Untitled CAV Paper

Alexander Legg[1], Leonid Ryzhyk[2], and Nina Narodytska[2]

[1] NICTA* and UNSW
alexander.legg@nicta.com.au
[2] Samsung Research

Abstract. This is the abstract

1 Introduction

Reactive systems are ubiquitous in real-world problems such as circuit design, industrial automation, or device drivers. Automatic synthesis can provide a *correct by construction* controller for a reactive system from a specification. However, the reactive synthesis problem is 2EXPTIME-complete so naive algorithms are infeasible on even simple systems.

Reactive synthesis is formalised as a game between the *controller* and its *environment*. In this work we focus on safety games, in which the controller must prevent the environment from forcing the game into an error state. Much of the complexity of reactive synthesis stems from tracking the set of states in which a player is winning.

There are several techniques that aim to mitigate this complexity by representing states symbolically. Historically the most successful technique has been to use *Binary Decision Diagrams* (BDDs). BDDs efficiently represent a relation on a set of game variables but in the worst case the representation may be exponential. This means that BDDs are not a one-size-fits-all solution for all reactive synthesis specifications.

Advances in SAT solving technology has prompted research into its applicability to synthesis as an alternative to BDDs. One approach is to find sets of states in CNF [**?**]. Another approach is to eschew states and focus on *runs* of the game. Previous work has applied this idea to realizability of bounded games [3] by forming abstract representations of the game. In this paper, we extend this idea to unbounded games by constructing approximate sets of winning states from abstract trees.

2 Reactive Synthesis

A *safety game*, $G = \langle X, L_c, L_u, \delta, I, E, \rangle$, consists of a set of boolean state variables, sets of controllable and uncontrollable label variables, a transition relationship $\delta : (X, L_c, L_u) \rightarrow X$, an initial state, and an error state. The *controller* and

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

environment players choose controllable and uncontrollable labels respectively and the game proceeds according to δ.

An *run* of a game $(x_0, c_0, u_0), (x_1, c_1, u_1) \dots (x_n, c_n, u_n)$ is a chain of state and label pairs of length n s.t. $x_{k+1} \leftarrow \delta(x_k, c_k, u_k)$. A run is winning for the controller if $x_0 = I \wedge \forall i \in \{1..n\}(x_i \neq E)$. In a bounded game of rank n all runs are restricted to length n, whereas unbounded games consider runs of infinite length.

A *controller strategy* $\pi^c : X \rightarrow L_c$ is a mapping of states to controllable labels. A controller strategy is winning in a bounded game of rank n if all runs $(x_0, \pi^c(x_0), u_0), (x_1, \pi^c(x_1), u_1) \dots (x_n, \pi^c(x_n), u_n)$ are winning. Bounded *realizability* is the problem of determining the existence of such a strategy for a bounded game.

An *environment strategy* $\pi^e : (X, L_c) \rightarrow L_u$ is a mapping of states and controllable labels to uncontrollable labels. A bounded run is winning for the environment if $x_0 = I \wedge \exists i \in \{1..n\}(x_i = E)$ and an environment strategy is winning for a bounded game if there exists a run $(x_0, c_1, \pi^e(x_1, c_1)), (x_1, c_1, \pi^e(x_1, c_1)) \dots (x_n, c_n, \pi^e(x_n, c_n))$ that wins for the environment. Safety games are zero sum, therefore the existence of a controller strategy implies the nonexistence of an environment strategy and vice versa.

2.1 Abstract Game Trees

A set of runs can be symbolically represented by a tree of labels. Each edge of the tree is either a *fixed* valuation of controllable or uncontrollable variables, or it is *unfixed* denoting any values are possible. Also, the tree must alternate between controllable and uncontrollable edges to reflect the alternating turns of the game. The entire set of runs of a bounded game of rank n is symbolically represented by a tree of depth $2n$ and width 1 populated by unfixed edges. Reducing the set of runs in a game forms an abstract game, which can be represented symbolically by an *abstract game tree*.

A strategy is equivalent to the set of all runs with the player's labels obeying the strategy mapping π. Therefore, a strategy can also be represented by a tree. Relaxing the restriction of the strategy mapping allows for a *partial strategy* in which multiple labels are now possible for a single state.

A strategy or partial strategy can also be thought of as an abstract game. A partial strategy for the controller is a restriction only on the controllable labels in the game. So if the environment can not win in the abstract game equivalent to the controller's partial strategy, then all strategies allowable by that partial strategy must be winning.

An abstract game tree can be checked for the existence of a winning run by a SAT solver[3]. The tree must be encoded into CNF by making copies of the transition relation for each game step in tree. The formula must also check whether the error state has been reached in no branch/in all branches for the controller and environment respectively. The functions that produce these formulas are shown in Algorithm 1.

Algorithm 1 Tree formulas for Controller and Environment respectively

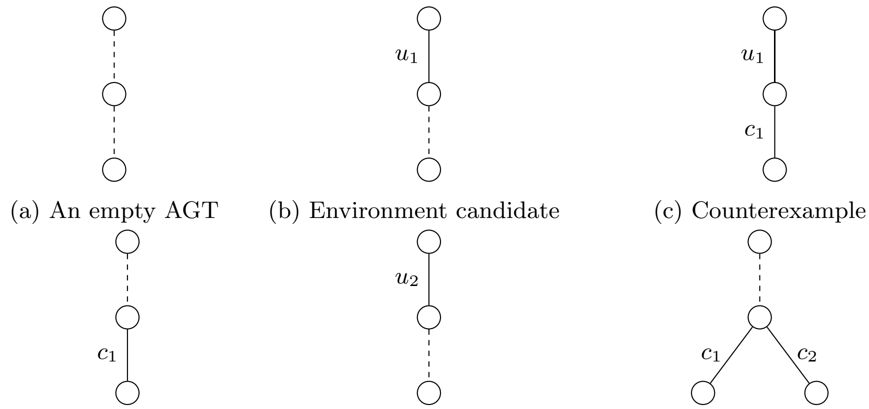
function $\textsc{treeFormula}$(gt)
  if $\textsc{rank}(gt) == 0$ **then**
    return $\neg \mathrm{E}(x^{gt})$
  else
    return $\neg \mathrm{E}(x^{gt}) \wedge \bigwedge_{n \in \mathrm{SUCC}(gt)} (\delta(n) \wedge \textsc{label}(n) \wedge \textsc{treeFormula}(n))$
  end if
end function
function $\overline{treeFormula}$(gt)
  if $\textsc{rank}(gt) == 0$ **then**
    return $\mathrm{E}(x^{gt})$
  else
    return $\mathrm{E}(x^{gt}) \vee \bigvee_{n \in \mathrm{SUCC}(gt)} (\delta(n) \wedge \textsc{label}(n) \wedge \overline{treeFormula}(n))$
  end if
end function

When we produce a formula using $\textsc{treeFormula}$ the SAT solver is playing on behalf on the controller. Any unfixed labels in the tree, controllable or uncontrollable, will be existentially quantified. This means that if there exists a way for the game to be solved when both players cooperate the SAT solver will find it. If no winning run exists in an abstract game even when the players are cooperating then there is definitely no winning run when the opponent is playing adversarily.

2.2 Counterexample Guided Bounded Synthesis

By checking for the existence of a winning run for the environment in an abstract game tree constructed from a partial controller strategy we are checking if the partial strategy is winning. If no spoiling run can be found then the partial strategy must always win. If a spoiling run is found then we then we have a counterexample that proves that the controller strategy is not winning. This forms the basis of a counterexample guided abstraction refinement framework that operates on candidate strategies.

The bounded synthesis algorithm described in [3] begins with an empty game as seen in Figure 1a. Initially we are playing on behalf of the environment because it chooses the first move in each step. The empty game is passed to the SAT solver, which searches for a candidate environment strategy. If a candidate is found (Figure 1b) then it is checked for a spoiling strategy by solving for the controller. If no spoiling strategy exists, that means our candidate is a winning strategy and the algorithm terminates. Otherwise we find a counterexample (Figure 1c), which is used to refine the empty game tree to include the first move from the controller's spoiling strategy (Figure 1d. The algorithm continues by finding a new candidate for the environment (Figure 1e), and a new counterexample to refine the game abstraction once again (Figure 1f).



(a) An empty AGT (b) Environment candidate (c) Counterexample

(d) Refined AGT (e) New candidate (f) Refined AGT

Fig. 1: Abstract game trees

3 Unbounded Synthesis

Bounded synthesis can be used to prove the existence of a winning strategy for the environment by providing a witness. For the controller, the strongest claim that can be made is that the strategy is winning as long as the game does not extend longer than the bound.

In model checking the maximum bound is decided based on the states of the game [1]. The naïve approach is to use size of the state space as the bound ($2^{|X|}$). With a bound of this size all states may be explored. One optimisation is to use the diameter of the game, which is the smallest number d such that for any state x there is a path of length $\leq d$ to all other reachable states. However, for large games these bounds are intractable.

When performing model checking or synthesis with BDDs [2] the set of states that are winning for the environment is iteratively constructed by computing the states from which the environment can force the game into the previous winning set. Eventually this process reaches a fixed point and the total set of environment winning states is known.

A similar concept can be applied to the bounded synthesis algorithm to iteratively increase the bound of the game and terminate when a fixed point is reached. When a strategy is found to be winning on an abstract game tree, we record as winning the states from which the opponent could find no counterexample. To find these states we use interpolation of subformulas of the game tree.

3.1 Learning States with Interpolants

Given two formulas A and B such that $A \wedge B$ is unsatisfiable, it is possible to construct and interpolant $\mathcal{I}$ such that $A \rightarrow \mathcal{I}$, $B \wedge \mathcal{I}$ is unsatisfiable, and $\mathcal{I}$

refers only to the intersection of variables in A and B. An interpolant can be constructed efficiently from a resolution proof of the unsatisfiability of $A \wedge B$ [4].

References

1. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without bdds. In: Cleaveland, W. (ed.) Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 1579, pp. 193–207. Springer Berlin Heidelberg (1999)
2. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. In: Symposium on Logic in Computer Science. pp. 428–439 (1990)
3. Narodytska, N., Legg, A., Bacchus, F., Ryzhyk, L., Walker, A.: Solving games without controllable predecessor. In: Computer Aided Verification. pp. 533–540 (2014)
4. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. Journal of Symbolic Logic 62(3), 981–998 (1997)