Template-Based Controller Synthesis for Timed Systems

Bernd Finkbeiner and Hans-Jörg Peter

Reactive Systems Group
Universität des Saarlandes, Germany

Abstract. We present an effective controller synthesis method for real-time systems modeled as timed automata with safety requirements. Under the realistic assumption of partial observability, the problem is undecidable in general, and prohibitively expensive (2ExpTime-complete) if a bound on the granularity of the controller is set in advance. We investigate the synthesis of controllers from templates, given as timed automata with parametric control structure. Template-based synthesis is significantly cheaper (PSpace-complete) than standard synthesis and produces much simpler controllers. We present an efficient symbolic synthesis algorithm based on automatic abstraction refinement and report on encouraging experimental results from an implementation in the timed verification and synthesis tool Synthia.

1 Introduction

In controller synthesis, we automatically transform a given model of a plant and a safety requirement into a finite-state controller that monitors and affects the ongoing behavior of the plant to ensure the safety of its operation. There has been a lot of recent progress [1,8,5,24,23] in synthesizing controllers for real-time systems, where the plant is given as a timed automaton. Notably, the Uppaal-Tiga tool [5], which is based on the popular timed model checker Uppaal, has extended the highly efficient state-space traversal based on symbolic zone representations from verification to synthesis.

Unfortunately, timed controller synthesis quickly turns into an intractable problem if one makes the realistic assumption that the controller does not have access to the full state of the plant, but rather only sees a subset of the events. Under *partial observability*, the controller synthesis problem is undecidable in general, and remains prohibitively expensive (2ExpTime-complete) if the problem is made decidable by fixing a bound on the granularity of the controller in advance [6]. Furthermore, since the size of the controller may be doubly-exponential in the size of the plant, it is often infeasible to actually construct the controller.

In this paper, we propose a new synthesis approach, where the size and general shape of the controller is fixed in advance in the form of a *template*. A template is a timed or untimed automaton with parametric control structure. Figures 1

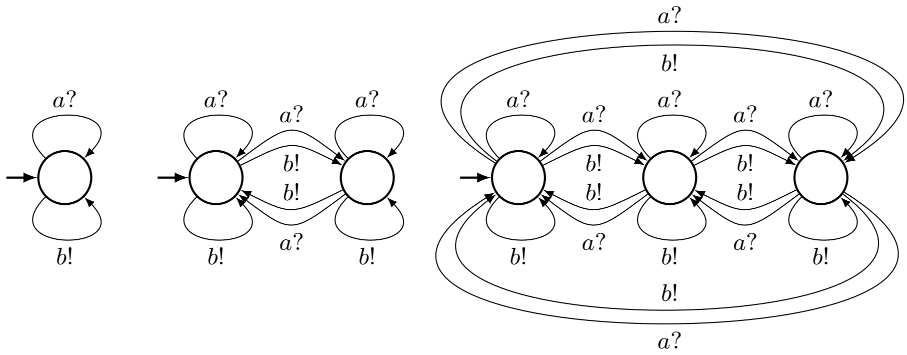

Fig. 1. Full template with one, two, and three locations

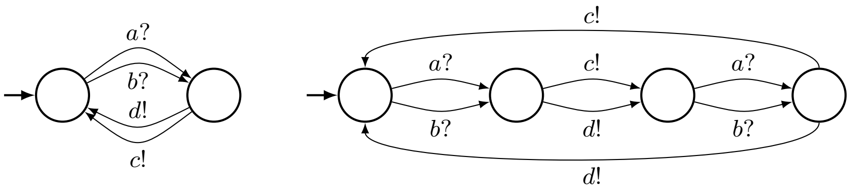

Fig. 2. Cyclic-executive template with two and four locations

and 2 show two example template families. In the *full template*, shown in Figure 1, every pair of locations is connected by an edge for every possible action. In the *cyclic-executive template*, shown in Figure 2, the controller implements some schedule according to which the actions are handled; the controller alternates between waiting for an uncontrollable action and responding with some controllable action. The template families are organized according to the number of locations. Typically, we start the synthesis process with small templates and then iteratively increase the size until an optimal controller is found. A controller *matches* a template if it can be obtained by removing a subset of the edges. We encode the presence of edges using Boolean parameter variables and combine the resulting parametric timed automaton with the plant automaton: any valuation of the Boolean parameters under which only safe states are reachable represents a correct controller.

Template-based synthesis has several attractive features: Since the observations of the controller are limited by the template, template-based synthesis naturally solves the controller synthesis problem with partial observability. The size of the controller is also limited by the size of the template. Because the templates model standard types of controllers, the synthesized controllers are well-structured, resembling a manually built controller.

In terms of complexity, it is not surprising that template-based synthesis is much simpler than standard synthesis. The problem is PSpace-complete, matching the complexity of model checking. Template-based synthesis can in fact be understood as parametric model checking, where we verify a timed automaton that is parametrized with Boolean variables.

The technical challenge in developing fast algorithms for template-based synthesis thus lies in the efficient manipulation of the potential valuations of the parameters. In the paper, we present a solution to this challenge based on automatic abstraction refinement. Starting with an initial abstraction that considers all parameter valuations, a refinement loop incrementally focuses the search towards smaller and smaller sets of parameter valuations. The loop terminates as soon as, for the remaining parameter valuations, only safe states are reachable. New refinements are computed by identifying situations where an unsafe state is reachable for a subset of the parameter valuations.

Our experimental results indicate that template-based synthesis is not only an effective solution to the controller synthesis problem with partial observability, it is an attractive alternative to standard synthesis also in the simpler case of complete observability, outperforming tools like Uppaal-Tiga on benchmarks where structurally simple controllers, corresponding to the available templates, exist.

Related Work. The basic timed controller synthesis problem for timed automata [2] with complete observability was defined by Maler et al. [21,4] in the setting of turn-based timed games. In their fundamental work, the decidability of the problem was shown by demonstrating that the standard discrete attractor construction [26] on the region graph suffices to obtain winning strategies. Henzinger and Kopke showed that this construction is theoretically optimal by proving that the problem is ExpTime-complete [17]. D'Souza and Madhusudan investigated the complexity of timed controller synthesis against external specifications [12]. Bouyer et al. continued this line of research and also investigated the impact of partial information [6].

A first more practical approach to timed controller synthesis, implemented in the tool SynthKro, was proposed by Altisen and Tripakis [1]. The approach requires, however, an expensive preprocessing step. Cassez et al. presented a symbolic algorithm, implemented in Uppaal-Tiga, that avoids the upfront state explosion by combining the backward attractor construction with a forward zone graph exploration [8,5]. Our timed verification and synthesis tool Synthia [23] is based on abstraction refinement techniques that combine symbolic representations for the discrete and the continuous state components [14] and exploit the compositional structure of the timed system [24]. We have implemented the approach presented in this paper as an extension of Synthia.

Compared to the significant body of work on timed controller synthesis with complete observability, there has been comparatively little work on the more realistic setting of partial observability. Fundamental results on the undecidability of the general problem and the complexity for fixed granularity are due to Bouyer et al. [6]. Cassez et al. proposed a pragmatic approach to handle partial information, which restricts the choices and the observability of the controller so that a zone-based synthesis algorithm remains possible [9]. An extension of this work uses *alternating timed simulation relations* to efficiently control partially observable systems [10]. This approach has also been implemented in Uppaal-Tiga.

Template-based synthesis is related to the *bounded synthesis* approach [25], where one fixes the size (but not the structure) of the controller. Bounded synthesis has so far been limited to purely discrete systems. There are efficient algorithms for bounded synthesis based on SMT-solving [16], antichains [15], and BDDs [13], which, however, unfortunately do not seem to have straightforward extensions to the timed case. Another interesting restriction on the type of controllers to be considered has been proposed by Lustig et al.: *synthesis from component libraries* [20] attempts to construct a controller by assembling routines from a given library. The difference to template-based synthesis is that the synthesized controller is a combination of predefined components rather than an instantiation of a parametric template. Currently, this approach is also limited to discrete systems.

2 Timed Systems

Timed Automata. The components of a timed system are represented by *timed automata*. A timed automaton [2] is a tuple $\mathcal{A} = (L, l_0, \Sigma, \Delta, X, I)$, where L is a finite set of (control) locations, $l_0 \in L$ is the initial location, Σ is a finite set of actions, $\Delta \subseteq (L \times \Sigma \times \mathcal{C}(X) \times 2^X \times L)$ is an edge relation, X is a finite set of real valued clocks, $I : L \to \mathcal{C}(X)$ maps each location to an invariant, and $\mathcal{C}(X)$ is the set of clock constraints over X. A (rectangular) clock constraint $\varphi \in \mathcal{C}(X)$ is of the form

$$\varphi = \mathbf{true} \mid x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2,$$

where x is a clock in X and c is a constant in $\mathbb{N}_0$. A *clock valuation* $\boldsymbol{t} : X \to \mathbb{R}_{\geq 0}$ assigns a nonnegative value to each clock and can also be represented by a $|X|$-dimensional vector $\boldsymbol{t} \in \mathcal{R}$, where $\mathcal{R} = \mathbb{R}_{\geq 0}^X$ denotes the set of all clock valuations.

The states of a timed automaton are pairs $(l, \boldsymbol{t})$ of locations and clock valuations. Timed automata have two types of transitions: *timed transitions*, where only time passes and the location remains unchanged, and *discrete transitions*, where no time passes, the current location can be changed and some clocks can be reset to zero. In a timed transition, denoted by $(l, \boldsymbol{t}) \xrightarrow{d} (l, \boldsymbol{t} + d \cdot \mathbf{1})$, the same nonnegative value $d \in \mathbb{R}_{\geq 0}$ is added to all clocks such that, for each $0 \leq d' \leq d$, $\boldsymbol{t} + d'$ satisfies the location invariant $I(l)$. A discrete transition, denoted by $(l, \boldsymbol{t}) \xrightarrow{a} (l', \boldsymbol{t}')$ for some action $a \in \Sigma$, corresponds to an edge $\delta = \langle l, a, \varphi, \lambda, l' \rangle$ of Δ such that $\boldsymbol{t}$ satisfies the clock constraint φ of δ, and $\boldsymbol{t}' = \boldsymbol{t}[\lambda := 0]$ is obtained from $\boldsymbol{t}$ by setting the clocks in λ to 0 and satisfies the location invariant $I(l')$. For two states s and s', we write $s \xrightarrow{\delta} s'$ if there is a delay $d \in \mathbb{R}_{\geq 0}$ and an edge δ with action a such that there is an s'' with $s \xrightarrow{d} s''$ and $s'' \xrightarrow{a} s'$.

We say that a state s is *reachable* if there is a finite sequence of transitions of the form $s_0 \xrightarrow{\delta_0} s_1 \dots s_{n-1} \xrightarrow{\delta_{n-1}} s$ such that $\delta_0, \dots, \delta_{n-1} \in \Delta$ are edges in Δ,

$s_0 = (l_0, \mathbf{0})$ is the initial state (where $\mathbf{0}$ is the zero vector), and for all $1 \leq i \leq n$, the individual $s_i = (l_i, \boldsymbol{t}_i)$ are states of the automaton. We define $\mathsf{Reach}(\mathcal{A})$ as the set of all forward reachable states of a timed automaton $\mathcal{A}$.

Composition. Timed automata can be composed to networks, in which the automata run in parallel and synchronize on shared actions. For two timed automata $\mathcal{A} = (L_1, l_0^1, \Sigma_1, \Delta_1, X_1, I_1)$ and $\mathcal{A}' = (L_2, l_0^2, \Sigma_2, \Delta_2, X_2, I_2)$ with disjoint clock sets $X_1 \cap X_2 = \emptyset$, the *parallel composition* $\mathcal{A}_1 \| \mathcal{A}_2$ is the timed automaton $(L_1 \times L_2, (l_0^1, l_0^2), \Sigma_1 \cup \Sigma_2, \Delta, X_1 \cup X_2, I)$, where $I(l_1, l_2) = I_1(l_1) \wedge I_2(l_2)$ for all $l_1 \in L_1$ and $l_2 \in L_2$, and Δ is the smallest set that 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 $a \in \Sigma_1 \setminus \Sigma_2$, $\langle (l_1, l_2), a, \varphi_1, \lambda_1, (l_1', l_2) \rangle$ if $\langle l_1, a, \varphi_1, \lambda_1, l_1' \rangle \in \Delta_1$, and
- for $a \in \Sigma_2 \setminus \Sigma_1$, $\langle (l_1, l_2), a, \varphi_2, \lambda_2, (l_1, l_2') \rangle$ if $\langle l_2, a, \varphi_2, \lambda_2, l_2' \rangle \in \Delta_2$.

Finite Semantics. The decidability of the reachability problem of timed automata relies on the existence of the *region equivalence relation* [2] on $\mathcal{R}$ which has a finite index.

For a timed automaton $\mathcal{A} = (L, l_0, \Sigma, \Delta, X, I)$, we call the value of a clock $x \in X$ *maximal* if it is strictly greater than the highest constant c_{max} any clock is compared to. We say that two clock valuations $\boldsymbol{t}_1, \boldsymbol{t}_2 \in \mathcal{R}$ are in the same *clock region*, denoted $\boldsymbol{t}_1 \sim_R \boldsymbol{t}_2$, if

- the set of clocks with maximal value is the same in $\boldsymbol{t}_1$ and in $\boldsymbol{t}_2$ ($\forall x \in X : \boldsymbol{t}_1(x) > c_{max} \Leftrightarrow \boldsymbol{t}_2(x) > c_{max}$), and
- $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ agree (1) on the integer parts of the clock values, (2) on the relative order of the noninteger parts of the clock values, and (3) on the equality of the noninteger parts of the clock values with 0. That is, for all clocks x and y with nonmaximal value, it holds that (1) $\lfloor \boldsymbol{t}_1(x) \rfloor = \lfloor \boldsymbol{t}_2(x) \rfloor$, (2) $\widehat{\boldsymbol{t}}_1(x) \leq \widehat{\boldsymbol{t}}_1(y) \Leftrightarrow \widehat{\boldsymbol{t}}_2(x) \leq \widehat{\boldsymbol{t}}_2(y)$, and (3) $\widehat{\boldsymbol{t}}_1(x) = 0$ if, and only if, $\widehat{\boldsymbol{t}}_2(x) = 0$, where $\widehat{\boldsymbol{t}}_i(x) = \boldsymbol{t}_i(x) - \lfloor \boldsymbol{t}_i(x) \rfloor$ for $i \in \{1, 2\}$.

We denote with $[\boldsymbol{t}]_R = \{\boldsymbol{t}' \in \mathcal{R} \mid \boldsymbol{t} \sim_R \boldsymbol{t}'\}$ the clock region $\boldsymbol{t}$ belongs to. We say that two states $s_1 = (l_1, \boldsymbol{t}_1)$ and $s_2 = (l_2, \boldsymbol{t}_2)$ of $\mathcal{A}$ are *region-equivalent*, denoted by $s_1 \sim_R s_2$, if their locations are the same ($l_1 = l_2$) and the clock valuations are in the same clock region ($\boldsymbol{t}_1 \sim_R \boldsymbol{t}_2$), and denote with $[(l, \boldsymbol{t})]_R = \{(l, \boldsymbol{t}') \in L \times \mathcal{R} \mid \boldsymbol{t} \sim_R \boldsymbol{t}'\}$ the equivalence class of region-equivalent states that $(l, \boldsymbol{t})$ belongs to.

Regions are a suitable semantics for the abstraction of timed automata because they essentially preserve the language: if there is a discrete transition $s \xrightarrow{a} s'$ from a state s to a state s' of a timed automaton, then there is, for all states r with $r \sim_R s$, a state r' with $r' \sim_R s'$ such that $r \xrightarrow{a} r'$ is a discrete transition with the same label. For timed transitions, a slightly weaker property holds: if there is a timed transition $s \xrightarrow{t} s'$ from a state s to a state s', then there is, for all states r with $r \sim_R s$, a state r' with $r' \sim_R s'$ such that there is a timed transition $r \xrightarrow{t'} r'$ (but possibly with $t' \neq t$).

The *finite semantics* of a timed automaton $\mathcal{A} = (L, l_0, \Sigma, \Delta, X, I)$ is the finite graph $[\![\mathcal{A}]\!] = (Q, q_0, T)$ where

- the symbolic state set $Q = \{[(l, \boldsymbol{t})]_R \mid (l, \boldsymbol{t}) \in L \times \mathcal{R}\}$ of $[\![\mathcal{A}]\!]$ is the set of equivalence classes of region-equivalent states of $\mathcal{A}$, with
- the initial state $q_0 = [(l_0, \boldsymbol{t}_0)]_R$, and
- the set $T = \{(q, q') \in Q \times Q \mid \exists r \in q,\ r' \in q', a \in \Sigma \cup \mathbb{R}_{\geq 0}.\, r \xrightarrow{a} r'\}$ of transitions.

The finite semantics is reachability-preserving:

Lemma 1. *[2] For a timed automaton $\mathcal{A} = (L, l_0, \Sigma, \Delta, X, I)$ there is a finite path from a state $(l, \boldsymbol{t})$ to a state $(l', \boldsymbol{t}')$ if, and only if, there is a finite path from $\big[(l, \boldsymbol{t})\big]_R$ to $\big[(l', \boldsymbol{t}')\big]_R$ in $[\![\mathcal{A}]\!]$.*

Assuming a binary encoding of the constants in the clock constraints, the number of states of the finite semantics is exponential in the number of clocks and in the magnitude of the constants:

Lemma 2. *[2] For a timed automaton $\mathcal{A} = (L, l_0, \Sigma, \Delta, X, I)$, with c_x as the maximal constant appearing in any constraint of $\mathcal{A}$, the number of states of $[\![\mathcal{A}]\!]$ is bounded by*

$$|L| \cdot |X|! \cdot 2^{|X|-1} \cdot \prod_{x \in X} O(c_x) = |L| \cdot |X|! \cdot O(c_x)^{|X|}.$$

As it turns out, the finite semantics is a theoretically optimal state space representation for deciding reachability:

Theorem 1. *[2] For a timed automaton $\mathcal{A}$ and a set of states B, testing whether $\mathsf{Reach}(\mathcal{A}) \cap B = \emptyset$ is* PSPACE*-complete.*

In practice, instead of deciding $\mathsf{Reach}(\mathcal{A}) \cap B = \emptyset$ based on an explicit construction of the finite semantics, tools like SYNTHIA or UPPAAL use the much coarser *clock zones* as the fundamental representation of clock values.

3 Template-Based Controller Synthesis

In this section, we formalize controller templates and the template instantiation problem. A *controller template* is a tuple $(\mathcal{T}, P, \Pi)$ consisting of a timed automaton $\mathcal{T} = (L, l_0, \Sigma, \Delta, X, I)$, a finite set of Boolean parameters P, and a total function $\Pi : \mathcal{P} \to 2^{\Delta}$ defining which edges are enabled for a given parameter valuation, where $\mathcal{P} = P \to \mathbb{B}$ is the set of all parameter valuations. In the following, we will assume that the timed automaton modeling the environment (or plant) is already integrated (by parallel composition) in $\mathcal{T}$. As usual, we assume that the controller does neither reset plant clocks, inhibit plant actions, nor introduce timelocks.

Definition 1. *For a controller template $(\mathcal{T}, P, \Pi)$ with $\mathcal{T} = (L, l_0, \Sigma, \Delta, X, I)$ and a set of bad states B, the* instantiation problem *asks for a parameter valuation $\boldsymbol{p} \in \mathcal{P}$ such that $\mathcal{I} = (L, l_0, \Sigma, \Pi(\boldsymbol{p}), X, I)$ and $\mathsf{Reach}(\mathcal{I}) \cap B = \emptyset$.*

We call an instantiation of the template that satisfies the condition of the definition *feasible.* Synthesizing a template-based controller corresponds to *statically* finding a feasible instantiation. This is in contrast to the classical formulation of the timed controller synthesis problem [21,4], where the controller is an arbitrary timed automaton whose behavior depends *dynamically* on the observed events of the plant.

The complexity of the template instantiation problem is the same as the complexity of standard timed model checking: the exponential size of the region graph dominates the size of the search space (the possible valuations of the parameters).

Theorem 2. *The instantiation problem for a controller template* $(\mathcal{T}, P, \Pi)$ *and a set of bad states* B *is* PSpace*-complete.*

We note that the synthesis of controllers with full observability is already ExpTime-complete [17]. The case where the controller can only observe a subset of the events of the plant, is even undecidable in general, or 2ExpTime-complete if the granularity (number and precision of the clocks) of the controller is bounded in advance [6]. Furthermore, the size of the controller can be exponential or even, in case of partial observability, doubly-exponential in the size of the plant. Template-based synthesis is not only much cheaper (PSpace-complete), it also has the advantage that the size of the controller is fixed in advance.

Template-based synthesis thus provides a much more promising setting for effective controller synthesis than the standard approach. The remainder of the paper is devoted to the development of an efficient template-based synthesis algorithm and an experimental evaluation.

4 Symbolic Parameter Synthesis

We now present a symbolic algorithm for finding feasible instantiations for a given controller template $(\mathcal{T}, P, \Pi)$ with $\mathcal{T} = (L, l_0, \Sigma, \Delta, X, I)$ and a set of bad states $B \subseteq S$, where S is the set of states of $\mathcal{T}$. In the rest of this section, we assume that $\mathcal{T}$, P, Π, S, and B are fixed.

We develop the algorithm in three steps: first, we describe the immediate, exact, computation of the set of feasible instantiations based on forward and backward propagation; then we give an approximate computation based on an abstraction of the template; finally, we describe an abstraction refinement procedure, which increases the precision of the approximate computation until either a feasible instantiation has been found, or it has been shown that no feasible instantiation exists.

4.1 Precise Computation of the Feasible Instantiations

The precise set of feasible instantiations can be computed in a standard fixed point construction that either starts from the initial state and propagates, in a forward manner, the reachable combinations of states and parameter valuations, or starts with the bad states and propagates, in a backward manner, those combinations of states and parameter valuations that have a path to the bad states.

To accommodate both directions, we define a successor and a predecessor propagation function $\mathsf{Succ}, \mathsf{Pred} : 2^{S\times\mathcal{P}} \to 2^{S\times\mathcal{P}}$ with

$$\mathsf{Succ}(Y) = \left\{(s', \boldsymbol{p}) \in S \times \mathcal{P} \mid \exists\delta \in \Pi(\boldsymbol{p}) : \exists s \in S : (s, \boldsymbol{p}) \in Y \wedge s \xrightarrow{\delta} s'\right\} \text{ and}$$
$$\mathsf{Pred}(Y') = \left\{(s, \boldsymbol{p}) \in S \times \mathcal{P} \mid \exists\delta \in \Pi(\boldsymbol{p}) : \exists s' \in S : (s', \boldsymbol{p}) \in Y' \wedge s \xrightarrow{\delta} s'\right\}.$$

The set FR of forward-reachable states and parameter valuations and the set BR of backward-reachable states and parameter valuations are obtained by the following fixed point computations (the index identifies the round of the fixpoint iteration):

$$\begin{array}{ll@{\qquad\qquad}ll}
\mathsf{FR}_0 & = \{(l_0, \mathbf{0})\} \times \mathcal{P} & \mathsf{BR}_0 & = B \times \mathcal{P} \\
\mathsf{FR}_{i+1} & = \mathsf{Succ}(\mathsf{FR}_i) \cup \mathsf{FR}_i & \mathsf{BR}_{i+1} & = \mathsf{Pred}(\mathsf{BR}_i) \cup \mathsf{BR}_i \\
\mathsf{FR} & = \lim_i \mathsf{FR}_i & \mathsf{BR} & = \lim_i \mathsf{BR}_i.
\end{array}$$

Clearly, if there is some $(s, \boldsymbol{p}) \in \mathsf{FR}_i$ then this means that state s is reached after $i \in \mathbb{N}$ forward steps for parameter valuation $\boldsymbol{p}$, which corresponds to a path $s_0 \xrightarrow{\delta_1} s_1 \xrightarrow{\delta_2} \ldots \xrightarrow{\delta_i} s$, where each $\delta_1, \delta_2, \ldots, \delta_i$ is in $\Pi(\boldsymbol{p})$. Dually, if there is some state $(s, \boldsymbol{p}) \in \mathsf{BR}_i$ then this means that state s is reached after $i \in \mathbb{N}$ backward steps for parameter valuation $\boldsymbol{p}$, which corresponds to a path $s \xrightarrow{\delta_1} s_1 \xrightarrow{\delta_2} \ldots \xrightarrow{\delta_i} b$, where $b \in B$ and each $\delta_1, \delta_2, \ldots, \delta_i$ is in $\Pi(\boldsymbol{p})$.

We can obtain the feasible instantiations either by looking for parameter valuations in FR that are not paired up with bad states, or by looking for parameter valuations in BR that are not paired up with the initial state. Both constructions identify the same set of feasible instantiations.

Theorem 3. *The set*

$$G = \{\boldsymbol{p} \in \mathcal{P} \mid (B \times \{\boldsymbol{p}\}) \cap \mathsf{FR} = \emptyset\} = \{\boldsymbol{p} \in \mathcal{P} \mid ((l_0, \mathbf{0}), \boldsymbol{p}) \notin \mathsf{BR}\}$$

consists of exactly the feasible instantiations.

In practice, neither construction performs well. The problem is that it is difficult and expensive to maintain the correlation between parameter valuations and reachable states; typically, each parameter valuation results in a different set of states.

Instead of directly computing the precise set of parameter valuations, in the next subsection, we will present an abstraction technique that allows us to reason about approximations of parameter valuations.

4.2 The Focus Abstraction

We now consider an abstraction of the template based on a given set $P \subseteq \mathcal{P}$ of parameter valuations, which we call *focus*. We use the parameter valuations in P to obtain an over- or underapproximation of the sets FR and BR, by considering P as an equivalence class: we require that a transition must exist for some or all parameter valuations in P, respectively. In the following, we use an overapproximation for the forward construction and an underapproximation for the backward construction; obviously, all constructions can also be dualized. We obtain the following approximate successor and predecessor functions: $\overline{\mathsf{Succ}}^P, \underline{\mathsf{Pred}}^P : 2^S \to 2^S$ with

$$\overline{\mathsf{Succ}}^P(Y) = \left\{ s' \in S \mid \exists \boldsymbol{p} \in P : \exists \delta \in \Pi(\boldsymbol{p}) : \exists s \in Y : s \xrightarrow{\delta} s' \right\} \text{ and}$$
$$\underline{\mathsf{Pred}}^P(Y') = \left\{ s \in S \mid \forall \boldsymbol{p} \in P : \exists \delta \in \Pi(\boldsymbol{p}) : \exists s' \in Y' : s \xrightarrow{\delta} s' \right\}.$$

Replacing the precise Succ and Pred operators in the fixed point construction from Subsection 4.1, we obtain two new fixed point constructions for the approximations $\overline{\mathsf{FR}}^P$ and $\underline{\mathsf{BR}}^P$:

$$\begin{array}{ll@{\qquad\qquad}ll}
\overline{\mathsf{FR}}_0^P & = \{(l_0, \mathbf{0})\} & \underline{\mathsf{BR}}_0^P & = B \\
\overline{\mathsf{FR}}_{i+1}^P & = \overline{\mathsf{Succ}}^P(\overline{\mathsf{FR}}_i^P) \cup \overline{\mathsf{FR}}_i^P & \underline{\mathsf{BR}}_{i+1}^P & = \underline{\mathsf{Pred}}^P(\underline{\mathsf{BR}}_i^P) \cup \underline{\mathsf{BR}}_i^P \\
\overline{\mathsf{FR}}^P & = \lim_i \overline{\mathsf{FR}}_i^P & \underline{\mathsf{BR}}^P & = \lim_i \underline{\mathsf{BR}}_i^P .
\end{array}$$

Clearly, if there is some state $s \in \overline{\mathsf{FR}}_i^P$ then this means that state s is reached after $i \in \mathbb{N}$ forward steps for a set of parameter valuations P, which corresponds to a path $s_0 \xrightarrow{\delta_1} s_1 \xrightarrow{\delta_2} \ldots \xrightarrow{\delta_i} s$, where, for each δ_i, there is a $\boldsymbol{p}_i \in P$ such that δ_i in $\Pi(\boldsymbol{p}_i)$. Dually, if there is some state $s \in \underline{\mathsf{BR}}_i^P$ then this means that state s is reached after $i \in \mathbb{N}$ backward steps for a set of parameter valuations P, which corresponds to a path $s \xrightarrow{\delta_1} s_1 \xrightarrow{\delta_2} \ldots \xrightarrow{\delta_i} b$, where $b \in B$ and, for each δ_i and each $\boldsymbol{p} \in P$, we have δ_i in $\Pi(\boldsymbol{p})$.

The following lemma clarifies the relationships between the approximate and precise versions of FR and BR: $\overline{\mathsf{FR}}^P$ overapproximates FR on P, $\underline{\mathsf{BR}}^P$ underapproximates BR on P.

Lemma 3. *For every set $P \subseteq \mathcal{P}$ of parameter valuations, it holds that*

$$\overline{\mathsf{FR}}^P \supseteq \{s \in S \mid \exists \boldsymbol{p} \in P : (s, \boldsymbol{p}) \in \mathsf{FR}\} \text{ and}$$
$$\underline{\mathsf{BR}}^P \subseteq \{s \in S \mid \exists \boldsymbol{p} \in P : (s, \boldsymbol{p}) \in \mathsf{BR}\}.$$

Combining Lemma 3 with Theorem 3, we obtain that the focus abstraction allows us to approximate the set of feasible instantiations: A set of parameter valuations P definitely represents feasible instantiations if no bad states appear in $\overline{\mathsf{FR}}^P$. Dually, the parameter valuations in P definitely represent infeasible instantiations if the initial state appears in $\underline{\mathsf{BR}}^P$. Hence, we obtain the following lower and upper bounds for the set of feasible instantiations.

Theorem 4. *Let G be the precise set of feasible instantiations. For every set $P \subseteq \mathcal{P}$, it holds that*

$$\left\{\boldsymbol{p} \in P \mid B \cap \overline{\mathsf{FR}}^P = \emptyset\right\} \subseteq G \subseteq \left\{\boldsymbol{p} \in \mathcal{P} \mid \boldsymbol{p} \in P \Rightarrow (l_0, \mathbf{0}) \notin \underline{\mathsf{BR}}^P\right\}.$$

In the next subsection, we will describe an automatic refinement algorithm for the Focus abstraction.

4.3 Abstraction Refinement

We now describe a refinement procedure that computes an increasingly precise approximation of the set of feasible instantiations. The procedure starts with the set $\mathcal{P}$ of all parameter valuations, and then splits the set into smaller and smaller subsets, until either a feasible instance is found, or it is established that no feasible instance exists.

Algorithm 1. $\mathsf{Solve}(P)$: The algorithm computes a safe subset of a given set P of parameter valuations, or returns fail if no safe subset exists.

```
1: if $P = \emptyset$ then
2:   return fail
3: else if $(l_0, \mathbf{0}) \in \underline{\mathsf{BR}}^P$ then
4:   return fail
5: else if $\overline{\mathsf{FR}}^P \cap \underline{\mathsf{BR}}^P = \emptyset$ then
6:   return $P$
7: else
8:   $P_1 := \mathsf{Refine}(P)$
9:   $R_1 := \mathsf{Solve}(P_1)$
10:  if $R_1 \neq \mathsf{fail}$ then
11:    return $R_1$
12:  else
13:    $P_2 := P \setminus P_1$
14:    return $\mathsf{Solve}(P_2)$
```

The procedure is shown as Algorithm 1. The input to the procedure is the current focus P, for which we initially use $\mathcal{P}$. Unless the (un)reachability of some bad state can be surely established, after each refinement step, Solve recurs on the refined focus. In each call of Solve, the set of bad states are augmented with the states in $\underline{\mathsf{BR}}^P$. This is justified by the following lemmas, which state that the old underapproximation $\underline{\mathsf{BR}}^P$ is a subset of the new underapproximation $\underline{\mathsf{BR}}^{P'}$ for a refinement $P' \subset P$, and that excluding $\underline{\mathsf{BR}}^P$ from $\overline{\mathsf{FR}}^P$ does not affect the resulting upper bound on the feasible instantiations.

Lemma 4. *For two sets $P, P' \subseteq \mathcal{P}$ of parameter valuations such that $P' \subset P$, it holds that $\underline{\mathsf{BR}}^P \subseteq \underline{\mathsf{BR}}^{P'}$.*

Lemma 5. *For every set $P \subseteq \mathcal{P}$ of parameter valuations, it holds that*

$$\left\{\boldsymbol{p} \in P \mid B \cap \overline{\mathsf{FR}}^P = \emptyset\right\} = \left\{\boldsymbol{p} \in P \mid \underline{\mathsf{BR}}^P \cap \overline{\mathsf{FR}}^P = \emptyset\right\}.$$

It remains to specify the function Refine, which is called in procedure Solve to find an appropriate subset of $\mathcal{P}$ to split on. Since $\mathcal{P}$ is finite, we could, in principle, choose any strict (and non-empty) subset of $\mathcal{P}$ during the refinement step. In the following we describe a heuristic choice that has proved useful in practice: we choose a set of parameter valuations that are guaranteed to increase $\underline{\mathsf{BR}}^P$ in the next iteration.

Suppose the termination conditions of procedure Solve are not true yet, i.e., the initial state is not in $\underline{\mathsf{BR}}^P$ and there are still states in $\overline{\mathsf{FR}}^P \cap \underline{\mathsf{BR}}^P$. We choose a state $s \in \overline{FR}^P \setminus \underline{\mathsf{BR}}^P$ and a state $s' \in \underline{BR}^P$, such that there exists a transition $\delta \in \Pi(\boldsymbol{p})$ that leads from s to s' for some $\boldsymbol{p} \in P$, but not for all $\boldsymbol{p} \in P$. The refinement proceeds with the parameter valuations that allow a transition from s to s':

$$\mathsf{Refine}(P) = \{\boldsymbol{p} \in P \mid \exists \delta \in \Pi(\boldsymbol{p}) : s \xrightarrow{\delta} s'\}$$

Since such a pair s, s' of states can be found until the termination conditions of procedure Solve become true, we obtain that Refine always ensures progress of our refinement algorithm.

Lemma 6. *For every set of parameter valuations $P \subseteq \mathcal{P}$, if $P \neq \emptyset$, $(l_0, \mathbf{0}) \notin \underline{\mathsf{BR}}^P$, and $\overline{\mathsf{FR}}^P \cap \underline{\mathsf{BR}}^P \neq \emptyset$, then there is a state $s \in \overline{\mathsf{FR}}^P \setminus \underline{\mathsf{BR}}^P$ and a state $s' \in \underline{\mathsf{BR}}^P$ such that*

$$\emptyset \subset \mathsf{Refine}(P) \subset P.$$

Putting everything together, we obtain the following correctness theorem for $\mathsf{Solve}(\mathcal{P})$, where Lemma 6 guarantees termination and Theorem 4 guarantees soundness of the result.

Theorem 5. *Called with the set $\mathcal{P}$ of parameter valuations, Procedure $\mathsf{Solve}(\mathcal{P})$ terminates after at most $|\mathcal{P}|$ refinement steps and either computes a feasible template instantiation or reports failure, in which case no feasible template instantiation exists.*

5 Experimental Results

In this section, we report on experimental results based on a prototype implementation of the symbolic instantiation algorithm from Section 4.

Implementation. We have implemented the symbolic instantiation algorithm from Section 4 in the SYNTHIA tool [23]. SYNTHIA is a verification and synthesis tool for timed automata extended with bounded integer variables. The tool provides facilities for automatic abstraction refinement and combines reduced ordered binary decision diagrams (ROBDDs) [7] with difference bound matrices (DBMs) [11] to obtain symbolic state representations for both discrete and continuous state components.

While standard SYNTHIA already includes a game-based synthesis algorithm for timed controllers with complete observability, we only use the verification functionality of SYNTHIA for template-based synthesis. For a template instantiation problem given by a controller template $(\mathcal{T}, P, \Pi)$ and a set of bad states, we encode the Boolean parameters P by global integer variables whose initial values are left undefined.

The algorithm from Section 4 is realized as a specialized refinement procedure for SYNTHIA's standard (location-based) abstraction refinement loop. Whenever an edge for refinement is found, we identify the parameter valuations associated with that edge and split the global abstraction with these valuations. In the subsequent refinement step, we focus (i.e., we restrict the forward exploration) on the identified parameters of the last refinement.

Benchmarks. In the *Chinese Juggler* benchmark [19], a performer needs to stabilize spinning plates to prevent them from falling. After a certain amount of time has passed since a plate was stabilized, it can nondeterministically become unstable. If no restabilization takes place, it ultimately falls down. The plates have different sizes, and hence, different times to become unstable. It takes the performer one time unit to stabilize a certain plate. During that time, he cannot stabilize another plate. The controller synthesis task consists in finding a safe strategy for the performer such that no plate will ever fall down. The benchmark size is parametrized in the number of plates n. For the template-based synthesis, we use a generic cyclic-executive template (cf. Section 1) with n locations. In each step of the cyclic execution, the controller decides which plate should be stabilized next.

In the *Dam* benchmark, a controller is to be synthesized that determines the speed of the inflow to a dam. The controller can either stop the inflow or choose between a slow or a fast inflow speed. The bounded reachability requirement is that the fill level should reach a certain value between a minimal and maximal bound. While a fast speed might reach the desired fill level more quickly, the variance of the actual inflow is larger so that the maximal level might be exceeded. On the other hand, being in slow mode, it takes longer to reach the desired fill level, but the variance is not so high so that it is always possible to exactly reach a desired fill level. Thus, being in one mode all the time is not feasible, since a feasible controller must alternate between fast and slow at least once to fulfill the requirement. The benchmark size is parametrized in the degree of precision in which the fill level and the inflow amount is digitized. For the template-based synthesis, we use a controller template that models a parametric two-phase program: in the first phase, a certain inflow speed is set until a threshold of the current fill level is passed. Then, the controller enters the second phase with a possibly different speed. The controller stops as soon as a desired fill level is reached. The first and the second speed, as well as the phase-switching threshold are parameters, for which feasible instantiations are to be found.

Results. We compare the performance of the template-based extension of SYNTHIA against standard SYNTHIA and UPPAAL-TIGA 0.16 [5]. In Table 1, the columns show, from left to right, the benchmark instance, the number of refinement steps and abstract locations in the final abstraction of the parameter synthesis algorithm, the running time and memory consumption of our template-based implementation, the performance of SYNTHIA's standard controller syn-

Table 1. Experimental evaluation of template-based synthesis. We compare the performance of the template-based extension of Synthia against standard Synthia and Uppaal-Tiga.

	Template-based Synthia				Standard Synthia				Uppaal-Tiga		
Benchmark	Steps	Abs	Time	Mem	Steps	Abs	Time	Mem	States	Time	Mem
Juggler 2	6	19	0	53	2	5	1	53	57	0	6
Juggler 3	38	136	0	61	6	9	1	61	477	0	6
Juggler 4	110	421	2	87	TIMEOUT				6755	6	57
Juggler 5	423	1899	59	247	TIMEOUT				81292	1095	79
Juggler 6	1445	8335	1932	1335	TIMEOUT				TIMEOUT		
Juggler 7	TIMEOUT				TIMEOUT				TIMEOUT		
Dam 5	58	100	1	80	230	149	4	80	88592	2	65
Dam 25	268	380	13	87	1115	718	1182	91	3114648	307	443
Dam 50	530	730	87	105	TIMEOUT				13545848	5018	2355
Dam 75	793	1080	329	111	TIMEOUT				TIMEOUT		
Dam 100	1055	1430	927	143	TIMEOUT				TIMEOUT		
Dam 125	1318	1780	1949	149	TIMEOUT				TIMEOUT		
Dam 150	1580	2130	3483	153	TIMEOUT				TIMEOUT		
Dam 175	1843	2480	5127	213	TIMEOUT				TIMEOUT		
Dam 200	TIMEOUT				TIMEOUT				TIMEOUT		

thesis algorithm, Uppaal-Tiga's number of explored states, running time and memory consumption. For Uppaal-Tiga, we measured the performance for various parameters and always selected the best results. Running times are given in seconds, memory consumption in MB, the time limit was set to 2 hours, and the memory limit was set to 4 GB. All experiments were conducted on a 2.6 GHz AMD Opteron computer running Ubuntu 10.04.

For both benchmarks, template-based Synthia clearly outperforms the game-based synthesis techniques implemented in standard Synthia and Uppaal-Tiga. A closer look at the *Chinese Juggler* example reveals that a major source of complexity results from the subtraction operation that occurs in the backwards computation of the winning states. Subtraction is expensive because it does not preserve convexity, and therefore requires a split into multiple zones. The much better performance of template-based synthesis is due to the fact that template-based synthesis is based on model checking, rather than game solving, and model checking does not require such nonconvex operations. In the *Dam* example, we observe that the size of the abstraction, and, thus, the running time, of the template-based approach increases polynomially in the size of the benchmark, while both standard Synthia and Uppaal-Tiga suffer from an exponential blow-up.

6 Conclusion

Our results demonstrate that template-based synthesis is an attractive alternative to the standard game-based approach to timed synthesis. Template-based synthesis has the better worst-case complexity, is easier to implement with symbolic data structures such as DBMs, and produces nicely structured controllers with a small number of locations.

In future work, we plan to expand the class of templates considered by the synthesis algorithm. Particularly interesting is the introduction of parameters in the clock constraints. Results from parametric timed model checking [3] indicate that the analysis of such templates is in general undecidable. However, subclasses of parametric timed automata, such as L/U automata [18], for which the emptiness problem is decidable, are promising candidates for a more expressive and yet computationally feasible class of templates.

The long-term goal is to obtain a succinct but comprehensive library of standard templates that serves as a basis for a fully automatic template-based synthesis approach.

Acknowledgments. The authors would like to thank Alexandre David for pointing out the *Chinese Juggler* benchmark and Christoph Scholl for helpful comments on an early draft of the paper.

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).

References

1. Altisen, K., Tripakis, S.: Tools for controller synthesis of timed systems. In: 2nd Workshop on Real-Time Tools, RT-TOOLS (2002)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: STOC, pp. 592–601 (1993)
4. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Lafay, J.-F. (ed.) Proc. 5th IFAC Conference on System Structure and Control, pp. 469–474. Elsevier (1998)
5. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: Time for Playing Games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
6. Bouyer, P., D'Souza, D., Madhusudan, P., Petit, A.: Timed Control with Partial Observability. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 180–192. Springer, Heidelberg (2003)
7. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)
8. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-Fly Algorithms for the Analysis of Timed Games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005)
9. Cassez, F., David, A., Larsen, K.G., Lime, D., Raskin, J.-F.: Timed control with observation based and stuttering invariant strategies. In: [22], pp. 192–206
10. Chatain, T., David, A., Larsen, K.G.: Playing games with timed games. In: Giua, A., Silva, M., Zaytoon, J. (eds.) Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS 2009), Zaragoza, Spain (September 2009)

11. Dill, D.L.: Timing Assumptions and Verification of Finite-State Concurrent Systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
12. D'Souza, D., Madhusudan, P.: Timed Control Synthesis for External Specifications. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 571–582. Springer, Heidelberg (2002)
13. Ehlers, R.: Symbolic Bounded Synthesis. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 365–379. Springer, Heidelberg (2010)
14. Ehlers, R., Mattmüller, R., Peter, H.-J.: Combining Symbolic Representations for Solving Timed Games. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 107–121. Springer, Heidelberg (2010)
15. Filiot, E., Jin, N., Raskin, J.-F.: An Antichain Algorithm for LTL Realizability. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 263–277. Springer, Heidelberg (2009)
16. Finkbeiner, B., Schewe, S.: SMT-based synthesis of distributed systems. In: Proceedings of the 2nd Workshop on Automated Formal Methods (AFM 2007), November 6, pp. 69–76. ACM Press, Atlanta (2007)
17. Henzinger, T.A., Kopke, P.W.: Discrete-time control for rectangular hybrid automata. Theoretical Computer Science 221(1-2), 369–392 (1999)
18. Hune, T., Romijn, J., Stoelinga, M., Vaandrager, F.W.: Linear Parametric Model Checking of Timed Automata. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 189–203. Springer, Heidelberg (2001)
19. Larsen, K.G., Behrmann, G., Skou, A.: Exercises for Uppaal, `http://www.cs.aau.dk/~bnielsen/TOV08/ESV04/exercises`
20. Lustig, Y., Vardi, M.Y.: Synthesis from Component Libraries. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 395–409. Springer, Heidelberg (2009)
21. Maler, O., Pnueli, A., Sifakis, J.: On the Synthesis of Discrete Controllers for Timed Systems (An Extended Abstract). In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
22. Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.): ATVA 2007. LNCS, vol. 4762. Springer, Heidelberg (2007)
23. Peter, H.-J., Ehlers, R., Mattmüller, R.: Synthia: Verification and Synthesis for Timed Automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 649–655. Springer, Heidelberg (2011)
24. Peter, H.-J., Mattmüller, R.: Component-based abstraction refinement for timed controller synthesis. In: Baker, T.P. (ed.) IEEE Real-Time Systems Symposium, pp. 364–374. IEEE Computer Society (2009)
25. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: [22], pp. 474–488
26. Thomas, W.: On the Synthesis of Strategies in Infinite Games. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)