Discrete Event Models + Temporal Logic = Supervisory Controller: Automatic Synthesis of Locomotion Controllers

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 address the problem of the synthesis of controller programs for a variety of robotics and manufacturing tasks. The problem we choose for test and illustrative purposes is the standard "Walking Machine Problem," a representative instance of a real hybrid *problem with both logical/discrete and continuous properties and strong mutual influence without any reasonable separation. We aim to produce a "compiler technology" for this class of problems in a manner analogous to the development of the so-called "Silicon Compilers" for the VLSI technology. To cope with the difficulties inherent to the problem, we resort to a novel approach that combines many key ideas from a variety of disciplines: namely,* Discrete Event Supervisory Systems *[14]* Petri Nets *approaches [8], [10] and* Temporal Logic *[5].*

1 Introduction and Our Goals

This paper describes a "*Controller Synthesis System*" and its application in the development of a *walking machine*. Our (admittedly ambitious) goal is to build a comprehensive controller synthesis system based on the Ramadge and Wonham's DES theory [14] and a form of temporal logic [5] widely used in the field of *verification*.

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.

However, the problem is not as straightforward as it may seem at the first glance. We need to address the problem resulting from inadequate formulations of the close interplay between the discrete and the "underlying" continuous levels. In the walking machine case, the reciprocal influences between these levels must be taken into account in order to produce a reasonable integrated controller. The examples appearing in the DES literature (mostly related to manufacturing) seem to be tractable otherwise. We believe that, in the case of walking machine, the difficulties in specifying the desired behavior arise from the fact that the system is inherently *tightly coupled*. In contrast, most examples seen in the DES literature appear to be *loosely coupled*.

Other works in DES theory [1] and [17] pose many interesting problems, especially with respect to the inherent complexity of the manageability of the systems[1]. For some related ideas, we refer the readers to [2] and [7].

This paper is organized as follows. In the next section, we describe the model of the walking machine we are using. We discuss its DES components at length and its kinematics/dynamics briefly. Finally, we describe how our work fits in with other ongoing research at our Laboratory. The next two sections are devoted

*This research has been partially supported by the National Science Foundation under grant number CCR-9202900.

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

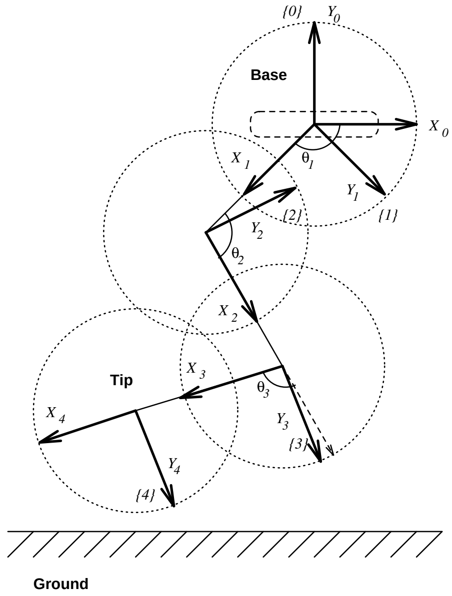

Figure 1: *Leg with frame assignments. Note that* $\{B\} = \{0\}$ *and* $\{T\} = \{4\}$*, following standard terminology.*

i = link#	α_i	a_i	d_i	θ_i
1	0	0	0	θ_1
2	0	l_1	0	θ_2
3	0	l_2	0	θ_3
4	0	l_3	0	0

Table 1: *The Denavit-Hartenberg Link Parameters assignment.*

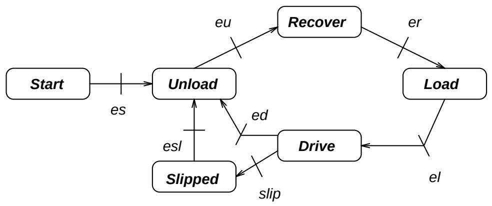

Figure 2: *Model of the discrete transitions of a single leg with an* uncontrollable *event (*slip*).*

to an explanation of the interplay between the temporal logic used, a *Model Checker* for it and how it simplifies the use of DES theory for the walking machine system. We conclude the paper by pointing out some problems we encountered and worthy of further exploration.

2 Walking Machine Model

The Walking Machine Model we are building is a four legged system based on Microcontroller and Direct Drive Technology. We model the system as comprising of a *discrete* and of a *continuous* layer. Our system follows standard modeling techniques proposed in [6], [12] and [15].

2.1 Leg Model

We use a standard three link leg model which we analyzed as a planar manipulator. Figure 2.1 shows the geometry of the leg and the assignment of the standard coordinate frames. The Denavit-Hartenberg parameters for the Leg are shown in table 1.

At this point, it is fairly straightforward to derive the kinematic and dynamic equations for the leg in order to build position and force controllers for the joints. Yet, our main interest remains to be in exploring the interactions between the "low level" continuous control and the "discrete" synchronization scheme of the whole walking machine.

2.2 Discrete Event Model

Most studies on walking machines use a *Finite State Machine* (FSM) approach to the problem of "high level" synchronization ([6], [12]). Using DES Theory we are able to take into account possible undesirable behaviors. The finite state model we use for a single leg is depicted in fig. 2. The six states Start, Unload, Recover, Load, Drive and Slipping correspond to different "movements" of the leg. E.g. in the Recover state the leg is moving "forward" without contacts with the ground; in the Slipping state, the leg has just lost the stance on the ground and is conceivably not supporting the hip anymore. In the spirit of DES theory, the events es, eu, er, el, ed and esl[2] are all *controllable*, slip is instead *uncontrollable*.

The walking machine is modeled by slightly different equations in each state (with the possible exception of the Slipping state, for which we assume no model). The role of the "high level" Discrete Controller is to choose the appropriate set of control laws for each discrete state. Such Discrete Controller is synthesized using the Supervisor Synthesis schemes of DES[3] [13].

We build the actual FSM for the DES "plant" language L (in DES terminology) by taking the *shuffle*

[2]The prefix e- is intended to mean "end of".

[3]We assume familiarity with the standard DES terminology. Refer otherwise to [14] for a survey of the original DES Theory.

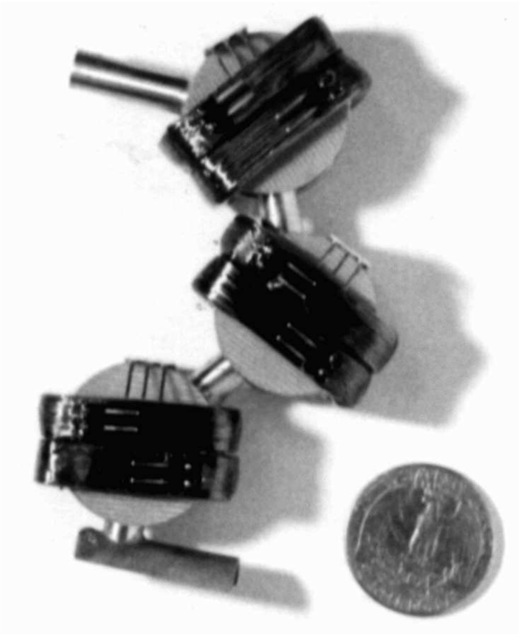

Figure 3: *Prototypes of the mini actuator links for the legs of the Walking Machine built by Richard Wallace and Fred hansen.*

product of the four distinct FSM's, one for each leg. This yields a FSM with 1296 states and 5184 transitions. How this model will be used to synthesize the actual DES Supervisor will be deferred to the next sections.

2.3 System Software and Simulation

Our *controller compiler* was rapidly prototyped 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 "post processing" process) of low level Assembly, C or Ada modules 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* [16]. In collaboration with this group, we have been designing direct drive walking machines and constructing these out of mini-actuators. Fig. 3 shows prototype leg joints.

3 Controller Synthesis

We modified the standard DES Theory approach to the Supervisor Synthesis Problem by using a *Temporal Logic* formalism for the specification and verification of *properties* of the desired language K.

3.1 $\mathcal{CTL}$ Specifications and Supervisory Synthesis

The Temporal Logics we use are the standard *Branching Time Logics* of the $\mathcal{CTL}$ family [5]. We are not the first to explore the use of Temporal Logic in the DES realm (see [9]), but our approach (and the Logic used) is different.

$\mathcal{CTL}$ is a Logic whose semantics is defined in terms of a *Kripke Structure* which can easily be reinterpreted in terms of FSM's.

The well formed formulas of $\mathcal{CTL}$ are listed in table 2 along with their semantics in terms of the underlying FSM. This logic has been extensively used in the field of Verification of VLSI circuits [3, 11]. Its usefulness comes from the existence of a linear time *Model Checking* algorithm that works by recursively labeling the underlying finite state machine. When coupled with hierarchical and *State Space Compression* techniques, such algorithm becomes usable in a wide range of cases [4].

We specify the *desired discrete behavior K* by marking off "undesirable states". We do so by using a modified version of the Model Checking Algorithm. A $\mathcal{CTL}$ formula that specifies this for a train of legs (left or right) is the following

$$\begin{aligned}\mathbf{AG}(\ & \neg\texttt{state}([\mathsf{Drive}_1, \mathsf{Drive}_2]) && \wedge \\ & \neg\texttt{state}([\mathsf{Recover}_1, \mathsf{Recover}_2]) && \wedge \\ & \neg\texttt{state}([\mathsf{Slipping}_1, \mathsf{Slipping}_2])).\end{aligned}$$

The meaning of this formula is simply the statement of a *state avoidance problem*. We do not want the system to be in a state where both legs of a train are both recovering, or driving, or (worse) slipping.

Another property that we would like to enforce is what physiologists and zoologists call *rear-to-front* waves in animal gaits [6]. This is a constraint on the sequencing of events in the leg system. Specifying this sequencing constraint for a train of legs is rather easy using $\mathcal{CTL}$.

$$\begin{aligned}&\mathbf{AG}.\big(\texttt{state}([\mathsf{Drive}_1, \mathsf{Recover}_2]) \\ &\quad \Rightarrow \ \neg\mathbf{EX}.(\texttt{state}([\mathsf{Unload}_1, \mathsf{Recover}_2]))\big).\end{aligned}$$

The meaning of this formula is that whenever the rear leg (number 2) is *recovering*, the front leg (number 1) cannot start unloading.

The Supervisory Synthesis theory does guarantee that a Supervisor exists and that it is "minimally restrictive" [13]. Yet no guarantee is given that the supervisor will maintain all the properties that we may

Syntax	Semantics	Description
	Base Formulæ	
p	$p \in \mathcal{A}(s)$	A proposition
$f_1 \vee f_2$	$f_1 \in \mathcal{A}(s)$ or $f_2 \in \mathcal{A}(s)$	A disjunction
$f_1 \wedge f_2$	$f_1 \in \mathcal{A}(s)$ and $f_2 \in \mathcal{A}(s)$	A conjunction
$\neg f$	$f \notin \mathcal{A}(s)$	A negation
$f_1 \Rightarrow f_2$	$f_1 \notin \mathcal{A}(s)$ or $f_2 \in \mathcal{A}(s)$	An implication
	Temporal Formulæ	
$\mathbf{EX}(f)$	$f \in \mathcal{A}(s')$ and s' is a successor state of s	f will be true in some next state
$\mathbf{AX}(f)$	$f \in \mathcal{A}(s')$ for every s' successor of s	f will be true in all the next states
$\mathbf{E}[f_1\,\mathbf{U}\,f_2]$	If $s_0, s_1, \ldots, s_n$ is a sequence of states and at each of them $f_1 \in \mathcal{A}(s_i)$ for $i < n$ and $f_2 \in \mathcal{A}(s_n)$	There is a sequence of states where f_1 holds *until* f_2 will.
$\mathbf{A}[f_1\,\mathbf{U}\,f_2]$	For any sequence of states $s_0, s_1, \ldots, s_n$ at each of them $f_1 \in \mathcal{A}(s_i)$ for $i < n$ and $f_2 \in \mathcal{A}(s_n)$	There is a sequence of states where f_1 holds *until* f_2 will.
$\mathbf{EF}(f)$	There is *a sequence* of states where f will *eventually* hold (this is actually an abbreviation for $\mathbf{E}[\mathrm{True}\,\mathbf{U}\,f]$)	This formula represents a *potential* event.
$\mathbf{AF}(f)$	For *any sequence* of states f will *eventually* hold (this is actually an abbreviation for $\mathbf{A}[\mathrm{True}\,\mathbf{U}\,f]$)	This formula represents a *necessary* event.
$\mathbf{EG}(f)$	There is *a sequence* of states, f will *always* hold (this is actually an abbreviation for $\neg\,\mathbf{AF}(\neg f)$)	The formula f will always hold on some path.
$\mathbf{AG}(f)$	For *all sequences* of states, f will *always* hold (this is actually an abbreviation for $\neg\,\mathbf{EF}(\neg f)$)	This formula states a *global* and *invariant* property of the system.

Table 2: *Syntax and informal Semantics for $\mathcal{CTL}$. Note that $\mathcal{A}$ is an assignment of propositions and formulæ to each state. A proposition p (or, recursively, a formula f) is* **True**, *or* holds *in a state s when $p \in \mathcal{A}(s)$.*

specify. We use the Modified Model Checker again to debug the Synthesized Supervisor. This was actually our original motivation for the use of the $\mathcal{CTL}$.

As an example, our first attempts at the Supervisory Synthesis for a train of legs, kept removing the states $[\mathsf{drive}_1, \mathsf{recover}_2]$ and $[\mathsf{recover}_1, \mathsf{drive}_2]$. We were able to discover this fact only by means of graphical simulation. The Model Checker turned out to be an excellent tool for the debugging, significantly reducing the turnaround time. Moreover we were able to prove fancier properties for the controlled system. As an example, we could check some *liveness* conditions such as

$$\mathbf{AG}(\mathtt{state}([\mathsf{drive}_1, ?]) \Rightarrow \mathbf{AF}(\mathtt{state}([?, \mathsf{drive}_2]))),$$

and the fact that the supervised system were still able to reach the states $[\mathsf{drive}_1, \mathsf{recover}_2]$ and $[\mathsf{recover}_1, \mathsf{drive}_2]$.

3.2 Continuous Control Constraints

The "desired behavior" of the Walking Machine system is obviously not completely specified by the constraints we posed on the discrete level. The transitions between states are ruled by measurements taken from sensors. We used only position information in order to allow the transition from one state to the other of the discrete control. This is sufficient to get nice simulations and already poses interesting problems for the control synthesis procedure.

The geometric model that we use for our Walking Machine is depicted in fig. 3.2. By following the

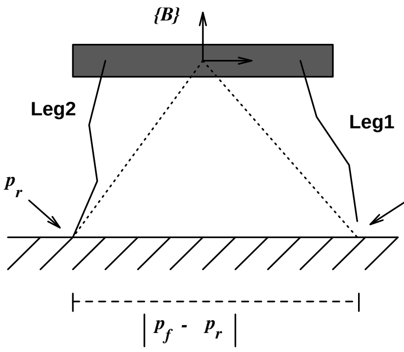

Figure 4: *Simplified Geometric Model of the Walking Machine. $\{B\}$ is a coordinate frame set in the body. All measurements are taken with respect to it.*

standard methodology, we obtain a discrete supervisor that allows for

$$\mathbf{EF}.\mathtt{state}([\mathsf{load}_1, \mathsf{load}_2])$$

to be true for one train of legs[4]. In this state, the supervisor has to choose which transition to make next to either state $[\mathsf{load}_1, \mathsf{drive}_2]$ or $[\mathsf{drive}_1, \mathsf{load}_2]$. Since both transitions are controllable and not forbidden by the supervisor, the system might end up "taking a step longer than the leg" by cycling one too many times

[4]With respect to the "start" state. Actually we can prove

$$\mathbf{EG}(\mathtt{state}([\mathsf{Load}_1, \mathsf{Load}_2]) \Rightarrow \mathbf{EX}(\mathbf{EF}(\mathtt{state}([\mathsf{Load}_1, \mathsf{Load}_2])))).$$

through the [$\mathsf{load}_1, \mathsf{load}_2$] state [5]. In Petri Net terminology, this is called a *conflict* and it really represents a situation where "extra information" is needed (or assumed) in the system.

We solved this problem by studying some algorithms that will allow us to identify these "conflict states" in order to reduce the actual behavior of the system[6] to a "geometrically acceptable" one. In this task we are doing something similar to [10].

When we consider a train of two legs, the transition er1 for the front leg (leg 1) in state [$\mathsf{Recover}_1, \mathsf{Drive}_2$] causes the difference in the position of the feet $\Delta(p_{feet}) = \mid p_f - p_r \mid$ to change in the following way

$$\Delta(p_{feet})[\mathsf{Load}_1, \mathsf{Drive}_2] = \Delta(p_{feet})[\mathsf{Recover}_1, \mathsf{Drive}_2] + \frac{1}{2}\mathtt{step},$$

if we assume the rear leg moved a "very small" distance,

$$\Delta(p_{feet})[\mathsf{Load}_1, \mathsf{Drive}_2] = \Delta(p_{feet})[\mathsf{Recover}_1, \mathsf{Drive}_2] + 2\,\mathtt{step}$$

if we assume the both legs moved (almost) the full `step` distance.

We can repeat this reasoning for all the other states. This "interval" computation for the transitions can be reconstructed from the description of the state in which it is taking effect, hence we can set up a simple graph traversal which will mark the states where a given constraint *could* (but not necessarily would) be violated. In our case the simple constraint we would like to maintain is

$$\Delta(p_{feet}) \leq \ell,$$

where ℓ is derived from the mechanics of the Walking Machine.

The graph traversal simply maintains for each node traversed a possible maximum and minimum value for $\Delta(p_{feet})$ while following only the controllable transitions enabled by the Supervisor. Whenever there are two or more such transitions outgoing a state s and one of the reachable states (or the state itself) possibly violates the constraint, then s is marked as a "choice point". Eventually, we will be able to equip the runtime of the system with appropriate tests that will avoid the controllable transitions that in specific occasions (usually after a few tours around a cycle in the state space) would violate the constraint.

[5]This argument applies also when we consider two legs in alternation – left and right – and *virtual legs*.

[6]Note that we will be giving up some of the properties of the language found by the approximation algorithm. I.e. we will be imposing further restrictions on the *supremal* language.

3.3 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[7]:

```
(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`). To remove the undesirable states we run the Model Checker, which, as a side effect, marks the states that do not satisfy the formula.

```
CMUCL 4> (model-check legs
            '(AG (and (not (state (d1 d2)))
                      (not (state (r1 r2)))
                      (not (state (sl1 sl2))))))
NIL
```

The `NIL` result tells us that the unregulated shuffle does not satisfy the property.

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 = ((D1 SL2) (SL1 D2))
---------------
;; Debugging deleted...
>> OMEGA(1): removable states = NIL
#<Representation for the approximation to K>
CMUCL 8>
```

4 Conclusions and Open Problems

We have presented an application of DES theory to a standard problem in robotics: the Walking Machine. Our goal was to build an easy-to-use "supervisor compiler" system for a wide range of robotics and

[7]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.

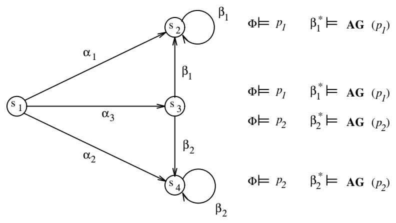

Figure 5: *A case where a naïve $\mathcal{CTL}$ Supervisor Synthesis approach fails. In state s_2 we have that $\mathbf{AG}(p_1)$ is true for language β_1^*, while in state s_4 we have $\mathbf{AG}(p_2)$ true for β_2^*.*

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.

One open problem that we are investigating concerns the *direct synthesis* of the supervisory map from a set of $\mathcal{CTL}$ specifications; i.e. without specifying K as a language. This brought up some interesting questions about the satisfiability of $\mathcal{CTL}$ formulæ under the control action of the supervisor: see fig. 4 for an example. Suppose we want a supervisor that achieves a language satisfying the formula

$$\mathbf{AX}(\mathbf{AG}(p_1)) \vee \mathbf{AX}(\mathbf{AG}(p_2))$$

under the assignment $\mathcal{A}(s_2) = p_1$ and $\mathcal{A}(s_4) = p_2$. Then the maximal controllable sublanguage K is not unique and hence not well-defined. The problem can be traced to non-monotonicity of $\mathcal{CTL}$ modal operators. We are investigating a solution based on a restriction on the logic called $\mathcal{CTL}^-$ [11], which circumvents this problem. The resulting algorithm also reduces the complexity of the synthesis by one order of magnitude.

Acknowledgments We thank Mohsen Jafari of Rutgers, Fred Hansen and Richard Wallace of NYU Robotics Lab for their help and suggestions.

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] 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.

[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] L. E. Holloway and B. H. Krogh. Synthesis of Feedback Control Logic for a Class of Controlled Petri Nets. *IEEE Transactions on Automatic Control*, 35(5):514–523, may 1990.

[9] J.-Y. Lin and D. Ionescu. Analysiss and Synthesis Procedured of Discrete Event Systems in a Temporal Logic Framework. In *International Symposium on Intelligent Control*. IEEE, 1992.

[10] 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.

[11] B. Mishra and E. M. Clarke. Hierarchical Verification of Asynchronous Circuits Using Temporal Logic. *Theoretical Computer Science*, 38:269–291, 1985.

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

[13] 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.

[14] P. J. G. Ramadge and W. M. Wonham. The Control of Discrete Events Systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[15] S. Song and K. J. Waldron. *Machines that Walk: The Adaptive Suspension Vehicle*. MIT Press, 1989.

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

[17] 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.