AbsSynthe: an abstract synthesis tool

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

***Abstract*—We describe a synthesis algorithm for safety specifications given as circuits. Our algorithm is based on fixpoint computations, abstraction and refinement. It uses binary decision diagrams as symbolic data structure.**

I. Introduction

a) Input of the Algorithm: AbsSynthe takes as input a safety specification described by a file in the *extended AIGER format*. The original AIGER format [1] is a safety property specification used for model checking purposes. The extended version of it has been recently introduced by Jacobs [2] as a standard way of representing safety objectives for controller synthesis.

b) Notation: 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.

c) The specifcation: We are interested in synthesizing controllers for synchronous sequential circuits given with safety specifications, 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 configurations.

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

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 configuration.

II. Concrete algorithm

We discuss here the main steps of the *plain fixpoint* algorithm.

The classic approach is to build a boolean function representing the full transition relation, i.e. a function $T(L, X_u, X_c, L')$ where L' is a primed copy of the latches representing the next step value of them. However, to be efficient, the explicit construction of the BDDs for the transition relation needs to be avoided [3], our solution is to use substitution of variables with BDDs as in [4] to directly compute the effect of the transition relation backwardly. To achieve this we use the `compose` procedure available in most BDD packages (see, e.g. [5]).

Substitution can be formalized as follows. 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, $f[y \leftarrow g_y]_{y \in Y}(X \setminus Y, Z) = \exists Y.\ f(X) \wedge \Big(\bigwedge_{y \in Y} y \Leftrightarrow g_y(Z)\Big)$.

Thus, our version of the "uncontrollable predecessors" operator is given by

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

where S is a boolean formula over L. Let $\mathcal{U}(L)$ be a boolean formula which holds if and only if an error configuration is reached. The set of latch configurations from which controller can ensure the safety property holds corresponds to the complement of the fixpoint of UPRE on $\mathcal{U}$ (see, e.g. [6]).

III. Abstract algorithm

Let P be a set of boolean variables, which we will call *predicates*. Each $p \in P$ has an associated boolean formula $f_p(L)$. P contains a predicate p_I which holds if and only if all latches are set to 0 and a predicate p_U which corresponds to $\mathcal{U}$ (see previous section).

We define the *concretization function* $\gamma : \mathcal{P}(Q^a) \to \mathcal{P}(Q)$ as

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

The dual operation is abstraction; we define two *abstraction functions* $\overline{\alpha}, \underline{\alpha} : \mathcal{P}(Q) \to \mathcal{P}(Q^a)$ as follows:

$$\overline{\alpha}(S)(P) = \exists L : S(L) \wedge \Big(\bigwedge_{p \in P} p \Leftrightarrow f_p(L)\Big),$$

$$\underline{\alpha}(S) = \exists L : \neg(\neg S(L) \wedge (\bigwedge_{p \in P} p \Leftrightarrow f_p(L))).$$

We define over and under approximations of UPRE which operate on valuations of P. 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 $\overline{\mathsf{UPRE}}_a(S^a), \underline{\mathsf{UPRE}}_a(S^a)$ operators can be computed as shown below.

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

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

where S^a is a boolean formula over P.

We also need to define an operator which yields, given a boolean formula S^a over P and a boolean formula $\Lambda^{env}(P, X_u)$, an over-approximation of the configurations that can be reached. Formally, we define $\overline{\mathsf{post}}(S^a, \Lambda^{env})$ as

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

A. Algorithm

If the fixpoint of the under-approximated operator contains the initial configuration, we are certain that the specification is unrealizable. Conversely, if the fixpoint of the over-approximated operator does not contain the initial configuration, we know that the specificaiton is realizable. Since the operators are not exact, the converse of the above assertions does not hold. In this case we add aditional predicates (in fact, we apply *localization reduction* [7]) to our model in order to have a finer abstraction.

We now outline the main stages of the algorithm.

1) Add to $P = \{p_I, p_U, p_R\}$, where $p_R \equiv 1$, some additional predicates each of which hold if and only if a latch holds.
2) Denote by W_u the fixpoint of $\underline{\mathsf{UPRE}}_a$ on p_U. If W_u contains p_I, then the specification is unrealizable.
3) Repeat the following until p_R is not updated:
   a) Denote by W_o the fixpoint of $\overline{\mathsf{UPRE}}_a$ on W_u doing conjunction of each intermediary BDD with p_R. If W_o does not contain p_I, then the specification is realizable;
   b) otherwise, we extract a boolean formula $\Lambda^{env}(P, X_u)$, which corresponds to a relation that chooses X_u so that p_U can be forced from p_I. Compute the fixpoint of $\overline{\mathsf{post}}$ on it, and set it to be $p_R \in P$.
4) Add more single latch predicates to P and go back to the second step.

The inner loop corresponds to the exhaustion of the information one can gather from a fixed set of predicates. Even if the initial configuration is indeed contained in the fixpoint of $\overline{\mathsf{UPRE}}_a$, one can then over-approximate the configurations that are *reachable if environment can force error configurations*. This reduces the set of configurations to be considered.

IV. Optional optimizations

We have implemented a version of the classic fixpoint computation which receives as input some reachability information and uses it to minimize the size of the BDDs handled. More specifically, we choose k different subsets of L which we use as predicates and compute the fixpoint of $\overline{\mathsf{post}}$ on p_I for it. This yields k over-approximations of the set of reachable configurations of the system. After each UPRE operation we `restrict` the result with each one of these over-approximations. The `restrict` procedure is a pervasive one in BDD packages. It is used to minimize the size of a BDD using an additional "don't care" BDD (see, e.g. [8]). We assume that `restrict` is "safe", that is, if the operation increases the size of the BDD then the result is dropped in favor of the original BDD.

V. A word on synthesis

Denote by $\mathsf{UPRE}^*(\mathcal{U})$ the fixpoint of UPRE on $\mathcal{U}$. We have $\gamma(W_u) \subseteq \mathsf{UPRE}^*(\mathcal{U})$, so $\gamma(W_u)^c$ is an over-approximation of desired set. Then $\mathsf{UPRE}^*(\gamma(W_u))^c$ gives the set of configurations from which controller can ensure the property holds.

As a final optimization, we describe how we use `restrict` in the final phase of the synthesis algorithm to minimize the synthesized circuit size.

After extracting the desired circuit, which we describe by formulae $f_c(L, X_u)$ for all $c \in X_c$, one can launch a last reachability analysis. Namely, we compute the reachable configurations of the latches when allowing only transitions that agree with $\bigwedge_{c \in X_c} c' \Leftrightarrow f_c(L, X_u)$. The resulting set of valuations of L can be used to `restrict` all $f_c(L, X_u)$ formulae.

References

[1] A. Biere. Aiger format and toolbox. [Online]. Available: http://fmv.jku.at/aiger/

[2] S. Jacobs. (2014, February) Extended aiger format for synthesis (v0.1). [Online]. Available: http://www.syntcomp.org/wp-content/uploads/2014/02/Format.pdf

[3] J. Burch, E. M. Clarke, and D. Long, "Symbolic model checking with partitioned transition relations," *Computer Science Department*, p. 435, 1991.

[4] O. Coudert and J. C. Madre, "A unified framework for the formal verification of sequential circuits," in *ICCAD*, 1990, pp. 126–129.

[5] F. Somenzi, "Binary decision diagtams," *Calculational system design*, vol. 173, p. 303, 1999.

[6] W. Thomas, "On the synthesis of strategies in infinite games," in *STACS 95*. Springer, 1995, pp. 1–13.

[7] R. P. Kurshan, "Automata-theoretic verification of coordinating processes," in *11th International Conference on Analysis and Optimization of Systems Discrete Event Systems*. Springer, 1994, pp. 16–28.

[8] Y. Hong, P. A. Beerel, J. R. Burch, and K. L. McMillan, "Safe bdd minimization using don't cares," in *Proceedings of the 34th annual Design Automation Conference*. ACM, 1997, pp. 208–213.