HYBRID CONTROLLERS FOR ROBOTICS

BUD MISHRA[1]

Robotics Laboratory, Courant Institute, New York University

ABSTRACT

In this paper, we survey various theoretical and algorithmic questions related to a compiler that takes a specification of the desired behavior of a hybrid system (i.e., combining both discrete and continuous operations) and builds a hybrid supervisory controller for the system. A version of the compiler has been constructed at NYU with Marco Antoniotti and has been used in such areas as robotics and manufacturing. We briefly describe how the system works with an example of a controller for bipedal locomotion.

KEYWORDS: Controller, Temporal Logic, Walking Machine.

INTRODUCTION

We describe a research project (with collaborators from NYU, Rutgers and ICSI Berkeley) for rapid prototype construction of discrete controllers based on a practical approach that we have developed and implemented on small experimental robotics systems successfully. The research described here involves two components.

Theoretical Work: This involves formalizing the specification, synthesis, verification, simulation and query-processing of a very powerful hybrid controller (involving both continuous time and discrete event structure) for large scale robotics and manufacturing systems.

Experimental Work: This involves implementing a large distributed software system that automatically generates a controller (either directly or in a two step process, with the intermediate structure described by GRAFCETS) and the associated visualization and interface tools.

THEORETICAL FRAMEWORK

A crucial component of our controller synthesis algorithm is based on the *modal control* approach to the construction of supervisory controller for a discrete event system. However, we differ in one significant way: namely, by the specification formalism based on the classical propositional temporal logic.

[1]Address: 251 Mercer Street, New York, NY, 10003, U.S.A. E-mail: `mishra@nyu.edu`

We are grateful to many of our colleagues for their help, advice and comments: Dr. Marco Antoniotti of ICSI Berkeley who contributed to all the ideas described here, Prof. Mohsen Jafari of Rutgers, Prof. E.M. Clarke of CMU, Prof. P.K. Khosla of CMU, Dr. R. Kurshan of Bell Labs, Prof. R. Wallace of Lehigh, Mr. F. Hansen of NYU, and Dr. M. Teichmann also of NYU.

Discrete Event System

We start with a discussion of the original work of Ramadge and Wonham [9].

For our purpose, a *discrete event system* stands for any dynamical system that can be represented in terms of finite state automata. Thus our discrete event system is represented as an automaton $G = (U, X, f, x_0)$ where U = set of inputs/controls[2], X = set of states, $f: U \times X \to X$ = the state transition functions and $x_0 \in X$ represents the initial state of the system. We also allow each state to be labeled by a set of propositional formulas that hold true in that state. Classically, such a labeled finite state automaton is called a Kripke structure. The inputs, U is assumed to be a disjoint union of U_c = the set of controllable events and $U_u = U \setminus U_c$ = the set of uncontrollable events.

The modal controller works by simply disabling some subset of controllable events and the choice of this subset is based solely upon the current state of the dynamical system (i.e., plant). Equivalently, we may base the choice of the disabled controllable events upon some propositional (possibly modal/temporal) formula that holds in the current state. Note that, the controller has *no control* over the uncontrollable events and they cannot be disabled.

A controller for the given dynamical system is then simply specified by a map: $\eta : X \to \Gamma : x \mapsto \gamma_x$, where $U_u \subseteq \gamma_x \subseteq U$, and γ_x represents all the inputs and controls that have *not been disabled*. Such a γ_x is called an *admissible control* and $\Gamma \subseteq 2^U$ is the class of admissible controls.

Given a plant as defined earlier, we can describe the language generated by the plant (assuming it is a trim automaton with every state being a final state) by $M \subseteq U^*$. The controller's job is then to modify the plant's behavior by restricting the accepted language to some subset $L \subseteq M$, where L is to be specified by the user.

The discussion above should serve well to expose one of the major pragmatic problems with Ramadge-Wonham formalism. Namely, the task of accurately modeling the desired behavior by the language L is rather cumbersome. The problem is two folds: There may be more than one such language that should be acceptable to the designer. However, the choice is somewhat arbitrary. Furthermore, great care has to be taken in order not to restrict the language too strongly. Secondly, a good specification of L depends heavily on the designer's understanding of the plant model and the associated language M. In a hierarchically organized system, the designer makes relatively poor usage of the underlying structure. Furthermore, a minor change in any component can significantly alter the choice of the language L. In order to alleviate this problem, we propose using only a very high level temporal logic specification of the desired behavior. Starting with fairly general description of the plant model (e.g., in SADT and IDEFO), we construct automatically both the language M and a sub-language $L \subseteq M$ satisfying the temporal properties.

The second problem with the approach explained in this subsection, is that the language L may not be controllable, in the sense that it may not allow a supervisory controller achieving the language L. Ramadge and Wonham in their seminal work[9] solve this problem elegantly by proposing the concept of a *supremal controllable* sub-

[2]These represent the very primitive events in the system.

language $L^{\uparrow} \subseteq L \subseteq M$, for which there is always a supervisory controller. The supremal controllable sub-language $L^{\uparrow}$ is simply the largest controllable sub-language of L, and since the controllable sub-languages of L is nonempty (the empty-set $\emptyset \subseteq L$ is controllable) and is closed under set union operation, it can be easily defined as

$$L^{\uparrow} = \bigcup \Big\{ K : (K \subseteq L) \ \wedge \ (\overline{K}U_u \cap M \subseteq \overline{K}) \Big\}.$$

This also leads to a fixed point characterization of $L^{\uparrow}$ which allows it to be computed fairly efficiently. However, in the approach we propose, the controllable language $L \subseteq M$ satisfying the temporal properties can be computed directly and the synthesis of supervisor is then fairly straightforward.

Temporal Logic

The logic that we use to specify is a propositional temporal logic of branching time, called *CTL* (*Computation Tree Logic*)[7]. The syntax for CTL is as follows: Let $\mathcal{P}$ be the set of all the atomic propositions in a given language, $\mathcal{L}$. Then (1) Every atomic proposition P in $\mathcal{P}$ is a formula in CTL. (2) If f_1 and f_2 are CTL formulas, then so are $\neg f_1$, $f_1 \wedge f_2$, $\mathbf{AX}\ f_1$, $\mathbf{EX}\ f_1$, $\mathbf{A}[f_1\ \mathbf{U}\ f_2]$ and $\mathbf{E}[f_1\ \mathbf{U}\ f_2]$.

In this logic the propositional connectives $\neg$ and $\wedge$ have their usual meanings of *negation* and *conjunction*, respectively. The temporal operators **AX** and **EX** are the unary *nexttime* operators. The intuitive meaning of $\mathbf{AX}\ f_1$ (respectively, $\mathbf{EX}\ f_1$) is that f_1 holds in every (respectively, in some) immediate successor state of the current state. The temporal operators **AU** and **EU** are the binary (*strong*) *until* operators. For the sake of readability, we write $\mathbf{A}[f_1\ \mathbf{U}\ f_2]$ and $\mathbf{E}[f_1\ \mathbf{U}\ f_2]$ instead of the customary $\mathbf{AU}(f_1, f_2)$ and $\mathbf{EU}(f_1, f_2)$. The intuitive meaning of $\mathbf{A}[f_1\ \mathbf{U}\ f_2]$ (respectively, $\mathbf{E}[f_1\ \mathbf{U}\ f_2]$) is that for every computation path (respectively, for some computation path), there exists an initial prefix of the path such that f_2 holds at the last state of the prefix and f_1 holds at all other states along the prefix.

We also use the following syntactic abbreviations:

$f_1 \vee f_2 \equiv \neg(\neg f_1 \wedge \neg f_2)$, $f_1 \rightarrow f_2 \equiv \neg f_1 \vee f_2$, **and** $f_1 \leftrightarrow f_2 \equiv (f_1 \rightarrow f_2) \wedge (f_2 \rightarrow f_1)$.

$\mathbf{AF}\ f_1 \equiv \mathbf{A}[\text{True}\ \mathbf{U}\ f_1]$ **and** $\mathbf{EF}\ f_1 \equiv \mathbf{E}[\text{True}\ \mathbf{U}\ f_1]$

$\mathbf{AG}\ f_1 \equiv \neg\, \mathbf{EF}\ \neg f_1$ **and** $\mathbf{EG}\ f_1 \equiv \neg\, \mathbf{AF}\ \neg f_1$

The semantics of a CTL formula is as follows. A CTL structure is a triple $\mathcal{M} = (S, R, \Pi)$ where (1) S *is a finite set of states*, (2) R *is a total binary relation on* S *(*$R \subseteq S \times S$*) and denotes the possible transitions between states*, and (3) Π *is an assignment of atomic proposition to states, i.e.* $\Pi : S \rightarrow 2^{\mathcal{P}}$.

A *computation path* is an infinite sequence of states $(s_0, s_1, s_2, \ldots)$ such that $\forall_i [\langle s_i, s_{i+1} \rangle \in R]$. For any structure $\mathcal{M} = (S, R, \Pi)$ and state $s_0 \in S$, there is an *infinite computation tree* with root labeled s_0 such that $s \rightarrow t$ is a directed edge in the tree *if and only if* $\langle s, t \rangle \in R$.

The truth in a structure is expressed by $\mathcal{M}, s_0 \models f$, meaning that the temporal formula f is satisfied in the structure $\mathcal{M}$ at state s_0. The semantics of temporal formulas is defined inductively as follows:

$$
\begin{array}{lll}
s_0 \models P & \textit{iff} & P \in \Pi(s_0). \\
s_0 \models \neg f & \textit{iff} & s_0 \not\models f. \\
s_0 \models f_1 \wedge f_2 & \textit{iff} & s_0 \models f_1 \text{ and } s_0 \models f_2. \\
s_0 \models \mathbf{AX}\ f_1 & \textit{iff} & \text{for all states } t \text{ such that } \langle s_0, t\rangle \in R, t \models f_1. \\
s_0 \models \mathbf{EX}\ f_1 & \textit{iff} & \text{for some state } t \text{ such that } \langle s_0, t\rangle \in R, t \models f_1. \\
s_0 \models \mathbf{A}[f_1\ \mathbf{U}\ f_2] & \textit{iff} & \text{for all computation paths } (s_0, s_1, s_2, \ldots), \\
 & & \quad \exists_{i\geq 0}[s_i \models f_2 \wedge \forall_{0\leq j<i}[s_j \models f_1]]. \\
s_0 \models \mathbf{E}[f_1\ \mathbf{U}\ f_2] & \textit{iff} & \text{for some computation path } (s_0, s_1, s_2, \ldots), \\
 & & \quad \exists_{i\geq 0}[s_i \models f_2 \wedge \forall_{0\leq j<i}[s_j \models f_1]].
\end{array}
$$

The Model Checker for CTL can now be thought of as an algorithm that determines the satisfiability of a given temporal formula f_1 in a model $\mathcal{M}$. Thus, given a plant description with the language $M \subseteq U^*$, our first goal is to find a sub-language $L \subseteq M$ such that if the computation paths of the plant's computation tree is restricted to the strings of L, then the resulting "restricted" computation tree is a model for the CTL specification, denoting the desired overall behavior.

However, in the general setting, the associated operators are non-monotonic; the synthesis problem may not have any or unique solution and may not be computationally tractable. In a recent paper, we have shown that in fact the problem in its full generality ("Unrestricted CTL Supervisory Synthesis Problem") is NP-complete [5]. In particular, we have shown that disjunctions of arbitrary CTL formulas, do not admit an efficient algorithm for the synthesis of a supervisor. The resulting implementation in the Control-D system [1], therefore either allowed only disjunctions with only one *path* disjunct, or used some heuristics to synthesize the supervisor.

CONTROLLER FOR A WALKING MACHINE

Using Control-D, a hybrid controller for the walking machine was implemented in Common Lisp with Motif-based graphical animation and was extensively tested. The desired behavior of the system was specified in CTL.

Each leg was modeled by a finite state machine with five states corresponding to the following activities of an individual leg. Start (`s`): The leg is in a rest position. Unload (`u`): The leg begins not to support any weight anymore. Recover (`r`): The leg is brought forward in a "flying motion". Load (`l`): The leg starts to bear weight. Drive (`d`): The leg thrusts forward the hip exerting a force on the ground. Slipping (`sl`): The leg was not able to firmly stand on the ground.

The events `es`, `eu`, `er`, `el`, `ed` and `esl` (corresponding respectively to 'end-of-start,' 'end-of-unload,' 'end-of-recover,' 'end-of-load,' 'end-of-drive,' and 'end-of-slipping') are all *controllable*, slip ('begin-of-slipping') is in contrast *uncontrollable*. For our testbed, we chose to introduce an uncontrollable event, slip, which indicates a condition where the leg has somehow lost the necessary "stance" on the ground.

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.

Now, we need to ensure that the bipedal system (when augmented with a controller) guarantees various desirable properties. For example:

Safety Condition: $\neg\,\mathbf{EF}\,(\mathtt{r}_i \wedge \mathtt{r}_j), \quad i,j \in \{1,2\}, i \neq j.$

> It is not the case that for some computation path, there exists a state on the path where both legs are recovering simultaneously. In other words, we wish to avoid situations where both legs are "up in the air."

Liveness Condition: $\mathbf{AG}\,[\mathtt{d}_i \rightarrow \mathbf{AF}\,(\mathtt{d}_j)] \quad i,j \in \{1,2\}, i \neq j.$

> For each state in which one of the legs drives, it must be the case that in every computation path starting there, for some state on the path the other leg drives. Thus, we wish to have each leg contact the ground infinitely often.

"Rear-to-Front" Wave: $\mathbf{AG}\,[(\mathtt{d}_i \wedge \mathtt{r}_j) \rightarrow \neg\,\mathbf{EX}\,(\mathtt{u}_i \wedge \mathtt{r}_j)] \quad i,j \in \{1,2\}, i \neq j.$

> Both legs alternate in driving in a wave like manner.

In order to give a flavor of the current usage of our system, we present an excerpt of a session where we consider the behavior of one train of legs (i.e. a *front* (1) and a *rear* (2) leg). Figure 1 shows the state machine representing the behavior of one leg and the machine representing the interleaving of the legs (i.e. `legs` is the `shuffle` of the two machines (`leg1` and `leg2`)).

```
(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)
    )
```

(a)

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

(b)

```
>> (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>
>>
```

(c)

Figure 1: (a) State machine for one leg. (b) Combined state machine. (c) Results derived by Control-D.

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 shown in (c) of Figure 1. We can now 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.

FINAL REMARKS

While there is a tremendous push in robotics and manufacturing to simplify the very basic building blocks (e.g, RISC: Reduced Intricacies in Sensing and Control, Reactive Robotics, or Reconfigurable Robotics), there is a complementary drive to be able to automatically combine a large number of such building blocks to obtain a huge gain in functionality (e.g., Swarm Robotics or Collective Robotics). We believe that in this setting, our techniques offer many fundamental contributions.

References

[1] M. Antoniotti. *Synthesis and Verification of Controllers for Robotics and Manufacturing Devices with Temporal Logic and the Control-D System.* Ph.D. Thesis, N.Y.U., NY 1995.

[2] M. Antoniotti, M. Jafari and B. Mishra. "Applying Temporal Logic Verification and Synthesis to Manufacturing Systems." *Proc. IEEE SMAC*, Vancouver, 1995.

[3] M. Antoniotti and B. Mishra. "Automatic Synthesis Algorithms for Supervisory Controllers." In *Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, Troy, NY, October 10–12, 1994.

[4] M. Antoniotti and B. Mishra. "Discrete Event Models + Temporal Logic = Supervisory Controller: Automatic Synthesis of Locomotion Controllers," *Proc. ICRA'95*, Nagoya, 1995.

[5] M. Antoniotti and B. Mishra. *The Supervisor Synthesis Problem for unrestricted* CTL *is* $\mathcal{NP}$-*complete*, Tech Report, NYU No. 707, No. `ICSI TR-95-062`, Nov 1995.

[6] E.M. Clarke, E.A. Emerson, and A.P. Sistla. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications." *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[7] E.A. Emerson and E.M. Clarke. "Characterizing Properties of Parallel Programs as Fixpoints." *Proc. ICALP'85*, LNCS 85, Springer 1981.

[8] B. Mishra and E.M. Clarke. "Hierarchical Verification of Asynchronous Circuits using Temporal Logic." *Theoretical Comp. Sc.*, 38:269–291, 1985.

[9] P.R. Ramadge and W.M. Wonham. "Supervisory Control of a Class of Discrete-Event Processes." *SIAM J. Cont. Opt.*, 25:206–230, 1987.

[10] P.R. Ramadge and W.M. Wonham. "Modular Feedback Logic for Discrete Event Systems." *SIAM J. Cont. Opt.*, 25, 1987.

[11] W.M. Wonham and P.R. Ramadge. "On the Supremal Controllable Sublanguage of a Given Language." *SIAM J. Control Optim.*, 25:637–659, 1987.