Anzu: A Tool for Property Synthesis[⋆]

Barbara Jobstmann, Stefan Galler, Martin Weiglhofer, and Roderick Bloem

Graz University of Technology

Abstract. We present the tool ANZU. ANZU takes a formal specification of a design and generates a functionally correct system if one exists. The specification is given as a set of linear temporal logic (LTL) formulas belonging to the class of *generalized reactivity* of rank 1. Such formulas cover the majority of the formulas used in practice. ANZU is an implementation of the symbolic reactive(1) approach to synthesis by Piterman, Pnueli, and Sa'ar. If the specification is *realizable* ANZU provides the user with a Verilog module that represents a correct finite-state system.

1 Introduction

Automatically constructing a system from a logical specification has been one of the more ambitious dreams in computer science for almost half a century [Chu62]. Given a specification over the signals $\mathcal{I} \cup \mathcal{O}$ the Synthesis Problem is to construct a reactive system with input signals $\mathcal{I}$ and output signal $\mathcal{O}$ such that all infinite input-output sequences produces by this system adhere to φ, in case such a system exists.

Rabin [Rab69] and Büchi and Landweber [BL69] were the first to provide solutions to the Synthesis Problem for S1S. Pnueli and Rosner [PR89] reconsidered the problem for specifications in *linear temporal logic* (LTL) and provided an algorithm to construct *open systems*, systems that behave correctly independent of the surrounding environment. Furthermore, they proved that synthesis of LTL formulas is 2EXPTIME-complete [Ros92].

In order to overcome the complexity issues, research concentrated in part on subsets of LTL. Recently, Pnueli, Piterman, and Sa'ar [PPS06] proposed an efficient symbolic algorithm to synthesize specification of reactive(1) designs, in time N^3 where N is the size of the state space of the design. ANZU implements this approach and extends it to generate compact circuits.

Anzu can be found at http://www.ist.tugraz.at/staff/jobstmann/anzu/.

2 Case Studies

We start by describing our case studies as we will use parts of them to show how ANZU works. We have tried ANZU on two examples. The first is an arbiter for ARM's AMBA Advanced High-Performance Bus (AHB). The AHB is an on-chip bus that connects components like processor cores, DMA controllers, and cache. It has up to 16 masters,

[⋆] This work was supported in part by the European Union under contract 507219 (PROSYD).

```
[INPUT_VARIABLES]                         [OUTPUT_VARIABLES]
  RtB0;                                     BtR0;
  ...                                       ...
[ENV_INITIAL]                             [SYS_INITIAL]
  ...                                       ...
[ENV_TRANSITIONS]                         [SYS_TRANSITIONS]
  G((BtR0=1 * RtB0=1) -> X(RtB0=1));        G((RtB0=1 * BtR0=1) -> X(BtR0=1));
  G( BtR0=0 -> X(RtB0=0));                  G( RtB0=1 -> X(BtR0=0));
  ...                                       ...
[ENV_FAIRNESS]                            [SYS_FAIRNESS]
  ...                                     ...
```

Fig. 1. Part of the input file to synthesis GenBuf with ANZU

which initiate transfers, and up to 16 clients, which are mostly passive. The AHB supports different transfer types, including different length bursts. From the official specification for the bus [ARM99], which is written in English, we have distilled a set of 4 assumptions and 11 guarantees.

The second case study is a generalized buffer (henceforth *GenBuf*) from IBM[1]. This design forwards data from n senders to two receivers using a handshake protocol on either side and a FIFO to store data temporarily. The design is used at IBM and came with a very clear and relatively complete (although not fault-free) formal specification. Our full specification consists of 13 guarantees and 5 assumptions.

3 Technical Approach

ANZU is written in Perl and the symbolic algorithms rely on PerlDD, a Perl extension of the the CUDD BDD Package [Som]. We use the handshake protocol between GenBuf and a receiver to illustrate a specification. Genbuf communicates with receiver i using the signals BtoRi and RtoBi. The handshake between GenBuf and a receiver consists of four phases: (1) GenBuf requests receiver i (by raising BtoRi) and may not cancel the request. (2) Receiver i eventually answers the request (by raising RtoBi). (3) In the next step, GenBuf lowers the request, and (4) after a further step the receiver deasserts the signal RtoBi. In order to ensure a correct handshake, GenBuf and the receiver have to meet the following requirements. GenBuf has to guarantee $\bigwedge_{i\in\{0,1\}}(\mathsf{G}((\text{BtoRi} \wedge \neg\text{RtoBi}) \rightarrow \mathsf{X}\,\text{BtoRi}) \wedge \mathsf{G}(\text{RtoBi} \rightarrow \mathsf{X}\,\neg\text{BtoRi}))$. Receiver i has to fulfill $\mathsf{G}(\text{BtoRi} \rightarrow \mathsf{F}\,\text{RtoBi}) \wedge \mathsf{G}((\text{BtoRi} \wedge \text{RtoB}) \rightarrow \mathsf{X}\,\text{RtoBi}) \wedge \mathsf{G}(\neg\text{BtoRi} \rightarrow \mathsf{X}\,\neg\text{RtoBi})$. Fig.1 shows part of the requirements as we write them for ANZU.

ANZU takes a text file holding the definitions of the input and output signals (see Fig.1, Line 1-3) and a specification in a subset of LTL. The specification needs to comply with simple syntax rules [PPS06]: A specification consists of two parts: *assumptions* on the environment (Fig.1, Col.1) and *guarantees* the system has to keep, if the environment fulfills its assumptions (Fig.1, Col.2). Each part is defined by a set of LTL formulas over input signals $\mathcal{I}$ and output signals $\mathcal{O}$. The signals in $\mathcal{I}$ are controlled by the environment, signals in $\mathcal{O}$ are controlled by the system. An assumption or a guarantee defines either allowed *initial* states, possible *transitions*, or *fairness* obligations that all accepting runs have to fulfill.

[1] See http://www.haifa.ibm.com/projects/verification/RB_Homepage/tutorial3/.

A formula that does not fall in one of these three classes can usually (but not always) be transformed to a suitable format by using *deterministic monitors*. For such a formula we construct a deterministic Büchi word automaton We create a Boolean encoding of the state space of the automaton using a new set of variables, and describe its behavior using an initial constraint, a constraint on the transition relation, and a fairness constraint, as before. For instance, for the formula $\mathsf{G}(\text{BtoR0} \to \mathsf{F}\,\text{RtoB0})$ we add a variable s and encode a corresponding deterministic monitor with the formulas $\neg s$, $\mathsf{G}((\neg s \wedge (\neg\text{BtoR0} \vee \text{RtoB0})) \to \mathsf{X}\,\neg s)$, $\mathsf{G}((\neg s \wedge \text{BtoR0} \wedge \neg\text{RtoB0}) \to \mathsf{X}\,s)$, $\mathsf{G}((s \wedge \neg\text{RtoB0}) \to \mathsf{X}\,s)$, $\mathsf{G}((s \wedge \text{RtoB0}) \to \mathsf{X}\,\neg s)$, and $\mathsf{G}\,\mathsf{F}\,\neg s$.

ANZU follows the approach of [PPS06]: it builds a transition system over the input and output variables from the formulas restricting the transitions. On the transition system ANZU searches for states from which the system can force an accepting run independent of the input values the environment chooses. A run is accepting if either a fairness obligation of the environment is violated or all fairness obligation of the system are fulfilled. If the initial states belong to this set of states, the specification is realizable and ANZU builds a BDD representing a set of possible implementations. Otherwise, ANZU reports that the specification is not realizable and quits. From the BDD, ANZU extracts a circuit with $|\mathcal{I}| + |\mathcal{O}|$ flip flops and combinational logic expressed on the gate level. The relation between the variables of the specification and the signals of the circuit are depict in Fig.2. (Primed variables denote next-state values.) The construction of the combinational circuitry turns out to be the most time consuming part of synthesis. We have tried approaches based on [KS00] and on the use of cofactors [BGJ^{+}07] and we are actively researching better ways of constructing the circuits.

Results. We have synthesized GenBuf for upto 10 senders. (Ten senders seem enough considering that the original circuit handles only four.) The specification for 10 senders consists of 13 fairness formulas (2 for the environment and 11 for the system) and 121 guarantees and 27 assumptions that restrict the transition relation. Figure 3 shows how the size of the circuit grows as a function of the number of senders for two different methods of constructing the circuit. (The y axis shows the “standard cell grid count” for the synthesized circuit, which allows for a fair comparison of the sizes of different circuits, even though the absolute values may not be very informative.) Note that even with the cofactor method, the circuit is still about a factor of 10 larger than a handwritten one. The number of latches needed in the circuit grows from 13 to 40 and synthesis goes through in under a minute for 10 masters.

The AHB arbiter is harder. We are able to synthesize it for up to 10 masters, after which our tool runs out of memory. The case study is described in detail in [BGJ^{+}07]. The number of latches needed for the largest arbiter is 60 and the time for synthesis is 6.5 hours, 85% of which is spent reordering BDDs.

4 Benefits and Drawbacks of Synthesis

The generalized reactivity(1) framework in which we stated the specification turned out to be expressible enough for anything we wanted state. Deciding realizability is relatively simple in comparison to building a circuit. Furthermore, the specifications are relatively small and easy to read.

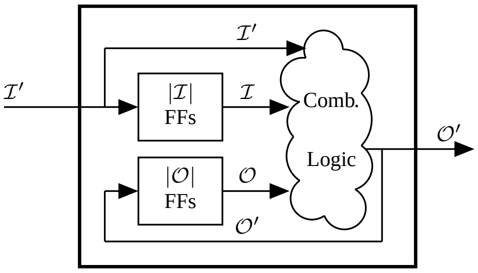

Fig. 2. Design of the generated circuits

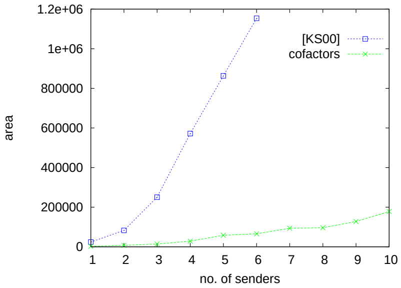

Fig. 3. Size of the generalized buffer

On the other hand, writing a (complete!) formal specification is sometimes hard, although the introduction of new signals usually helped. (We see great potential for tools that help the user debug the specification.) Furthermore, the performance of the tool depends heavily on the formulation of the specification. The circuits produced by ANZU are relatively large and their size depends heavily on the value of the parameter, much more strongly than is expected for manually coded circuits.

Concluding, ANZU allows for synthesis of real, though modest sized, industrial examples from specifications and ANZU's performance is improving rapidly.

Acknowledgments The authors would like to thank Karin Greimel and Milan Milinkovic for their help with the implementation, and Nir Piterman and Amir Pnueli for interesting discussions.

References

[ARM99] ARM Ltd. AMBA Specification (Rev. 2). Available from www.arm.com, 1999.

[BGJ+07] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Automatic hardware synthesis from specifications: A case study. In *Proceedings of the Conference on Design, Automation and Test in Europe*, 2007.

[BL69] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[Chu62] A. Church. Logic, arithmetic and automata. In *Proceedings International Mathematical Congress*, 1962.

[KS00] J. H. Kukula and T. R. Shiple. Building circuits from relations. In *12th Conference on Computer Aided Verification (CAV'00)*, pages 113–123, 2000. LNCS 1855.

[PPS06] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *Proc. Verification, Model Checking, and Abstract Interpretation*, pages 364–380, 2006.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. Symposium on Principles of Programming Languages (POPL '89)*, pages 179–190, 1989.

[Rab69] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[Ros92] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.

[Som] F. Somenzi. *CUDD: CU Decision Diagram Package*. University of Colorado at Boulder, ftp://vlsi.colorado.edu/pub/.