The investigated case study represents a Gear Production Stack[2] (GPS) which is depicted in Figure 4. The pipeline-like architecture sequentializes a series of stations, each specialized in a certain workpiece processing method. When a workpiece is loaded into the machine it first arrives at station 1. After some time, station 1 finishes and the workpiece is transported to station 2. After station 2 finishes processing, the piece is transported to station 3, and so on. In the extended version of the benchmark, each station comprises five additional sub-processing units, represented as one extra component per station. The task is now to synthesize a controller for the machine that transports the pieces from station to station right in time. As a critical safety property, it is required that a workpiece must be processed by all stations within a given time bound.

The incremental refinement strategy used by our prototype starts with a subnetwork that comprises only the last station in the pipeline that can cause an immediate violation of the critical timeout property. On each refinement step, our heuristic selects the preceding station to be added to the subnetwork. Each local synthesis procedure yields an approximation of the global attractor set that, in turn, is used in the subsequent synthesis procedures.

Table 1 shows the results of our evaluation. The first three columns describe the examined benchmark instance: the number of stations, whether a controller exists w.r.t. the timeout property (Sat), and whether the precise model of the processing stations (including their substations) is explicitly given as extra components (Ext). This is followed by two columns containing the explored states and running time of UPPAAL-TIGA. The last five columns summarize the results from our prototype implementation. They comprise the accumulated number of post and pre steps, i.e., the sum of all forward and backward game solving steps of all local synthesis procedures that were necessary to obtain the final result, the sizes (in number of clock zones) of the reachable state space and the attractor set of the global system, and finally the total running time of our tool.

The most significant observation is that in contrast to nonincremental controller synthesis, our approach only considers the relevant components. If the sub-processing units are included in the system description, they are ignored by the refinement algorithm. This is documented in the table by the almost equal workload of the nonextended and extended benchmark instances.

Interestingly, even though the refinement process terminates with the global controller synthesis problem, the accumulated post and pre steps are an order of magnitude smaller than those needed when synthesizing the global controller with nonincremental synthesis in the first place. This documents the effectiveness of our pruning rules based on the intermediate approximations of the global attractor set.

6 Conclusions and Outlook

We presented a novel abstraction refinement technique for timed controller synthesis that exploits the compositional nature in the design of reactive systems. Therefore, we extended the classic computational model of timed game automata by a notion of modality, which in turn, gives the possibility to obtain under- and over-approximations of the attractor set. We presented a refinement algorithm that computes a chain of abstractions with increasing precision. In each refinement cycle, a local synthesis problem on the current abstraction is solved and an intermediate attractor set approximation is computed that can be used for (1) early termination and (2) simplification of the subsequent refinement cycles. We proved the soundness of the abstractions and the termination of the refinement algorithm. An empirical evaluation provides evidence for the effectiveness of our approach in real-world applications.

In the abstraction refinement loop, the selection of transitions on which to refine and components to include in the next subnetwork can potentially be made

[2]The UPPAAL-TIGA model of the benchmark is available at `http://www.avacs.org/Benchmarks/Open/gps_controller.tgz`

			UPPAAL-Tiga		Component-based Abstraction Refinement				
Benchmark	Sat	Ext	States	Time	Post	Pre	Reach	Attr	Time
GPS 2	No	No	67	0.01	8	11	5	5	0.00
GPS 3	No	No	400	0.01	25	30	5	5	0.00
GPS 4	No	No	2751	0.03	90	97	5	5	0.00
GPS 5	No	No	14534	0.21	347	356	5	5	0.02
GPS 6	No	No	62573	1.24	1372	1383	5	5	0.15
GPS 7	No	No	205834	5.05	5469	5482	5	5	1.30
GPS 2	No	Yes	12832	0.23	8	11	6	5	0.00
GPS 3	No	Yes	1781742	362.65	25	30	6	5	0.00
GPS 4	No	Yes	MEMOUT		90	97	6	5	0.01
GPS 5	No	Yes	MEMOUT		347	356	6	5	0.03
GPS 6	No	Yes	MEMOUT		1372	1383	6	5	0.19
GPS 7	No	Yes	MEMOUT		5469	5482	6	5	1.73
GPS 2	Yes	No	67	0.00	24	30	17	16	0.01
GPS 3	Yes	No	416	0.00	93	113	65	60	0.00
GPS 4	Yes	No	3042	0.04	366	445	257	236	0.02
GPS 5	Yes	No	22891	0.34	1455	1811	1025	940	0.16
GPS 6	Yes	No	184823	3.89	5808	7475	4097	3756	1.43
GPS 7	Yes	No	1621644	47.11	23217	31004	16385	15020	37.71
GPS 2	Yes	Yes	52852	1.14	22	28	17	16	0.00
GPS 3	Yes	Yes	18767076	17457.02	87	107	65	60	0.01
GPS 4	Yes	Yes	MEMOUT		344	424	257	236	0.03
GPS 5	Yes	Yes	MEMOUT		1369	1734	1025	940	0.19
GPS 6	Yes	Yes	MEMOUT		5466	7185	4097	3756	1.74
GPS 7	Yes	Yes	MEMOUT		21851	29891	16385	15020	55.13

Table 1. Comparison of our approach with the timed game solver UPPAAL-Tiga using the Gear Production Stack (GPS) benchmark. The columns (from left to right) describe the size of the instance (number of stations), whether a controller exists (Sat), whether the stations are explicitly modeled as extra components (Ext), the explored states and the (user) running time (in seconds) of UPPAAL-Tiga, the accumulated post and pre steps of our prototype, the sizes (in terms of zones) of the reachable state space and the attractor set of the global system, as well as the (user) running time (in seconds) of our prototype. All benchmarks were measured on an AMD Opteron processor with 2.6 GHz and 4GB RAM.

in a more sophisticated way. In particular, we believe that a high-level analysis of the communication structure of the component network, a low-level analysis of the arising clock zones, or a thorough analysis of abstract strategies found so far are good starting points to develop effective refinement heuristics.

Another interesting direction for further investigation is a fully symbolic representation of both discrete and continuous parts of the reachable state space and the attractor set. Particularly our approach, in which an under- and over-approximation of the attractor set in the refinement loop is maintained, would greatly benefit from a fully symbolic state space treatment. Promising starting points are symbolic techniques from the real-time and hybrid model checking domain [21, 24, 11].

Acknowledgment. This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

The authors want to thank Bernd Finkbeiner for useful comments and fruitful discussions.

References

[1] K. Altisen and S. Tripakis. Tools for controller synthesis of timed systems. In *2nd Workshop on Real-Time Tools (RT-TOOLS'02)*, 2002.

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

[3] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In J.-F. Lafay,

editor, *Proc. 5th IFAC Conference on System Structure and Control*, pages 469–474. Elsevier, 1998.

[4] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. UPPAAL-Tiga: Time for playing games! In W. Damm and H. Hermanns, editors, *Proc. 19th International Conference on Computer Aided Verification (CAV'07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007.

[5] J. Bengtsson. *Clocks, DBM, and States in Timed Systems*. PhD thesis, Uppsala University, 2002.

[6] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. of the American Mathematical Society*, 138:295–311, 1969.

[7] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In M. Abadi and L. de Alfaro, editors, *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005.

[8] F. Cassez, J. J. Jessen, K. G. Larsen, J.-F. Raskin, and P.-A. Reynier. Automatic synthesis of robust and optimal controllers - an industrial case study. In R. Majumdar and P. Tabuada, editors, *Proc. 12th International Conference on Hybrid Systems: Computation and Control (HSCC'09)*, volume 5469 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2009.

[9] A. Church. Logic, arithmetic and automata. In V. Stenstrom, editor, *Proc. International Congress of Mathematicians 1962*, pages 23–25, Uppsala, 1963.

[10] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. A. Emerson and A. P. Sistla, editors, *Proc. 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[11] W. Damm, S. Disch, H. Hungar, S. Jacobs, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Exact state set representations in the verification of linear hybrid systems with large discrete state space. In K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *Proc. 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, volume 4762 of *Lecture Notes in Computer Science*, pages 425–440. Springer, 2007.

[12] L. de Alfaro, T. A. Henzinger, and R. Majumdar. Symbolic algorithms for infinite-state games. In K. G. Larsen and M. Nielsen, editors, *Proc. 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 536–550. Springer, 2001.

[13] L. de Alfaro and P. Roy. Solving games via three-valued abstraction refinement. In L. Caires and V. T. Vasconcelos, editors, *Proc. 18th International Conference on Concurrency Theory (CONCUR'07)*, volume 4703 of *Lecture Notes in Computer Science*, pages 74–89. Springer, 2007.

[14] K. Dräger, B. Finkbeiner, and A. Podelski. Directed model checking with distance-preserving abstractions. *STTT*, 11(1):27–37, 2009.

[15] D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In H. Alt and A. Ferreira, editors, *Proc. 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 571–582. Springer, 2002.

[16] S. Graf, B. Steffen, and G. Lüttgen. Compositional minimisation of finite state systems using interface specifications. *Formal Aspects of Computation*, 8(5):607–616, 1996.

[17] T. A. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP'03)*, volume 2719 of *Lecture Notes in Computer Science*, pages 886–902. Springer, 2003.

[18] M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In D. Sands, editor, *Proc. 10th European Symposium on Programming Languages and Systems (ESOP'01)*, volume 2028 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2001.

[19] J. J. Jessen, J. I. Rasmussen, K. G. Larsen, and A. David. Guided controller synthesis for climate controller using Uppaal Tiga. In J.-F. Raskin and P. S. Thiagarajan, editors, *Proc. 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 2007.

[20] K. G. Larsen. Modal specifications. In J. Sifakis, editor, *Proc. International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 232–246. Springer-Verlag, 1990.

[21] K. G. Larsen, C. Weise, W. Yi, and J. Pearson. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.

[22] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In E. W. Mayr and C. Puech, editors, *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995.

[23] S. Tripakis and K. Altisen. On-the-fly controller synthesis for discrete and dense-time systems. In *World Congress on Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 233–252. Springer, 1999.

[24] F. Wang. Region encoding diagram for fully symbolic verification of real-time systems. In *Proc. 24th International Computer Software and Applications Conference (COMPSAC'00)*, pages 509–515. IEEE Computer Society, 2000.