Automatic Synthesis Algorithms for Supervisory Controllers (Preliminary Report)

Marco Antoniotti　　　Bud Mishra
Robotics Laboratory
Courant Institute of Mathematical Sciences
New York University
719 Broadway
New York, NY, 10003, U.S.A.
marcoxa@cs.nyu.edu　　　mishra@nyu.edu

Abstract

In this paper we describe our experience with a prototype system capable of synthesizing Supervisor Controller Programs *based largely on the theory of discrete event systems (DES) first proposed by Ramadge and Wonham [13]. We augment the theory by also allowing continuous time trajectories modeling transitions between events. We illustrate our approach by an example,* the discrete control of a walking machine, *which poses some challenges on the applicability of the theory and finally, discuss some possible solutions.*

1 Introduction

In manufacturing and robotics, we are constantly struggling with two related tasks: *modeling* the devices and "processes," we want to control and more importantly, *implementing the necessary software components* that realize these models.

The modeling task has been solved in many cases, especially for linear time-invariant control problems [16], but not for more general (hence, realistic) systems in spite of their prevalence. In this paper, we focus on such generalizations that include many challenging and interesting applications.

The implementation task is still largely a question of individual skills. To the best of our knowledge, very little has been done toward the (partial) automation of the process that goes from a model of the system to the actual programs to be run on micro-controller boards and/or standard PLC's. Notable successful examples in this direction include: [1], [2] and [15].

The aim of our research is to produce a comprehensive computer environment for the synthesis of controller programs for a variety of manufacturing and robotics systems, while simultaneously tackling the modeling and the implementation tasks. In order to achieve this goal, our initial approach has been to sort through a large body of proposals and select (and devise) carefully the algorithms that efficiently implement the step from model to implementation. The effort also involves substantial theoretical work that deal with expressibility problem, query processing algorithms and probabilistic phenomena.

1.1 Problems Encountered

This paper deals only with our experience with an initial prototype and points out the practical obstacles that one encounters in trying to synthesize a controller for a real application (namely, a walking machine). The problems encountered can be classified into two broad categories: *linguistic* and *model based*.

Linguistic problems denote the difficulties we encountered trying to express some properties of our examples starting directly from the theoretical specification of DES. Model based problems denote a set of more fundamental difficulties which we found in the original DES formulation. In the later category, the primary problem seems to arise from an inadequate formulation of the close interplay between the discrete and the "underlying" continuous levels.

Other authors have applied the theory of DES to the modeling and control of systems, see [1] and [15]. While posing many interesting problems, especially with respect to the inherent complexity of the manageability of the systems[1], they avoid the typical ques-

[1]The algorithms used run against a *state space explosion*, which is mostly unavoidable.

tions involved in the construction of an integrated environment for the practitioners.

2 "Dreadful Gaits": Control Synthesis for a Walking Machine

We have chosen the problem of synthesizing the controller for a *walking machine* as a testbed for our research. Traditionally, the problem of walking machine control has been dealt with either as an *architectural problem* (see [6]) or as a standard control problem (see [11]).

From our point of view, the problem is interesting because it encompasses both the *discrete* (leg stances) and *continuous* (leg kinematics and dynamics) levels of control. The control of the kinematics and dynamics of a mechanical leg is typically done within the realm of classical continuous time control theory and yet, it is obvious that the dynamics equations *do* change when the leg is in contact with the ground, in contrast to the situation when the leg is moving freely in the air.

Problems like this are usually classified as **hybrid**[2].

Our synthesizer accepts a model of the legs (both continuous and discrete) and a set of goals (expressible in temporal logic) and automatically synthesizes a controller that controls the legs. The controlled walking machine exhibits behaviors that are guaranteed not to violate any of the desired goals. This class of behaviors of the legs are called "gaits" and are expected to depend on the leg model and desired goals. We also graphically simulate the gaits to gather insights about the formulation and hope to provide the designer feedback on how to make design changes. The complete system has been facetiously dubbed as "Dreadful Gaits," or more simply, DG.

2.1 Some Assumptions

In general, the class of problems of interest to us, share the following characteristics:

1. The hybrid model comprises a continuous and a discrete part.

2. The control of the discrete part of our model determines the kind of models (and their control policies) used at the continuous level (in a very similar way to the model used by Nicollin et al. in [10]). That is, we assume that the continuous part evolves until a discrete change happens which might also change the kind of model used in the continuous part.

3. We assume that *time* is "global". That is, we consider a global "clock" from which all the elements of our model (synchronously) read off the current time.

[2]For a different example, the classic *peg-in-the-hole* problem, see [9].

2.2 The Model

We model the discrete behavior of each leg with a standard finite state machine. The simplest model we would like to use is shown in Fig. 1. The five states correspond to the following activities of an individual leg.

- Start: The leg is in a rest position.

- Unload: The leg begins not to support any weight anymore.

- Recover: The leg is brought forward in a "flying motion".

- Load: The leg starts to bear weight.

- Drive: The leg thrusts forward the hip exerting a force on the ground.

- Slipping: The leg was not able to firmly stand on the ground.

Note that we do not specify final (or even "marked", in DES terminology) states in the model, since we assume that any sequence of transitions is acceptable[3].

In the spirit of DES theory, the events es, eu, er, el, ed and esl are all *controllable*, slip is instead *uncontrollable*.

Since we want to model a system with many legs we proceed in standard way by taking the *interleaving product* of the machines for each leg. This yields a "combined machine" that effectively represents the discrete behavior of the comprehensive, unregulated system. That is to say, in DES terminology, the language is $L(\mathcal{G})$ (or, more simply L).

For our testbed, we chose to introduce an uncontrollable event, slip, (see fig. 1), which indicates a condition where the leg has somehow lost the necessary "stance" on the ground[4]

[3]I.e. the language described by the state machine is *closed*.

[4]Note that the way to "sense" when such an event happens is by measuring some - sampled - *continuous* information such as the force exerted on the links. This raises many interesting questions about hybrid systems.

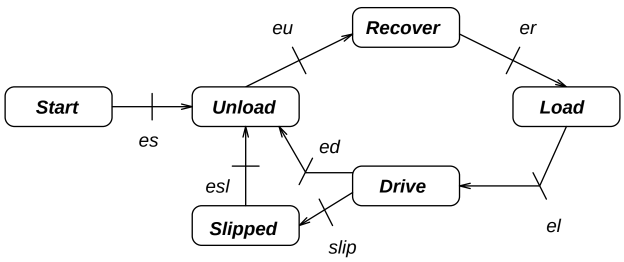

Figure 1: Model of the discrete transitions of a single leg with *uncontrollable* events slip and stop.

2.3 Supervisor Prototyping Methodology

First, we describe a simple methodology which forms the basis of our collection of software tools for supervisor synthesis. Fig. 2 depicts the typical steps that our collection of software tools are intended to provide. The definitions of *controllability* and *the supremal controllable set* (denoted by $K^{\uparrow}$) are taken from the standard DES terminology and will be defined explicitly subsequently.

This flowchart is evinced from the literature on DES theory. Yet many of the steps are not immediately realizable from the theory. They require further research before they could be embodied in a usable tool for the practitioners.

2.3.1 Applying the Theory

The model we described in fig. 1 yields a comprehensive model with 1296 states and 5184 transitions (for 4 legs). This comprehensive model represents the *physically admissible* behavior of the 4 leg system.

The next steps are to formulate a control law (or a *desired behavior*), represented by a language K and then to use this law to guide the synthesis of the supervisor[5].

There are two choices for this step:

1. Specify K as a *new* language,
2. Specify a set of *restrictions* on the comprehensive language L in order to define K.

[5]Formally a supervisor $\mathcal{S}$ is a couple (S, ϕ), where

- S is a state machine defined on the same alphabet of the system being controlled,
- $\phi : X \times \Sigma \rightarrow \{0,1\}^{\Sigma}$ where X is the set of states of S and Σ is the set of symbols recognizable in the system.

ϕ is called the *state feedback map* of the system. Its action is to "enable" (1) or "disable" (0) transitions in the system.

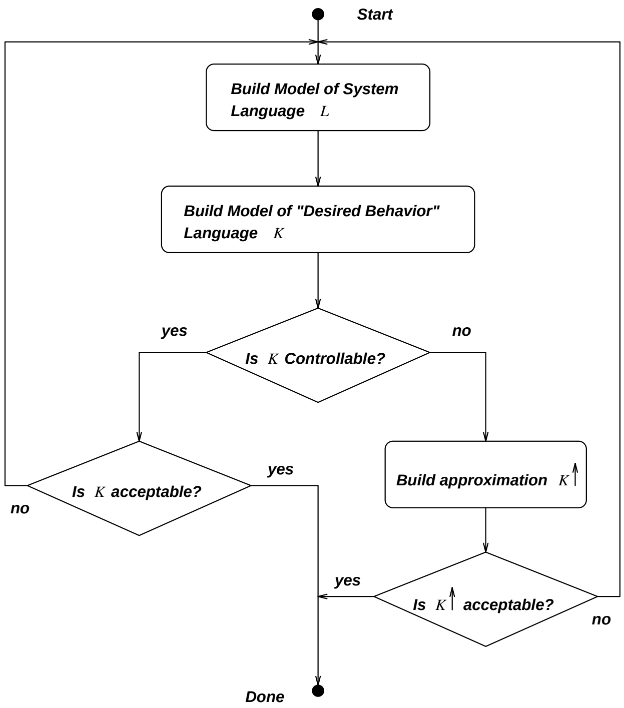

Figure 2: Schematic flowchart of the Supervisor building steps.

While the first alternative seems the most straightforward (and it is definitively suitable for "small" cases) it is very laborious and error prone. The main problems stem from the necessity to define a language K that must be a subset of L. This can be difficult specifically for large examples and requires extra machinery to test for language containment. Things get even more complicated when the original model is extended to take into account *uncontrollable events*: one is more likely to specify behaviors which result in trivial or useless controllers.

The second alternative seemed more viable. Roughly, we specify which states of the system do not meet some requirements and remove them from the comprehensive system. This has some appealing side effects.

- It does not violate the property that $K \subseteq L$.
- It allows us to specify "local" criteria for desirability of a state.
- It is easily implementable.

There are some drawbacks to this approach as will be pointed out later.

2.3.2 Controllability of the Discrete Model

One of the key notions in the DES theory is that of *controllability*, which is expressed in terms of L and K, using the following terminology.

If we indicate by $\bar{L}$ (respectively, $\bar{K}$) the language of the *prefixes* of all the elements of L (respectively, K), the notion of controllability in the discrete case amounts to ensuring that any initial segment of the desired behavior, followed by an uncontrollable event, must still constitute a prefix of our desired behavior K. Formally this is expressed with

$$\left(\bar{K}\Sigma_u \cap L\right) \subset \bar{K} \tag{1}$$

The algorithm that tests for controllability is basically a simple graph search and we augmented it with a diagnostic capability that will return a "counterexample run" of the system, that violates (1). The complexity of this algorithm is therefore linear in the size of the machine[6] for K.

2.3.3 Approximation of the Control Law

When the desired behavior K fails the controllability test, we know that we can compute an *approximation* to it, denoted $K^{\uparrow}$, which is called the *supremal controllable sublanguage* of K [13]. We implemented one of the algorithms described in [12].

Once an approximation is computed, the actual definition of the supervisor state feedback map is very simple[7].

2.4 Expressiveness Problems

There are several pragmatic problems in the synthesis of supervisors. While the "restriction" method for providing a viable first cut specification, is rather successful, we did notice some shortcomings of this approach. Namely, it is very difficult to express properties that are inherent to *subsequences* of events.

As an example of this difficulty, we were interested in modeling what physiologists and zoologists call *rear-to-front* waves in animal gaits [6]. In four and six legged animals, this gait pattern makes the recovering movement in a "train" of legs—either the left or the right legs, start from the rear most one and proceeds to the front one. This is typically a constraint on the sequencing of events in the leg system.

In order to express these sorts of constraints, we must resort to other formalisms. One such formalisms is *Temporal Logic*[8]. Temporal Logic (see for example [8] and [3]) has been heavily used in the field of *verification* of concurrent and real time systems [5]. In one such formalisms, CTL, the "rear-to-front" condition can be expressed as

$$\begin{aligned}&\mathbf{AG}.\big[(state_1 = \mathsf{drive} \wedge state_2 = \mathsf{recover})\\ &\quad\Rightarrow \neg\mathbf{EX}.(state_1 = \mathsf{unload} \wedge state_2 = \mathsf{recover})\big].\end{aligned}$$

The meaning of this formula is that whenever the rear leg (number 2) is *recovering*, the front leg (number 1) cannot make a direct transition from the drive to the unload state.

Removing this transitions from the machine representing the unregulated state is once again a matter of doing a simple graph traversal. Moreover, a lot of work has been done in order to efficiently manipulate within the TL formalisms (see [4]) and we will heavily draw on this body of work.

2.5 Our System

As a preliminary feasibility study, we have built a system that implements our approach as described earlier. Given the exploratory nature of our task, we chose to implement a rapidly prototyped system in Common Lisp. Such a choice had many advantages over a more traditional one, given the flexibility of the Lisp environment. Moreover, it does not hinder the actual production (through a "compilation" process) of low level Assembly or C code for some of the architectures currently used in our laboratory (Motorola MC68332[tm] boards and VxWorks[tm]).

Under the supervision of R. Wallace, our robotics laboratory has developed an inexpensive yet powerful technology of *mini actuators* [14]. In collaboration with this group, we have been designing direct drive walking machines and constructing these out of mini-actuators together with necessary simulation models and controllers. Fig. 3 shows prototype leg joints.

2.5.1 Example

In order to give a flavor of the current usage of our system, we give some excerpts of a session where we consider the behavior of one train of legs (i.e. a *front* (1) and a *rear* (2) leg).

The state machine representing the behavior of one leg is represented as follows[9]:

[6]Which, in turn and unfortunately, is *exponential* in the size of the its components.

[7]The details can also be found in [13].

[8]Abbreviated to TL, henceforth.

[9]The notation and the tricks used are standard Common Lisp. `leg2` represents the state machine for the rear leg; `s2` represents the relative `start` state and so on. `define-state-machine` is a simple macro that extends the language.

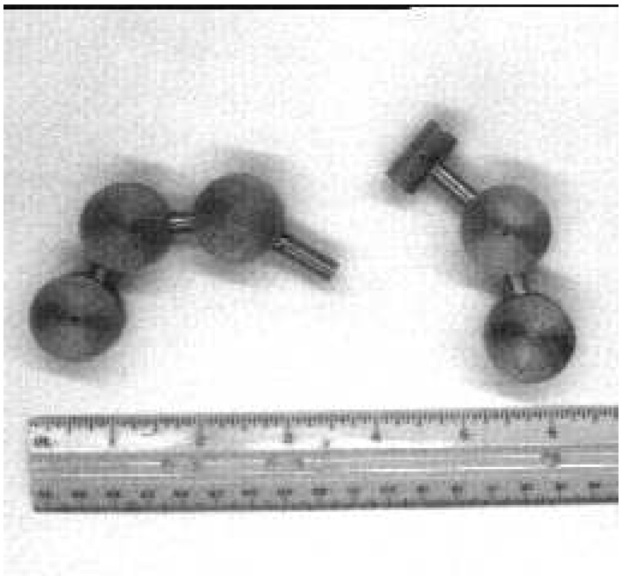

Figure 3: Prototypes of the mini actuator links for the legs of the Walking Machine.

```
(define-state-machine leg2
    :states (s2 r2 l2 d2 u2 sl2)
    :start s2
    :alphabet (es2 er2 el2 ed2 eu2 esl2 slip2)
    :uncontrollable (slip2)
    :delta ((s2 es2 u2) (r2 er2 l2)
            (l2 el2 d2) (d2 ed2 u2)
            (u2 eu2 r2)
            (d2 slip2 u2) (sl2 esl2 u2)
            )
    :final-states (s2 r2 l2 d2 u2 sl2)
    )
```

In order to specify the machine representing the interleaving of the discrete events we write

```
(define-state-machine legs
    :op (shuffle leg1 leg2))
```

which states that `legs` is the `shuffle` of the two machines at hand (`leg1` and `leg2`)

We build a representation for the desired behavior K by removing from the machine `legs` those states which are "inconsistent" with our aims. As an example, we are not interested in those situations when both legs are recovering: this state is represented as `(r1 r2)`.

The resulting language K is not controllable, hence we need to build an approximation for it. In this case the approximation algorithm terminates after two iterations. The results are as follows:

```
CMUCL 7> (omega-op K legs uncontrollable-events)
;; Debugging deleted...
>> OMEGA(0): removable states = ((R1 D2) (D1 R2)
                                 (D1 U2) (D1 SL2)
                                 (U1 D2) (SL1 D2))
---------------
;; Debugging deleted...
>> OMEGA(1): removable states = NIL
#<Representation for the approximation to K>
CMUCL 8>
```

From which we can immediately infer that the gait of the train of legs will be completely "stable", since the states `(r1 d2)` and `(d1 r2)` (which represent states of the train where the system is "unbalancing" in order to proceed) have been removed from the supervisor.

Some more testing by means of the graphical simulation shows that the system loses the necessary alternation between the front and the rear leg. The source of the problem is the "unconstrained choice" the supervisor makes in state `(l1 l2)` between which event to enable next.

A way around this is, for example to introduce extra *fairness* constraints in the specification of the desirable behavior K. Of course this raises more problems in terms of the definition of a viable specification language. Once again, TL seems to be a very good candidate.

2.5.2 Interactions with the Continuous Level

We tested our system through a simple graphical simulation. The necessities of the simulation brought up all the problems relative to the interaction between the *discrete* and the *continuous* levels of the walking machine system. For related ideas, see [10].

Our system simulates each leg through a Common Lisp function which is run at regular intervals. The only parameter we actually measure (we do not implement any continuous closed loop control scheme at this level) is the position of leg. According to the position and discrete state, the simulation advances the leg or keeps it in place.

The simulation architecture is straightforward. We produce the next enabled discrete event via the synthesized supervisor and run each leg at the same time. A leg may find itself in one of the following *meta* states:

1. It can make the transition,

2. It needs another transition which is not enabled,

3. It does not need a "discrete" transition, but it can move on "continuously" .

In case 1 the transition is made in (apparent) no time and it is reflected in the supervisor. In case 2 the leg is blocked in its "current" state; its position is not changed. In case 3 the leg state is not changed, but its position is.

We are currently researching ways to achieve the following goals: definition of a language for a smoother integration of the continuous level of a system; analysis of the algorithms for the synthesis of supervisors in the hybrid case; better system support and choices of target language (the NYU ADA-9X translator is a good candidate).

3 Final Remarks

We have presented an application of DES theory to a standard problem in robotics: the walking machine. The main goal was to understand the difficulties that arise in such applications and gain insights toward the realization of a robust and easy-to-use "supervisor compiler" for a wide range of robotics and manufacturing systems.

There are still many open problems which we expect to face before actually producing a viable software environment capable of aiding the practitioner in the production of code for PLC's or microcontroller programs. Our current experience points some out. Other are well known problems from the field of hybrid systems and are now under active investigation by researchers around the globe[7].

Acknowledgments

We are grateful to many of our colleagues for their help, advice and comments: Prof. Mohsen Jafari of Rutgers for getting us involved in similar problems arising in manufacturing; Profs. Ken Perlin, Jack Schwartz and Richard Wallace, all of NYU Robotics Lab for motivating several examples from human-computer interface, multimedia and robotics.

References

[1] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. Supervisory Control of a Rapid Thermal Multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059, jul 1993.

[2] A. Benveniste, M. Le Borgne, and P. Le Guernic. Hybrid Systems: the SIGNAL approach. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 230–254. Springer-Verlag, 1993.

[3] M. Browne, E. M. Clarke, D. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, c-35(12):1035–1044, 1986.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *5th LICS*, pages 428–439, 1990.

[5] J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors. *Real-Time: Theory in Practice (REX Workshop)*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[6] M. D. Donner. *Real-Time Control of Walking*, volume 7 of *Progress in Computer Science*. Birkhäuser, 1986.

[7] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[8] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[9] B. J. McCarragher and H. Asada. A Discrete Event Approach to the Control of Robotic Assembly Tasks. In *IEEE International Conference on Robotics and Automation*, pages 331–336. IEEE, 1993.

[10] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An Approach to the Description and Analysis of Hybrid Systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 149–178. Springer-Verlag, 1993.

[11] M. H. Raibert. *Legged Robots That Balance*. MIT Press, 1986.

[12] P. J. Ramadge and W. M. Wonham. On the Supremal Controllable Sublanguage of a Given Language. *SIAM J. Control and Optimization*, 25(3):637–659, may 1987.

[13] P. J. G. Ramadge and W. M. Wonham. Supervisory Control of a Class of Discrete Events Processes. *SIAM J. Control and Optimization*, 25(1):206–203, 1987.

[14] R. S. Wallace. Miniature Direct Drive Rotary Actuators. *Robotics and Autonomous Systems*, 11:129–133, 1993.

[15] R. A. Williams, B. Benhabib, and K. C. Smith. A Hybrid Supervisory Control System for Flexible Manufacturing Workcells. In *IEEE International Conference on Robotics and Automation*, pages 2551–2556. IEEE, 1994.

[16] W. M. Wonham. *Linear Multivariate Control – A Geometric Approach*. Springer–Verlag, 3rd edition, 1985.