Verification of Requirement Specifications Using Counter Automata

K Vasanta Lakshmi K V Raghavan

Dept. of Computer Science and Automation
Indian Institute of Science, Bangalore
{kvasanta,raghavan}@csa.iisc.ernet.in

1. INTRODUCTION

Collecting the requirements specification is the first step in the software development cycle. All the subsequent steps are dependent on it. So it is important that these specifications are unambiguous, consistent, complete and correct. Constructing a model from the specifications and verifying the model can help catch some bugs early in development cycle. Many modeling techniques used to model requirements specifications, such as petri nets [9], hybrid automata [7] and counter automata [8, 2], view the system as a state transition system. It is known that model-checking of finite state transition systems is decidable. However, for many of the infinite-state transition systems even answering reachability of a state is not decidable.

Several partitioning-based algorithms [7, 6] have been proposed to answer reachability and other temporal properties of infinite-state systems. The input to these algorithms is any initial partitioning of the set of states described by the system (e.g., into initial and non-initial states), and output is a refinement of the initial partitioning that satisfies a certain property. These algorithms refine the initial partitioning by iteratively computing the predecessor set of states of the partitions, until a fix point is reached. For the backward analysis algorithm that we implement the final partitioning satisfies the following property:
If there exists an edge from partition P_1 to partition P_2, then for every state $s_1 \in P_1$ there exists a state $s_2 \in P_2$, such that there exists a transition from s_1 to s_2 in the given infinite-state transition system.
If the algorithm terminates then its output will be a finite set of partitions of the state space, with transitions among these partitions, which essentially simulate the original system. Using this finite system one can answer reachability of *some* state in a given target partition from any other state in any other partition.

Similar to backward analysis an algorithm can perform a forward analysis by computing the successor sets of the partitions in the refinement step. The output of the forward algorithm that we propose satisfies the following property:
If there exists an edge from partition P_1 to partition P_2, then for every state $s_2 \in P_2$ there exists a state $s_1 \in P_1$, such that there exists a transition from s_1 to s_2 in the given infinite state transition system.[1]
If the algorithm terminates then its output will be a finite set of partitions of the state space. In the finite transition system formed by these partition, a path between two partitions P_i and P_j means that *every* state in P_j is reachable from atleast one state in P_i. Clearly this partitioning is more powerful than that produced by the backward analysis, in the sense that it can be used to answer reachability of *any* state from any other state.

The classes of systems for which backward and forward algorithms terminate, respectively, are not the same. They overlap, but neither is contained in the other. In the Appendix we give an example for which our forward analysis algorithm terminates, but backward analysis algorithms do not terminate.

Some generalized results have been proposed by Finkel et. al [5, 3, 4] that give sufficient conditions for termination of reachability-testing algorithms that are based on saturation of reachable sets (as opposed to partitioning). Their fundamental sufficient condition is that the given infinite-state system be a *well-structured transition system* (WSTS). This characterization applies to diverse sorts of infinite-state systems such as petri nets, hybrid automata, counter automata, and lossy-channel systems.

2. OUR WORK

In our talk we will present some case studies of applying a backward-analysis partitioning algorithm and our forward-analysis partitioning algorithm to several real-world systems, from domains such as streaming applications [1], banking [11] and finance [10]. The motivation for our case studies was primarily to understand the real-life applicability of the various approaches and concepts discussed earlier, and to identify directions for future work to improve their applicability. In particular, we wished to

- To understand the idioms and patterns exhibited by real systems that cause both kinds of algorithms, respectively, to terminate or non-terminate, respectively.
- To identify systems (if any) that satisfy the WSTS property, and yet do not result in a finite partitioning when processed by either partitioning algorithm (forward analysis or backward analysis).
- To understand whether or not systems on which at least one of the partitioning algorithms terminate invariably satisfy the WSTS property.
- To identify systems on which at least one of the partitoning-based algorithm terminates, but that does not satisfy the WSTS property. The previous point as well as

[1]We are not aware of any partitioning-based algorithm that performs forward analysis.

this one are basically motivated by the desire to compare and contrast the partitioning and set-saturation approaches when applied to practical settings.

In our opinion the above objectives have not been addressed adequately by the existing literature. In addition, the results of our case study are likely to point to interesting future work directions in identifying better (i.e., broader) sufficient conditions (relative to WSTS) for answering temporal properties of infinite-state systems, as well as improved versions of partitioning-based algorithms (i.e., that terminate on a broader class of systems).

3. REFERENCES

[1] S. Agrawal, W. Thies, and S. Amarasinghe. Optimizing stream programs using linear state space analysis. In *Proc. 2005 Intl. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '05, pages 126–136. ACM, 2005.

[2] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 268–279, London, UK, 1998. Springer-Verlag.

[3] A. Finkel and J. Goubault-Larrecq. Forward Analysis for WSTS, Part I: Completions. In *STACS*, volume 3 of *LIPIcs*, pages 433–444. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[4] A. Finkel and J. Goubault-Larrecq. Forward Analysis for WSTS, Part II: Complete WSTS. In *Proceedings of the 36th Internatilonal Colloquium on Automata, Languages and Programming*, ICALP '09, pages 188–199, Berlin, Heidelberg, 2009. Springer-Verlag.

[5] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[6] B. S. Gulavani, T. A. Henzinger, Y. Kannan, A. V. Nori, and S. K. Rajamani. Synergy: a new algorithm for property checking. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 117–127, New York, NY, USA, 2006. ACM.

[7] T. Henzinger. The theory of hybrid automata. *Logic in Computer Science, Symposium on*, pages 278–292, 1996.

[8] O. H. Ibarra, J. Su, Z. Dang, T. Bultan, and R. A. Kemmerer. Counter machines: Decidable properties and applications to verification problems. In *MFCS '00: Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 426–435, London, UK, 2000. Springer-Verlag.

[9] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[10] M. Simmons. *Securities Operations: A Guide to Trade and Position Management*. John Wiley, 2002.

[11] *Terms and conditions* of popular credit cards.

APPENDIX

Consider a banking system that processes deposit and withdrawal transactions, one transaction at a time. The items

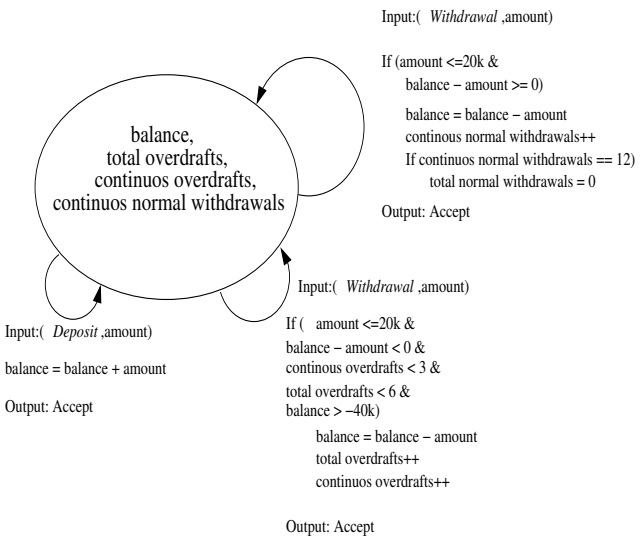

Figure 1: Counter Automata

given below represent the requirements specification of the system. This system can be modeled using the counter automaton given in Figure 1.

- Individual withdrawals have an upper limit of Rs. 20,000.
- Overdrafts (i.e., withdrawing more than the current balance) are allowed.
- Overdrafts are not allowed when
  - When the balance is below negative Rs. 40,000
  - When the previous three withdrawals were overdrafts.
- After six overdrafts, no more overdrafts are allowed until 12 normal withdrawals are completed.

This system has infinite state space because the balance can take unbounded positive values. Also, this system has temporal characteristics, as the occurrence of some events is dependent on past events. Figure 2 shows a part of the finite final partitioning that is given by our forward-analysis partitioning algorithm. Each oval represents a partition, with the predicate inside the oval describing the set of states (in the original system) that belong to this partition. Partition P1 contains the initial state, where all counters are set to 0. Partition P1 contains the set of states that can be reached after one overdraft from the initial state. P2 and P3 are other partitions that contain states reachable from the initial partition. P4 contains a subset of unreachable states in the system; note this partition is not reachable from the initial partition (or, in fact, from any other partition).

Some example verification properties that are of interest in this system are:

- Reachability of a state. *Example*: Is the state wherein balance = -20,000, total overdrafts = 2, continuous overdrafts = 3, continuos normal withdrawals = 0, reachable? The answer is *no*, as total overdrafts $\geq$ continuous overdrafts must hold as per requirements specification. The reachability of this state can be answered by checking if there is a path from the initial partition to the partition to which this given state belongs, i.e., partition P4 in Figure 2.

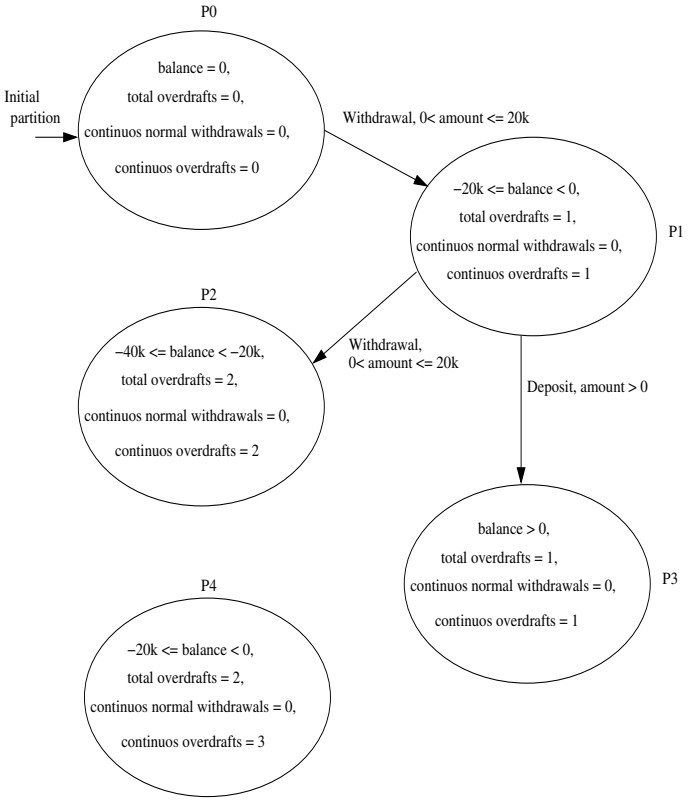

Figure 2: Transition System

- Reachability of a state. *Example*: Is the state wherein balance = -30,000, total overdrafts = 2, continuous overdrafts = 2, continuos normal withdrawals = 0, reachable? The answer is *yes*. This state belongs to partition P2 in Figure 2.
- Temporal property. *Example*: Can an overdraft be rejected without any overdraft being accepted in the past. The answer is *no*. This property can be checked by analyzing all the paths from the initial partition to all partitions containing a state where overdraft is rejected that pass through an overdraft edge where system output is accept.