AbsSynthe: abstract synthesis from succinct safety specifications[*]

Romain Brenguier, Guillermo A. Pérez, Jean-François Raskin
Ocan Sankur
Université Libre de Bruxelles – Brussels, Belgium
{rbrengui,gperezme,jraskin,osankur}@ulb.ac.be

In this paper, we describe a synthesis algorithm for safety specifications described as circuits. Our algorithm is based on fixpoint computations, abstraction and refinement, it uses binary decision diagrams as symbolic data structure. We evaluate our tool on the benchmarks provided by the organizers of the synthesis competition organized within the SYNT'14 workshop.

1 Introduction

The *model-checking* approach to verification of reactive systems is as follows. Given a model of the system together with a description of the environment in which it is embedded, and a specification that formalizes a property of interest of the system, an algorithm verifies that all the possible behaviors of the system within its environment comply with the specification. Model-checking has been proposed in the eighties and is now a standard technique to improve the reliability of reactive systems.

Synthesis goes a step further: synthesis only requires a model of the environment together with a specification of a property that the system must enforce within the environment, it does not require a model of the system. From the description of the environment and the property, an algorithm tries to build automatically a system that is correct by construction, i.e. a system which enforces the specification. If such a system does not exist then the synthesis algorithm can also provide feedback in the form of a strategy for the environment that enforces the negation of the specification and so shows why the specification cannot be realized.

The synthesis problem can be formalized as a two-player game on a graph with an omega-regular objective. While the theory that underlines those games is now well understood, see e.g. [27], there are only a few implementations available [2, 11, 13, 24] and the sizes of systems on which synthesis has been applied are usually much smaller than the sizes of systems for which model-checking has been successfully applied.

This paper describes our experiences with building a prototype of tool to participate to the first synthesis competition organized within the SYNT'14 workshop. The set up for this competition is as follows. Given a sequential circuit description with exactly one output signal and a partition of its input signals into *controllable* inputs that belongs to the system (to synthesize) and *uncontrollable* inputs that belong to the environment, decide if there is a strategy to choose the controllable input signals such that no matter how the uncontrollable input signals are updated along the execution of the circuit, the output of the circuit is always low. In terms of two player games, the winning condition (the specification) for the system is thus a safety objective. If a winning strategy exists for the system, then build a circuit which implements this strategy.

[*] This work was supported by the ERC inVEST (279499) project.

K. Chatterjee, R. Ehlers, and S. Jha (Eds.):
Third Workshop on Synthesis (SYNT 2014)
EPTCS 157, 2014, pp. 100–116, doi:10.4204/EPTCS.157.11

© R. Brenguier et al.
This work is licensed under the
Creative Commons Attribution License.

The realizability problem for safety specifications is known to be solvable in linear time with respect to the size of the underlying game graph (see, e.g. [14,27]). However, here the underlying graph is given implicitly and succinctly by the circuit description and in this case the problem is known to be complete for EXP (see, e.g. [22]). To combat the state explosion problem, we adapt two classical techniques that have proven useful in the context of model-checking: we use binary decision diagrams [3] as a data structure to represent and manipulate symbolically sets of configurations of the circuit specification, and we use abstraction and refinement to simplify the underlying game and lower its dimension (the number of Boolean variables that are necessary for its description). The abstraction refinement algorithm that we have defined can be seen as combining the ideas of [1] and [16] together with binary decision diagrams (BDDs) and some additional heuristics, all this formalized with abstract interpretation as in [17].

We have implemented a fixpoint computation together with several optimizations that lead to a synthesis algorithm that is able to handle circuits with several tens of latches and a few hundreds of gates. We report on the experiments that we have conducted on all the benchmarks provided by the organizers of the synthesis competition that were available at the time of submission. In a vast majority of the benchmarks the best performing version of our algorithm is the *plain fixpoint* algorithm that does not use abstraction at all. However, to be efficient, the explicit construction of the BDDs for the transition relation needs to be avoided [4], our solution is to use substitution of variables with BDDs as in [8] to directly compute the effect of the transition relation backwards. Nevertheless, for some examples, abstraction and refinement are necessary: our algorithm based on abstraction and refinement terminates while the basic concrete fixpoint computation does not. We think that the lack of good performance of abstraction in our experiments is partly due to the fact that there is no explicit structure in the circuit description on which we apply our analysis. In fact, we consider circuits given in a low-level description, which is neither hierarchical nor compositional, so the usual techniques used in program verification [5] does not seem to be applicable here. Another reason could also be that the benchmarks considered here are control intensive and not data oriented where abstraction seems to perform better [6]. We strongly believe that more research is necessary for understanding how to recover interesting aspects of the structure present in the circuits from their low level descriptions and use this structure in the abstraction procedure. Finally, we also report on how to synthesize a circuit from the winning region computed by our algorithm and how to exploit reachability information to decrease the size of the synthesized circuit with minimization of BDDs using don't care sets [18].

Related works Recent efforts to find efficient algorithms for synthesis have been reported in [2] where solutions based on solvers for QBF and SAT are investigated. In that paper, the authors compare their solutions with a BDD implementation that constructs explicitly the transition relation. The conclusions of their paper resemble our conclusions: the BDD implementation usually outperforms the QBF-SAT algorithms with the exception of a few examples. Our BDD implementation that does not construct the transition relation is usually largely more efficient than the one that constructs the BDD for the transitions relation.

In [21] the authors present an algorithm for synthesis which searches for a small set of plays that witness a winning strategy for one of the players. They report their tool works well for games in which winning strategies admit compact representations.

The problem of minimization of Boolean functions used in circuit constructions has been widely studied in the logic synthesis community (e.g. [18,20]).

2 Preliminaries

We will present our algorithms in set-theoretic notation. However, in order to provide symbolic representations of sets and the implementation of set operators we will also represent sets by Boolean functions, and use both notations interchangeably. Formally, we let $\mathbb{B} = \{0,1\}$, and if L denotes a finite set of variables, a function $L \to \mathbb{B}$ is called a *valuation of* L. Note that a valuation v defines a subset $v^{-1}(1)$ of L. We will also consider Boolean functions $\mathbb{B}^L \to \mathbb{B}$ to denote sets of valuations.

We will describe Boolean functions by *first-order logic* formulas on a given set of variables V, which are made of propositional logic and first-order quantification on V. A *formula* f whose free variables are X will be written $f(X)$. If the free variables are $X \cup Y$ for two sets X,Y, we may also write $f(X,Y)$. When we quantify over a set of variables L, we will write $\exists L$ instead of $\exists l_1 \exists l_2 \dots \exists l_n$ if $L = \{l_1,\dots,l_n\}$, and similarly for universal quantification.

Let X,Y,Z be three sets of variables such that $Y \subseteq X$ and $X \cap Z = \varnothing$. Consider a formula $f(X)$ and a set of formulas $(g_y(Z))_{y\in Y}$ (one for each element in Y). We denote by $f[y \leftarrow g_y]_{y\in Y}$ the formula f in which every $y \in Y$ has been substituted by the corresponding g_y. Formally, $f[y \leftarrow g_y]_{y\in Y}(X \setminus Y,Z) = \exists Y.\, f(X) \wedge \big(\bigwedge_{y\in Y} y \Leftrightarrow g_y(Z)\big)$. This work has been implemented using BDDs [3] to perform all symbolic operations on Boolean functions. BDD packages provide optimized procedures to do substitution (see e.g. function *compose* in [25]).

Circuit specifications We are interested in synthesizing controllers for synchronous sequential circuits enforcing a given safety specification, where some inputs are *controllable*, and others are *uncontrollable*. Intuitively, controllable inputs are to be determined by the controller to be synthesized, while uncontrollable inputs cannot be restricted, and are determined by the environment. A distinguished latch indicates if an error has occurred. Formally, a *synchronous sequential circuit* is a tuple $\langle X_u, X_c, L, (f_l)_{l\in L}, f_{\mathsf{BAD}}\rangle$, where:

- X_u, X_c, L are finite sets of boolean variables representing *uncontrollable inputs*, *controllable inputs*, and *latches* respectively;
- for each latch $l \in L$, $f_l \colon \mathbb{B}^{X_u} \times \mathbb{B}^{X_c} \times \mathbb{B}^L \to \mathbb{B}$ is the *transition function* that gives the valuation of l in the next step;
- f_{BAD} is the *error function* $f_{\mathsf{BAD}} \colon \mathbb{B}^{X_u} \times \mathbb{B}^{X_c} \times \mathbb{B}^L \to \mathbb{B}$, which evaluates to true in error states.

Given a circuit, our goal is to synthesize a *controller* which, given any valuation of the latches and uncontrollable inputs, sets the controllable inputs, in order to ensure that the overall system never enters an error state.

We assume that (i) there is a latch $\mathsf{BAD} \in L$ which, once it becomes true, stays true, and (ii) that the latches are initialized to 0, i.e. the initial valuation is $v(l) = 0$ for all l.

Reachability and Safety Games The problem of controller synthesis can be formalized as a game between two players, namely, environment and controller, played on a graph (see, e.g. [27]). Formally, an *arena* is a tuple $G = \langle Q, q_I, \Sigma_u, \Sigma_c, \Delta\rangle$ where: (i) Q is a finite set of states; (ii) $q_I \in Q$ is the initial state; (iii) Σ_u is a finite set of uncontrollable actions; (iv) Σ_c a finite set of controllable actions; (v) $\Delta \subseteq Q \times \Sigma_u \times \Sigma_c \times Q$ is a transition relation.

The game is initially in state q_I and is played in rounds. At every round, from state q, environment chooses an action a_u from Σ_u and controller responds by choosing an action a_c from Σ_c and a successor state $s \in Q$ such that $(q,a_u,a_c,s) \in \Delta$. We write δ instead of Δ if the transition relation is functional.

A *play* in such a game consists of an infinite sequence of states, i.e. $q_0q_1\ldots \in Q^\omega$, where $q_0 = q_I$. For a play $\pi = q_0q_1\ldots$, we denote by $\pi[n]$ its $(n+1)$-th state, i.e q_n. A *strategy of environment* is a function $\lambda^{env}\colon Q^* \to \Sigma_u$ which given a sequence of states, chooses an uncontrollable action. A *strategy of controller* is a function $\lambda^{ctrl}\colon Q^* \times \Sigma_u \to \Sigma_c$ which given a sequence of states and an uncontrollable action, choses a controllable action. For $\pi = q_0q_1\ldots q_n \in Q^*$, we denote by $\mathsf{last}(\pi)$ the last state from π, i.e. q_n. We say λ^{env} is a *memoryless strategy of environment* if for any $\pi,\pi' \in Q^*$ then $\mathsf{last}(\pi) = \mathsf{last}(\pi')$ implies $\lambda^{env}(\pi) = \lambda^{env}(\pi')$. Similarly, λ^{ctrl} is a *memoryless strategy of controller* if for any $\pi,\pi' \in Q^*, a_u \in \Sigma_u$, then $\mathsf{last}(\pi) = \mathsf{last}(\pi')$ implies $\lambda^{ctrl}(\pi,a_u) = \lambda^{ctrl}(\pi',a_u)$.

A play π is *consistent* with a pair of strategies $(\lambda^{env},\lambda^{ctrl})$ if for all $i \geq 0$:

$$\pi[i+1] = \delta(\pi[i],\lambda^{env}(\pi[i]),\lambda^{ctrl}(\pi[i],\lambda^{env}(\pi[i]))).$$

Given a strategy λ^{ctrl} of controller, we write $\mathsf{Plays}(G,\lambda^{ctrl})$ the set of plays that are consistent with $(\lambda^{env},\lambda^{ctrl})$ for some λ^{env}.

A *safety game* is a pair $\langle G,\mathscr{U}\rangle$ where $\mathscr{U} \subseteq Q$ is a set of *unsafe* states. The objective of controller is to keep the play within the states $Q\setminus\mathscr{U}$ at all times. We say that λ^{ctrl} is *winning for controller* if for any play $\pi \in \mathsf{Plays}(G,\lambda^{ctrl})$, for all $n \geq 0$, $\pi[n] \notin \mathscr{U}$. Otherwise, π is *winning for environment*, and we denote by i_π the first turn in which a state in $\mathscr{U}$ is visited, that is $i_\pi = \min\{i \geq 0 \mid \pi[i] \in \mathscr{U}\}$. Note that in safety games, the objective of environment is to *reach* $\mathscr{U}$. From the point of view of environment, these are in fact *reachability games*.

In this work we study finite safety and reachability games for which it is known that *memoryless strategies* suffice for either player (see, e.g. [14]). Thus in what follows, when we speak about strategies, we mean memoryless strategies and we take strategies for environment and controller to be of the form $\lambda^{env} : Q \to \Sigma_u$ and $\lambda^{ctrl} : Q \times \Sigma_u \to \Sigma_c$, respectively.

Safety Games For Circuits We formalize the controller synthesis problem for circuits as safety games. Given a circuit specification $\langle X_u, X_c, L, (f_l)_{l\in L}, f_{\mathsf{BAD}}\rangle$, we define the game $\langle G,\mathscr{U}\rangle$ with $G = \langle Q, q_I, \Sigma_u, \Sigma_c, \delta\rangle$, where $Q = \mathbb{B}^L$, $q_I = 0^L$ (i.e. the valuation that assings 0 to all L), $\Sigma_u = \mathbb{B}^{X_u}$, and $\Sigma_c = \mathbb{B}^{X_c}$. So states (resp. actions) in G are valuations on latches (resp. inputs). Let q,s be valuations on latches. We define accordingly the transition function as $\delta(q,\sigma_u,\sigma_c) \mapsto s$ if $s(l) = f_l(q,\sigma_u,\sigma_c)$ for all $l \in L$.

3 Realizability

Basic Fixpoint Algorithm We recall the basic fixpoint computation for solving safety games, applied here on safety games for circuits. Let $C = \langle X_u, X_c, L, (f_l)_{l\in L}, f_{\mathsf{BAD}}\rangle$ be a circuit specification and $G_C = \langle Q, q_I, \Sigma_u, \Sigma_c, \delta, \mathscr{U}\rangle$ the associated safety game. The set of the states from which there is no controller's strategy to ensure the safety objective can be computed by iterating an *uncontrollable predecessors* operator. For any $S \subseteq Q$, the *uncontrollable predecessors* of S is defined as

$$\mathsf{UPRE}(S) = \{q \in Q \mid \exists \sigma_u \in \Sigma_u.\ \forall \sigma_c \in \Sigma_c : \delta(q,\sigma_u,\sigma_c) \in S\}.$$

We denote by $\mathsf{UPRE}^*(S) = \mu X.(S \cup \mathsf{UPRE}(X))$, the *least fixpoint* of the function $F : X \to S \cup \mathsf{UPRE}(X)$ in the μ-calculus notation (see [12]). Note that F is defined on the powerset lattice, which is finite. It follows from Tarski-Knaster theorem [26] that, because F is monotonic, the fixpoint exists and can be computed by iterating the application of F starting from the least value of the lattice, i.e. $\emptyset$.

The following is a well-known result about the relationship between safety games and the UPRE operator. The second part of the claim follows from the determinacy of finite safety games [14].

Proposition 1. *Let C be a circuit specification and G_C the associated safety game. Then (i) environment has a winning strategy in G_C if and only if $q_I \in \mathsf{UPRE}^*(\mathscr{U})$; and (ii) controller has a winning strategy in G_C if and only if $q_I \notin \mathsf{UPRE}^*(\mathscr{U})$.*

Symbolic implementation of UPRE There is a plethora of symbolic algorithms to do forward and backward state space exploration in large systems defined succinctly, e.g. [4, 7, 9] to mention a few classic works on the topic. The construction of a symbolic (monolithic or partitioned) transition relation is the first step of those algorithms. For deterministic systems, where the transition relation is functional, a *transition function vector* can be used to represent the transitions (that is, one distinct function for each latch). This is known to improve the performance of state space exploration algorithms in some systems, although this is not the case systematically; see [7,9].

We consider both monolithic and partitioned transition relations in this work. We present 1) a version of the operators using the monolithic transition relation $T(L,X_u,X_c,L')$ constructed once at the beginning of the algorithm, and 2) an alternative version using only the partitioned transition relation. Our results also confirm that the preference between the two depends on the type of circuit (see Section 7).

More precisely, the monolithic transition relation is defined as $T(L,X_u,X_c,L') = \bigwedge_{l\in L} l' \Leftrightarrow f_l(X_u,X_c,L)$, where L' represents the next step states. $\mathsf{UPRE}(S)$ can then be computed symbolically by the formula

$$\mathsf{UPRE}(S) = \exists X_u.\ \forall X_c.\ \exists L' : T(L,X_u,X_c,L') \wedge S(L').$$

Alternatively, we observe that since we have that $\forall l' \in L' : l' \Leftrightarrow f_l(X_u,X_c,L)$ we can directly substitute all l' into $S(L')$ to obtain the desired set without using $T(L,X_u,X_c,L')$, i.e.

$$\mathsf{UPRE}(S) = \exists X_u.\ \forall X_c : S(L')[l' \leftarrow f_l(X_u,X_c,L)]_{l\in L}.$$

4 Abstractions of Safety Games

4.1 Conservative Abstractions

Computing the fixpoint of UPRE in safety games for circuits may be infeasible due to their large state spaces. For such circuits, we consider *abstractions*, which are games with smaller state spaces on which fixpoint computations are feasible. We follow the *abstract interpretation* framework [10] to build *conservative abstractions*, so as to make sure that if the abstract game can be won by controller, then the concrete game can also be won by her.

Let C be a circuit specification and $G_C = \langle Q, q_I, \Sigma_u, \Sigma_c, \delta, \mathscr{U}\rangle$ its associated safety game. Intuitively, abstractions will be obtained by partitioning the state space of G_C and defining transitions between the elements of the partition. Formally, a game $G_C^a = \langle Q^a, q_I^a, \Sigma_u, \Sigma_c, \Delta^a, \mathscr{U}^a\rangle$ is a *conservative abstraction of G_C* if

- Q^a is a partition of Q;
- $q_I^a = \{q_I\} \in Q^a$;
- $(s,\sigma_u,\sigma_c,s') \in \Delta^a$ if $\exists q \in s.\ \exists q' \in s' : \delta(q,\sigma_u,\sigma_c) = q'$; and
- $\mathscr{U} \subseteq \bigcup_{u\in\mathscr{U}^a} u$.

Notice that we require the abstractions to distinguish the initial states, and the abstract safety specification $\mathscr{U}^a$ to cover $\mathscr{U}$. Conservative abstractions give more power to environment [17]. We will show that if controller wins in a conservative abstraction, then it wins in the original game. We will refer to the states of G_C as *concrete states*, and those of G^a as *abstract states*.

We define the *concretization function* $\gamma : \mathscr{P}(Q^a) \to \mathscr{P}(Q)$ for this abstraction, defined by $\gamma(S^a) = \bigcup_{s^a \in S^a} s^a$, which gives the set of concrete states covered by a given set of abstract states. The dual operation is abstraction; we define two *abstraction functions* $\overline{\alpha}, \underline{\alpha} : \mathscr{P}(Q) \to \mathscr{P}(Q^a)$ as follows: $\overline{\alpha}(S) = \{q^a \in Q^a \mid S \cap q^a \neq \emptyset\}$, and $\underline{\alpha}(S) = \{q^a \in Q^a \mid S \subseteq q^a\}$. Intuitively, $\overline{\alpha}(S)$ is the smallest set of abstract states covering S; while $\underline{\alpha}(S)$ is the largest set of abstract states entirely included in S. The pair $(\overline{\alpha}, \gamma)$ defines a Galois connection:

Lemma 1 (from [10])**.** *The pair $(\overline{\alpha}, \gamma)$ is a Galois connection, that is, for all $s \subseteq Q$ and $t \subseteq Q^a$, we have that $\overline{\alpha}(s) \subseteq t$ if, and only if $s \subseteq \gamma(t)$.*

The following lemma shows the relation between $\overline{\alpha}$ and $\underline{\alpha}$, which are, respectively, over- and under-approximations of given sets.

Lemma 2. *For any $S \subseteq Q$, we have $\gamma(\underline{\alpha}(S)) \subseteq S \subseteq \gamma(\overline{\alpha}(S))$, and $\underline{\alpha}(S) = Q^a \setminus \overline{\alpha}(Q \setminus S)$.*

4.2 Predicate Abstraction and Localization Reduction

In order to effectively construct abstractions from the concrete circuit safety game we use *predicate abstraction* [15] and *localization reduction* [19]. Predicate abstraction consists in defining abstractions by partitioning the state space by predicates, and is used e.g. in CEGAR methods [5]. Localization reduction is a special case of predicate abstraction in which predicates consist of single latches.

Consider any circuit specification $C = \langle X_u, X_c, L, (f_l)_{l \in L}, f_{\mathsf{BAD}} \rangle$, and the associated safety game $G = \langle Q, q_I, \Sigma_u, \Sigma_c, \delta, \mathscr{U} \rangle$. Let P be a set of boolean variables, also called *predicates*, and $(f_p(L))_{p \in P}$ be a set of formulas. We assume that there exist $p_I, p_U \in P$ such that $f_{p_I} \equiv q_I$, and $f_{p_U} \equiv \mathscr{U}$. The predicates P partition the state space Q, i.e. $\mathbb{B}^L = \biguplus_{v \in \mathbb{B}^P} \bigcap_{p \in P} f_p^{-1}(v(p))$. We will consider the conservative abstraction defined on this partition.

Formally, we consider the state space $Q^a = \mathbb{B}^P$. Given $S^a \subseteq Q^a$, the concretization function is

$$\gamma(S^a)(L) = S^a(P)[p \leftarrow f_p(L)]_{p \in P}.$$

The abstraction functions are defined accordingly:

$$\begin{aligned} \overline{\alpha}(S)(P) &= \exists L : S(L) \wedge (\bigwedge_{p \in P} p \Leftrightarrow f_p(L)), \\ \underline{\alpha}(S)(P) &= \exists L : \neg(\neg S(L) \wedge (\bigwedge_{p \in P} p \Leftrightarrow f_p(L))). && \text{from Lemma 2} \end{aligned}$$

The transition relation Δ^a is given by $(q^a, \sigma_u, \sigma_c, r^a) \in \Delta^a \Leftrightarrow q^a, \sigma_u, \sigma_c, r^a \models T^a$, where

$$T^a(P, X_u, X_c, P') = \exists L, L' : T(L, X_u, X_c, L') \wedge (\bigwedge_{p \in P} p \Leftrightarrow (f_p(L)) \wedge (\bigwedge_{p' \in P'} p' \Leftrightarrow (f_{p'}(L')).$$

4.3 Abstract uncontrollable predecessors

We now define the uncontrollable predecessors operators in the abstract games. We define two operators, one yielding an over-approximation of the usual UPRE, and another one yielding an under-approximation. We let

$$\overline{\mathsf{UPRE}}_a(S^a) = \{q^a \mid \exists \sigma_u.\ \forall \sigma_c.\ \exists r^a : (q^a, \sigma_u, \sigma_c, r^a) \in \Delta^a \text{ and } r^a \in S^a\},$$
$$\underline{\mathsf{UPRE}}_a(S^a) = \{q^a \mid \exists \sigma_u.\ \forall \sigma_c.\ \forall r^a : (q^a, \sigma_u, \sigma_c, r^a) \in \Delta^a \text{ implies } r^a \in S^a\}.$$

Given a formula $S^a(P)$, representing a set of states from Q^a, the operators can be easily computed symbolically as follows:

$$\overline{\mathsf{UPRE}}_a(S^a) = \exists X_u.\ \forall X_c.\ \exists P' : T^a(P, X_u, X_c, P') \wedge S^a(P'),$$
$$\underline{\mathsf{UPRE}}_a(S^a) = \exists X_u.\ \forall X_c.\ \forall P' : T^a(P, X_u, X_c, P') \Rightarrow S^a(P').$$

The following lemma shows the relation between the abstract uncontrollable predecessor operator and the concrete one.

Lemma 3. *For any set $S^a \subseteq Q^a$, $\gamma(\underline{UPRE}_a(S^a)) \subseteq UPRE(\gamma(S^a)) \subseteq \gamma(\overline{UPRE}_a(S^a))$.*

Proof. We show the inequalities hold from left to right. Let $q^a \in \underline{\mathsf{UPRE}}_a(S^a)$. There is σ_u such that for all σ_c, for any state r^a, $(q^a, \sigma_u, \sigma_c, r^a) \in \Delta^a$ implies $r^a \in S^a$. So by construction of Δ^a, for any state q in $\gamma(q^a)$, for all σ_c, $\delta(q, \sigma_u, \sigma_c) \in \gamma(S^a)$. Hence $q \in \mathsf{UPRE}(\gamma(S^a))$ and $\gamma(q^a) \subseteq \mathsf{UPRE}(\gamma(S^a))$. Therefore $\gamma(\underline{\mathsf{UPRE}}_a(S^a)) \subseteq \mathsf{UPRE}(\gamma(S^a))$.

Let $q \in \mathsf{UPRE}(\gamma(S^a))$, then there is σ_u such that for all σ_c, $\delta(q, \sigma_u, \sigma_c) \in \gamma(S^a)$. By definition of Δ^a, for all σ_c $(\overline{\alpha}(q), \sigma_u, \sigma_c, \overline{\alpha}(\delta(q, \sigma_u, \sigma_c))) \in \Delta^a$. Since $\overline{\alpha}(\delta(q, \sigma_u, \sigma_c)) \in S^a$ for all σ_c, we have $\overline{\alpha}(q) \in \overline{\mathsf{UPRE}}_a(S^a)$ and $q \in \gamma(\overline{\mathsf{UPRE}}_a(S^a))$. Hence $\mathsf{UPRE}(\gamma(S^a)) \subseteq \gamma(\overline{\mathsf{UPRE}}_a(S^a))$.

□

Lemma 3 implies, by induction, the following.

Lemma 4. *$\gamma(\underline{UPRE}^*_a(\mathscr{U}^a)) \subseteq UPRE^*(\mathscr{U}) \subseteq \gamma(\overline{UPRE}^*_a(\mathscr{U}^a))$.*

This yields the following Theorem.

Theorem 1. *Let C be a circuit specification, G_C be its associated safety game and G_C^a a conservative abstraction of it. If controller wins G_C^a then she also wins G_C.*

4.4 Optimizations

Because we consider conservative abstractions, if environment wins in the abstract game, one cannot conclude unrealizability right away. However, we can still use the information gathered during the computation of the abstract uncontrollable predecessors $\overline{\mathsf{UPRE}}_a$. In fact, we will show that the states and actions that witness the uncontrollable predecessors in each iteration of $\overline{\mathsf{UPRE}}_a$ define a *set* of strategies which contains any concrete winning strategy for environment (Prop. 2 below). We then use this information to restrict future UPRE operations to these strategies.

Quasi-strategies of environment Formally, a *quasi-strategy of environment* in the conservative abstraction G^a of a game G is a function $\Lambda^{env} : Q^a \to \mathscr{P}(\Sigma_u)$ which maps any abstract state to a *set* of uncontrollable actions. Thus, a *quasi-strategy* can be seen as a non-deterministic strategy defined on a subset of states (in fact, Λ^{env} can map some states to the empty set). We denote by $\gamma(\Lambda^{env})$ the concrete quasi-strategy in G given by $\gamma(\Lambda^{env})(q) \mapsto \Lambda^{env}(\overline{\alpha}(q))$ for any $q \in Q$.

Let $S^a \subseteq Q^a$, and $W = \overline{\mathsf{UPRE}}^*_a(S^a)$. The set W describes a quasi-strategy Λ^{env}_W, defined by $\Lambda^{env}_W(q) \mapsto \{\sigma_u \in \Sigma_u \mid \forall \sigma_c.\ \exists s \in W : (q, \sigma_u, \sigma_c, s) \in \Delta^a\}$, for any $q \in Q^a$. This quasi-strategy corresponds to the set of uncontrollable actions environment can play from states in W to stay within W. Note that not all strategies respecting Λ_W^{env} are winning for environment; although all winning strategies for environment choose actions prescribed by Λ_W^{env}.

Proposition 2. *Let G^a be a conservative abstraction of a game G, Λ^{env} be the quasi-strategy for environment defined by $\overline{\mathsf{UPRE}}_a^*(\mathscr{U}^a)$ and λ^{env} a strategy for environment in G. If λ^{env} is a winning strategy for environment in G, then $\forall \pi \in \mathit{Plays}(G, \lambda^{env}).\ \forall i < i_\pi : \lambda^{env}(\pi[i]) \in \gamma(\Lambda^{env})(\pi[i])$.*

Proof. Let $S^a \subseteq Q^a$ and $\mathsf{St}_{env}^i(S^a) = \{(q, \sigma_u) \in Q^a \times \Sigma_u \mid \forall \Sigma_c.\ \exists r : (q, \sigma_u, \sigma_c) \in \Delta^a \text{ and } r \in \overline{\mathsf{UPRE}}_a^i(S^a)\}$. Clearly $\mathsf{St}_{env}^{i-1}(S^a) \sqsubseteq \Lambda^{env}$.

Let $\imath = \max\{i_\pi \mid \pi \in \mathsf{Plays}(G, \lambda^{env})\}$, which is finite since λ^{env} is memoryless and winning for environment.

Consider any play $\pi \in \mathsf{Plays}(G, \lambda^{env})$. We show that $\lambda^{env}(\pi[\imath - j]) \in \gamma(\mathsf{St}_{env}^j(\mathscr{U}^a))$.

By definition of $\imath$, for all $\sigma_c \in \Sigma_c$, $\delta(\psi[\imath-1], \lambda^{env}(\psi[\imath-1]), \sigma_c) \in \mathscr{U}$. By construction of Δ^a, we have that for all $\sigma_c \in \Sigma_c$ there exists r^a such that $(\overline{\alpha}(\psi[\imath-1]), \lambda^{env}(\psi[\imath-1]), \sigma_c, r^a) \in \Delta^a$ where $r^a \in \overline{\alpha}(\mathscr{U})$. This implies that $(\overline{\alpha}(\psi[\imath-1]), \lambda^{env}(\psi[\imath-1])) \in \mathsf{St}_{env}(\overline{\alpha}(\mathscr{U}))$. Note that from the definition of $\mathscr{U}^a$ we get that $\overline{\alpha}(\mathscr{U}) \subseteq \mathscr{U}^a$ and that since St_{env} is monotone $(\overline{\alpha}(\psi[\imath-1]), \lambda^{env}(\psi[\imath-1])) \in \mathsf{St}_{env}(\mathscr{U}^a)$. Thus $\lambda^{env}(\psi[\imath-1]) \in \gamma(\Lambda^{env})(\psi[\imath-1])$.

Consider now $2 \leq j \leq \imath$. For all $\sigma_c \in \Sigma_c$, $\delta(\psi[\imath-j], \lambda^{env}(\psi[\imath-j]), \sigma_c) \in \mathsf{UPRE}^{j-1}(\mathscr{U})$. It follows that for any $\sigma_c \in \Sigma_c$, there exists $r^a \in \overline{\alpha}(\mathsf{UPRE}^{j-1}(\mathscr{U}))$ with $(\overline{\alpha}(\psi[\imath-j]), \lambda^{env}(\psi[\imath-j]), \sigma_c, r^a) \in \Delta^a$. We have $r^a \in \overline{\alpha}(\mathsf{UPRE}^{j-1}(\gamma(\mathscr{U}^a)))$ by monotonicity of UPRE, and by Lemma 4, we get $r^a \in \overline{\mathsf{UPRE}}_a^{j-1}(\mathscr{U}^a)$. Hence, $(\overline{\alpha}(\psi[\imath-j]), \lambda^{env}(\psi[\imath-j])) \in \mathsf{St}_{env}(\overline{\mathsf{UPRE}}_a^{j-1}(\mathscr{U}^a)) \subseteq \mathsf{St}_{env}^j(\mathscr{U}^a)$. □

Guiding UPRE using Λ_W^{env} For convenience, let $\Lambda^{env} = \Lambda_W^{env}$. We define the concrete UPRE operator restricted to the quasi-strategy $\gamma(\Lambda^{env})$ as follows.

$$\mathsf{UPRE}_{\gamma(\Lambda^{env})}(S) = \{q \in Q \mid \exists \sigma_u \in \gamma(\Lambda^{env})(q).\ \forall \sigma_c \in \Sigma_c : \delta(q^a, \sigma_u, \sigma_c) \in S\}.$$

$\mathsf{UPRE}_{\gamma(\Lambda^{env})}(S)$ yields the set states from which environment can force to reach states in S by using actions compatible with the given quasi-strategy. Proposition 2 implies that because the quasi-strategy was extracted from the abstract uncontrollable predecessors fixpoint, this restriction is no loss of generality. Indeed, if environment has a winning strategy it is included in the quasi-strategy. It follows that if the abstract game is winning for controller, then this will be detected by UPRE restricted to $\gamma(\Lambda^{env})$.

Theorem 2. *Let G^a be a conservative abstraction of a game G, and Λ^{env} be the quasi-strategy for environment defined by $\overline{\mathsf{UPRE}}_a^*(\mathscr{U}^a)$. $q_I \notin \mathsf{UPRE}_{\gamma(\Lambda^{env})}^*(\mathscr{U})$ if and only if $q_I \notin \mathsf{UPRE}^*(\mathscr{U})$.*

Proof. Observe that since $\mathsf{UPRE}_{\gamma(\Lambda^{env})}$ is a restricted version of UPRE, we have that $\mathsf{UPRE}_{\gamma(\Lambda^{env})}^i(S) \subseteq \mathsf{UPRE}^i(S)$ for any $S \subseteq Q$ and all $i \geq 0$. Thus, if $q_I \notin \mathsf{UPRE}^*(\mathscr{U})$ then $q_I \notin \mathsf{UPRE}_{\gamma(\Lambda^{env})}^*(\mathscr{U})$.

For the other direction recall that from Proposition 1 we have that $q_I \in \mathsf{UPRE}^*(\mathscr{U})$ if and only if environment has a winning strategy in G. Assume $q_I \in \mathsf{UPRE}^*(\mathscr{U})$ and that λ^{env} is a winning strategy for environment in G. By Proposition 2 we get that

$$\bigcup_{\substack{\pi \in \mathsf{Plays}(G, \lambda^{env}) \\ 0 \leq j \leq i_\pi}} \pi[j] \subseteq \mathsf{UPRE}_{\gamma(\Lambda^{env})}^*(\mathscr{U}).$$

In particular, this implies that $q_I \in \mathsf{UPRE}_{\gamma(\Lambda^{env})}^*(\mathscr{U})$, which concludes the proof. □

Reachable states under Λ^{env}_W As a second optimization, we restrict the exploration of both the concrete and abstract state spaces to the set of states which are reachable from the initial state when environment plays according to a winning strategy. This will allow us to prune the search space. As we will show, the set of states that are winning for environment but not reachable from the initial state, or those states reached by strategies losing for environment can be safely ignored.

Let $\mathsf{post}(S,\lambda^{env})$ be the set of states reachable from $s \in Q$ if environment plays according to λ^{env}. We now formally define

$$\mathscr{R}(G) = \bigcup_{\substack{\lambda^{env} \text{ winning for } env \\ \pi \in \mathsf{Plays}(G,\lambda^{env}) \\ i \geq 0}} \pi[i].$$

Note that $\mathscr{R}(G)$ is empty if the circuit is controllable. We will omit G from $\mathscr{R}(G)$ when it is clear from the context. Ideally, we would like to restrict our computations to $\mathscr{R}(G)$. However, computing $\mathscr{R}(G)$ is clearly as difficult as solving realizability of the safety game G, so we will rather consider over-approximations of this set computed on conservative abstractions of G^a. For any $S^a \subseteq Q^a$, and Λ^{env} a quasi-strategy for *environment* in G^a, the *possible successors under* Λ^{env} are defined as follows.

$$\mathsf{post}(S^a,\Lambda^{env}) = \{r^a \in Q^a \mid \exists q^a \in S^a.\ \exists \sigma_u. \in \Lambda^{env}(q^a).\ \exists \sigma_c \in \Sigma_c : (q^a,\sigma_u,\sigma_c,r^a) \in \Delta^a\}.$$

Note that the post operator can be computed symbolically as follows.

$$\mathsf{post}(S^a,\Lambda^{env}) = \exists P.\ \exists X_u.\ \exists X_c : T^a(P,X_u,X_c,P') \wedge S^a(P) \wedge \Lambda^{env}(P,X_u).$$

Let G^a be a conservative abstraction of a game G and Λ^{env}_W the quasi-strategy defined by $\overline{\mathsf{UPRE}}^*_a(\mathscr{U}^a)$. Now, Prop. 2 implies the following result.

Proposition 3. *Let G^a be a conservative abstraction of a game G and Λ^{env}_W the quasi-strategy defined by $\overline{\mathit{UPRE}}^*_a(\mathscr{U}^a)$. Then $\mathscr{R}(G) \subseteq \gamma(\mathit{post}^*(q^a_I,\Lambda^{env}_W))$.*

Now, the first purpose of defining over-approximations of $\mathscr{R}(G)$ is to restrict the fixpoint computations on the abstract game to these states, so that the considered sets of states are smaller. This will, hopefully, lead to smaller BDDs. We define the $\overline{\mathsf{UPRE}}_a$ fixpoint computation restricted to over-approximations of $\mathscr{R}$.

Theorem 3. *Let G^a be a conservative abstraction of a safety game G, and let $R^a \subseteq Q^a$ with $\mathscr{R} \subseteq \gamma(R^a)$. Then $\gamma(\mu X.\ \mathscr{U}^a \cup \overline{\mathit{UPRE}}_a(X)) \cap \mathscr{R} = \gamma(\mu X.\ (\mathscr{U}^a \cup \overline{\mathit{UPRE}}_a(X)) \cap R^a) \cap \mathscr{R}$.*

The same idea can be applied to the post operator.

Theorem 4. *Let G^a be a conservative abstraction of a safety game G, and let $R^a \subseteq Q^a$ with $\mathscr{R} \subseteq \gamma(R^a)$. Then $\gamma(\mu X.\ \{q^a_I\} \cup \mathit{post}(X)) \cap \mathscr{R} = \gamma((\mu X.\ \{q^a_I\} \cup \mathit{post}(X,\Lambda^{env})) \cap R^a) \cap \mathscr{R}$.*

Using Abstract Partitioned Transition Relation As mentioned earlier, in some circuits, one can improve performance by using only a partitioned transition relation and avoiding the computation of the monolithic transition relation. In this paragraph, we explain how this can be achieved and combined with the reachability analysis in abstract games.

Note that partitioning the transition relation works well in instances in which the transition relation is large (i.e. the size of the BDD needed to represent T^a is large) but the fixpoint is reached in a small number of steps. On the contrary, if too many iterations are needed to obtain the fixpoint, then it is often more efficient to construct T^a once and use it to compute the operators $\overline{\mathsf{UPRE}}_a$ and $\underline{\mathsf{UPRE}}_a$, as the cost will be amortized in the long run. These observations are illustrated in the section on experiments.

Let $\psi_p(L,X_u,X_c) = f_p(L')[l' \leftarrow f_l(X_u,X_c,L)]_{l\in L}$. Then the $\overline{\mathsf{UPRE}}_a, \underline{\mathsf{UPRE}}_a$ operators can be computed as shown below.

Lemma 5. *For any G^a,*

$$\overline{UPRE}_a(S^a) = \exists X_u.\ \forall X_c : \overline{\alpha}(S^a(P'))[p' \leftarrow \psi_p(X_u,X_c,L)]_{p\in P})$$
$$\underline{UPRE}_a(S^a) = \neg(\forall X_u.\ \exists X_c : \overline{\alpha}(\neg S^a(P')[p' \leftarrow \psi_p(X_u,X_c,L)]_{p\in P})).$$

We also present an operator yielding an over-approximation of the set of reachable states which can be computed with partitioned transition relations. Let $S^a \in Q^a$ and Λ^{env} be a quasi-strategy for environment in G^a.

$$\overline{\mathsf{post}}(S^a,\Lambda^{env}) = \exists L,X_u : (S^a(P) \wedge \Lambda^{env}(P,X_u))[p \leftarrow f_p(L)]_{p\in P} \wedge \bigwedge_{p\in P} \exists X_c : p' \Leftrightarrow \psi_p(X_u,X_c,L).$$

Note that $\overline{\mathsf{post}}$ is defined, from post, simply pushing the quantification over X_c inside. In fact, the exact definition of post contains the transition relation T^a, which we want to avoid computing.

The following lemma shows that this yields over-approximations.

Lemma 6. *The set of abstract states reachable from S^a in one step if environment plays according to Λ^{env} is contained in $\overline{post}(S^a,\Lambda^{env})$. That is, $post(S^a,\Lambda^{env}) \subseteq \overline{post}(S^a,\Lambda^{env})$.*

Note that one could also push the quantification over X_u inside the conjunction in order to obtain coarser over-approximations. However, this alternative definition was not faster to compute, nor did it improve overall performance in our experiments.

5 Yet another CEGAR algorithm

We present a CEGAR-based synthesis algorithm, given in Algorithm 1, based on a combination of ideas introduced in [1] and [16]. The algorithm constructs abstractions using – initially – three predicates, namely, p_I describing the initial state, p_U an under-approximation of the losing states, and p_R an over-approximation of the states reachable from the initial state under winning strategies of environment. The algorithm further refines the abstraction by localization reduction. In fact, the initial abstraction consists of the conservative abstraction defined by these three predicates, and at each refinement loop, some latch is made "visible", that is, added as a predicate.

We give an informal description of the algorithm. Given a conservative abstraction, the algorithm first computes W_u at line 1, the fixpoint of $\underline{\mathsf{UPRE}}_a$, restricted to R^a which overapproximates $\mathscr{R}(G)$. If the initial state belongs to W_u, then by Lemma 4, controller has no winning strategy (line 3). Otherwise, in the while loop of line 7, we compute W_o, the fixpoint for $\overline{\mathsf{UPRE}}_a$ restricted to R^a – which is an over-approximation on reachable states under winning strategies of environment. In this case, if the initial state does *not* belong to W_o, then nor does it belong to the fixpoint of UPRE and we conclude that the circuit is controllable. Otherwise, we recompute the fixpoint for $\overline{\mathsf{UPRE}}_a$ by decreasing R^a: we first compute, at line 13, the quasi-strategy for environment allowing her to stay inside W_o, then restrict R^a, at line 14, to states that are reachable under this quasi-strategy. These restrictions are justified since any winning strategy for environment is compatible with these (see Proposition 2). If we were not able to conclude, then the abstraction is too coarse and needs to be refined. At line 17, we compute the concrete UPRE of W_u restricted to the quasi-strategy and to R^a. If it turns out that W_u was already a fixpoint for UPRE, then we know that the circuit is controllable (line 19) since W_u does not contain the initial state. Otherwise, we refine the abstraction by making a latch visible, but also increasing $\mathscr{U}^a$ using the information computed with $\underline{\mathsf{UPRE}}_a$. The refinement step is given by the `refine` function described in Algorithm 2.

The algorithm is initially called with the abstraction given by the three predicates $\{p_I, p_U, p_R\}$ defined by $p_I \equiv \{q_I\}$, $p_U \equiv \mathscr{U}$, and $p_R \equiv Q$.

```
Algorithm 1: abs_synth($G, G^a, R^a$)
Data: Safety game $G = \langle Q, q_I, \Sigma_u, \Sigma_c, \delta, \mathscr{U} \rangle$, abstraction $G^a = \langle Q^a, q_I^a, \Sigma_u, \Sigma_c, \Delta^a, \mathscr{U}^a \rangle$ and $R^a \supseteq \mathscr{R}$.
$W_u := \mu X.\ (\mathscr{U}^a \cup \underline{\mathsf{UPRE}}_a(X)) \cap R^a$;
if $q_I^a \in W_u$ then
    return not controllable;
end
$prev := \emptyset$;
while $R^a \neq prev$ do
    $prev := R^a$;
    $W_o := \mu X.\ (W_u \cup \overline{\mathsf{UPRE}}_a(X)) \cap R^a$;
    if $q_I^a \notin W_o$ then
        return controllable;
    end
    $\Lambda^{env}$ := quasi-strategy defined by $(W_o)$;
    $R^a := \mu X.\ (q_I^a \cup \mathsf{post}(X, \Lambda^{env})) \cap R^a$;
end
$W_u' := (\mathsf{UPRE}_{\gamma(\Lambda^{env})}(\gamma(W_u))) \cap \gamma(R^a)$;
if $W_u' \subseteq \gamma(W_u)$ then
    return controllable;
end
$Q_2^a :=$ refine($Q^a, W_u' \cup \gamma(W_u), \gamma(R^a)$);    // $\underline{\alpha}_2, \overline{\alpha}_2$ are the associated abstraction operators
$\mathscr{U}_2^a := \underline{\alpha}_2(W_u' \cup \gamma(W_u))$;
return abs_synth($G, G_2^a, \overline{\alpha}_2(\gamma(R^a))$);
```

Refinement is achieved symbollicaly by adding a new predicate to our predicate set P. Besides having $W_u' \cup \gamma(W_u)$ and $\gamma(R^a)$ replace the previous p_U and p_R, respectively, we also make a new latch "visible". Latches that depend on the value of other visible latches are given priority by Algorithm 2.

```
Algorithm 2: refine($P, U(L), R(L)$)
Data: Predicate set $P = \{p_I, p_U, p_R, l_{\alpha_1}, \dots, l_{\alpha_m}\}$, and sets $U'(L)$ and $R'(L)$
$P' := P \setminus \{p_U, p_R, p_I\}$;
$interesting := \{m \in L \setminus P' \mid m \not\Rightarrow U$ and $\neg m \not\Rightarrow U\}$;
$useful := \{m \in interesting \mid \mathsf{supp}(f_m) \cap P' \neq \emptyset\}$;
if $useful \neq \emptyset$ then
    $e$ := an element from $useful$;
else
    $e$ := an element from $interesting$;
end
return $P' \cup \{e, U'(L), R'(L), p_I\}$;
```

Theorem 5. *Let G be a safety game, G^a a conservative abstraction of it and $R^a \supseteq \mathsf{post}^*(q_I^a, \Lambda^{env})$ where Λ^{env} is the quasi-strategy defined by $\overline{\mathsf{UPRE}}_a^*(\mathscr{U}^a)$. If Algorithm 1 returns* controllable *for* (G, G^a, R^a)

then controller has a winning strategy in G; if it returns not controllable *then environment has a winning strategy in G. Moreover, the algorithms always terminates.*

To prove the correctness of the algorithm, we first show the following invariants.

Lemma 7. *The following invariants hold:*

$$\mathcal{R} \subseteq \gamma(R^a), \tag{1}$$
$$\mathcal{U} \cap \mathcal{R} \subseteq \gamma(\mathcal{U}^a) \subseteq \mathit{UPRE}^*(\mathcal{U}), \tag{2}$$
$$\mathcal{U} \cap \mathcal{R} \subseteq \gamma(W_u) \subseteq \mathit{UPRE}^*(\mathcal{U}), \tag{3}$$
$$\mathcal{R} \subseteq \gamma(W_o). \tag{4}$$

Proof. We prove these invariants by induction on the number of recursive calls. Initially, $R^a \equiv Q^a$ which satisfies (1) by Proposition 2, and (2) is satisfied since $\gamma(\mathcal{U}^a) = \mathcal{U}$. Consider any recursive call of the algorithm, and assume that (1) and (2) hold at line 1.

W_u is defined at line 1. Let us show that $\mathcal{U} \cap \mathcal{R} \subseteq \gamma(W_u)$. This holds at any iteration of the fixpoint defining W_u. In fact, we have $\mathcal{U} \cap \mathcal{R} \subseteq \gamma(\mathcal{U}^a)$, and $\mathcal{R} \subseteq \gamma(R^a)$, so any iterate contains $\mathcal{U}^a \cap R^a$. The result follows since $\mathcal{U} \cap \mathcal{R} \subseteq \gamma(\mathcal{U}^a) \cap \gamma(R^a) \subseteq \gamma(\mathcal{U}^a \cap R^a)$. To show the right hand side inequality, it suffices to observe that $\gamma(\mu X.(\mathcal{U}^a \cup \underline{\mathsf{UPRE}}_a(X))) \subseteq \mathsf{UPRE}^*(\mathcal{U})$, which holds since $\gamma(\mathcal{U}^a) \subseteq \mathsf{UPRE}^*(\mathcal{U})$. The inequality then follows by monotonicity.

Now we analyze the while loop of line 7 to prove (4) and (1) hold. Let us define $W'_o = \mu X.(W_u \cup \overline{\mathsf{UPRE}}_a(X))$. Note that we just showed $\mathcal{U} \cap \mathcal{R} \subseteq \gamma(W_u)$ so $\mathsf{UPRE}^*(\mathcal{U} \cap \mathcal{R}) \subseteq \gamma(\overline{\mathsf{UPRE}}^*_a(\mathcal{U} \cap \mathcal{R})) \subseteq \gamma(W'_o)$. But $\mathcal{R} \subseteq \mathsf{UPRE}^*(\mathcal{U} \cap \mathcal{R})$ by the definition of $\mathcal{R}$. Moreover, by Theorem 3, $\gamma(W_o) \cap \mathcal{R} = \gamma(W'_o) \cap \mathcal{R}$. It follows that $\mathcal{R} \subseteq \gamma(W_o)$. We proved the invariant for arbitrary R^a satisfying $\mathcal{R} \subseteq \gamma(R^a)$.

We now prove invariant (1) on this while loop. In fact, because $\mathcal{R} \subseteq \gamma(W_o)$, the strategy Λ^{env} defined on line 13 contains all winning strategies for environment, in the sense of Prop. 2. Now, if we denote $R' = \mu X.(q_I^a \cup \mathsf{post}(X, \Lambda^{env}))$, then $\mathcal{R} \subseteq \gamma(R')$ by Prop. 2. By Theorem 4, it follows that $\mathcal{R} \subseteq \gamma(R^a)$.

It remains to show that the invariants hold on the recursive call at line 23. Variable R^a is not modified, so we need to show (2), that is, $\mathcal{U} \cap \mathcal{R} \subseteq \gamma(\mathcal{U}_2^a) \subseteq \mathsf{UPRE}^*(\mathcal{U})$. By the definition of W'_u at line 17, we have that $W'_u \subseteq \mathsf{UPRE}_{\gamma(\Lambda^{env})}(\gamma(W_u))$, and since $\gamma(W_u) \subseteq \mathsf{UPRE}^*(\mathcal{U})$, we get that $W'_u \subseteq \mathsf{UPRE}^*(\mathcal{U})$. Thus, $W'_u \cup \gamma(W_u) \subseteq \mathsf{UPRE}^*(\mathcal{U})$, and $\gamma_2(\mathcal{U}_2^a) = \gamma_2(\underline{\alpha}_2(W'_i \cup \gamma_2(W_u))) \subseteq \gamma_2(\underline{\alpha}_2(\mathsf{UPRE}^*(\mathcal{U}))) \subseteq \mathsf{UPRE}^*(\mathcal{U})$. To show that other inclusion, it suffices to note that $\mathcal{U} \cap \mathcal{R} \subseteq \gamma_2(W_u) \subseteq \gamma_2(\underline{\alpha}_2(\gamma_2(W_u)) \subseteq \gamma_2(\mathcal{U}_2^a)$. □

Proof of Theorem 5. Assume that the algorithm answers not controllable, on line 3. By (3), we have $\gamma(W_u) \subseteq \mathsf{UPRE}^*(\mathcal{U})$ so $q_I^a \in W_u$ implies $\{q_I\} = \gamma(q_I^a) \subseteq \mathsf{UPRE}^*(\mathcal{U})$, which means that environment has a winning strategy.

Assume that the algorithm answers controllable on line 10. By (4), we have $\mathcal{R} \subseteq \gamma(W_o)$, so $q_I^a \notin W_o$ means that $q_I \notin \mathcal{R}$, so controller has a winning strategy.

Last, assume that the algorithm returns controllable on line 18. We have that $q_I \in \mathsf{UPRE}^*(\mathcal{U})$ iff $q_I \in \mathsf{UPRE}^*(\gamma(W_u))$ iff $q_I \in \mathsf{UPRE}^*_{\gamma(\Lambda^{env})}(\gamma(W_u))$ iff $q_I \in \mathsf{UPRE}^*_{\gamma(\Lambda^{env})}(\gamma(W_u)) \cap \mathcal{R}$. The test of line 17 means that $\gamma(W_u)$ is already a fixpoint of the latter equation. Moreover, we know that $q_I^a \notin W_u$ by line 2. It follows that $q_I \notin \mathsf{UPRE}^*(\mathcal{U})$, and the returned result is correct.

Now, termination follows from the fact that at each recursive call, a new latch is made visible, so at most after $|L|$ iterations, we obtain the concrete game. In this case, $\overline{\mathsf{UPRE}}_a = \underline{\mathsf{UPRE}}_a = \mathsf{UPRE}$, thus W_u and W_o are complementary inside R^a. So the algorithm will either output not controllable on line 3, or controllable on line 10. □

6 Strategy Synthesis

The first step of the strategy synthesis is to obtain the *winning region* for controller, that is, the set W of all winning states for controller. With the basic fixpoint algorithm – without abstractions, the algorithm computes $\mathsf{UPRE}^*(\mathscr{U})$ to decide that the circuit is controllable, so the complement of this set is the winning region. When Algorithm 1 determines the controllability of a given game, we compute a winning region as follows. We have, by Invariant (3), that $\gamma(W_u) \subseteq \mathsf{UPRE}^*(\mathscr{U})$, so $\gamma(W_u)^c$ is an over-approximation of the winning region. Then $\mathsf{CPRE}^*(\gamma(W_u)^c)$ gives the winning region, where $\mathsf{CPRE}^*(X) = \nu Y.(X \cap \mathsf{CPRE}(Y))$, and $\mathsf{CPRE}(X) = \{q \mid \forall \sigma_u \in \Sigma_u, \exists \sigma_c \in \Sigma_c, \delta(q, \sigma_u, \sigma_c) \in X\}$.

Let $\mathscr{S}$ denote such a winning region. As a first step, it is easy to derive a quasi-strategy for controller from $\mathscr{S}$: We define λ as $\lambda(q, \sigma_u) = \{\sigma_c \in \Sigma_c \mid \delta(q, \sigma_u, \sigma_c) \in \mathscr{S}\}$ for all $q \in \mathscr{S}$, and $\sigma_u \in \Sigma_u$, and arbitrarily on other states. We denote by $\mathscr{R}(\lambda)$ the set of states reachable from q_I when controller plays any strategy compatible with the quasi-strategy λ. It is clear that $\mathscr{R}(\lambda) \cap \mathscr{U} = \emptyset$. In other terms, any strategy compatible with λ is winning for controller from states $\mathscr{S}$.

We are interested in synthesizing a circuit implementing a winning strategy. However, the quasi-strategy we just constructed is non-deterministic, so it cannot be directly mapped as a circuit. We are going to extract a deterministic strategy from λ, and show how the implementing circuit can be produced.

Algorithm 3: `det_strat`$(\lambda(L, X_u, X_c), R(L))$

```
  Data: Winning quasi-strategy $\lambda(L, X_u, X_c)$, and set $\mathscr{R}(\lambda)$
  Result: A circuit for each $\sigma_c \in \Sigma_c$, implementing a strategy compatible with $\lambda$
1 for $x \in X_c$ do
2     $f(L, X_u, x) := \exists X_c \setminus \{x\} : \lambda(L, X_u, X_c)$;
3     $f_x(L, X_u) := f(L, X_u, x)[x \leftarrow 1]$;
4     $f_{\overline{x}}(L, X_u) := f(L, X_u, x)[x \leftarrow 0]$;
5     $care(L, X_u) := R(L) \wedge (\neg f_x(L, X_u) \vee \neg f_{\overline{x}}(L, X_u))$;
      /* could also be $(\neg f_{\overline{x}}(L, X_u)) \Downarrow \mathrm{care}(L, X_u)$        */
6     $g_x := f_x(L, X_u) \Downarrow \mathrm{care}(L, X_u)$;
7     $\lambda := \lambda \wedge (x \Leftrightarrow g_x(L, X_u))$;
8 end
9 return $(g_x)_{x \in X_c}$;
```

The idea of Algorithm 3 is to extract functions for each $x \in X_c$ incrementally, so that at the i-th iteration, the quasi-strategy yields a unique value for the first i controllable inputs. To extract deterministic strategies, we use the *restrict* operation implemented in most BDD packages (see [23]). Given two formulas $f(X)$ and $D(X)$, the *restriction of* $f(X)$ *to* $D(X)$ is defined by $(f \Downarrow D)(X)$, and has the following property.

Lemma 8 (from [9])**.** *For any two formulas* $f(X)$, $D(X)$, $(f \Downarrow D)(X)$ *is a set that agrees with* $f(X)$ *on the domain* $D(X)$. *In other terms,* $\forall X.D(X) \Rightarrow \big((f \Downarrow D)(X) \Leftrightarrow f(X)\big)$.

This operation is useful when one needs an *arbitrary* set which complies with $D(X)$ since the size of the BDD representing $(f \Downarrow D)(X)$ is guaranteed to be not larger than $f(X)$, and is often smaller.

We will use this operation to extract functions as follows. In Algorithm 3, given $x \in X_c$, we identify the set $care(L, X_u)$ on which the strategy being constructed yields a unique value for x given L, X_u, while we know that outside this set x could get any value. We then define the strategy for x on this set, and (arbitrarily) extend to the whole domain by the restrict operation. Note that the restrict operation is an optimization; we could instead simply set $g_x := f_x(L, X_u)$ on line 6.

Theorem 6. *Let G be a safety game, R a winning region for controller, and λ quasi-strategy of controller winning from R. Then the strategy λ' returned by Algorithm 3 is winning for controller.*

Proof. Let $x_1, x_2, \ldots$ be the ordered sequence of controllable inputs taken by the loop. Note that the function g_{x_i} is defined on iteration i. Let us denote by λ_0 the quasi-strategy given in input.

We show that the following invariant holds at the beginning of iteration $i \geq 1$:

$$\forall L, X_u, X_c. R(L) \wedge \lambda(L, X_u, X_c) \Rightarrow \lambda_0(L, X_u, X_c). \tag{5}$$

$$\forall L, X_u \exists X_c. \lambda(L, X_u, X_c). \tag{6}$$

$$\forall j = 1 \ldots i-1, \forall L, X_u : \neg\big((\exists X_c \setminus \{x_j\}. \lambda(L, X_u, X_c) \wedge x_j) \wedge (\exists X_c \setminus \{x_j\}. \lambda(L, X_u, X_c) \wedge \neg x_j)\big). \tag{7}$$

In words, the first invariant says that at all states in $R(L)$, the partial strategy computed so far is compatible with λ_0; and the second invariant says that λ is satisfiable given any L, X_u. This will ensure that λ is always a *winning* quasi-strategy. The third invariant states the functionality of λ for the first $i-1$ variables. In fact, it states that, given L, X_u, there is only one possible value of x_j that satisfies λ. Thus, at the end of the algorithm, these invariants will yield that λ is a function compatible with λ_0 which is what we want.

The claim holds trivially for $i = 1$. Consider $i \geq 2$. On lines 3 and 4, we define $f_x(L, X_u)$ (resp. $f_{\bar{x}}(L, X_u)$), the subset of L, X_u on which x_i can be set to true (resp. false) by λ. On line 5, the set *care* is defined as the set L, X_u where $R(L)$ holds, and x_j can only be set to either true or false. Intuitively, λ must be defined uniquely on this set, whereas it can be defined arbitrarily outside. In fact, outside $R(L)$ we do not care about λ since it does not matter for winning; and outside $(\neg f_x(L, X_u) \vee \neg f_{\bar{x}}(L, X_u))$, we know that x_j can take both values. On line 6, we define the function $g_x(L, X_u)$ by the restrict operator $\Downarrow$, which gives an arbitrary function compatible with $f_x(L, X_u)$ on the set $care(L, X_u)$. This means that x_i is set to 1 when $R(L) \wedge f_x(L, X_u)$ holds, and to 0 when $R(L) \wedge f_{\bar{x}}(L, X_u)$ holds. It follows that, by construction, the updated λ is still compatible with λ_0. Moreover, since $g_x(L, X_u)$ is a function, λ is also functional on variables $x_1, \ldots, x_i$. □

Finally, we present a possible further optimization. One could execute the algorithm once, recover the new strategy λ' and re-run the algorithm with $R \equiv \mathscr{R}(\lambda')$, which is clearly a winning region of controller. This would still guarantee the invariants hold and is thus sound.

7 Experimental results

We evaluated four different algorithms: (C) the classical fixpoint computation with a precomputed transition relation; (C-TL) the classical fixpoint computation using the partitioned transition relation; (A) the algorithm 1 with a precomputed abstract transition relation; (A-TL) the algorithm 1 using abstract operators implemented to avoid using a transition relation (this implies $\overline{\mathsf{post}}$ was used instead of the exact post operator). The benchmarks that we used for the evaluation are provided for the SyntComp (Synthesis Competition)

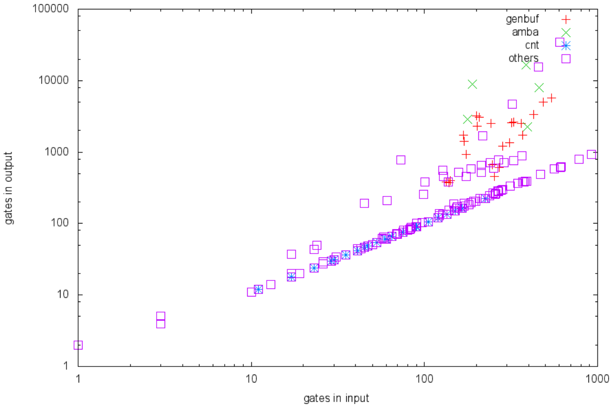

Figure 3: Size of the synthesized strategy.

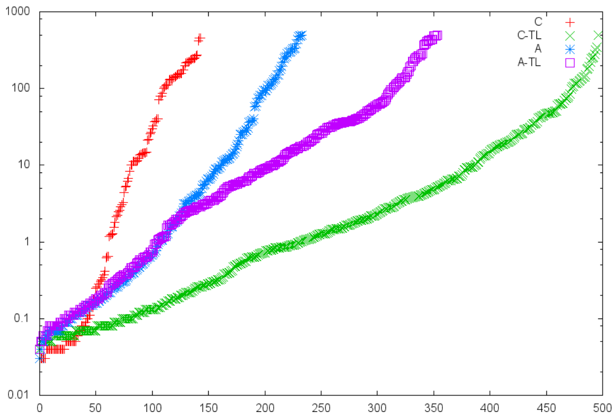

Figure 1: Time (in seconds) to check realizability.

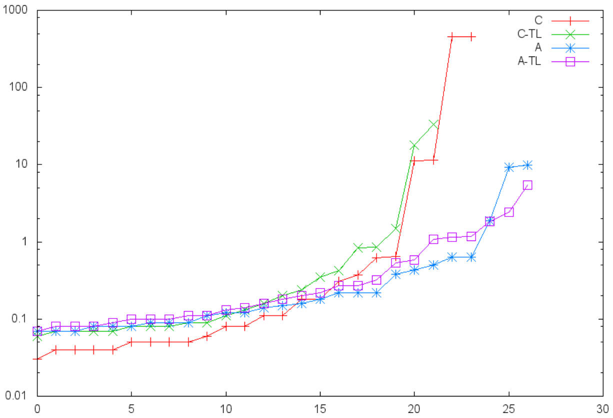

Figure 2: Time (in seconds) for *cnt* benchmarks.

`https://syntcompdb.iaik.tugraz.at/`. At the time of submission of this work, there were 432 benchmarks provided by the organizers of the competition. We have submitted 102 additional benchmarks.

Figure 1 summarizes performances of the algorithms on all our benchmarks. The horizontal axis is the number of instances that can be solved within the time limit given by the vertical axis. In general C-TL performs better, however the algorithms that use abstraction perform better on some examples. This is in particular the case for the “cnt” benchmark, as can be seen in Figure 2. In these benchmarks, there is a counter (its size depends on the benchmark), the adversary can increment it and the controller should reset it at the right moment. The set of reachable states is enormous but the winning strategy is quite simple which may explain why abstraction works better. The abstract algorithms were able to solve more of these examples within the time limit of 500s that we fixed.

In Figure 3 we compared the size of the synthesized circuits with the size of the input circuits for the different sets of benchmarks. Most of the time, our method allows to find solutions that are not too big when compared to the input circuit.

It is worth mentioning that from the 534 benchmarks considered, we were able to determine realizability in under 500s for all but less than 30 of them. Amongst these, 369 are known to be realizable. We were able to synthesize a circuit, again in under 500s, for all but less than 35.

8 Acknowledgements

We thank Robert Könighofer for providing us their implementation of the classic fixpoint computation algorithm as well as a benchmarking framework for it. This implementation [2] was the starting point for our tool.

References

[1] Luca de Alfaro & Pritam Roy (2010): *Solving games via three-valued abstraction refinement*. Information and Computation 208(6), pp. 666–676, doi:10.1016/j.ic.2009.05.007.

[2] Roderick Bloem, Robert Könighofer & Martina Seidl (2014): *SAT-Based Synthesis Methods for Safety Specs*. In: *VMCAI*, *LNCS* 8318, Springer, pp. 1–20, doi:10.1007/978-3-642-54013-4_1.

[3] Randal E. Bryant (1986): *Graph-based algorithms for boolean function manipulation*. Computers, IEEE Transactions on 100(8), pp. 677–691, doi:10.1109/TC.1986.1676819.

[4] Jerry R. Burch, Edmund M. Clarke & David E. Long (1991): *Symbolic Model Checking with Partitioned Transistion Relations*. In: VLSI, pp. 49–58.

[5] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu & Helmut Veith (2000): *Counterexample-guided abstraction refinement*. In: CAV, LNCS 1855, Springer, pp. 154–169, doi:10.1007/10722167_15.

[6] Edmund Clarke, Orna Grumberg, Muralidhar Talupur & Dong Wang (2003): *High level verification of control intensive systems using predicate abstraction*. In: MEMOCODE, IEEE, pp. 55–64, doi:10.1109/MEMCOD.2003.1210089.

[7] Olivier Coudert, Christian Berthet & Jean Christophe Madre (1990): *Verification of synchronous sequential machines based on symbolic execution*. In: Automatic verification methods for finite state systems, LNCS 407, Springer, pp. 365–373, doi:10.1007/3-540-52148-8_30.

[8] Olivier Coudert & Jean Christophe Madre (1990): *A Unified Framework for the Formal Verification of Sequential Circuits*. In: ICCAD, pp. 126–129.

[9] Olivier Coudert, Jean Christophe Madre & Christian Berthet (1991): *Verifying temporal properties of sequential machines without building their state diagrams*. In: CAV, LNCS 531, Springer, pp. 23–32, doi:10.1007/BFb0023716.

[10] Patrick Cousot & Radhia Cousot (1977): *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*. In: POPL, ACM, pp. 238–252, doi:10.1145/512950.512973.

[11] Rüdiger Ehlers (2010): *Symbolic Bounded Synthesis*. In: CAV, LNCS 6174, Springer, pp. 365–379, doi:10.1007/s10703-011-0137-x.

[12] E. Allen Emerson & Charanjit S. Jutla (1991): *Tree automata, mu-calculus and determinacy*. In: FOCS, IEEE, pp. 368–377, doi:10.1109/SFCS.1991.185392.

[13] Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2009): *An Antichain Algorithm for LTL Realizability*. In: CAV, LNCS 5643, Springer, pp. 263–277, doi:10.1007/978-3-642-02658-4_22.

[14] Erich Grädel (2004): *Positional Determinacy of Infinite Games*. In: STACS, LNCS 2996, Springer, pp. 4–18, doi:10.1007/978-3-540-24749-4_2.

[15] Susanne Graf & Hassen Saïdi (1997): *Construction of abstract state graphs with PVS*. In: CAV, LNCS 1254, Springer, pp. 72–83, doi:10.1007/3-540-63166-6_10.

[16] Thomas A. Henzinger, Ranjit Jhala & Rupak Majumdar (2003): *Counterexample-guided control*. In: ICALP, LNCS 2719, Springer, pp. 886–902, doi:10.1007/3-540-45061-0_69.

[17] Thomas A. Henzinger, Rupak Majumdar, Freddy Y. C. Mang & Jean-François Raskin (2000): *Abstract Interpretation of Game Properties*. In: SAS, pp. 220–239, doi:10.1007/978-3-540-45099-3_12.

[18] Youpyo Hong, Peter A Beerel, Jerry R Burch & Kenneth L McMillan (1997): *Safe BDD minimization using don't cares*. In: Proceedings of the 34th annual Design Automation Conference, ACM, pp. 208–213, doi:10.1145/266021.266068.

[19] Robert P. Kurshan (1994): *Automata-theoretic verification of coordinating processes*. In: 11th International Conference on Analysis and Optimization of Systems Discrete Event Systems, Springer, pp. 16–28, doi:10.1007/BFb0033528.

[20] Edward J. McCluskey (1956): *Minimization of Boolean Functions**. Bell system technical Journal 35(6), pp. 1417–1444, doi:10.1002/j.1538-7305.1956.tb03835.x.

[21] Nina Narodytska, Alexander Legg, Fahiem Bacchus, Leonid Ryzhyk & Adam Walker (2014): *Solving Games without Controllable Predecessor*. In: CAV, Springer.

[22] Christos H. Papadimitriou & Mihalis Yannakakis (1986): *A note on succinct representations of graphs*. Information and Control 71(3), pp. 181 – 185, doi:10.1016/S0019-9958(86)80009-2.

[23] Martin Sauerhoff & Ingo Wegener (1996): *On the complexity of minimizing the OBDD size for incompletely specified functions*. IEEE Trans. on CAD of Integrated Circuits and Systems 15(11), pp. 1435–1437, doi:10.1109/43.543775.

[24] Saqib Sohail & Fabio Somenzi (2009): *Safety first: A two-stage algorithm for LTL games*. FMCAD, pp. 77–84, doi:10.1007/s10009-012-0224-3.

[25] Fabio Somenzi (1999): *Binary Decision Diagrams*. In: Calculational system design, 173, IOS Press, p. 303.

[26] Alfred Tarski et al. (1955): *A lattice-theoretical fixpoint theorem and its applications*. Pacific journal of Mathematics 5(2), pp. 285–309, doi:10.2140/pjm.1955.5.285.

[27] Wolfgang Thomas (1995): *On the synthesis of strategies in infinite games*. In: STACS, Springer, pp. 1–13, doi:10.1007/3-540-59042-0_57.