Proc. of 36th IEEE Conf. on *Decision and Control*, Dec. 1997, San Diego, CA, USA

On Computing Optimal Controllers for Finite State Systems

Enrico Tronci[1]
Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, Coppito 67100 L'Aquila, Italy
tronci@univaq.it

Abstract

From a computational point of view many control problems involve synthesis of controllers for Finite State Systems *(FSSs). In this paper we address the problem of automatic synthesis of* Optimal Controllers *(OCs) for FSSs. Such problem has been widely studied and many algorithms are known for it. However, because of* state explosion *their use is limited to quite small dynamical system.*

We devised a symbolic algorithm for automatic synthesis of OCs for FSSs. In this paper we show practical usefulness of our techniques by giving experimental results on the use of our symbolic algorithm to synthesize a C program implementing an OC for a semiconductor manufacturing facility. This entails computing OCs for plants with about $3{*}10^9$ *states. To the best of our knowledge no previous algorithm can handle systems of such size.*

Key Words Optimal Control, Finite State Systems, Discrete Event Systems, Supervisory Control, Hybrid Systems, Manufacturing Systems, Boolean Symbolic Programming, Boolean First Order Logic, Binary Decision Diagrams

1 Introduction

From a computational point of view many control problems involve synthesis of controllers for *Finite State Systems* (FSSs). E.g. finite state *Discrete Event Systems* (DESs) can be modeled as FSSs in order to synthesize (optimal) supervisory controllers (e.g. see [1, 5, 10, 12, 14, 13]). After sampling and discretization many continuous systems can be approximated, for computational purposes, as FSSs (e.g. see [9, 17]). FSSs also arise in modeling and control of *Hybrid Systems* (HS) (e.g. see [4, 11]). Thus automatic synthesis of controllers for FSSs plays an important role in automatic synthesis of controllers in general.

In this paper we address the problem of automatic synthesis of *Optimal Controllers* (OCs) for FSSs. Informally our *Optimal Control Problem* (OCP) can be described as follows. We are given a *plant* (i.e. a deterministic FSS), a set of plant states (*final states*) and a *cost* (i.e. a positive integer) for each plant transition. Plant transitions are triggered by *events*. The set of events is finite. A (state feedback) controller is a function (control law) associating to each plant state an event. I.e. the controller observes the plant state and *triggers* an event accordingly to the control law. Given a controller an *objective function* (or *cost index*) associates to each plant state a cost on the base of plant transition costs and of the control law. We want to find an *Optimal Controller* (OC), i.e. a controller that drives the plant to a final state and minimizes a given objective function.

Not surprisingly the above OCP has been widely studied. In fact a FSS can be modeled as a graph which vertices represent states and edges represent transitions. Thus our OCP is the problem of finding for each state a shortest path to a final state. There are well known *classical* algorithms to solve such graph problem (e.g. Dijkstra, Dial, see [2]). Unfortunately the number of states of a dynamical system is typically so large (*state explosion*) that even the system cannot be explicitly represented in memory with a graph. Let alone finding an OC. This limits the use of the above mentioned algorithms to *small* dynamical systems.

Symbolic techniques (based on *Binary Decision Diagrams*, BDDs, [7]) have been very successful in contrasting state explosion in automatic verification of FSSs via *Model Checking* (e.g.[8], [6]) as well as in automatic synthesis of *Supervisory Controllers* satisfying given specifications (e.g. see [5]). When the objective function ranges on a finite domain a symbolic algorithm for automatic synthesis of *Optimal Supervisory Controllers* (OSCs) is given in [16]. Here we are interested in finding an optimal controller in the *traditional* sense (i.e., using a supervisory control terminology, for us all events are controllable). We could use the symbolic algorithm in [16]. However exploiting the fact that in our OCP all events are controllable and the plant is deterministic it is possible to design a symbolic algorithm much more efficient than the one presented in [16].

Using our symbolic algorithm we succeeded in computing OCs for plants with about $3 * 10^9$ states. To the best of our knowledge no previous algorithm can handle systems of this size. Note that the symbolic algorithm in [16] can only handle systems with about 10^4 states. Moreover, although we are using BDDs, our symbolic algorithm does not require us to know in advance the largest value taken by the objective function when using an optimal controller as instead is the case for the symbolic algorithm in [16].

In this paper we show practical usefulness of our symbolic algorithm by presenting (sect. 2) experimental results on the automatic synthesis of a C program implementing a minimum time OC for the INTEL semiconductor manufacturing facility modeled in [3].

2 Experimental Results

We implemented a BDD based compiler for the *Boolean Symbolic Programming* language outlined in [15]. Using such tool we run our automatic synthesis experiments.

Essentially our symbolic algorithm defines a *suitable* breadth first visit of the plant transition graph.

Our plant is the INTEL semiconductor manufacturing facility described in [3, figs. 2.1, 2.2]. We modified the model in [3] as follows: lot size is 1 for all processes, buffers are finite.

[1] This research has been partially supported by MURST funds.

Buffers size	State bits (n)	State Space Size	CPU (min)	Max BDD	OC	J^*	Parts	Avg
1	22	775,680	17	1,283,944	961	469	9	52.1
2	28	8,835,480	100	2,000,046	1335	765	15	51.0
3	28	49,643,520	200	2,000,046	1757	1071	21	51.0
7	34	3,177,185,280	3800	2,000,046	2300	2295	45	51.0

Figure 1: Experimental Results on SUN Sparc Station 20 with 128MB of RAM

The manufacturing facility is formed by 3 machines and 6 buffers. Each machine can execute one of two (mutually exclusive) processes. Thus there are 6 processes. In our plant we keep the same processing times as in [3, table 2.1]. We use a time unit (clock) of 5 minutes. Our goal is to find a controller that empties the plant in minimum time. Thus we have just one final state: all buffers and machines are empty. Note that here we are looking for a state feedback controller. As well known this has many advantages w.r.t. a static machine scheduling (e.g. machine failures are automatically handled).

Fig. 1 reports our experimental results.

All buffers have the same size given by column 'Buffer Size'. The state of each buffer is represented with $buf\_bits = \lceil \log_2(\text{'Buffer Size'} + 1) \rceil$ bits.

Column 'State bits' gives the number of bits we used to represent the state of the all plant (n in the following). Column 'State Space Size' is the product of the number of backward reachable states (i.e. states reachable from a final state) of buffers (i.e. $(\text{'Buffer Size'} + 1)^6$) and machines (i.e. 12120). This is less than 2^n.

Column 'Max BDD' gives the size of the larger BDD built during the computation when garbage collection is invoked at (about) 2,000,046 BDD nodes.

Column 'OC' gives the number of BDD vertices needed to represent the r $(= 9)$ BDDs defining our optimal controller $\mathbf{f}$. From the BDDs representing $\mathbf{f}$ our tool can automatically generate a C program, say C($\mathbf{f}$), implementing $\mathbf{f}$. C($\mathbf{f}$) closely follows the BDD representation of $\mathbf{f}$. Thus column 'OC' also gives the size (lines) of C($\mathbf{f}$). The running time of C($\mathbf{f}$) is O($r * n$). This could be made $O(n)$ if we had used a hardware (parallel) implementation for our controller.

Column 'J^*' gives the length of a longest path to a final state in the optimally controlled plant. Thus it is the maximum time (in time units of 5 minutes) needed to empty the plant using an optimal controller.

Column 'Parts' gives the maximum number of parts present in the plant. This is 6*('Buffer Size') + 3.

Column 'Avg' gives 'J^*'/'Parts' which measures the average number of time units a part spends in the optimally controlled plant.

As to be expected, using buffers of size 2 gives a smaller avg time than using buffers of size 1. However when using an optimal controller buffers of size 2, 3 or 7 give the same avg time.

From the last two raws of the table in fig. 1 we get a running time for our symbolic algorithm of $O(s^{0.7})$ where s is the size of the state space (i.e. s = 'State Space Size'). Note that this is better than the running time $O(s)$ of an explicit breadth first visit of the plant transition graph. Indeed to the best of our knowledge no previous algorithm can synthesize optimal controllers for systems with state spaces as large as the ones we considered here. Note however that for small plants (graphs) an explicit breadth first visit of the plant transition graph is faster than our symbolic approach.

References

[1] A. Aziz, F. Balarin, R. K. Brayton, M. D. DiBenedetto, A. Saldanha, A. L. Sangiovanni-Vincentelli, *Supervisory Control of Finite State Machines*, CAV 95, LNCS 939, Springer-Verlag

[2] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, *Network Flows: theory, algorithms and applications*, Prentice-Hall, 1993

[3] E. Adl, M. K. Rodriquez, A.A. Tsakalis, S. Kostas, *Hierarchical Modeling and Control of Re-entrant Semiconductor Facilities*, Proc. of 35th IEEE Conf. on "Decision and Control", 1996, Kobe, Japan

[4] P. J. Antsaklis, J. A. Stiver, M. Lemmon, *Hybrid System Modeling and Autonomous Control Systems*, Hybrid Systems, LNCS 736, Springer-Verlag, 1993

[5] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, G. F. Franklin, *Supervisory Control of a Rapid Thermal Multiprocessor*, IEEE Trans. on Automatic Control, Vol. 38, N. 7, July 1993

[6] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang, *Symbolic model checking: 10^{20} states and beyond*, Information and Computation 98, (1992)

[7] R. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transaction on Computers, Vol. C-35, N.8, Aug. 1986

[8] E. A. Emerson, *Temporal and Modal Logic*, Handbook of Theoretical Computer Science, Elsevier 1990

[9] G. Kreisselmeier, T. Birkhölzer *Numerical Nonlinear Regulator Design* , IEEE Trans. on Automatic Control, Vol. 39, N.1, Jan. 1994

[10] R. Kumar, V. Garg, *Optimal Supervisory Control of Discrete Event Systems*, SIAM Journal of "Control and Optimization", March 1995, 419-439

[11] M. Lemmon, J. A. Stiver, P. J. Antsaklis, *Event Identification and Intelligent Hybrid Control*, Hybrid Systems, LNCS 736, Springer-Verlag, 1993

[12] K. M. Passino, P. J. Antsaklis, *On the Optimal Control of Discrete Event Systems*, Proc. 28th IEEE Conf. Decision and Control, 1989

[13] P. J. Ramadge, W. M. Wonham, *The Control of Discrete Event Systems*, Proceedings of the IEEE, 77(1):81-98, Jan. 1989,

[14] R. Sengupta, S. Lafortune, *A Graph-Theoretic Optimal Control Problem for Terminating Discrete Event Processes*, Discrete Event Dynamic Systems: Theory and Applications 2, (1992): 139–172, Kluwer

[15] E. Tronci, *Hardware Verification, Boolean Logic Programming, Boolean Functional Programming*, Proc. 10th IEEE Conf. on "Logic In Computer Science" 1995, San Diego, CA, USA

[16] E. Tronci, *Optimal Finite State Supervisory Control*, Proc. of 35th IEEE Conf. on "Decision and Control", 1996, Kobe, Japan

[17] J. N. Tsitsiklis, *Efficient Algorithms for Globally Optimal Trajectories*, IEEE Trans. on Automatic Control, Vol. 40, N.9, 1995