Combining Symbolic Representations for Solving Timed Games

Rüdiger Ehlers[1], Robert Mattmüller[2], and Hans-Jörg Peter[1]

[1] Reactive Systems Group
Saarland University, Germany
{ehlers | peter}@cs.uni-saarland.de
[2] Foundations of Artificial Intelligence Group
Freiburg University, Germany
mattmuel@informatik.uni-freiburg.de

Abstract. We present a general approach to combine symbolic state space representations for the *discrete* and *continuous* parts in the synthesis of winning strategies for timed reachability games. The combination is based on abstraction refinement where *discrete* symbolic techniques are used to produce a sequence of abstract timed game automata. After each refinement step, the resulting abstraction is used for computing an under- and an over-approximation of the timed winning states. The key idea is to identify large relevant and irrelevant parts of the precise weakest winning strategy already on coarse, and therefore simple, abstractions. If neither the existence nor nonexistence of a winning strategy can be established in the approximations, we use them to guide the refinement process. Based on a prototype that combines binary decision diagrams [7,9] and difference bound matrices [5], we experimentally evaluate the technique on standard benchmarks from timed controller synthesis. The results clearly demonstrate the potential of the new approach concerning running time and memory consumption compared to the classical on-the-fly algorithm implemented in Uppaal-Tiga [10,4].

1 Introduction

In the last two decades, the timed automaton model by Alur and Dill [2] has become a de-facto standard for modeling timed asynchronous systems. A natural extension to their classical definition is to distinguish between internally and externally controllable behaviors [15,3]. The automated analysis of such so-called *open* systems requires a game-based computational model where an internal controller plays against an external environment. Solving problems defined on open systems (such as, e.g., timed controller synthesis [15]) is an active area of research [15,3,13,1,10,4,8,16] and usually corresponds to computing winning strategies in two-player games played on timed automata.

A predominant source of complexity in this setting is the large size of the concurrent control structure induced by a network of communicating timed automata. Current timed game solvers such as Uppaal-Tiga [10,4] represent the continuous parts symbolically, but the location information explicitly. Hence,

such *semi-symbolic* representations fail in the analysis of timed systems with many concurrent components causing a *discrete* blow-up in the state space.

In this paper, we tackle this problem by introducing an abstraction refinement approach that uses *discrete symbolic techniques* (e.g., binary decision diagrams (BDDs) [7,9,18]) to produce a sequence of *syntactic* abstractions with increasing precision of the input network of timed automata. We obtain the abstractions by merging locations such that the abstract control structure strictly weakens one player and strengthens her opponent. For each abstraction, we apply traditional solving algorithms to obtain an under- and an over-approximation of the winning states of the reachability player (e.g., one can use [10], which works fine for timed games with few locations, to obtain under-approximations).

Instead of solving the original game on the most precise control structure directly, our key idea is to solve a sequence of simpler games where each solving process reuses the approximations obtained from the previous one. That is, we use winning state set approximations computed on coarse (and therefore simple) abstractions to (1) characterize interpolants for refinement that ensure an increase in precision, and (2) derive pruning rules and optimizations that accelerate subsequent game solving processes over finer abstractions. Both soundness and effectiveness of our approach rely on the fact that whenever an abstract state appears in an under-approximation, all subsumed concrete states are surely winning, and dually, whenever an abstract state is not contained in an over-approximation, all subsumed concrete states are surely not winning.

In our prototype, the use of BDDs allows us to represent sets of locations efficiently and to refine abstract games arbitrarily while retaining an algorithmically simple check for the existence of abstract transitions. Based on (federations of) difference bound matrices (DBMs) [5], we use our own implementation of [10] for obtaining winning state set approximations.

Example. Consider the timed game automaton $\mathcal{G}$ given in Fig. 1(a), where l_0 is the initial and l_3 is the goal location. The reachability player ($\Diamond$) controls the dashed edges and wants to reach l_3 while the safety player ($\Box$) controls the solid edges and wants to avoid l_3. We abstract $\mathcal{G}$ by merging its locations. The abstract locations so obtained are connected via abstract transitions which are either (1) surely available, i.e., there is a corresponding concrete transition from each represented location, or (2) potentially available, i.e., there is a corresponding concrete transition from some represented location. Figures 1(b) and 1(c) show abstract automata with one abstract location representing the concrete goal location l_3 and one abstract location representing the remaining locations l_0, l_1, and l_2. In $\lceil\mathcal{G}\rceil_0$, we strengthen $\Diamond$ by letting her play on the potentially available transitions and weaken $\Box$ by letting him play on the ones that are surely available. Dually, we weaken $\Diamond$ and strengthen $\Box$ in $\lfloor\mathcal{G}\rfloor_0$. In $\lceil\mathcal{G}\rceil_0$, the abstract initial location (together with the initial clock valuation $x = 0$) is winning for $\Diamond$, whereas in $\lfloor\mathcal{G}\rfloor_0$, it is winning for $\Box$. Hence, we cannot determine the winner of $\mathcal{G}$ based on this coarse initial abstraction. Therefore, we refine the abstraction by separating l_0 in an additional abstract location. The finer abstractions are shown in Figures 1(d) and 1(e). Now, $\Diamond$ wins in $\lfloor\mathcal{G}\rfloor_1$ and, therefore, also in $\mathcal{G}$.

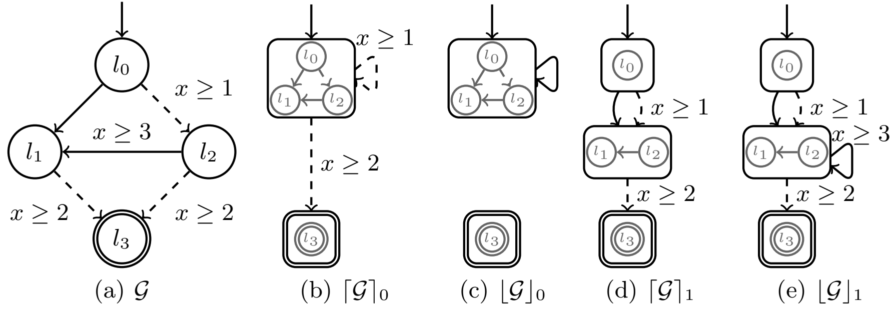

Fig. 1. Game automaton $\mathcal{G}$, abstractions $\lceil \mathcal{G} \rceil_0 / \lfloor \mathcal{G} \rfloor_0$, and refinements $\lceil \mathcal{G} \rceil_1 / \lfloor \mathcal{G} \rfloor_1$

Related Work. The definition of the game solving problem in the framework of timed automata [2] was given by Maler et al. [15,3]. In their fundamental work, the decidability of the problem was shown by demonstrating that the standard discrete attractor construction [19] on the region graph suffices to obtain winning strategies. Henzinger and Kopke showed that this construction is theoretically optimal by proving EXPTIME-completeness of the problem [13]. A first on-the-fly solving technique was proposed by Altisen and Tripakis which, however, requires an expensive preprocessing step [1]. As a remedy to this problem, Cassez et al. proposed a fully on-the-fly solving algorithm that combines the backward attractor construction with a forward zone graph exploration [10,4]. Recently, we developed an incremental variant that takes the compositional nature of networks of timed automata into account [16]. As a continuation of this line of research, the approach presented here can be seen as a further generalization that (1) provides the general basis for more fine-grained (location-based) abstractions, and (2) reports on a concrete application combining BDDs and DBMs.

Henzinger et al. adapted *counterexample-guided abstraction refinement* for games [12] where abstractions are defined over game states, counterexamples are abstract strategies, and refinement corresponds to the splitting of abstract game states. De Alfaro et al. introduced three-valued abstractions [11] where the refinement process is guided by differences between under- and over-approximations of the game states. Due to their *semantic* (i.e., state-based) natures, both techniques can be seen as a generalization of the approach presented in this paper. However, due to the lack of suitable data structures, an immediate implementation of these techniques for timed games, resulting in an efficient solving algorithm, appears not (yet) possible. We argue that location-based abstractions are an interesting sweet spot between granularity and implementability.

We propose an optimization technique that prunes irrelevant moves early in the refinement process which resembles *slicing* from model checking. Brückner et al. proposed a technique that combines slicing with abstraction refinement [6]. While their work only considers model checking problems for closed systems, our approach is capable of handling the strictly more general class of open systems.

Outline. Section 2 recalls the necessary foundations. In Sect. 3, we first formalize the notion of abstract games and give an algorithm that constructs an abstract game from a given concrete game and a location partition. Based on that, Sect. 4 describes an approximation-guided refinement loop, which is the formal basis for our prototype implementation presented in Sect. 5.

2 Preliminaries

2.1 Timed Games

We consider two-player, zero-sum reachability games played on timed automata. We distinguish between the *reachability* player $\Diamond$ whose objective is to eventually reach some goal state, and the *safety* player $\Box$ whose objective is to always avoid goal states. Following the setting of [3], we assume that timed automata are strongly nonzeno, i.e., there are no cycles where a player can play a time-convergent sequence of moves.

Timed Game Automata. A *timed game automaton* (TGA) [2,15,3] $\mathcal{G}$ is a tuple $(L, I, \varDelta, X, G)$, where L is a finite set of locations, $I \subseteq L$ is a set of initial locations, $\varDelta \subseteq L \times \mathcal{C}(X) \times \mathcal{P}(X) \times L$ is the set of transitions[3], X is a finite set of real valued clocks, and $G \subseteq L$ is a set of goal locations. We distinguish between controller $\varDelta_\Box$ and environment transitions $\varDelta_\Diamond$ such that $\varDelta = \varDelta_\Box \uplus \varDelta_\Diamond$. The clock constraints $\varphi \in \mathcal{C}(X)$ are recursively defined as $\varphi = \mathbf{true} \,|\, x \bowtie c \,|\, \varphi_1 \wedge \varphi_2$, where x is a clock in X, c is a constant in $\mathbb{N}_0$, $\bowtie \in \{<, \leq, \geq, >\}$, and φ_1, φ_2 are constraints in $\mathcal{C}(X)$. A *clock valuation* $\boldsymbol{t} : X \to \mathbb{R}_{\geq 0}$ assigns a non-negative 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. For a constraint $\varphi \in \mathcal{C}(X)$, we define $[\![\varphi]\!] = \{\boldsymbol{t} \in \mathcal{R} \mid \boldsymbol{t} \models \varphi\}$. We denote clock resets as $\boldsymbol{t}[\lambda := 0]$, for a set $\lambda \subseteq X$, and uniform time elapse as $\boldsymbol{t} + d$, for a $d \in \mathbb{R}_{\geq 0}$.

A *partition* of the locations L of a TGA $\mathcal{G}$ is a set $\varPi = \{\pi_1, \dots, \pi_n\} \in \mathcal{P}(\mathcal{P}(L) \setminus \{\emptyset\})$ such that $\bigcup_{i=1}^n \pi_i = L$ and $\pi_i \cap \pi_j = \emptyset$ for $i \neq j$. We say that a partition $\varPi'$ is *finer* than a partition $\varPi$, written as $\varPi \prec \varPi'$, iff $|\varPi| < |\varPi'|$ and $\forall \pi' \in \varPi' : \exists \pi \in \varPi : \pi' \subseteq \pi$. The *refinement* of a partition $\varPi$ with a set $R \subseteq L$, is defined as $\varPi|R = \bigcup_{\pi \in \varPi} \{\pi \cap R, \pi \setminus R\} \setminus \{\emptyset\}$. The most fine-grained partition is denoted as $\widetilde{\varPi}$ with $|\widetilde{\varPi}| = |L|$.

Timed Game Structures. The semantics of timed game automata is defined in terms of *timed game structures*. A timed game structure (TGS) $\mathcal{S}$ is a tuple $(S, S_0, \varGamma_\Box, \varGamma_\Diamond)$ where S is an infinite set of states, $S_0 \subseteq S$ are the initial states, and $\varGamma_\Box, \varGamma_\Diamond \subseteq S \times S$ are the moves of the players. A TGA $(L, I, \varDelta_\Box \cup \varDelta_\Diamond, X, G)$ induces a timed game structure $\mathcal{S} = (L \times \mathcal{R}, I \times \{\mathbf{0}\}, \varGamma_\Box, \varGamma_\Diamond)$ with

$$\begin{aligned}\varGamma_p = &\big\{(s, s') \mid \exists d > 0 \colon s' = s + d\big\} \cup \\ &\big\{((l, \boldsymbol{t}), (l', \boldsymbol{t}')) \mid \exists \langle l, \varphi, \lambda, l' \rangle \in \varDelta_p \colon \boldsymbol{t} \models \varphi \wedge \boldsymbol{t}' = \boldsymbol{t}[\lambda := 0]\big\},\end{aligned}$$

[3] For the sake of simplicity, we omit transition labels in our formal definition since control-related concepts such as synchronization or integer variables are just technicalities in the construction of the symbolic discrete transition relation.

for a player $p \in \{\Box, \Diamond\}$, where, for a game state $s = (l, \boldsymbol{t}) \in S$ and a delay $d \in \mathbb{R}_{\geq 0}$, we write $s + d$ for $(l, \boldsymbol{t} + d)$.

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 over a history of events. However, for reachability games under complete information, it suffices to consider state-based (or memoryless) strategies [15,3]. Formally, a *memoryless strategy* for player p is a function $f_p : S \to S$ such that all $s \in S$ are mapped to an s' with $(s, s') \in \Gamma_p$. Such an s' always exists because even if there is no successor location of the current location, it is always possible to play a time elapse move.

The notion of an *outcome* of a pair of strategies $f_\Diamond$ and $f_\Box$ defines the set of states that are reached if player $\Diamond$ sticks to $f_\Diamond$ and player $\Box$ sticks to $f_\Box$. Let s and s' be two states in S with $s' = f_p(s)$, $p \in \{\Box, \Diamond\}$, then the *time elapse* between s and s', written as $\delta(s, s')$, is defined as (1) $\delta(s, s') = d$, if $s' = s + d$, for a $d \in \mathbb{R}_{>0}$; (2) $\delta(s, s') = 0$, otherwise. The set $\mathsf{Outcome}(f_\Diamond, f_\Box) \subseteq S$ is the smallest subset of S (wrt. set inclusion), such that the following holds:

- $S_0 \subseteq \mathsf{Outcome}(f_\Diamond, f_\Box)$;
- if $s \in \mathsf{Outcome}(f_\Diamond, f_\Box)$, then
  $f_\Box(s) \in \mathsf{Outcome}(f_\Diamond, f_\Box)$, if $\delta(s, f_\Box(s)) < \delta(s, f_\Diamond(s))$, and
  $f_\Diamond(s) \in \mathsf{Outcome}(f_\Diamond, f_\Box)$, if $\delta(s, f_\Diamond(s)) \leq \delta(s, f_\Box(s))$.

With this definition of $\mathsf{Outcome}$, we assume that (1) player $\Diamond$ chooses the initial state, and (2) the scheduler resolving concurrent moves is always playing in favor for player $\Diamond$. Note that this captures the controller synthesis problem accurately since any actual controller implementation (player $\Box$) has to be robust wrt. any low-level scheduling policy or arbitrary environment (player $\Diamond$). Moreover, along with complementary winning objectives, the timed games considered here are always determined and their semantics is equivalent to Uppaal-Tiga [10,4].

Timed Reachability Games. Let $\mathcal{S} = (S, S_0, \Gamma_\Box, \Gamma_\Diamond)$ be a TGS and $K \subseteq S$ a set of goal states. Then $(\mathcal{S}, K)$ represents a *timed reachability game*. Player $\Diamond$ wins $(\mathcal{S}, K)$ iff she can enforce a visit to K. More formally, player $\Diamond$ wins iff $\exists f_\Diamond \forall f_\Box : \mathsf{Outcome}(f_\Diamond, f_\Box) \cap K \neq \emptyset$. A TGA $\mathcal{G}$ with goal locations G induces a timed reachability game $\mathsf{Game}(\mathcal{G}) = (\mathcal{S}, K)$ such that $\mathcal{S}$ is the game structure induced by $\mathcal{G}$ and $K = G \times \mathcal{R}$ is the set of $\mathcal{G}$'s goal states.

2.2 Solving Timed Games

Solving a timed reachability game $(\mathcal{S}, K)$ means computing the set of states from which player $\Diamond$ has a strategy to enforce an outcome that contains some states from K. Before we come to the actual solving algorithm, we formalize the notion of controllability. For a TGS $\mathcal{S} = (S, S_0, \Gamma_\Box, \Gamma_\Diamond)$, the *timed enforceable predecessor operator* $\mathsf{PreEnf} : \mathcal{P}(S) \to \mathcal{P}(S)$ for player $\Diamond$ is defined as

$$\mathsf{PreEnf}(Y) = \big\{ r \in S \mid \exists s \in Y : (r, s) \in \Gamma_\Diamond \wedge (\forall d > 0 : s = r + d \Rightarrow \forall 0 \leq d' < d : \forall (r', s') \in \Gamma_\Box : r' = r + d' \Rightarrow s' \in Y) \big\}.$$

Intuitively, $\mathsf{PreEnf}(Y)$ comprises the source states of $\Diamond$-moves leading to Y that (1) change the location or (2) delay with no spoiling $\Box$-move in between. It was shown in [15,3] and later in [10] that PreEnf can be effectively computed using clock regions or clock zones.

We define those states from which player $\Diamond$ has a winning strategy to enforce an outcome that eventually visits some state in K as the *attractor* of K. For a reachability game $(\mathcal{S}, K)$, the computation of the attractor $\mathsf{Attr}(\mathcal{S}, K) \subseteq S$ is carried out by iteratively applying PreEnf in a least fixed point construction on K, i.e., $\mathsf{Attr}(\mathcal{S}, K)$ corresponds to $\mu A.A \cup K \cup \mathsf{PreEnf}(A)$ [15,3]. Note that any starting point A', with $K \subseteq A' \subseteq \mathsf{Attr}(\mathcal{S}, K)$, converges to $\mathsf{Attr}(\mathcal{S}, K)$ in the fixed point construction. We will write $\mathsf{Attr}(\mathcal{G})$ as an abbreviation for $\mathsf{Attr}(\mathsf{Game}(\mathcal{G}))$. Player $\Diamond$ wins $\mathsf{Game}(\mathcal{G})$ iff $S_0 \cap \mathsf{Attr}(\mathcal{G}) \neq \emptyset$. Dually, player $\Box$ wins $\mathsf{Game}(\mathcal{G})$ iff player $\Diamond$ does not win, i.e., $S_0 \cap \mathsf{Attr}(\mathcal{G}) = \emptyset$.

Theorem 1. *[13] For a TGA $\mathcal{G}$, constructing $\mathsf{Attr}(\mathcal{G})$ is complete for EXPTIME.*

From a practical point of view, a careful analysis shows that the application of the (nonconvex) symbolic PreEnf operator is very expensive compared to a zone-based forward analysis. For this purpose, the authors of [10] propose an on-the-fly game solving algorithm based on an interleaved fixed point construction that alternates between a forward exploration of the reachable states and a backward propagation of the attractor. Here, the number of PreEnf applications is reduced at the cost of introducing forward steps. In combination with the abstraction refinement technique presented in this paper, we use this algorithm to compute attractor under-approximations. In the following, we refer to algorithms such as [10] as *backend solving algorithms.*

2.3 Boolean Functions & Binary Decision Diagrams

Sets of locations can be represented by Boolean functions (BFs) $F : \mathcal{P}(\mathcal{B}) \to \mathbb{B}$ for some finite set of variables $\mathcal{B}$. In practice, *reduced ordered binary decision diagrams* (BDDs) [7,9] are the predominantly used data structure for this task. Since the usual operations on Boolean functions such as conjunction, disjunction and negation can be implemented as manipulations of BDDs, we treat Boolean functions and BDDs interchangeably here. In addition, BDDs support existential (and universal) abstraction. Given a set of variables $\mathcal{B}' \subseteq \mathcal{B}$ and a BDD F, the existential abstraction of F wrt. $\mathcal{B}'$ is written as $\exists \mathcal{B}'.F$ and denotes the BDD that maps those and only those $x \subseteq \mathcal{B}$ to **true** for which there exists some $x' \subseteq \mathcal{B}'$ such that $F(x' \cup (x \setminus \mathcal{B}')) = \textbf{true}$.

3 Abstract Timed Games

We abstract a TGA by merging its locations such that the resulting abstract control structure strictly privileges one player and penalizes her opponent. We can do this asymmetric abstraction in two ways: (1) we can weaken player $\Diamond$ and strengthen player $\Box$ to obtain a *weakened* reachability game; (2) we can

strengthen player $\Diamond$ and weaken player $\Box$ to obtain a *strengthened* reachability game. In this section, we first introduce the formal model which acts as a basis for showing soundness and completeness of our approach. Then, we describe a Boolean function-based algorithm for constructing abstractions.

3.1 Abstract Timed Game Automata

A TGA $\mathcal{G} = (L, I, \Delta, X, G)$ with $\Delta = \Delta_\Diamond \uplus \Delta_\Box$ and a partition Π of L induce a *weakened TGA* $\lfloor \mathcal{G} \rfloor_\Pi$ and a *strengthened TGA* $\lceil \mathcal{G} \rceil_\Pi$:

$$\lfloor \mathcal{G} \rfloor_\Pi = (\Pi, \lfloor I \rfloor_\Pi, \lfloor \Delta_\Diamond \rfloor_\Pi \cup \lceil \Delta_\Box \rceil_\Pi, X, \lfloor G \rfloor_\Pi);$$
$$\lceil \mathcal{G} \rceil_\Pi = (\Pi, \lceil I \rceil_\Pi, \lceil \Delta_\Diamond \rceil_\Pi \cup \lfloor \Delta_\Box \rfloor_\Pi, X, \lceil G \rceil_\Pi).$$

Here, the *weak abstracting operator* $\lfloor \cdot \rfloor$ and the *strong abstracting operator* $\lceil \cdot \rceil$ are defined as follows. For any set $L' \subseteq L$ (and in particular I and G), we define

$$\lfloor L' \rfloor_\Pi = \{\pi \in \Pi \mid \pi \subseteq L'\} \text{ and}$$
$$\lceil L' \rceil_\Pi = \{\pi \in \Pi \mid \pi \cap L' \neq \emptyset\}.$$

Furthermore, for any set $\Delta' \subseteq \Delta$, we define

$$\lfloor \Delta' \rfloor_\Pi = \{\langle \pi, \varphi, \lambda, \pi' \rangle \mid \forall l \in \pi : \exists l' \in \pi' : \langle l, \varphi, \lambda, l' \rangle \in \Delta'\} \text{ and}$$
$$\lceil \Delta' \rceil_\Pi = \{\langle \pi, \varphi, \lambda, \pi' \rangle \mid \exists l \in \pi : \exists l' \in \pi' : \langle l, \varphi, \lambda, l' \rangle \in \Delta'\}.$$

Intuitively, transitions in $\lfloor \Delta' \rfloor_\Pi$ are *surely* available, while transitions in $\lceil \Delta' \rceil_\Pi$ are *potentially* available. It is easy to see that $\lfloor Y \rfloor_\Pi \subseteq \lceil Y \rceil_\Pi$, for every set of locations or transitions Y. We say that a pair of abstract locations $(\pi_1, \pi_2) \in \Pi \times \Pi$ represents a *potential connection* in a TGA $\mathcal{G}$ wrt. a partition Π iff there is a connecting transition from π_1 to π_2 in $\lceil \Delta \rceil_\Pi$. The following lemma states that a refinement never introduces new potential connections.

Lemma 1. *For a TGA $\mathcal{G}$ with locations L and partitions Π and Π' of L with $\Pi \prec \Pi'$, if $(\pi_1, \pi_2) \in \Pi \times \Pi$ is not a potential connection in $\mathcal{G}$ wrt. Π, then there is no potential connection $(\pi'_1, \pi'_2) \in \Pi' \times \Pi'$ in $\mathcal{G}$ wrt. Π' with $\pi'_1 \subseteq \pi_1 \wedge \pi'_2 \subseteq \pi_2$.*

In order to compare abstract attractor sets for different partitions, we need to *flatten* them: let $a = (\pi, \boldsymbol{t}) \in \mathcal{P}(L) \times \mathcal{R}$ and $A \subseteq \mathcal{P}(L) \times \mathcal{R}$. Then, the flattenings of a and A are defined as $\widehat{a} = \{(l, \boldsymbol{t}) \mid l \in \pi\}$ and $\widehat{A} = \bigcup_{a \in A} \widehat{a}$. Recall that $\widetilde{\Pi}$ denotes the most fine-grained partition. With these definitions, we can state the central soundness lemma.

Lemma 2. *Let $\mathcal{G}$ be a TGA with locations L and Π be a partition of L. Then,* $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)} \subseteq \widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}})} = \mathsf{Attr}(\mathcal{G}) = \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_{\widetilde{\Pi}})} \subseteq \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_\Pi)}$.

On the one hand, Lemma 2 guarantees the soundness of our abstractions: once an abstract state $(\pi, \boldsymbol{t})$ appears in the attractor under-approximation, every subsumed concrete state $(l, \boldsymbol{t})$, for any $l \in \pi$, is surely winning for player $\Diamond$.

Dually, once an abstract state $(\pi', \boldsymbol{t}')$ is not contained in the attractor over-approximation, every subsumed concrete state $(l', \boldsymbol{t}')$, for any $l' \in \pi'$, is surely winning for player $\Box$. On the other hand, the lemma ensures that every refinement process eventually ends up with the precise attractor (e.g., when $\Pi = \widetilde{\Pi}$).

Soundness of Zeno Abstractions. Location-based abstractions of timed systems may contain zeno loops giving rise to the existence of physically unmeaningful, and therefore spuriously too powerful, winning strategies. Note that this does not affect the soundness of our approach: there can only be zeno loops in an over-approximating control structure since we require the original system to be strongly nonzeno. Then, giving spuriously more power to an over-approximated player $\Box$ is consistent with the abstraction. On the other hand, giving zeno moves to an over-approximated player $\Diamond$ does not increase her winning possibilities since no moves leading to goal states are added.

We do not require a special treatment of zeno behavior in the backend solving algorithm; we only expect that, for zeno inputs, the algorithm reports sound (though zeno) strategies (which is the case for [10]).

3.2 Constructing Abstractions using Boolean Functions

The key motivation for considering location-based abstractions is the possibility to use BFs for the construction of the abstract control structure. In this section, we describe an algorithm that constructs abstract (both weakened and strengthened) TGAs from a concrete TGA $\mathcal{G} = (L, I, \Delta, X, G)$ and a location partition Π. Note that we only use BFs for the *construction* of abstract TGAs but not for the actual game solving: abstract TGAs are represented using the standard (explicit location) representation [10].

In a preparation step, we encode Δ as a BF. The set of BF variables $\mathcal{B}$ that we use for our symbolic encoding consists of three disjoint sets $\mathcal{B}_L$, $\mathcal{B}_{L'}$, and $\mathcal{B}_X$, where $\mathcal{B}_L$ and $\mathcal{B}_{L'}$ represent predecessor- and successor-locations of timed transitions (with $|\mathcal{B}_L| = |\mathcal{B}_{L'}| = \lceil \log_2 |L| \rceil$). Furthermore, $\mathcal{B}_X$ is a set containing one variable v_x for each clock x (for encoding resets in the transition relation) and one variable v_φ for each atomic constraint $\varphi = x \bowtie c$ in Δ (for encoding guards).

We formalize these encodings in the following *predicates* over $\mathcal{B}$. First of all, for each location $l \in L$, the predicate $(\!| l |\!)$ over $\mathcal{B}_L$ encodes l in binary form, and similarly, the predicate $(\!| l |\!)'$ over $\mathcal{B}_{L'}$ encodes the primed version of l as a successor location. Formally, $(\!| l |\!)$ and $(\!| l |\!)'$ are functions mapping an assignment to the variables in $\mathcal{B}_L$ and $\mathcal{B}_{L'}$ (respectively) to **true** iff the assignment corresponds to the location l. As long as the binary encoding of the locations guarantees that the locations add up to **true** and are disjoint ($\bigvee_{l \in L} (\!| l |\!) = \bigvee_{l \in L} (\!| l |\!)' = \mathbf{true}$ and for all locations $l_1, l_2 \in L$, $(\!| l_1 |\!) \wedge (\!| l_2 |\!) \equiv \mathbf{false}$ and $(\!| l_1 |\!)' \wedge (\!| l_2 |\!)' \equiv \mathbf{false}$ whenever $l_1 \neq l_2$), the details of the encoding are not important and are therefore not discussed here. We refer to Sect. 5 for further details. Additionally, we define $(\!| \varphi |\!) = v_\varphi$ for all atomic constraints φ appearing in Δ, $(\!| \varphi |\!) = \bigwedge_{i=1}^{n} (\!| \varphi_i |\!)$ for all nonatomic constraints $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_n$ and $(\!| \lambda |\!) = \bigwedge_{x \in \lambda} v_x \wedge \bigwedge_{x \in X \setminus \lambda} \neg v_x$ for all

Algorithm 1 BF-based construction of the transition relations of $\lfloor\mathcal{G}\rfloor_\Pi$ and $\lceil\mathcal{G}\rceil_\Pi$, for a given TGA $\mathcal{G}$ and a location partition Π.

```
1: for all p ∈ {□, ◇} do
2:   for all (π, π') ∈ Π × Π s.t. A ≡ ⦇π⦈ ∧ ⦇Δ_p⦈ ∧ ⦇π'⦈' ≢ false do
3:     for all φ ∈ C(X) and λ ⊆ X s.t. B ≡ A ∧ ⦇φ⦈ ∧ ⦇λ⦈ ≢ false do
4:       add ⟨π, φ, λ, π'⟩ to ⌈Δ_p⌉_Π
5:       if (⦇π⦈ ⇒ (∃B_L' ∪ B_X.B)) ≡ true then
6:         add ⟨π, φ, λ, π'⟩ to ⌊Δ_p⌋_Π
```

resets $\lambda \subseteq X$ appearing in Δ. These predicates are used to encode the guards of a transition and the respective resets in the transition relation.

For a set of locations $\pi \subseteq L$, we write $⦇\pi⦈$ for $\bigvee_{l\in\pi} ⦇l⦈$. The Boolean predicate that symbolically represents the concrete transition relation for a player $p \in \{\Box, \Diamond\}$ can be defined as $⦇\Delta_p⦈ \equiv \bigvee_{\langle s,\varphi,\lambda,t\rangle\in\Delta_p} ⦇s⦈ \wedge ⦇\varphi⦈ \wedge ⦇\lambda⦈ \wedge ⦇t⦈'$. Note that the extension of this definition by, e.g., an action-based synchronization of distributed components or discrete integer variables used in guards and update expressions is straightforward. However, for the sake of simplicity of our presentation, we stick to the minimalistic, monolithic setting, although our prototype implementation described in Sect. 5 supports these features. Then, for a network of timed automata, by building the transition relation for each automaton separately, explicitly enumerating all locations in the product automaton is avoided.

Finally, Algo. 1 describes the construction of the transition relations for the abstract TGAs $\lfloor\mathcal{G}\rfloor_\Pi$ and $\lceil\mathcal{G}\rceil_\Pi$ from the concrete TGA $\mathcal{G}$ and a partition Π. In the first two lines, the algorithm iterates over the players and all potential connections (π, π'), which are represented as the BF A. In line 3, we iterate over all combinations of guards φ and resets λ whose corresponding predicates satisfy A, and compute the BF B that represents all concrete transitions $(l, \varphi, \lambda, l')$, with $(l, l') \in \pi \times \pi'$. In line 4, the transition is added to the set of potentially available transitions. Then, in line 5, the algorithm tests if the transition is surely available, i.e., if it also needs to be added to the set of surely available transitions in line 6. It is easy to see that the abstract transition relations constructed by Algo. 1 satisfy the definition from Sect. 3.1.

Note that the iterations in lines 2 and 3 do *not* necessarily induce a global explicit blow-up, as we can use the following optimizations:

1. We use the algorithm only to *update* the abstract TGAs in an incremental way during the refinement process. This way, we only need to consider the abstract locations modified by the respective last refinement step in line 2.
2. According to Lemma 1, if two abstract locations $\pi_1 \in \Pi$ and $\pi_2 \in \Pi$ are not connected in $\lceil\mathcal{G}\rceil_\Pi$, we can safely assume that any pair of refined abstract locations $\pi_1' \subseteq \pi_1$ or $\pi_2' \subseteq \pi_2$ is also not connected in $\lceil\mathcal{G}\rceil_{\Pi'}$, where $\Pi \prec \Pi'$, $\pi_1' \in \Pi'$, and $\pi_2' \in \Pi'$.
3. In line 3, assuming that we use a BDD to represent the BF A, we can inspect the BDD structure of A to skip guard/reset combinations that do not occur for the chosen abstract states π and π'.

4 Approximation-Guided Abstraction Refinement

In an abstraction refinement loop, we incrementally solve a sequence of abstract games with increasing precision converging to the original game. In Sect. 4.1, we first describe how to obtain sets of concrete locations that serve as interpolants for refining abstract locations. Then, Sect. 4.2 describes the actual refinement loop. Finally, Sect. 4.3 investigates optimizations.

4.1 Abstract Location Refinement

We give a general characterization of sets of concrete locations that can be used as interpolants for splitting abstract locations, i.e., location partitions in Π. All interpolants selected by any concrete refinement heuristic must satisfy this characterization. Due to the lack of space, we only describe the refinement for *enlarging* attractor under-approximations; shrinking attractor over-approximations is just the dual case and can be done analogously.

Definition 1. *Let $\mathcal{G} = (L, I, \Delta, X, G)$ be a TGA, Π be a partition of L, and $\lfloor \mathsf{A} \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)$. A set of concrete locations $R \subseteq L$ is defined to be an* effective interpolant *if and only if there is at least one $\pi \in \Pi$ with $\emptyset \subsetneq \pi \cap R \subsetneq \pi$ such that either*

(1) there is at least one transition $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lceil \Delta_\Diamond \rceil_\Pi \setminus \lfloor \Delta_\Diamond \rfloor_\Pi$ such that

$$\forall l \in \pi \cap R : \exists l' \in \pi' : \langle l, \varphi, \lambda, l' \rangle \in \Delta_\Diamond \quad \textit{and}$$
$$\exists \boldsymbol{t} \in [\![\varphi]\!] : (\pi, \boldsymbol{t}) \notin \lfloor \mathsf{A} \rfloor \ \wedge \ (\pi', \boldsymbol{t}[\lambda := 0]) \in \lfloor \mathsf{A} \rfloor, \quad \textit{or}$$

(2) there is at least one transition $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lceil \Delta_\Box \rceil_\Pi \setminus \lfloor \Delta_\Box \rfloor_\Pi$ such that

$$\forall l \in \pi : (\exists l' \in \pi' : \langle l, \varphi, \lambda, l' \rangle \in \Delta_\Box) \Rightarrow l \in R \quad \textit{and}$$
$$\exists \boldsymbol{t} \in [\![\varphi]\!] : (\pi', \boldsymbol{t}[\lambda := 0]) \notin \lfloor \mathsf{A} \rfloor.$$

In other words, an effective interpolant R refines some abstract locations whose transitions are either spuriously too weak for player $\Diamond$ or spuriously too powerful for player $\Box$. More precisely, guided by an attractor under-approximation, R is defined based on transitions whose appearance generates winning $\Diamond$-moves or whose disappearance removes spoiling $\Box$-moves. There always exists an effective interpolant unless the abstraction is most precise:

Lemma 3. *If $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)} \subsetneq \mathsf{Attr}(\mathcal{G})$, then there exists an effective interpolant.*

Refinements with effective interpolants always ensure progress:

Lemma 4. *If $R \subseteq L$ is an effective interpolant, then $\Pi \prec \Pi|R$.*

Refinements leading to an increase of precision are based on effective interpolants:

Lemma 5. *Let $R \subseteq L$, $\lfloor \mathsf{A} \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)$, and $\lfloor \mathsf{A}' \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi|R})$, where $\lfloor \mathsf{A} \rfloor$ is the starting point for computing $\lfloor \mathsf{A}' \rfloor$.*
If $\widehat{\lfloor \mathsf{A} \rfloor} \subsetneq \widehat{\lfloor \mathsf{A}' \rfloor}$, then R is an effective interpolant.

4.2 Refinement Loop

For a TGA $\mathcal{G} = (L, I, \Delta, X, G)$, we construct a finite sequence of location partitions of the form $\Pi_0 \prec \Pi_1 \prec \ldots \prec \Pi_n$ in a refinement loop, where n is a natural number and $\Pi_0 = \{L \setminus G, G\}$ is the trivial initial partition that separates non-goal locations from goal locations. We use the Boolean function-based technique from Sect. 3.2 to initially construct and incrementally update a sequence of abstract TGAs converging to $\mathcal{G}$. Note that, instead of constructing the complete abstract TGA in each refinement cycle, we incrementally update the previous one by letting Algo. 1 iterate only over those partitions that were affected by the previous refinement step. Each refinement step is guided by a *refinement heuristic* that determines an effective interpolant as defined in Sect. 4.1. More precisely, after each cycle i, for an effective interpolant $R_i \subseteq L$, we obtain the succeeding partition $\Pi_{i+1} = \Pi_i | R_i$.
We compute the initial and intermediate attractor approximations as follows:

$$\lfloor \mathsf{A}_0 \rfloor = \widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi_0})} \qquad \text{and} \qquad \lfloor \mathsf{A}_{i+1} \rfloor = \lfloor \mathsf{A}_i \rfloor \cup \widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi_{i+1}})}\,;$$

$$\lceil \mathsf{A}_0 \rceil = \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_{\Pi_0})} \qquad \text{and} \qquad \lceil \mathsf{A}_{i+1} \rceil = \lceil \mathsf{A}_i \rceil \cap \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_{\Pi_{i+1}})}\,.$$

Hence, every maximal sequence of approximations is of the form

$$\lfloor \mathsf{A}_0 \rfloor \subseteq \cdots \subseteq \lfloor \mathsf{A}_n \rfloor = \mathsf{Attr}(\mathcal{G}) = \lceil \mathsf{A}_n \rceil \subseteq \cdots \subseteq \lceil \mathsf{A}_0 \rceil.$$

The loop terminates whenever the existence (nonexistence) of a winning strategy can be established in an under-approximation (over-approximation):
– $(I \times \{\mathbf{0}\}) \cap \lfloor \mathsf{A}_i \rfloor \neq \emptyset$, i.e., player $\Diamond$ surely has a winning strategy, or
– $(I \times \{\mathbf{0}\}) \cap \lceil \mathsf{A}_i \rceil = \emptyset$, i.e., player $\Box$ surely has a winning strategy.
Clearly, this suffices for termination, since if neither of the two conditions is satisfied, Lemma 3 guarantees that some further refinement is possible.

Theorem 2. *The presented abstraction refinement loop always terminates and yields a sound winning strategy for one of the players upon termination.*

4.3 Optimizations

Our abstraction refinement algorithm greatly benefits from several optimizations which can be applied *early* in the refinement loop. They are based on (1) pruning irrelevant moves that do not affect the winning capabilities of either player and (2) identifying surely winning states for player $\Diamond$ based on a strengthened TGA. For any abstract TGA $\mathcal{G} = (\Pi, I, \Delta, X, G)$ and its induced game structure $(S, S_0, \Gamma_\Box, \Gamma_\Diamond)$, with attractor under-approximation $\lfloor \mathsf{A} \rfloor$ and over-approximation $\lceil \mathsf{A} \rceil$, one can apply the following optimizations.

States already determined. We can remove all moves that lead out of states that are already known to be winning for some player. According to Lemma 2, once a state appears in an attractor under-approximation, it is surely winning for player $\Diamond$, and once a state is no more contained in an attractor

over-approximation, it is surely winning for player $\Box$. Hence, it is safe to ignore all moves from $\{(s, s') \in \Gamma_\Box \cup \Gamma_\Diamond \mid s \in \lfloor \mathsf{A} \rfloor \vee s \notin \lceil \mathsf{A} \rceil\}$.

Moves already determined. We can remove all moves that lead to states that are already known to be winning for the opponent. Hence, it is safe to ignore all moves from $\{(s, s') \in \Gamma_\Box \mid s' \in \lfloor \mathsf{A} \rfloor\} \cup \{(s, s') \in \Gamma_\Diamond \mid s' \notin \lceil \mathsf{A} \rceil\}$.

States surely winning. Under the assumption that $\mathcal{G}$ is a strengthened TGA, an abstract state in S is surely winning for $\Diamond$ if *each* subsumed concrete state has *some* concrete move leading to $\lfloor \mathsf{A} \rfloor$. Hence, we can safely extend PreEnf by all states $(\pi, \boldsymbol{t}) \in \Pi \times \mathcal{R}$ where
$\pi \subseteq \{l \in L \mid \exists \pi' \in \Pi : \exists l' \in \pi' : (\pi', \boldsymbol{t}) \in \lfloor \mathsf{A} \rfloor \wedge ((l, \boldsymbol{t}), (l', \boldsymbol{t})) \in \Gamma_\Diamond\}$.

The first rule can easily be realized in the backend solving algorithm, when computing $\lfloor \mathsf{A}_{i+1} \rfloor$ (or $\lceil \mathsf{A}_{i+1} \rceil$), by not forward-exploring moves whose source states are already contained in $\lfloor \mathsf{A}_i \rfloor$ (or not contained in $\lceil \mathsf{A}_i \rceil$). The second rule is realized by reusing $\lfloor \mathsf{A}_i \rfloor$ as a starting point for $\lfloor \mathsf{A}_{i+1} \rfloor$ (and $\lceil \mathsf{A}_i \rceil$ for $\lceil \mathsf{A}_{i+1} \rceil$). The third rule is used to extend the results of PreEnf when constructing $\lfloor \mathsf{A}_i \rfloor$.

5 Experimental Results

5.1 Prototype Implementation

We implemented a prototype in C^{++}, where we combined the Cudd BDD library [18] for representing location partitions and the Uppaal-DBM library [5] for representing federations of clock zones in the attractors.

In the initialization phase, our tool registers all BDD variables after calling the Nova tool from the SIS toolset [17] for finding efficient assignments of control locations to BDD variable valuations. Then, as described in Sect. 3.2, we construct the symbolic discrete transition relation representing the control structure of the input network of TGAs. Note that, although not discussed in detail in the rest of the paper, in general our approach (and in particular our tool) is able to handle networks of communicating TGAs with integer variables: such pure discrete features are covered in the construction of the discrete transition relation. In the next initialization step, we use the discrete transition relation to compute an over-approximation of the reachable locations in a (cheap) BDD-based least fixed point computation. The initial partition splits this over-approximation into (1) the set of potentially reachable goal locations, (2) the set of potentially reachable locations from which no goal location is reachable, and (3) the remaining locations. At the end of the initialization phase, we use Algo. 1 to construct the initial weakened and strengthened TGAs, where we merge transitions with the same resets whose guards subsume each other.

In the automatic abstraction refinement loop, we use our implementation of the backend solving algorithm proposed in [10] to incrementally update an attractor under-approximation. After each iteration, we check if the concrete initial state is contained in the abstract attractor. In this case, we terminate since we can deduce that player $\Diamond$ surely wins. If this is not the case, we identify abstract transitions which are spuriously too weak for player $\Diamond$ and symbolically

compute corresponding effective interpolants (by applying the BDD-based pre-image operator). If there are no abstract transitions for player $\Diamond$, we identify abstract transitions which are spuriously too powerful for player $\Box$ and refine likewise. Then, we split the partition with each computed interpolant (by simple BDD-based conjunctions) and update the weakened TGA using Algo. 1. Each refinement step might split a single abstract location by multiple interpolants resulting in an exponential number of split operations. To address this issue, we fix a number $\mathcal{K}$ of maximal split operations per abstract location.

5.2 Benchmarks

We evaluated our approach on two standard benchmarks[4] for timed controller synthesis and compared the results with Uppaal-Tiga [4] version 4.1.3-0.14.

The *Production Cell* (Prodcell) example [14,10] represents a manufacturing plant consisting of a feeding belt, two robot arms, a press, and a departure belt. The timed game comes into play when synthesizing a controller for the robot arms such that all parts put onto the feeding belt are transported to the press right in time and are finally transported to the departure belt.

The *Gear Production Stack* (GPS) example [16] models a pipeline-like architecture that sequentializes a series of stations, each specialized in a certain processing method. The task is to synthesize a controller for the machine that ensures that the pieces are transported from station to station right in time. We investigate the nonextended version without sub-processing units.

Table 1 shows the results of our comparison where we fixed $\mathcal{K} = 1000$. From left to right, the first two columns describe the name of the benchmark, the length (in number of plates and stations, resp.), and whether there exists a controller implementation (i.e., a winning strategy for player $\Box$). The next three columns show the number of explored states, the running time, and the memory consumption of Uppaal-Tiga. The last four columns show the number of refinement steps, the final size (in number of locations) of the abstract TGA, the running time, and the memory consumption of our prototype. All benchmarks were executed on an AMD Opteron processor with 2.6 GHz and 4 GB RAM. The running times are given in seconds and the memory consumptions are given in MB. The time limit was set to four hours.

The most striking observation is that for both benchmarks, our approach almost always outperforms Uppaal-Tiga. Only for small benchmark instances, Uppaal-Tiga performs slightly better. This is due to the preprocessing phase where all BDD variables are registered and the symbolic discrete transition relation is constructed. However, for benchmark instances of nontrivial size, Uppaal-Tiga either runs out of memory or needs at least an order of magnitude more running time than our tool.

The impact of different values for $\mathcal{K}$ on the running time and memory consumption is shown in Table 2. Smaller values for $\mathcal{K}$ result in a higher number of refinement steps but lead to a lower memory peak consumption since fewer

[4] The Uppaal-Tiga models of the benchmarks are available at http://www.avacs.org/Benchmarks/Open/formats10.tgz

		Uppaal-Tiga			Our prototype			
Benchmark	Cont	States	Time	Mem	Steps	Abs	Time	Mem
Prodcell 3	No	15241	1	54	14	293	3	94
Prodcell 4	No	131999	5	74	14	935	13	156
Prodcell 5	No	1238698	240	309	14	2762	39	244
Prodcell 6	No	TIMEOUT			14	8212	150	538
Prodcell 7	No	TIMEOUT			15	24757	761	1936
Prodcell 8	No	TIMEOUT			15	75085	6543	2092
Prodcell 3	Yes	15206	1	54	14	294	3	113
Prodcell 4	Yes	133181	5	75	15	940	11	156
Prodcell 5	Yes	1255498	238	314	15	2772	42	246
Prodcell 6	Yes	TIMEOUT			15	8232	172	538
Prodcell 7	Yes	TIMEOUT			16	24792	1068	1936
Prodcell 8	Yes	TIMEOUT			16	75140	6444	2093
GPS 6	No	170470	4	69	14	274	2	81
GPS 7	No	1406744	40	190	16	560	3	117
GPS 8	No	12123700	545	1327	18	1134	6	133
GPS 9	No	MEMOUT			20	2284	20	250
GPS 10	No	MEMOUT			23	5518	91	402
GPS 11	No	MEMOUT			25	11128	307	948
GPS 12	No	MEMOUT			27	22368	1553	3550
GPS 6	Yes	190484	4	69	17	320	2	81
GPS 7	Yes	1647955	48	207	20	704	3	118
GPS 8	Yes	15187763	712	1551	23	1536	9	133
GPS 9	Yes	MEMOUT			26	3328	35	223
GPS 10	Yes	MEMOUT			29	7168	131	402
GPS 11	Yes	MEMOUT			32	15360	461	948
GPS 12	Yes	MEMOUT			35	32768	2207	3550

Table 1. Comparison of Uppaal-Tiga with our prototype.

split abstract locations have to be maintained during a single refinement step. If there is a player $\diamond$ winning strategy (Cont=No), more states can be pruned due to a more fine-grained refinement process. Consequently, the effect of pruning is weaker if there is no player $\diamond$ winning strategy (Cont=Yes). On the other hand, higher values for $\mathcal{K}$ result in a lower number of refinement steps but require more memory for a single refinement step. The decrease in the running times results from fewer calls of the backend solving algorithm which reuses the attractor under-approximation from the last call but has to recompute the reachable states.

Benchmark	Cont	$\mathcal{K}$	Steps	Abs	Time	Mem
GPS 12	No	50	28	16187	973	661
GPS 12	No	100	27	16215	1047	711
GPS 12	No	200	27	17974	1254	1201
GPS 12	No	300	27	20236	1341	2237
GPS 12	No	500	27	21859	1970	2893
GPS 12	No	1000	27	22368	1553	3550
GPS 12	No	2000	27	22368	1399	3454
GPS 12	No	5000	MEMOUT			
GPS 12	Yes	50	358	32768	13872	1947
GPS 12	Yes	100	190	32768	9041	1517
GPS 12	Yes	200	73	32768	4774	1585
GPS 12	Yes	300	44	32768	3167	2621
GPS 12	Yes	500	35	32768	2962	3277
GPS 12	Yes	1000	35	32768	2207	3550
GPS 12	Yes	2000	35	32768	1813	3454
GPS 12	Yes	5000	MEMOUT			

Table 2. Comparison of different values for $\mathcal{K}$.

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 Christoph Scholl for pointing out the SIS toolset [17] for finding efficient assignments of control locations to BDD variable valuations, and the anynomous reviewers for their helpful comments.

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. Theo. Comp. Sci. **126**(2) (1994)
3. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Proc. 5th IFAC Conference on System Structure and Control. (1998)
4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: Time for playing games! In: CAV. (2007)
5. Bengtsson, J.: Clocks, DBM, and States in Timed Systems. PhD thesis, Uppsala University (2002)
6. Brückner, I., Dräger, K., Finkbeiner, B., Wehrheim, H.: Slicing abstractions. Volume 89., IOS Press (2008) 369–392
7. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8) (1986) 677–691
8. Bulychev, P., Chatain, T., David, A., Larsen, K.G.: Efficient on-the-fly algorithm for checking alternating timed simulation. In: FORMATS. (2009)
9. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**(2) (1992)
10. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: CONCUR. (2005)
11. de Alfaro, L., Roy, P.: Solving games via three-valued abstraction refinement. In: CONCUR. (2007)
12. Henzinger, T.A., Jhala, R., Majumdar, R.: Counterexample-guided control. In: ICALP. (2003)
13. Henzinger, T.A., Kopke, P.W.: Discrete-time control for rectangular hybrid automata. Theoretical Computer Science **221**(1-2) (1999) 369–392
14. Lewerentz, C., Lindner, T., eds.: Formal Development of Reactive Systems - Case Study Production Cell. (1995)
15. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: STACS. (1995)
16. Peter, H.J., Mattmüller, R.: Component-based abstraction refinement for timed controller synthesis. In: RTSS. (2009)
17. Sentovich, E., Singh, K., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: SIS: A system for sequential circuit synthesis. Technical report, University of California (1992)
18. Somenzi, F.: CUDD: CU Decision Diagram package release 2.4.2 (2009)
19. Thomas, W.: On the synthesis of strategies in infinite games. In: STACS. (1995)

A Proof of Lemma 1

Lemma 1. *For a TGA $\mathcal{G}$ with locations L and partitions Π and Π' of L with $\Pi \prec \Pi'$, if $(\pi_1, \pi_2) \in \Pi \times \Pi$ is not a potential connection in $\mathcal{G}$ wrt. Π, then there is no potential connection $(\pi_1', \pi_2') \in \Pi' \times \Pi'$ in $\mathcal{G}$ wrt. Π' with $\pi_1' \subseteq \pi_1 \wedge \pi_2' \subseteq \pi_2$.*

Proof. Assume for contradiction that there is a pair $(\pi_1', \pi_2') \in \Pi' \times \Pi'$ with $\pi_1' \subseteq \pi_1$ and $\pi_2' \subseteq \pi_2$ that represents a potential connection in $\mathcal{G}$ wrt. Π'. Then, by definition, $(\pi_1', \pi_2') \in \lfloor \Delta \rfloor_{\Pi}$, i.e., there is a transition $\langle l, \varphi, \lambda, l' \rangle \in \Delta$ such that $l \in \pi_1'$ and $l' \in \pi_2'$. Since $\pi_1' \subseteq \pi_1$ and $\pi_2' \subseteq \pi_2$, also $l \in \pi_1$ and $l' \in \pi_2$, and hence π_1 and π_2 represents a potential connection in $\mathcal{G}$ wrt. Π. □

B Proof of Lemma 2

Lemma 2. *Let $\mathcal{G}$ be a TGA with locations L and Π be a partition of L. Then,* $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi})} \subseteq \widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}})} = \mathsf{Attr}(\mathcal{G}) = \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_{\widetilde{\Pi}})} \subseteq \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_{\Pi})}$.

Proof. We only show that $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi})} \subseteq \widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}})} = \mathsf{Attr}(\mathcal{G})$. The other direction $\mathsf{Attr}(\mathcal{G}) = \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_{\widetilde{\Pi}})} \subseteq \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_{\Pi})}$ can be shown similarly. For the rest of this proof, let $\mathcal{G} = (L, I, \Delta, X, G)$, $\lfloor \mathcal{G} \rfloor_{\Pi} = (\Pi, \lfloor I \rfloor_{\Pi}, \lfloor \Delta_{\diamond} \rfloor_{\Pi} \cup \lceil \Delta_{\square} \rceil_{\Pi}, X, \lfloor G \rfloor_{\Pi})$, and $\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}} = (\widetilde{\Pi}, \lfloor I \rfloor_{\widetilde{\Pi}}, \lfloor \Delta_{\diamond} \rfloor_{\widetilde{\Pi}} \cup \lceil \Delta_{\square} \rceil_{\widetilde{\Pi}}, X, \lfloor G \rfloor_{\widetilde{\Pi}})$. Moreover, let $K_{\Pi} = \lfloor G \rfloor_{\Pi} \times \mathcal{R}$ and $K_{\widetilde{\Pi}} = \lfloor G \rfloor_{\widetilde{\Pi}} \times \mathcal{R}$.

Proof of $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi})} \subseteq \widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}})}$. Consider the fixed-point definition of attractors. By this construction, there are state sets $\mathsf{A}_0, \ldots, \mathsf{A}_n$ such that $\mathsf{A}_0 = K_{\Pi}$, $\mathsf{A}_{i+1} = \mathsf{PreEnf}(\mathsf{A}_i) \cup \mathsf{A}_i$ for $i = 0, \ldots, n-1$, and $\mathsf{A}_n = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi})$. Similarly, we can construct sets $\mathsf{B}_0, \ldots, \mathsf{B}_n$ such that $\mathsf{B}_0 = K_{\widetilde{\Pi}}$ and $\mathsf{B}_{i+1} = \mathsf{PreEnf}(\mathsf{B}_i) \cup \mathsf{B}_i$ for $i = 0, \ldots, n-1$. Since $\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}})$ is the least fixed point of the latter construction, we get

$$\mathsf{B}_n \subseteq \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}}). \tag{1}$$

Moreover, by induction on i, we can show that for all $i = 0, \ldots, n$,

$$\widehat{\mathsf{A}_i} \subseteq \widehat{\mathsf{B}_i}. \tag{2}$$

For the base case we have to show that $\widehat{K_{\Pi}} \subseteq \widehat{K_{\widetilde{\Pi}}}$, for which it is sufficient to prove $\bigcup \lfloor G \rfloor_{\Pi} \subseteq \bigcup \lfloor G \rfloor_{\widetilde{\Pi}}$. Let $l \in \bigcup \lfloor G \rfloor_{\Pi}$. Then $l \in \pi$ for some $\pi \in \lfloor G \rfloor_{\Pi}$. By definition of $\lfloor G \rfloor_{\Pi}$, $\pi \subseteq G$, i.e., $l \in G$. Therefore $\pi' = \{l\} \subseteq G$, i.e., $\pi' \in \lfloor G \rfloor_{\widetilde{\Pi}}$ and hence $l \in \bigcup \lfloor G \rfloor_{\widetilde{\Pi}}$.

For the inductive case, assume $\widehat{\mathsf{A}_i} \subseteq \widehat{\mathsf{B}_i}$. To show that $\widehat{\mathsf{A}_{i+1}} \subseteq \widehat{\mathsf{B}_{i+1}}$, let $(l, \boldsymbol{t}) \in \widehat{\mathsf{A}_{i+1}}$. By definition of flattening, there exists $\pi \in \Pi$ such that $l \in \pi$ and $(\pi, \boldsymbol{t}) \in \mathsf{A}_{i+1} = \mathsf{PreEnf}(\mathsf{A}_i) \cup \mathsf{A}_i$. We distinguish two cases. If $(\pi, \boldsymbol{t}) \in \mathsf{A}_i$, then $(l, \boldsymbol{t}) \in \widehat{\mathsf{A}_i}$, and, by induction hypothesis, $(l, \boldsymbol{t}) \in \widehat{\mathsf{B}_i}$. Then $(\{l\}, \boldsymbol{t}) \in \mathsf{B}_i$ and, by the definition of the enforceable predecessor operator, $(\{l\}, \boldsymbol{t}) \in \mathsf{B}_{i+1}$, since $\mathsf{B}_i \subseteq \mathsf{B}_{i+1}$. Therefore, $(l, \boldsymbol{t}) \in \widehat{B_{i+1}}$.

Otherwise, if $(\pi, \boldsymbol{t}) \in \mathsf{PreEnf}(\mathsf{A}_i)$, then either (a) there exists $d > 0$ such that $(\pi, \boldsymbol{t} + d) \in \mathsf{A}_i$ and for all $0 \leq d' < d$ and all $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lceil \Delta_\square \rceil_\Pi$ with $\boldsymbol{t} + d' \models \varphi$ we have $(\pi', (\boldsymbol{t} + d')[\lambda := 0]) \in \mathsf{A}_i$, or (b) there exists a transition $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lfloor \Delta_\diamond \rfloor_\Pi$ such that $\boldsymbol{t} \models \varphi$ and $(\pi', \boldsymbol{t}[\lambda := 0]) \in \mathsf{A}_i$.

First consider case (a). By the definition of flattening and since $l \in \pi$, we know that $(l, \boldsymbol{t}+d) \in \widehat{\mathsf{A}_i}$. Moreover, from the definition of $\lceil \Delta_\square \rceil_\Pi$ we can conclude that, if $\langle l, \varphi, \lambda, l' \rangle \in \Delta_\square$, then $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lceil \Delta_\square \rceil_\Pi$ for all $\pi' \ni l'$, i.e., for all spoiling transitions $\langle l, \varphi, \lambda, l' \rangle \in \Delta_\square$ and $\pi' \ni l'$, we have $(\pi', (\boldsymbol{t} + d')[\lambda := 0]) \in \mathsf{A}_i$, in particular, $(l', (\boldsymbol{t} + d')[\lambda := 0]) \in \widehat{\mathsf{A}_i}$. Now we can apply the induction hypothesis to obtain that $(l, \boldsymbol{t} + d) \in \widehat{\mathsf{B}_i}$ and for all $0 \leq d' < d$ and all $\langle l, \varphi, \lambda, l' \rangle \in \Delta_\square$, also $(l', (\boldsymbol{t}+d')[\lambda := 0]) \in \widehat{\mathsf{B}_i}$. By the definition of flattening (wrt. $\widetilde{\Pi}$), $(\{l\}, \boldsymbol{t}+d) \in \mathsf{B}_i$ and for all $0 \leq d' < d$ and all $\langle \{l\}, \varphi, \lambda, \{l'\} \rangle \in \lceil \Delta_\square \rceil_{\widetilde{\Pi}}$, also $(\{l'\}, (\boldsymbol{t} + d')[\lambda := 0]) \in \mathsf{B}_i$. Therefore, $(\{l\}, \boldsymbol{t}) \in \mathsf{PreEnf}(\mathsf{B}_i) = \mathsf{B}_{i+1}$, and, by definition of flattening, $(l, \boldsymbol{t}) \in \widehat{\mathsf{B}_{i+1}}$.

Now consider case (b). There is a transition $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lfloor \Delta_\diamond \rfloor_\Pi$ such that $\boldsymbol{t} \models \varphi$ and $(\pi', \boldsymbol{t}[\lambda := 0]) \in \mathsf{A}_i$. By definition of $\lfloor \Delta_\diamond \rfloor_\Pi$, for all concrete locations in π, in particular for our fixed $l \in \pi$, there is a corresponding successor location $l' \in \pi'$ and a transition $\langle l, \varphi, \lambda, l' \rangle \in \lfloor \Delta_\diamond \rfloor_\Pi$ such that $\boldsymbol{t} \models \varphi$ and $(\pi', \boldsymbol{t}[\lambda := 0]) \in \mathsf{A}_i$, i.e., $(l', \boldsymbol{t}[\lambda := 0]) \in \widehat{\mathsf{A}_i}$. By induction hypothesis, $(l', \boldsymbol{t}[\lambda := 0]) \in \widehat{\mathsf{B}_i}$ and by the definition of flattening, $(\{l'\}, \boldsymbol{t}[\lambda := 0]) \in \mathsf{B}_i$. Since $\langle \{l\}, \varphi, \lambda, \{l'\} \rangle \in \lfloor \Delta_\diamond \rfloor_{\widetilde{\Pi}}$, by the definition of the enforceable predecessor operator, $(\{l\}, \boldsymbol{t}) \in \mathsf{B}_{i+1}$ and by flattening, $(l, \boldsymbol{t}) \in \widehat{\mathsf{B}_{i+1}}$. This concludes the inductive proof of Eq. 2.

Since $\mathsf{A}_n = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)$ and since $X \subseteq Y$ implies $\widehat{X} \subseteq \widehat{Y}$, from Eq. 1 and Eq. 2 for $i = n$, we get our claim that $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)} = \widehat{\mathsf{A}_n} \subseteq \widehat{\mathsf{B}_n} \subseteq \widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}})}$.

Proof of $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}})} = \mathsf{Attr}(\mathcal{G})$. We show that $\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}}$ is isomorphic to $\mathcal{G}$, specifically that $\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}}$ can be obtained from $\mathcal{G}$ by renaming all locations l to $\{l\}$. This follows immediately from the definition of the components of $\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}}$:

$$
\begin{aligned}
\widetilde{\Pi} &= \{\{l\} \mid l \in L\}, \\
\lfloor I \rfloor_{\widetilde{\Pi}} &= \{\pi \in \widetilde{\Pi} \mid \pi \subseteq I\} = \{\{l\} \mid l \in I\}, \\
\lfloor \Delta_\diamond \rfloor_{\widetilde{\Pi}} &= \{\langle \pi, \varphi, \lambda, \pi' \rangle \mid \forall l \in \pi \colon \exists l' \in \pi' \colon \langle l, \varphi, \lambda, l' \rangle \in \Delta_\diamond\} \\
&= \{\langle \{l\}, \varphi, \lambda, \{l'\} \rangle \mid \langle l, \varphi, \lambda, l' \rangle \in \Delta_\diamond\}, \\
\lceil \Delta_\square \rceil_{\widetilde{\Pi}} &= \{\langle \pi, \varphi, \lambda, \pi' \rangle \mid \exists \langle l, \varphi, \lambda, l' \rangle \in \Delta_\square \colon l \in \pi \wedge l' \in \pi'\} \\
&= \{\langle \{l\}, \varphi, \lambda, \{l'\} \rangle \mid \langle l, \varphi, \lambda, l' \rangle \in \Delta_\square\}, \text{ and} \\
\lfloor G \rfloor_{\widetilde{\Pi}} &= \{\pi \in \widetilde{\Pi} \mid \pi \subseteq G\} = \{\{l\} \mid l \in G\}.
\end{aligned}
$$

Because of this isomorphism between $\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}}$ and $\mathcal{G}$, the attractor sets satisfy

$$
\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}}) = \{(\{l\}, \boldsymbol{t}) \mid (l, \boldsymbol{t}) \in \mathsf{Attr}(\mathcal{G})\},
$$

and, by definition of flattening, $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\widetilde{\Pi}})} = \mathsf{Attr}(\mathcal{G})$. $\square$

C Proof of Lemma 3

Lemma 3. *If* $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)} \subsetneq \mathsf{Attr}(\mathcal{G})$*, then there exists an effective interpolant.*

Proof. Let $\lfloor A \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)$, let $K = G \times \mathcal{R}$ and $K_\Pi = \lfloor G \rfloor_\Pi \times \mathcal{R}$ be the sets of goal states of $\mathcal{G}$ and $\lfloor \mathcal{G} \rfloor_\Pi$, and assume $\widehat{\lfloor \mathsf{A} \rfloor} \subsetneq \mathsf{Attr}(\mathcal{G})$. Then there exists a state $(l, \boldsymbol{t}) \in \mathsf{Attr}(\mathcal{G})$ such that $(l, \boldsymbol{t}) \notin \widehat{\lfloor \mathsf{A} \rfloor}$. By definition of flattening, there is no $\pi \in \Pi$ such that $l \in \pi$ and $(\pi, \boldsymbol{t}) \in \lfloor \mathsf{A} \rfloor$, since otherwise we would have $(l, \boldsymbol{t}) \in \widehat{\lfloor \mathsf{A} \rfloor}$. On the other hand, since $(l, \boldsymbol{t}) \in \mathsf{Attr}(\mathcal{G})$, there is a finite index $i \geq 0$ such that $(l, \boldsymbol{t}) \in \mathsf{A}_{i+1}$, where $\mathsf{A}_0 = K$, $\mathsf{A}_{i+1} = \mathsf{PreEnf}(\mathsf{A}_i) \cup \mathsf{A}_i$ for $i = 0, \ldots, n-1$, and $\mathsf{A}_n = \mathsf{Attr}(\mathcal{G})$. First note that we do not have to consider the case that $(l, \boldsymbol{t}) \in \mathsf{A}_0 = K$, since if $(l, \boldsymbol{t}) \in K$, also $(\pi, \boldsymbol{t}) \in K_\Pi$, because in the initial partition $\Pi_0 = \{L \setminus G, G\}$, for all $\pi \in \Pi_0$, either $\pi \cap G = \emptyset$ or $\pi \subseteq G$, and this property holds inductively for all partitions Π, since partitions are only split and not merged. Therefore, if $l \in G$ and $l \in \pi$, then $\pi \subseteq G$ and hence, $\pi \in \lfloor G \rfloor_\Pi$ and therefore, $(\pi, \boldsymbol{t}) \in K_\Pi$, i.e., $(l, \boldsymbol{t}) \in \widehat{K_\Pi} \subseteq \widehat{\lfloor A \rfloor}$. In particular, there is a *least* such index i with $(l, \boldsymbol{t}) \in \mathsf{A}_{i+1}$, i.e., $(l, \boldsymbol{t}) \in \mathsf{A}_{i+1} \setminus \mathsf{A}_i$. For $(l, \boldsymbol{t}) \in \mathsf{A}_{i+1}$, there must be a move leading from $(l, \boldsymbol{t})$ to a state $(l', \boldsymbol{t}')$ already contained in A_i (and no spoiling move by the opponent $\Box$). Let $((l, \boldsymbol{t}), (l', \boldsymbol{t}')) \in \Gamma_\Diamond$ be such a move and $\pi, \pi' \in \Pi$ be the partitions with $l \in \pi$ and $l' \in \pi'$.

If one of the following two conditions were satisfied, then we would have $(\pi, \boldsymbol{t}) \in \lfloor \mathsf{A} \rfloor$, in contradiction to our assumption:

(1) $((l, \boldsymbol{t}), (l', \boldsymbol{t}')) \in \Gamma_\Diamond$ is an active action move induced by a transition $\langle l, \varphi, \lambda, l' \rangle \in \Delta_\Diamond$ with $\boldsymbol{t} \models \varphi$ and $\boldsymbol{t'} = \boldsymbol{t}[\lambda := 0]$, and $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lfloor \Delta_\Diamond \rfloor_\Pi$, and $(\pi', \boldsymbol{t}') \in \lfloor \mathsf{A} \rfloor$; or
(2) $((l, \boldsymbol{t}), (l', \boldsymbol{t}')) \in \Gamma_\Diamond$ is a passive wait move with $l' = l$ and $\boldsymbol{t}' = \boldsymbol{t} + d$ for some $d > 0$, such that $(\pi', \boldsymbol{t}') \in \lfloor \mathsf{A} \rfloor$, and there is no spoiling move by the opponent of the form $((l, \boldsymbol{t} + d'), (l'', \boldsymbol{t} + d'[\lambda := 0])) \in \Gamma_\Box$ for $0 \leq d' < d$ with $(l'', \boldsymbol{t} + d'[\lambda := 0]) \notin \lfloor \mathsf{A} \rfloor$, induced by some transition $\langle l, \varphi, \lambda, l'' \rangle \in \Delta_\Box$ with $\boldsymbol{t} + d' \models \varphi$, and $\langle \pi, \varphi, \lambda, \pi'' \rangle \in \lceil \Delta_\Box \rceil_\Pi$, where $\pi'' \in \Pi$ is the part with $l'' \in \pi''$.

Therefore, none of the two conditions is satisfied, i.e., conversely, one of the following has to hold: either

(a) $(\pi', \boldsymbol{t}') \notin \lfloor \mathsf{A} \rfloor$; or
(b) the active move $((l, \boldsymbol{t}), (l', \boldsymbol{t}')) \in \Gamma_\Diamond$ from (1) is not represented in the abstraction, i.e., $\langle \pi, \varphi, \lambda, \pi' \rangle \notin \lfloor \Delta_\Diamond \rfloor_\Pi$; or
(c) whereas the passive wait move $((l, \boldsymbol{t}), (l', \boldsymbol{t}')) \in \Gamma_\Diamond$ from (2) can be abstracted to an abstract move $((\pi, \boldsymbol{t}), (\pi, \boldsymbol{t} + d))$, there is an abstract spoiling move present in the abstraction that has no concrete counterpart, say $((\pi, \boldsymbol{t} + d'), (\pi'', \boldsymbol{t} + d'[\lambda := 0]))$ for $0 \leq d' < d$ with $(\pi'', \boldsymbol{t} + d'[\lambda := 0]) \notin \lfloor \mathsf{A} \rfloor$, induced by some transition $\langle \pi, \varphi, \lambda, \pi'' \rangle \in \lceil \Delta_\Box \rceil_\Pi$ with $\boldsymbol{t} + d' \models \varphi$, where $\pi'' \in \Pi$ is the part with $l'' \in \pi''$.

We may assume without loss of generality that (a) is not the culprit (i.e., that (a) is not satisfied), since if (a) were satisfied, we could replace $(\pi, \boldsymbol{t})$ in our argument by $(\pi', \boldsymbol{t}')$ and prove that $(l', \boldsymbol{t}') \notin \widehat{\lfloor \mathsf{A} \rfloor}$, but $(l', \boldsymbol{t}') \in \mathsf{Attr}(\mathcal{G})$. Note that this assumption is justified by the fact that in the worst case we only have to make finitely many such replacements of $(\pi, \boldsymbol{t})$ by $(\pi', \boldsymbol{t}')$, since $(l, \boldsymbol{t}) \in \mathsf{A}_{i+1} \setminus \mathsf{A}_i$, but $(l', \boldsymbol{t}') \in \mathsf{A}_i$, and i is finite. So, either (b) or (c) holds.

If (b) holds, fix $l, \boldsymbol{t}, \varphi, \lambda, l'$, and π as above, and let $R := \{l_2 \in \pi \mid \langle l_2, \varphi, \lambda, l' \rangle \in \Delta_\Diamond\}$. Then $R \neq \emptyset$, since $l \in R$, and $R \subsetneq \pi$, since if we had $R = \pi$, then we could conclude that $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lfloor \Delta_\Diamond \rfloor_\Pi$, in contradiction to (b). Moreover, $\langle \pi, \varphi, \lambda, \pi' \rangle \in \lceil \Delta_\Diamond \rceil_\Pi \setminus \lfloor \Delta_\Diamond \rfloor_\Pi$, and $\forall l_2 \in R \cap \pi : \exists l' \in \pi' : \langle l_2, \varphi, \lambda, l' \rangle \in \Delta_\Diamond$ by the definition of R, and $\boldsymbol{t} \in [\![\varphi]\!]$, $(\pi, \boldsymbol{t}) \notin \lfloor \mathsf{A} \rfloor$ and $(\pi', \boldsymbol{t}[\lambda := 0]) \in \lfloor \mathsf{A} \rfloor$. In this case, R satisfies part (1) of the statement of the definition of effective interpolants.

Otherwise, if (c) holds, fix l, $\boldsymbol{t}$, φ, λ, l'', d', and π as above, and let $R := \{l_2 \in \pi \mid \langle l_2, \varphi, \lambda, l'' \rangle \in \Delta_\Box\}$. Then $R \neq \emptyset$, since $l \in R$, and $R \subsetneq \pi$, since if we had $R = \pi$, then we could conclude that $\langle \pi, \varphi, \lambda, \pi'' \rangle \in \lfloor \Delta_\Box \rfloor_\Pi$, which is false. Moreover, $\langle \pi, \varphi, \lambda, \pi'' \rangle \in \lceil \Delta_\Box \rceil_\Pi \setminus \lfloor \Delta_\Box \rfloor_\Pi$, and $\forall l_2 \in \pi : \exists l'' \in \pi' : \langle l_2, \varphi, \lambda, l'' \rangle \in \Delta_\Box \Rightarrow l_2 \in R$ by the definition of R, and $\boldsymbol{t} \in [\![\varphi]\!]$ and $(\pi'', \boldsymbol{t} + d'[\lambda := 0]) \notin \lfloor \mathsf{A} \rfloor$. In this case, R satisfies part (2) of the statement of the definition of effective interpolants.

In both cases (b) and (c), we can find a set $R \subseteq L$ that satisfies (1) or (2) in the definition of effective interpolants, which concludes the existence proof of such a set R. ◻

D Proof of Lemma 4

Lemma 4. *If $R \subseteq L$ is an effective interpolant, then $\Pi \prec \Pi|R$.*

Proof. Since R is an effective interpolant, there must be at least one $\pi \in \Pi$ such that $\emptyset \subsetneq \pi \cap R \subsetneq \pi$, hence $\pi_1 = \pi \cap R \neq \emptyset$ and $\pi_2 = \pi \setminus R \neq \emptyset$. By definition of refinement, since $\pi_1 \neq \emptyset$ and $\pi_2 \neq \emptyset$, π is replaced by π_1 and π_2 and other locations π' other than π may or may not be split by R, and hence, $\Pi \prec \Pi|R$. ◻

E Proof of Lemma 5

Lemma 5. *Let $R \subseteq L$, $\lfloor \mathsf{A} \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)$, and $\lfloor \mathsf{A}' \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi|R})$, where $\lfloor \mathsf{A} \rfloor$ is the starting point for computing $\lfloor \mathsf{A}' \rfloor$.*
If $\widehat{\lfloor \mathsf{A} \rfloor} \subsetneq \widehat{\lfloor \mathsf{A}' \rfloor}$, then R is an effective interpolant.

Proof. For a TGA $\mathcal{G} = (L, I, \Delta, X, G)$, let Π be a location partition of L, $R \subseteq L$ be a set of locations, $\Pi' = \Pi|R$ be the refined partition. Let $\lfloor \mathsf{A} \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_\Pi)$ and $\lfloor \mathsf{A}' \rfloor = \mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi'})$ with $\widehat{\lfloor \mathsf{A} \rfloor} \subsetneq \widehat{\lfloor \mathsf{A}' \rfloor}$. Furthermore, we assume that $\lfloor \mathsf{A} \rfloor$ is a starting point for computing $\lfloor \mathsf{A}' \rfloor$, i.e.,

$$\widehat{\lfloor \mathsf{A} \rfloor} \subsetneq \widehat{\lfloor \mathsf{A} \rfloor} \cup \widehat{\mathsf{PreEnf}(\mathsf{A}^*)} \subseteq \widehat{\lfloor \mathsf{A}' \rfloor},$$

where A^* is the (unique) projection of $\lfloor \mathsf{A} \rfloor$ to $\Pi' \times \mathcal{R}$, i.e., $\widehat{\mathsf{A}^*} = \widehat{\lfloor \mathsf{A} \rfloor}$, and PreEnf is the enforceable predecessor operator defined over $\lfloor \mathcal{G} \rfloor_{\Pi'}$. We prove that R is an effective interpolant.
We distinguish between *active* and *delay* predecessors:

$$\begin{aligned}\mathsf{PreEnf}_a(Y) = \{(\pi_1', \boldsymbol{t}) \in \Pi' \times \mathcal{R} \mid \exists \pi_2', \varphi, \lambda : \langle \pi_1', \varphi, \lambda, \pi_2' \rangle \in \lfloor \Delta_\Diamond \rfloor_{\Pi'} \\ \wedge\ \boldsymbol{t} \models \varphi\ \wedge\ (\pi_2', \boldsymbol{t}[\lambda := 0]) \in Y\}; \\ \mathsf{PreEnf}_d(Y) = \mathsf{PreEnf}(Y) \setminus \mathsf{PreEnf}_a(Y).\end{aligned}$$

Assume $\widehat{\mathsf{PreEnf}_a(\mathsf{A}^*)} \setminus \widehat{\lfloor \mathsf{A} \rfloor} \neq \emptyset$. By definition of that assumption, we deduce that there is a transition $\langle \pi_1', \varphi, \lambda, \pi_2' \rangle \in \lfloor \Delta_\Diamond \rfloor_{\Pi'}$ such that there is a $\boldsymbol{t} \in [\![\varphi]\!]$ such that there are states $(\pi_2', \boldsymbol{t}[\lambda := 0]) \in \lfloor \mathsf{A}' \rfloor$ and $(\pi_1', \boldsymbol{t}) \in \lfloor \mathsf{A}' \rfloor$, but $(\pi, \boldsymbol{t}) \notin \lfloor \mathsf{A} \rfloor$, for the part $\pi \in \Pi$ with $\pi \supseteq \pi_1'$. We assume wlog. $\pi_2' \in \Pi$ and $\pi_2' \in \Pi'$. From $(\pi, \boldsymbol{t}) \notin \lfloor \mathsf{A} \rfloor$ follows that there is no transition $\langle \pi, \varphi, \lambda, \pi_2' \rangle \in \lfloor \Delta_\Diamond \rfloor_{\Pi}$. Thus, $\langle \pi, \varphi, \lambda, \pi_2' \rangle \in \lceil \Delta_\Diamond \rceil_{\Pi} \setminus \lfloor \Delta_\Diamond \rfloor_{\Pi}$, and by definition of $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$, we deduce $\emptyset \subsetneq \pi_1' \subsetneq \pi$. We conclude that π_1' fulfills all (1)-requirements for an effective interpolant.
Assume $\widehat{\mathsf{PreEnf}_d(\mathsf{A}^*)} \setminus \widehat{\lfloor \mathsf{A} \rfloor} \neq \emptyset$. By definition of that assumption, we deduce that for a delay $d > 0$, there are states $(\pi', \boldsymbol{t} + d) \in \lfloor \mathsf{A}' \rfloor$, $(\pi', \boldsymbol{t}) \in \lfloor \mathsf{A}' \rfloor$, and $(\pi, \boldsymbol{t} + d) \in \lfloor \mathsf{A} \rfloor$ but $(\pi, \boldsymbol{t}) \notin \lfloor \mathsf{A} \rfloor$, for the part $\pi \in \Pi$ with $\pi \supseteq \pi'$. Hence, by definition of PreEnf, there is a spoiling transition $\langle \pi, \varphi, \lambda, \pi_2 \rangle \in \lceil \Delta_\Box \rceil_{\Pi}$ such that there is a $\boldsymbol{t}' = \boldsymbol{t} + d'$, for a $d' < d$, with $\boldsymbol{t}' \in [\![\varphi]\!]$ such that there is a state $(\pi_2, \boldsymbol{t}'[\lambda := 0]) \in \lfloor \mathsf{A} \rfloor$. We assume wlog. $\pi_2 \in \Pi$ and $\pi_2 \in \Pi'$. From $(\pi', \boldsymbol{t}) \in \lfloor \mathsf{A}' \rfloor$ follows that there is no transition $\langle \pi', \varphi, \lambda, \pi_2 \rangle \in \lceil \Delta_\Box \rceil_{\Pi'}$. Thus, $\langle \pi, \varphi, \lambda, \pi_2 \rangle \in \lceil \Delta_\Box \rceil_{\Pi} \setminus \lfloor \Delta_\Box \rfloor_{\Pi}$, and by definition of $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$, we deduce $\emptyset \subsetneq \pi' \subsetneq \pi$. We conclude that π' fulfills all (2)-requirements for an effective interpolant. $\square$

F Proof of Theorem 2

Theorem 2. *The presented abstraction refinement loop always terminates and yields a sound winning strategy for one of the players upon termination.*

Proof. In this proof, we only cover the case of constructing a sequence of under-approximations. For over-approximations, note that Def. 1 as well as Lemma 3 can easily be dualized such that Def. 1 also covers shrinking attractor over-approximations, and that Lemma 3 guarantees the existence of an effective interpolant if $\mathsf{Attr}(\mathcal{G}) \subsetneq \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_{\Pi})}$ as well.

First, we show termination of the refinement loop. In a loop cycle i, from Lemma 3 follows that there always exists an effective interpolant R_i since as long as no termination criterion is satisfied the attractor approximations do not yet have full precision, i.e., $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi_i})} \subsetneq \mathsf{Attr}(\mathcal{G})$ and $\mathsf{Attr}(\mathcal{G}) \subsetneq \widehat{\mathsf{Attr}(\lceil \mathcal{G} \rceil_{\Pi_i})}$, where Π_i is the current partition. Wlog., we assume $\widehat{\mathsf{Attr}(\lfloor \mathcal{G} \rfloor_{\Pi_i})} \subsetneq \mathsf{Attr}(\mathcal{G})$. From Lemma 4 follows that, since R_i is an effective interpolant, $\Pi_i \prec \Pi_{i+1} = \Pi_i | R_i$,

hence $|\Pi_i| < |\Pi_{i+1}|$. On the other hand, every partition of L is coarser or equal to $\widetilde{\Pi}$, which is finite since L is finite. Hence, every partition ultimately converges to $\widetilde{\Pi}$ in a finite number of refinement steps. From Lemma 2 follows that if $\Pi_n = \widetilde{\Pi}$, for a finite $n \in \mathbb{N}_0$, the under- and over-approximations of the attractor coincide with the precise attractor. In this case, exactly one of the two termination conditions is trivially true.

Second, upon termination, Lemma 2 immediately implies the soundness of the computed attractor approximations, and hence, the soundness of the result of the algorithm. □