LTL2AIG v2.0: Compositional Benchmarks

Guillermo A. Pérez
Université Libre de Bruxelles – Brussels, Belgium
gperezme@ulb.ac.be

***Abstract*—We update our procedure to translate LTL synthesis specifications into the extended AIGER format for Synthesis. In particular, we are now able to translate all the benchmarks bundled with Acacia+ by constructing several automata – instead of a single monolithic one – from the input LTL specification, whenever possible.**

I. Introduction

The original AIGER format [1] is a safety property specification used for model checking purposes. The extended version of it was introduced in 2014 by Jacobs [2] as a standard way of representing safety objectives for controller synthesis. The first use of this format was the Synthesis Competition 2014, a satellite event of the 26th International Conference on Computer Aided Verification 2014.

In [3] the authors present an algorithm to synthesize strategies for objectives specified using linear temporal logic (LTL) (see, e.g. [4]). Their algorithms take as input an LTL formula and a partition of the atomic propositions into controllable and uncontrollable signals. The idea of the algorithm is to 1) translate the negation of the LTL formula into a universal co-Büchi automaton, and 2) play a k-co-Büchi game on the extended subset construction of it. That is, the controller is allowed to visit configurations with accepting states at most k times. They increase this k until a winning strategy for the controller is found or k goes above a threshold (which is doubly-exponential in the size of the input formula). At this point the controller has a winning strategy in the constructed k-co-Büchi game if and only if she has a winning strategy in the original universal co-Büchi game.

A. Specification Units

We note that some of the example benchmarks bundled with Acacia+ are given as LTL formulas partitioned into independent *spec units*. That is to say, the specification Φ is actually given by the conjunction of the formulas of each spec unit F_i. Each one of these spec units is transformed into an individual k-co-Büchi automaton by Acacia+. In practice, engineers and designers write their specifications in modules and not just as a big conjunction of individual and interdependent specifications. Dealing with each of the formulas individually reduces the time that the LTL-to-automaton program takes to output a deterministic machine and allows for several optimizations used by Acacia+.

In this update of the LTL2AIG tool [5], we follow the same idea. More specifically, we handle LTL formulas decomposable into spec units and construct several machines for them. We extend our translation from automata to AIG so that the error function in the final AIGER file corresponds to the disjunction of the acceptance condition of the constructed automata.

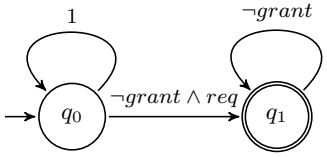

Fig. 1. Universal co-Büchi automaton for the negation of LTL formula 1.

II. LTL to AIGER

We have translated several benchmarks provided in the Acacia+ tool package [6], [7] into the extended AIGER format. The steps we follow closely resemble the algorithm presented in [3]. We take as input an LTL formula, a partition of the atomic propositions and a positive integer $k > 0$. Thus, if the AIGER safety objective specification is realizable, then the original LTL objective is also realizable (but the converse does not hold in general).

A. LTL formula to universal co-Büchi automata

We assume the reader has some familiarity with linear temporal logic. Consider the following LTL formula, where $grant$ is a controllable action and req is uncontrollable. This formula asks that every req emitted by the environment is eventually $grant$ed by the controller.

$$\square(req \rightarrow \lozenge grant); \tag{1}$$

The negation of this formula is then translated into its corresponding universal co-Büchi automaton via, say ltl2ba [8], [9]. A run of this machine is accepting if it visits accepting states infinitely often. The automaton accepts an (omega) word if and only if no run of it on the given word is accepting. Figure 1 depicts the automaton for Equation 1. Observe that the machine, upon receiving a req and no $grant$, may move to state q_1. From there, the machine stays in q_1 if no $grant$ is issued. Hence, to ensure no run of the automaton stays in q_1 forever, controller must eventually $grant$ every req issued by environment.

B. From universal co-Büchi automata to AIG

As explained in the introduction, we will construct a game in which the controller has to avoid visiting accepting states more than k times. We achieve this by having $k+2$ copies of

each state, one for each element from the sequence $0, \ldots, k+1$. Intuitively, we encode into the current state how many times the play has reached accepting states. This will be reflected in the new transition relation we define. If the play is in a copy of state q after having visited j accepting states and the next state visited, s, is accepting, then the actual state to which we transition is the $(j+1)$-th copy of s.

We now describe the translation in a formal fashion. Let Φ be an LTL formula consisting of spec units $F_1, \ldots, F_n$ with controllable and uncontrollable atomic proposition sets Σ_u, Σ_c. That is to say,

$$\Phi = \bigwedge_{1 \leq i \leq n} F_i$$

Denote by Q_i the set of states from the automaton constructed from $\neg F_i$, $\Delta \subseteq Q_i \times \Sigma_u \times \Sigma_c \times Q_i$ its transition relation and $\mathcal{B}_i \subseteq Q_i$ the set of accepting states.

AIGER specifications are composed of a set of latches L, a set of inputs I, a single error function and a set of gates. Additionally, in the extended AIGER format one is expected to mark inputs as controllable if they are so. We take the set of latches L to be $\bigcup_{i=1}^{n}(Q_i \times \{0, \ldots, k+1\})$.

In the sequel, we focus on building the transition function for latches in L_i which are used to encode the state-counter pairs from $Q_i \times \{0, \ldots, k+1\})$. For simplicity, we denote every latch by a state and number pair, e.g. $l_{(q_1,1)}$. For the special case of the initial configuration $(q_I, 0)$ we set

$$f_{l_{(q_I,0)}} = (\bigwedge_{l \in L} \neg l) \vee \bigvee_{(q,\sigma_u,\sigma_c,q_I) \in \Delta} l_{(q,0)} \wedge \sigma_u \wedge \sigma_c.$$

For every pair $s, i \in (Q_i \setminus (\mathcal{B}_i \cup \{q_I\})) \times \{0, \ldots, k+1\}$, we set the "next step" function of $l_{(s,i)}$ to be

$$f_{l_{(s,i)}} = \bigvee_{(q,\sigma_u,\sigma_c,s) \in \Delta} l_{(q,i)} \wedge \sigma_u \wedge \sigma_c$$

while for pairs $s, i \in (\mathcal{B} \setminus \{q_I\}) \times \{1, \ldots, k+1\}$ it is given by

$$f_{l_{(s,i)}} = \bigvee_{(q,\sigma_u,\sigma_c,s) \in \Delta} l_{(q,i-1)} \wedge \sigma_u \wedge \sigma_c.$$

The error function is then given by

$$f_{BAD}^{[i]} = \bigvee_{q \in Q_i} l_{(q,k+1)}.$$

We note the functions described above can be transformed into equivalent ones which only use logical conjunction and negation (required by the AIGER format). These, along with the input set $\Sigma_u \cup \Sigma_c$, define the transition function for the automaton corresponding to $\neg F_i$. The *global error function* is then given by

$$f_{BAD} = \bigvee_{1 \leq i \leq n} f_{BAD}^{[i]}.$$

This concludes the description of the circuit specification for the desired safety objective.

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] E. Filiot, N. Jin, and J.-F. Raskin, "Antichains and compositional algorithms for ltl synthesis," *Formal Methods in System Design*, vol. 39, no. 3, pp. 261–296, 2011.

[4] C. Baier, J.-P. Katoen *et al.*, *Principles of model checking*. MIT press Cambridge, 2008, vol. 26202649.

[5] G. A. Pérez. LTL2AIG. [Online]. Available: https://github.com/gaperez64/acacia_ltl2aig

[6] A. Bohy, V. Bruyère, E. Filiot, N. Jin, and J.-F. Raskin, "Acacia+, a tool for ltl synthesis," in *CAV*. Springer, 2012, pp. 652–657.

[7] E. Filiot. Acacia+. [Online]. Available: http://lit2.ulb.ac.be/acaciaplus/

[8] P. Gastin and D. Oddoux, "Fast ltl to büchi automata translation," in *CAV*. Springer, 2001, pp. 53–65.

[9] P. Gastin. Ltl2ba: fast translation from ltl formulae to büchi automata. [Online]. Available: http://www.lsv.ens-cachan.fr/ gastin/ltl2ba