Discrete synchronization of hybrid systems[1]

Paulo Tabuada and George J. Pappas

Department of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104
{tabuadap,pappasg}@seas.upenn.edu

Abstract

Control theory is currently faced with new paradigms and challenges that fall beyond traditional problems. Nowadays applications tend to be distributed, and require partial synchronization among their various subsystems. In this paper, we give initial steps towards discrete synchronization problems for systems which are compositions of several, possibly distributed, hybrid systems. Such problems arise frequently in the coordination of multi-agent systems, where each agent is modeled as a hybrid system. This results in control problems where the model is the composition of decoupled subsystems, but the specification is coupled across subsystems. A centralized solution to this problem requires computing the product hybrid system resulting in state explosion. We alternatively consider decentralized solutions to such discrete synchronization problems. Partially decentralized synchronization is achieved if each subsystem is allowed to communicate with the subsystems it needs to partially synchronize with. The required communication between agents is provided by modeling abstractions of the remaining agents. These abstractions, which are property-dependent, are then used to derive local controllers using global, but minimal, observations.

1 Introduction

Large scale systems are usually compositions of smaller subsystems. Furthermore, as communication and computation becomes more powerful and inexpensive, the system architecture of modern large scale applications, such as automated vehicles [20], and air traffic control [18], becomes more and more decentralized. In order to address the fundamental complexity issues involved in the analysis and design of such systems, compositional approaches to modeling, analysis, and control are vital.

Of particular interest is the composition of large numbers of so-called multi-agent systems, where each individual agent may be an aircraft, a car, a robot, or a molecule. In this paper, we focus on agents that can be modeled as hybrid systems. A collection of agents can then be simply modeled as the product or asynchronous composition of the corresponding hybrid systems. Even though the behavior of one hybrid system does not influence the behavior of another hybrid system, we consider global system specifications which enforce inter-agent coordination.

In particular, we address discrete synchronization problems for compositions of hybrid systems. Consider, for example, two hybrid systems modeling two aircraft. The two aircraft should not be concurrently in the same discrete state LAND. Similarly, given an automated platoon of cars on a highway, we should ensure that at least one of the cars in on the LEADER mode. This type of synchronization is *partial* as the agents may need to synchronize their actions in certain locations of the state space, and can be completely free in other locations.

The problem of controlling a hybrid system in order to achieve a discrete specification has been considered in [15], where supervisory control of hybrid systems is developed. In [2], the emphasis is placed on the computation of discrete abstractions of the continuous dynamics that will allow the design of discrete controllers for the hybrid system. Other discrete approaches to the control of hybrid systems include [3, 10]. In the above approaches, the controller design is centralized, and a centralized controller, as pictured in Figure 1a, is essential.

For purely discrete systems, the problem of local control has been successfully addressed by characterizing which specifications can be achieved with local controllers [14]. However, since both actuation and especially observations are local in nature, not many specifications can be implemented in a decentralized fashion.

[1]This research is partially supported by the National Science Foundation Information Technology Research Grant CCR01-21431, DARPA MICA Contract Number N66001-01-C-8076 and DARPA/AFRL Software-Enabled Control F33615-01-C-1848 grant.

In [13, 1] a more general decentralized setup, displayed in Figure 1b, is proposed by allowing communication between the controllers as an attempt to relax the restrictive conditions of decentralized control. Other related approaches to problems of decentralized control of discrete event systems include [6, 22, 5, 19].

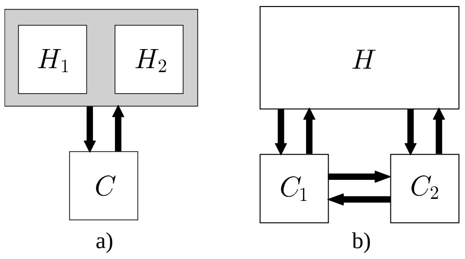

Figure 1: a) Centralized architecture, where the product of hybrid systems T_1 and T_2 is regarded as a single system. b) Decentralized architecture, where local controllers C_1 and C_2 communicate through an architecture specified a priori.

Our approach differs from the above as we exploit the decentralized nature of the plant by means of both *abstraction* and *composition*. Consider a composition of two hybrid systems H_1, H_2 which in general can be regarded as a single plant by computing the product hybrid system $H_1 \times H_2$ defined in some reasonable sense. In this product hybrid system, the aforementioned approaches could be applied, but the combinatorial explosion renders such approaches unfeasible, as the number of agents grows. Furthermore, it is also not possible, in general, to develop two local controllers with only *local* observations for each agent [14].

We advocate an intermediate approach that allows local, decentralized controllers, but possibly global observations. Such an approach allows the local controller of H_1 to have knowledge, if needed, of the discrete state of H_2 by equipping the agents with communication mechanisms. Depending on the specification, such approach will simplify controller design, however in the worst case a fully centralized solution may be obtained. A similar approach is described in [13], however no specification is considered.

In order to decompose the control problem, that is, in order to know *what* knowledge about H_2 should H_1 have, we determine an *abstraction* O_2 of H_2. This abstraction is sufficient to specify the synchronization as well as to design the local controller C_1 for H_1. Naturally, this process is also applied to H_2 by determining the relevant abstraction O_1 of H_1 for the local controller design. By proceeding this way, we reduce the design complexity and avoid rendering the problem unfeasible (as in the case of local actuation with local observation) by allowing the controller to observe the necessary information of the remaining systems through their abstractions. As scalability of the design algorithm is an important concern, we cannot build such an abstraction from the product $H_1 \times H_2$, instead we exploit compositionality. By computing an abstraction of each individual hybrid system we can obtain, by composition, an abstraction of $H_1 \times H_2$, see for example [16] for details on the compatibility between abstractions and compositionality. As a special case of our approach we recover the possibility of local control with local observation when the minimal required abstraction of the remaining agents reveal that no inter agent communication is necessary.

2 Discrete Specifications

2.1 Abstracting Specifications

A discrete synchronization specification on a group $\{H_1, H_2, H_3, \ldots, H_n\}$ of hybrid systems is a specification of the discrete behavior of the product hybrid system $H_1 \times H_2 \times H_3 \times \ldots \times H_n$. The naive approach to solve such a synchronization problem is to design a controller for the product hybrid system that will enforce the specification. However, this is not an efficient approach due to the state explosion involved in computing the product hybrid system. For simplicity of presentation we will consider only two hybrid systems H_1 and H_2, however the results extend to any number of hybrid systems. Furthermore, as the specification is discrete, we will only need to work with the discrete part of the hybrid systems that we model as two transition systems $T_1 \subseteq Q_1 \times L_1 \times Q_1$ and $T_2 \subseteq Q_2 \times L_2 \times Q_2$. Transition systems T_1 and T_2 encode the different modes of operation of hybrid systems H_1 and H_2, respectively, as well as the possible switching behavior between these modes. A discrete synchronization specification on H_1 and H_2 can now be modeled as a binary function S on the transition systems:

$$S : T_1 \times T_2 \longrightarrow \{0, 1\} \tag{1}$$

The specification map S, is interpreted as follows: if $S(a, b) = 1$ for $a = (q_1, \sigma_1, q_1') \in T_1$ and $b = (q_2, \sigma_2, q_2') \in T_2$ then the discrete evolution described by the pair (a, b) is allowed by the specification and otherwise forbidden if $S(a, b) = 0$. Other specification modeling alternatives include sublanguages of the language generated by $T_1 \times T_2$ as is usual in supervisory control of discrete event systems [12]. However, this form of specification, rules out branching specifications since two transition systems may generate the same language and yet not have the same branching structure. Control design for branching specifications is discussed in [8]. We will return to this discussion in Section 4 where other specification mechanisms will be discussed including temporal logic.

Our goal can now be succinctly described as follows: can we find a minimal abstraction O_2 of T_2 such that specification S can be given on $T_1 \times O_2$? Working on $T_1 \times O_2$ is clearly advantageous if this space is of lower cardinality then $T_1 \times T_2$. When it is possible to abstract T_2, there will exist a surjective map $\phi : T_2 \longrightarrow O_2$ describing which information on T_2 can be safely ignored while ensuring that a new specification map $\overline{S}$ exists on $T_1 \times O_2$. This map $\overline{S}$, represents the same specification as S, and therefore it has to agree with S in the sense that $S(a, b) = \overline{S}(a, \phi(b))$ for any $(a, b) \in T_1 \times T_2$. This equality can also be pictorially described by the following commutative diagram:

$$\begin{tikzcd} T_1 \arrow[d, "id"'] & \times & T_2 \arrow[r, "S"] \arrow[d, "\phi"'] & \{0,1\} \\ T_1 & \times & O_2 \arrow[ur, "\overline{S}"'] & \end{tikzcd} \tag{2}$$

We are not only interested in any abstraction O_2 for which S can be described as $\overline{S}$, but we are interested in the smallest such O_2, as we now formally state:

Problem 2.1 *Given a specification on the product transition system $T_1 \times T_2$ described as a map $S : T_1 \times T_2 \longrightarrow \{0, 1\}$, find the smallest set (abstraction) O_2 obtained from T_2 by a surjective map ϕ, for which there exists a specification $\overline{S} : T_1 \times O_2 \longrightarrow \{0, 1\}$ agreeing with S in the sense that $S(a, b) = \overline{S}(a, \phi(b))$ for any $(a, b) \in T_1 \times T_2$.*

To state the solution of the previous problem we introduce some necessary notation. Given a set A and an equivalence relation $R \subseteq A \times A$ on A we denote by A/R the set of equivalence classes defined by the relation R on A. The natural projection from A to A/R taking a point in A to its equivalence class in A/R is denoted by π_R.

Proposition 2.2 *Let T_1 and T_2 be two transition systems and S a specification $S : T_1 \times T_2 \longrightarrow \{0, 1\}$ on the product transition system. Denote by $R \subseteq T_2 \times T_2$ the following equivalence relation on T_2:*

$$(r, r') \in R \quad \Leftrightarrow \quad \forall t \in T_1 \quad S(t, r) = S(t, r')$$

Then the solution to Problem 2.1 is given by $O_2 = T_2/R$, $\phi = \pi_R$ and $\overline{S}$ is the function on $T_1 \times O_2$ induced by S.

Proof: We first show that the given solution makes diagram (2) commutative. Let o and o' be two points in the same equivalence class denoted by $[o]$, then $\overline{S}(t, [o]) = \overline{S}(t, \pi_R(o)) = S(t, o)$. Since $(o, o') \in R$ we have that $S(t, o) = S(t, o')$, by definition of R, therefore $\overline{S}(t, \pi_R(o')) = S(t, o') = S(t, o) = \overline{S}(t, \pi_R(o))$ which shows commutativity of (2). To show that O_2 is the smallest such set, consider for the sake of contradiction that there exist another triple $(O_2', \phi', \overline{S}')$ making (2) commutative and such that $O_2' \subset O_2$. Then there exist $(r, r') \in T_2$ such that $\phi(r) \neq \phi(r')$ and $\phi'(r) = \phi'(r')$. Commutativity of (2) requires that $S(t, r) = \overline{S}'(t, \phi'(r)) = \overline{S}'(t, \phi'(r')) = S(t, r')$. But this implies that $(r, r') \in R$ and therefore $\phi(r) = \phi(r')$, a contradiction. ∎

When the abstraction of T_2 turns out to be trivial (the set O_2 contains just one point) we can decouple the specification for transition system T_1 from the other transition system(s). This implies that the problem can be solved by a local controller with local observations.

The determination of the equivalence relation R can be a computationally intensive problem since it requires the analysis of the elements of the set $T_1 \times T_2$. However, for event synchronization, an explicit formula for R exists, which we now describe in detail.

2.2 Event Synchronization

When the specification S requires only synchronization of events, more detailed descriptions of the relation R can be obtained. Given two transition systems T_1 and T_2 we say that the product transition system $T_1 \times T_2$ is synchronized on the events $L_1 \cap L_2$ if, when system T_1 executes transition (q, l, q'), system T_2 executes a transition (p, l, p') simultaneously, for a $l \in L_1 \cap L_2$. Event synchronization leads to the following characterization of R:

Proposition 2.3 *Let T_1 and T_2 be two transition systems and S a specification requiring only synchronization of the events $L_1 \cap L_2$. Then equivalence relation $R \subseteq T_2 \times T_2$ is defined by the following equivalence classes:*

- $A_l = \{(p, l, p') \in T_2 \ : \ l \in L_1 \cap L_2\}$,
- $B = \{(p, l, p') \in T_2 \ : \ l \in (L_1 \cup L_2) \backslash (L_1 \cap L_2)\}$.

Proof: Let us denote by T the equivalence relation defined in Proposition 2.3 and show that for every $(t, t') \in T$ we have $(t, t') \in R$ as defined in Proposition 2.2. Let $t, t' \in A_l$, then $S(a, t) = S(a, t')$ for any $a \in T_1$ since $t = (p_1, l, p_2), t' = (p_1', l', p_2')$ and $l = l' \in L_1 \cap L_2$. Consequently $(t, t') \in R$. For $t, t' \in B$ we have $S(a, t) = 0 = S(a, t')$ for $a = (q, l, q') \in T_1$ such that $l \in L_1 \cap L_2$. When $l \in (L_1 \cup L_2) \backslash (L_1 \cap L_2)$ we have $S(a, t) = 1 = S(a, t')$ which shows that for any $a \in T_1$ we have $S(a, t) = S(a, t')$ for $t, t' \in C$.

We now show the converse, that is, for any $(t, t') \in R$ we have $(t, t') \in T$. For the sake of contradiction, we will assume that $(t, t') \in R$ and $(t, t') \notin T$. Since $(t, t') \notin T$ one of the following situations must occur:

1. $l, l' \in L_1 \cap L_2 \wedge l \neq l'$,
2. $l \in L_1 \cap L_2 \wedge l' \notin L_1 \cap L_2$,
3. $l \notin L_1 \cap L_2 \wedge l' \in L_1 \cap L_2$.

From $(t, t') \in R$ it follows that for any $a \in T_1$:

$$S(a, t) = S(a, t') \tag{3}$$

but if $l, l' \in L_1 \cap L_2$, (3) is only satisfied if $l = l'$ which is not the case as it implies that $(t, t') \in T$. For the second case, (3) is also not true as can be seen by choosing $a = (q, l, q')$ which leads to $S(a, t) = 1 \neq 0 = S(a, t')$. A similar argument works for the last case so that we have shown that $S(a, t) \neq S(a, t')$ for some $a \in T_1$, contradicting the fact that $(t, t') \in R$. ■

As we are interested in minimizing the upper bound of the controller design complexity we can use the plant cardinality as a meaningful measure of such complexity. We now assume that $L_2 \neq L_1 \cap L_2$ or $L_1 \neq L_1 \cap L_2$ and use Proposition 2.3 to show that the cardinality of O_2 is given by $|L_1 \cap L_2| + 1$. It then follows that the design for H_1 and H_2 has complexity $max\{|T_1|(|L_1 \cap L_2| + 1), |T_2|(|L_1 \cap L_2| + 1)\}$, respectively, where as if it is performed on the product transition system it has complexity $|T_1||T_2|$. Simple computations now show, that the abstraction methodology reduces the design complexity when:

$$|L_1 \cap L_2| \leq \frac{|T_1||T_2|}{max\{|T_1|, |T_2|\}} - 1 \tag{4}$$

By considering $|T_1| = |T_2| = 10$ we get $|L_1 \cap L_2| \leq |T_1| - 1 = 9$ meaning that if the synchronizing events are less then 9, the maximum complexity of the design process is reduced. However, the upper bound changes considerably as the number of agents increase.

3 Hybrid Synthesis

We now apply the previous results to design the switching logic of hybrid systems H_1 and H_2. We will assume that the guards associated with different transition are disjoint. If this is not the case, then we can always remove the nondeterminism by shrinking the guards. We start by reinterpreting the specification map S in terms of the hybrid systems. Since the discrete transitions are triggered by the continuous dynamics, we interpret $S(a, b) = 1$ for $a = (q_1, e, q_2) \in T_1$ and $b = (p_1, l, p_2) \in T_2$ as the discrete part of the state of hybrid systems H_1 and H_2 being q_1 and p_1, respectively, while the continuous part of the state belongs to the guards which enable transitions labeled by e and l, respectively. In this setting we also interpret a transition of the form (q_1, τ, q_2), where τ is (the empty string) a distinguished element not in L_1 as denoting a continuous evolution such that the continuous state does not enter any guard intersecting the invariant of mode q_1. With this interpretation of S we now see that designing the switching logic for H_1 simply amounts to disable the transitions $a = (q_1, e, q_2)$ such that $\overline{S}(a, \phi(b)) = 0$ for any $b \in T_2$. This is achieved by replacing each guard G of hybrid system H_1 with:

$$G \wedge \overline{S}(a, o)$$

where a is the transition of H_1 enabled by guard G obtained from the continuous and discrete state information of H_1 as already mentioned, and o is an element of O_2, communicated by H_2, representing the current abstract state of hybrid system H_2. The switching logic of H_2 is similarly designed by using an abstraction of T_1.

We note that, as a consequence of the assumption that the guards can be modified, the minimal abstraction required for the specification also represents the minimal amount of information required to design the controllers. This design scheme, therefore ensures, that one obtains the minimal communication local controllers for each agent. It is also important to mention, that as the specification is purely discrete, the interaction between the agents is also discrete and simply determines the enabling or disabling of the guards. When the guards are not available for design, the discrete communication between agents would then trigger continuous controllers specially designed for meeting the transition specifications. How to design continuous controllers for such discrete transition specifications is still an open problem. For the special case of affine control systems defined on invariants described by simplexes, an elegant solution has been proposed in [5]. An alternative approach to this problem would consider not the discrete transition system associated with the hybrid systems, but a discrete abstraction representing the discrete transitions for which continuous controllers have already been designed [7]. This would ensure that every discrete specification could be meet by the hybrid system.

4 Temporal logic as a specification mechanism

In this section we return to the problem of defining and representing discrete specifications for the synchronization of several hybrid systems. Both representations discussed so far, subsets of transition systems as described by (1) and sublanguages are not scalable with the number of agents. In fact, as the number of agents increases, so does the product hybrid system modeling their behavior and extracting a subset of the transition system or a sublanguage becomes a very complex and difficult procedure. We therefore need succinct specification mechanisms that do not require deep knowledge of the particular details of the plant. Based on these considerations, we advocate the use of temporal logic for defining the synchronization of hybrid systems as introduced by Pnueli in [11] for the specification of computer programs. In the context of synthesis, the use of logic as a specification mechanism is receiving some attention as described by the research activity described in [4, 9, 6].

In our context one could, for example, use Linear time Temporal Logic (LTL) to specify the synchronization of the hybrid systems. Choosing the set of atomic propositions P to be the union of the discrete state sets Q_i, (for each of the n agents) one recursively defines LTL formulas as:

- **true**, **false**, p, $\neg p$ are LTL formulas for all $p \in P$;
- if φ_1 and φ_2 are LTL formulas, then $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are LTL formulas;
- if φ_1 and φ_2 are LTL formulas, then $\bigcirc\varphi_1$ and $\varphi_1 U \varphi_2$ are LTL formulas.

As usual, implication $\varphi_1 \Rightarrow \varphi_2$ is defined as the abbreviation of $\neg\varphi_1 \vee \varphi_2$. The operator $\bigcirc$ is read as "next", with the meaning that the formula it precedes will be true in the next time step. The second operator U is read as "until" and the formula $\varphi_1 U \varphi_2$ specifies that φ_1 must hold until φ_2 holds. From the until operator one defines other two operators commonly used:

- $\Diamond\varphi = \textbf{true}\ U\,\varphi$, which is read as "eventually" and requires that φ will become true some time in the future.
- $\Box\varphi = \textbf{false}\ U\,\varphi$, which reads "always" and requires that φ is true for all time.

As the set of atomic propositions is the union of the discrete state sets, LTL formulas are interpreted over sequences of discrete state sets: $\sigma : \mathbb{N} \longrightarrow 2^P$ as follows:

For any $p \in P$, LTL formulas φ_1, φ_2, and $i \in \mathbb{N}$:

- $\sigma(i) \models p$ iff $p \in \sigma(i)$,
- $\sigma(i) \models \neg p$ iff $p \notin \sigma(1)$,
- $\sigma(i) \models \varphi \wedge \varphi_2$ iff $\sigma(i) \models \varphi_1$ and $\sigma(i) \models \varphi_2$,
- $\sigma(i) \models \varphi \vee \varphi_2$ iff $\sigma(i) \models \varphi_1$ or $\sigma(i) \models \varphi_2$,
- $\sigma(i) \models \bigcirc\varphi_1$ iff $\sigma(i+1) \models \varphi_1$,
- $\sigma(i) \models \varphi_1 U \varphi_2$ iff $\exists j \geq i$ such that for all k, $0 \leq k \leq j$ $\sigma(k) \models \varphi_1$ and $\sigma(j) \models \varphi_2$.

finally we say that a sequence σ satisfies formula φ iff $\sigma(0) \models \varphi$.

LTL allows to express properties that go beyond traditional control theoretical specifications. One can model conflict resolution as $\Box\neg(q \wedge q')$ for $q \in Q_1$ and $q' \in Q_2$ which asserts that it is always true that agent 1 will not be at state q while agent 2 is at state q'. If one replaces q and q' by the modes LAND, one recovers the aircraft example of the introduction. One can also model several type of ordinal constraints such as $\Box(q \Rightarrow \bigcirc q')$ which specifies that agent 2 must be at state q' in the next time step after agent 1 has been in state q. This allows to express complex ordering requirements involved in sequences of tasks to be executed by several agents.

After defining the specification as an LTL formula, controller design proceeds by refining that formula into the Buchi automaton B, that recognizes all the strings satisfying the formula. There are several algorithms available for this construction and we point the interested reader to the tutorial [21]. Once B has been constructed, one needs to determine the intersection of its language $\mathcal{L}(B)$ with the language $\mathcal{L}(T_1 \times T_2)$ generated by the product of the transition systems which can be performed as described in [17]. This provides a new automaton that now serves as a discrete transition specification for the continuous dynamics of the hybrid automata. Although LTL is a more expressive specification mechanism than the subsets of transition systems used in Section 2, a similar approach can be developed to simplify controller design. In this case we would abstract transition system T_2, not to retain the information provided by the map S (as in Section 2), but to retain the information contained in the language $\mathcal{L}(B) \cap \mathcal{L}(T_1 \times T_2)$. This is captured in the following expression:

$$\mathcal{L}(S) \cap \mathcal{L}(T_1 \times T_2) = \mathcal{L}(S) \cap \mathcal{L}(T_1 \times O_2)$$

that should be compared with diagram 2.

5 Conclusions

In this paper we gave preliminary steps towards the discrete synchronization of multi-agent hybrid systems. We considered decoupled hybrid systems working concurrently and coupled by synchronization specifications at the level of discrete states. We showed how local controllers with possible global observations can be designed through the use of abstraction and compositional methods. When the specifications are modeled by event synchronization we determine when it is preferable to design several local controllers with global, but minimal inter agent communication than to design a single, centralized controller. The discrete controllers are then interpreted as specifications for the continuous controlled dynamics in each discrete state of the several hybrid systems. We also discussed how specifications defined by subsets of transition systems and sublanguages do not scale with the number of agents and proposed temporal logic as an alternative specifications mechanism. The synthesis procedure that allows to design controllers based on temporal logic specifications has also been outlined.

References

[1] G. Barrett and S. Lafortune. Decentralized supervisory control with communicating controllers. *IEEE Transactions on Automatic Control*, 45(9):1620–1638, September 2000.

[2] A. Chutinan and B. H. Krogh. Computing aproximating automata for a class of hybrid systems. *Mathematical and Computer Modelling of Dynamical Systems*, 1998. Special Isuue on Discrete Event Models of Continuous Systems.

[3] J. M. Davoren, T. Moor, and A. Nerode. Hybrid control loops, A/D maps and dynamic specifications. In Claire Tomlin and Mark R. Greenstreet, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Sience. Springer-Verlag, 2002.

[4] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[5] L.C.G.J.M. Habets and J. H. van Schuppen. Control of piecewise-linear hybrid systems on simplices and rectangles. In Claire Tomlin and Mark R. Greenstreet, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Sience. Springer Verlag, 2002.

[6] S. Jiang and R. Kumar. Supervisory control of discrete event systems with CTL* temporal logic specifications. In *Proceedings of the of the 40th IEEE Conference on Decision and Control*, Orlando, FL, December 2001.

[7] T. J. Koo, G. J. Pappas, and S. Sastry. Mode switching synthesis for reachability specifications. In M. D. Di Benedetto and A. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 333–346. Springer-Verlag, 2001.

[8] P. Madhusudan and P.S. Thiagarajan. Branching time controllers for discrete event systems. *Theoretical Computer Science*, 274:117–149, March 2002.

[9] Z. Manna and P. Wolper. Synthesis of communication processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6:68–93, 1984.

[10] T. Moor, J. Raisch, and S.D. O'Young. Discrete supervisory control of hybrid systems based on l-complete approximations. *Journal of Discrete Event Dynamical Systems*, 12(1):83–107, 2002.

[11] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Providence, RI, November 1977.

[12] P. J. Ramadage and W. M. Wonham. The control of discrete event systems. *Proceedings of IEEE*, 77(1):81–98, 1989.

[13] K. Rudie, S. Lafortune, and F. Lin. Minimal communication in a distributed discrete-event control system. In *Proceedings of the American Control Conference*, pages 1965–1970, San Diego, CA, June 1999.

[14] K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.

[15] J. A. Stiver, X. D. Koutsoukos, and P. J. Antsaklis. An invariant based approach to the design of hybrid control systems. *International Journal of Robust and Nonlinear Control*, 11(5):453–478, April 2001. Special Issue on Hybrid Systems in Control.

[16] Paulo Tabuada, George J. Pappas, and Pedro Lima. Composing abstractions of hybrid systems. In Claire Tomlin and Mark R. Greenstreet, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Sience. Springer-Verlag, 2002.

[17] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume Volume B: Formal Methods and Semantics, pages 133–192. Elsevier Science Publishers, 1990.

[18] C. Tomlin, G. J. Pappas, and S. Sastry. Conflict resolution for air traffic management : A study in mutiagent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, April 1998.

[19] S. Tripakis. Undecidable problems of decentralized observation and control. In *Proceedings of the 40th IEEE Conference on Decision and Control*, Orlando, FL, 2001.

[20] P. Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, 1993.

[21] Pierre Wolper. Constructing automata from temporal logic formulas: A tutorial. In E. Brinksma, H. Hermanns, and J. P. Katoen, editors, *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[22] K.C. Wong, J.G. Thistle, R.P. Malhame, and H.H. Hoang. Supervisory control of distributed systems: conflict resolution. *Discrete Envent Dynamical Systems: Theory and Applications*, 10(1/2):131–186, 2000.