On the Design of Hybrid Control Systems Using Automata Models

Dang Van Hung * and Wang Ji**

International Institute for Software Technology
The United Nations University, P.O.Box 3058, Macau
email: {dvh, wj}@iist.unu.edu

Abstract. The paper gives a systematic way for the development of hybrid control systems, i.e. to refine specifications written in *DC* (Duration Calculus) into automata models. Firstly, *DC* formulas are extended with iteration form and a technique for deriving plant automata from *DC* formulas is presented. Then, a specification of control automata can be synthesized from plant automata with respect to requirements , based on a necessary and sufficient condition for a plant automaton and a requirement to have a control automaton. Water tank and Gas burner examples are demonstrated to illustrate our method.

1 Introduction

In this paper, we shall deal with hybrid control systems which consist of continuous plants controlled by decision makers via controllers. Typical hybrid systems can be found widely in the areas of robots, process control, aviation and so on. Because of safety critical properties of these systems, many efforts have been involved from computing science and control theory to get a well founded methodology for specification, design and analysis of hybrid control systems [1, 5]. Our aim is to provide a systematic way associated with formal techniques to support the refinement of hybrid systems from specifications written in *DC* (Duration Calculus) into automata models.

The model of hybrid systems used in this paper is taken from [3, 4, 9]. It consists of three distinct levels: Decision Maker, Controllers and Plant. The decision maker and controllers communicate via the interface that translates signals into symbols for the decision maker to use, and symbols into command signals for the controller input. A decision maker (control program) reads data of the plant provided by a controller (or sensor), computes the next control law, and imposes it on the controllers to control the plant. The plant will continue to use this control law until the next such intervention. The plant, the controllers

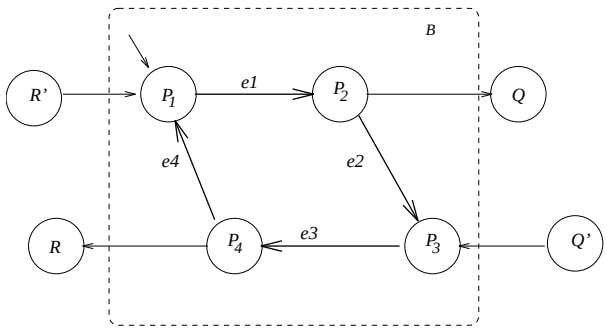
* On leave from the Institute of Information Technology, Nghia Do, Tu Liem, Hanoi, Vietnam.

** On leave from Department of Computer Science, Changsha Institute of Technology, Changsha 410073, P.R. China. email: wj@dns.cit.whnet.edu.cn. Partly supported by National NSF and National Hi-tech Programme of China.

and the interface are taken together to result in a so-called discrete state system *DES plant*.

The challenge is to develop methodologies which, given a performance specification and system description, extract control programs which will force the plants to meet their performance specifications. The key point is to link the different abstraction levels. We use the stepwise refinement approach to develop a systematic way for dealing with the problem. Starting with a top level specification and a description of the components of a system, we refine specifications of the system in more detail according to the physical laws and control laws in the problem domain. Iterate this step until it determines a plant automaton, of which each state consists of plant states controlled by the same controllers such that the necessary data of the plant given to the decision maker can be decided on entering the states. After obtaining the plant automaton, it is time to determine which control laws are needed to force the plant in which situations to ensure that the plant automaton has the allowable behaviors. It can be determined also whether a transition is caused by one of: time elapsing, disturbance or the control program. From this observation, a specification of the decision maker can be derived and the control program is designed from the specification.

Water Tank Example [7]. The aim is to design a decision maker to open and to close the valve such that the water level is maintained between 68 and 76. Let $w(t)$ be the water level at time t. One description of the system can be depicted as the automaton in Fig. 1.



$D1:$ the valve is closed — $D2:$ the valve is open

$P1 : 68 \leq w \leq 76 - \delta \wedge D1$ — $P2 : 76 - \delta < w \leq 76 \wedge D1$

$P3 : 68 + \delta \leq w \leq 76 \wedge D2$ — $P4 : 68 \leq w < 68 + \delta \wedge D2$

$Q : w > 76 \wedge D1,\ Q' : w > 76 \wedge D2$ — $R : w < 68 \wedge D2,\ R' : w < 68 \wedge D1$

$e1, e3$: time elapsing — $e2$: open the valve, $e4$: close the valve

Fig. 1. Automaton of Water Tank

The behaviors that satisfy the requirement $\forall t. 68 \leq w(t) \leq 76$ are represented by the graph B (indicated by the box B in Fig.1). In order to achieve these behaviors, started from $P1$, the automaton needs to receive from the decision maker the command 'open the valve' when it is in $P2$ and the command 'close

the valve' when it is in $P4$. On the other hand, on entering the states $P2$ and $P4$, the plant needs to send a signal to the decision maker (say, e.g. the name of the states $P2$ and $P4$ resp.). The decision maker, on receiving the signals, issues the commands 'open the valve' with the delay time less than $t1$ time units or 'close the valve' with the delay time less than $t2$ time units depending on the signals which are $P2$ and $P4$ respectively. The automaton achieving this behavior is depicted in Fig. 2a. The plant controlled by the decision maker (which is the parallel composition of the plant and the decision maker) is represented by the automaton depicted in Fig. 2b, which is the same as the graph B. In the figure, the transition named by the pair (f, e) indicates that the transition f of the decision maker occurs in parallel with the transition e of the plant. ϵ is the idle action. Here, for simplicity, we have assumed that the communication takes no time, and that on receiving a command the controllers can change the state of the plant automaton immediately. One thing should be noticed here is that it takes some time for the decision maker to give the commands. Thus, the roles of the automaton of the decision maker and of the DES plant automaton are not symmetrical.

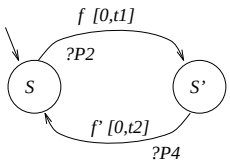


Fig. 2a

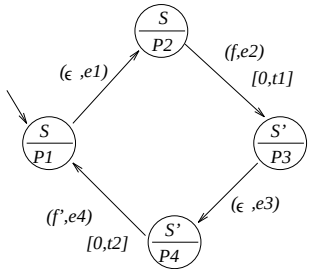


Fig. 2b

From the example we note the following: (1) From a description of the plant behaviors in the form of plant automaton, and from the set of the behaviors of the plant automaton that satisfy the requirement, a specification of the decision maker can be derived such that if its behaviors satisfy the specification, then the controlled plant will satisfy the requirement; (2) The DES plant may give no outputs, and transitions in the DES plant may have no inputs. However, it needs to be such that the pair of the sequence of inputs (commands) and the sequence of outputs of its behaviors that satisfy the requirement is distinguishable from that of the behaviors that do not satisfy the requirement. Hence, a possible approach to the design of hybrid control systems is to derive the DES plant automata from the requirement, and the description of the system such that input and output functions can be added to make the automata satisfy item (2) above. Then a decision maker can be derived directly. In this way, we can reduce the complexity of the proof of the correctness of the design.

In the following sections, we shall elaborate this approach in a formal model of hybrid control systems. In the next section, we present an automata model of hybrid control systems and show how to represent the behaviors of the DES plant

automaton by using the duration calculus (DC). Then in Section 3, we present a necessary and sufficient condition for a plant automaton and a requirement to have a decision maker and therefore give an algorithm for deriving a desired decision maker if it exists. Section 4 concludes with some comparison to the related work.

2 From DC^* to Real-time Communicating Automata

2.1 Real-time Communicating Automata

We add the input and output functions to real-time automata (see, e.g.[2]) to obtain real-time communicating automata.

Definition 1. *A* Real-time Communicating Automata A *is a tuple* $(S,\ I,\ O,\ E,\ d,\ s_0,\ \phi)$ *where:*

1. S *is a set of states,* $s_0 \in S$ *is the initial state,*
2. I *is an input alphabet,* O *is an output alphabet*
3. $E \subseteq S \times (I \cup \{\epsilon\}) \times S$ *is a set of transitions,* $d : E \to \{[a,b] : a,b \in \mathbf{R}^+ \cup \{\infty\}, a \leq b\}$ *is the delay-time function of transitions.*
4. $\phi : S \to O \cup \{\epsilon\}$ *is an output function.*

A real-time communicating automaton A is *deterministic* iff it is not the case that $(s,a,s') \in E \wedge (s,a,s'') \in E \wedge s' \neq s''$ and neither the case that $(s,\epsilon,s') \in E \wedge (s,a,s') \in E \wedge s \neq s'$ for some a. The transitions of the form (s,ϵ,s') represent internal transitions. In this paper, the symbol ϵ is treated as the empty word, and a singleton will be identified with its element.

Let $w = b_0 b_1 \ldots \in I^{\aleph}$. A *behavior* σ_w *of* A *corresponding to* w is $\sigma_w = (e_0,t_0)(e_1,t_1)\ldots$, where $e_i = (s_i, b_i, s_{i+1}) \in E$, $t_i \in d(e_i)$, $b_i \in I \cup \{\epsilon\}$ such that $b_0 b_1 \ldots = w$. t_i is the time that A stays in s_i (delay time of e_i) from when it is enabled (the automaton is in the state s_i and the input b_i is available). σ is a behavior if it is a behavior corresponding to w for some w. Notice that for a behavior σ, there exists only one w such that $\sigma = \sigma_w$, which is denoted by $in(\sigma)$. Output $out(\sigma)$ of the behavior σ is defined by $out(\sigma) = \phi(s_1)\phi(s_2)\ldots$. Therefore, A can be considered as a mapping

$$A(w) = \{out(\sigma_w) | \sigma_w \text{ is a behavior corresponding to } w\}.$$

Signal $signal(\sigma)$ of a behavior σ is defined by $(\phi(s_0),t_0)(\phi(s_1),t_1)\ldots$. By convention, $(\epsilon,t) = \epsilon$. An input with real-time constraints is $w_t = (a_0,t_0)(a_1,t_1)\ldots$, where $a_0 a_1 \ldots \in I^{\aleph}$. Informally, t_i is the time constraint on the processing of the input a_i. A behavior σ_{w_t} of A that has no internal transitions, corresponding to w_t is $\sigma_{w_t} = (e_0,t'_0)(e_1,t'_1)\ldots$, where $(e_0,t'_0)(e_1,t'_1)\ldots$ is a behavior corresponding to $a_0 a_1 \ldots$, and $t'_i = t_i$. A is considered as a mapping $A_t(w_t) = \{out(\sigma_{w_t}) | \sigma_{w_t}$ is a behavior corresponding to $w_t\}$. Notice that by our assumption on A, e_i is a transition with the input a_i. *Trace* $tr(\sigma)$ of the behavior σ is defined by $tr(\sigma) = e_0 e_1 \ldots$.

Let $Behaviors(A)$ be the set of all behaviors of A. For a $\sigma \in Behaviors(A)$, $\sigma = (e_0, t_0)(e_1, t_1)\ldots$ is an infinite behavior iff $\sum_{i=0}^{|\sigma|-1} t_i = \infty$. By adding some special transitions with delay-time ∞ into the definition of the real-time communicating automata, we can assume that every behavior of A can be extended to an infinite behavior. Our intention is to use real-time communicating automata to represent both the decision makers and the DES plants. However, the interaction between the two is not symmetrical as mentioned in the introduction. Furthermore, the plant automata may give no output on entering a new state, while the decision makers need to give a new command on entering a new state.

Definition 2. *Let $A = (S, I, O, E, d, s_0, \phi)$, and $A' = (S', O, I, E', d', s'_0, \phi')$ be real-time communicating automata. Assume that there is no internal transition in E.* Parallel composition *$A \times A'$ of A and A' is the* real-time *automaton defined by $A \times A' \mathrel{\widehat{=}} (S \times S', E'', D, (s_0, s'_0))$, where*

1. *$S \times S'$ is the set of states of $A \times A'$, and (s_0, s'_0) is the initial state $A \times A'$,*
2. *The set of transitions $E'' \subseteq (S \times S')^2$ and the delay-time function D are defined as follows: $e'' = ((s_1, s'_1), (s_2, s'_2)) \in E''$ iff*
   - *either $e = (s_1, \phi'(s'_1), s_2) \in E$, $e' = (s'_1, \phi(s_2), s'_2) \in E'$, and e, e' are not internal transitions; in this case, $D(e'') = d(e) \cap d'(e)$, or*
   - *$s_1 = s_2$, $e' = (s'_1, \epsilon, s'_2) \in E'$ and $\phi'(s'_1) = \epsilon$; $D(e'') = d'(e')$.*

   In both cases, we write $e'' = (e, e')$, where $e \in E \cup \{\epsilon\}$, $e' \in E'$.

Notice that the parallel composition operation of two real-time communicating automata is not symmetrical, and by our definition, the automaton A', when it sends a signal to the automaton A, must wait for a command from A, which means that in a state with a nonempty output, only transitions with a nonempty input can occur.

For a behavior $\sigma = ((e_1, e'_1), t_1)((e_2, e'_2), t_2)\ldots$ of $A \times A'$ (defined in usual way), the projection of σ on A and A' are defined respectively by

$$\sigma_A = (e_1, t_1)(e_2, t_2)\ldots, \qquad \sigma_{A'} = (e'_1, t_1)(e'_2, t_2)\ldots.$$

The following proposition follows directly from the definition.

Proposition 1. *1. For any behavior σ of $A \times A'$, σ_A is a behavior of A corresponding to $signal(\sigma_{A'})$, and $\sigma_{A'}$ is a behavior of A' corresponding to $out(\sigma_A)$.*

2. If σ_2 is a behavior of A' corresponding to w for some $w \in O^*$, and if σ_1 is a behavior of A corresponding to $signal(\sigma_2)$ such that $w = out(\sigma_1)$, then there exists a behavior σ of $A \times A'$ satisfying $\sigma_A = \sigma_1$ and $\sigma_{A'} = \sigma_2$.

2.2 Duration Calculus with Star

We present an extension of duration calculus (DC^*) that can represent the behaviors of real-time automata. Our extension is to add the closure of the modality ';' to DC: if D is a formula, then so is D^+. The semantic structure of DC^* shares the same with that of DC. An interpretation $\mathcal{I}$ is a function

$\mathcal{I} \in (V \to (time \to \{0,1\}))$, for which each $\mathcal{I}(X)$, $X \in V$ has at most finitely many discontinuity points in any interval $[a,b]$. The semantics of D^+ is defined as

$$\mathcal{I},[a,b] \models D^+ \text{ iff } \mathcal{I},[m_i,m_{i+1}] \models D,\ i=0,1,\ldots,n$$
$$\text{for some } a=m_0<m_1<\ldots<m_{n+1}=b,\ n\geq 0$$

For a DC^* formula D, we define $D^* \mathrel{\widehat{=}} D^+ \ \vee\ \lceil\ \rceil$.

The readers are referred to [12, 13] for the set of axioms and rules of DC which are the same as in DC^*. We just list some of the additional axioms and rules in the calculus. Let D and A be DC^* formulas.

1. (Monotonicity for star) If $D \Rightarrow A$ then $D^* \Rightarrow A^*$
2. D^+ holds iff there is $n>0$ such that D^n holds, where D^n is defined by $D^1 \widehat{=} D$, $D^{m+1} \widehat{=} D; D^m$ for all natural $m>1$
3. $D^*; D^+ \Rightarrow D^+$, and $D^+ \Rightarrow D^*$.

Theorem 1. *Let P_i, $i=1,2,\ldots,n$ be state expressions. Let $D \widehat{=} \lceil \bigvee_{i=1}^{n} P_i \rceil$. Then*

$$D \Leftrightarrow (\bigvee_{i=1}^{n} \lceil P_i \rceil)^+$$

2.3 Representing the Behavior of Real-time Automata

DC^* formulas can be used to describe the behavior of finite real-time automata (considered as a real-time communicating automaton with $I=O=\{\epsilon\}$) in the same way that the regular expressions represent the behavior of finite automata. For example, the real-time automaton in Fig. 3 is represented by the formula $(\lceil N \rceil \wedge \ell \geq 30; \lceil Leak \rceil \wedge \ell \leq 1)^*; (\lceil N \rceil \vee \lceil\ \rceil)$.

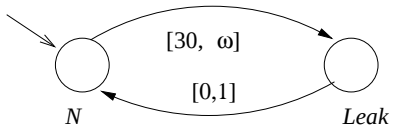


Fig. 3

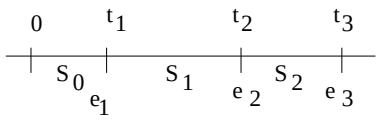


Fig. 4

Definition 3. *Let A be a real-time automaton with a set S of states which satisfies the condition that its every behavior can be extended to an infinite behavior. Each infinitely timed sequence of consecutive transitions $\sigma = (e_1,t_1)(e_2,t_2)\ldots$ of A defines an interpretation $\mathcal{I}^\sigma$ of state functions in S in the natural way (see Fig. 4). Namely, for a state $s \in S$,*

$$s_{\mathcal{I}^\sigma}(t) \widehat{=} 1 \Leftrightarrow \exists i>0, s' \in S \bullet e_i = (s,s') \wedge \sum_{j=1}^{i-1} t_j \leq t < \sum_{j=1}^{i} t_j.$$

A DC formula D is said to represent the behavior of A iff for all $infinite\ \sigma \in Behaviors(A) \Leftrightarrow \forall t \bullet \mathcal{I}^\sigma, [0,t] \models D$.

Definition 4. *Simple DC^* formulas are the DC^* formulas generated by the syntax:* $D ::= \lceil\ \rceil \mid \lceil P \rceil \mid \lceil P \rceil \wedge a \leq \ell \leq b \mid D_1; D_2 \mid D^* \mid D^+ \mid D_1 \vee D_2$, *where* P *stands for any state expression.*

For a real-time automaton A, let us denote by $(\lceil s \rceil \wedge a \leq \ell \leq b)$ the transition $e = (s, s')$ with $d(e) = [a, b]$. Then, in the same way as in the theory of formal languages (see, e.g. [8]) we have the following theorem:

Theorem 2. *For any real-time automaton A with a finite set of states and a finite set of transitions, there exists a simple DC^* formula D such that D represents the behaviors of A. Conversely, for any simple DC^* formula D such that $D \Rightarrow D; \ell = r$ (i.e. prefix-closed) and such that the time constraint of D (the replacements of $\lceil P \rceil$ by $\ell > 0$, $\lceil\ \rceil$ by $\ell = 0$ in D) is a valid DC^* formula, there exists a real-time automaton A with a finite set of states and a finite set of transitions such that the behaviors of A are represented by D.*

2.4 Refinement of *DC* Formulas into Automata

Now we present a systematic method for refining DC formulas into automata models. Our purpose is to derive from the physical descriptions of a system and the requirement written as a DC^* formula D a DES plant automaton A'. Starting from $D_1 = D$, by stepwise refinements, we find DC^* formulas $D_2, D_3, \ldots, D_n$ such that $D_i \Rightarrow D_{i-1}$, $i = 2, \ldots, n$, and D_n is a simple DC^* formula. By Theorem 7, A' is derived directly from D_n.

Gas Burner Example[12]. A physical description of the system (the plant automata) may be as follows.

1. States:
   Non-leak (N): *OnReq, OffReq, Burn, Idle, Rec* (N is an abstract state);
   Leak ($Leak$).
2. Events:

$On \mathrel{\widehat{=}}$ *(Idle, OnReq)*	(heat request)	$Off \mathrel{\widehat{=}}$ *(Burn, OffReq)*	(off request)
$R \mathrel{\widehat{=}}$ *(Leak, Rec)*	(recover)	$IgOK \mathrel{\widehat{=}}$ *(OnReq, Burn)*	(ignition OK)
$Igfl \mathrel{\widehat{=}}$ *(OnReq, Leak)*	(ignition failure)	$Ffl \mathrel{\widehat{=}}$ *(Burn,Leak)*	(flame failure)

3. Control laws: $C1$: to recover from *Leak*, $C2$: to become ready.

In [12], the requirement has been formalized as $D_1 \mathrel{\widehat{=}} \ell \geq 60 \Rightarrow \int Leak \leq 0.05 * \ell$ and the design decision is

$$D_2 \mathrel{\widehat{=}} (\Box(\lceil Leak \rceil \Rightarrow \ell \leq 1)) \wedge (\Box(\lceil Leak \rceil; \lceil N \rceil; \lceil Leak \rceil \Rightarrow \ell \geq 30))$$

It has been proved that $D_2 \Rightarrow D_1$ (see [12]). We wish to derive a plant automaton that has duration formulas built from physical states of the system and has behaviors satisfying the requirement. An abstract description of the system is $\lceil N \vee Leak \rceil$, which is $(\lceil N \rceil \vee \lceil\ \rceil); (\lceil Leak \rceil; \lceil N \rceil)^*; (\lceil Leak \rceil \vee \lceil\ \rceil)$ by Theorem 4. Those behaviors of the system that satisfy D_2 are represented by $(\lceil N \rceil \vee \lceil\ \rceil); ((\lceil Leak \rceil \wedge \ell \leq 1); (\lceil N \rceil \wedge \ell \geq 30))^*; ((\lceil Leak \rceil \wedge \ell \leq 1) \vee (\lceil Leak \rceil \wedge \ell \leq$

1); $\lceil N \rceil \vee \lceil\ \rceil$). A real-time automaton derived from this formula is depicted in Fig. 3. Since $\lceil N \rceil \wedge \ell \geq 30$ is not a state of the plant automaton, we need to refine the formula. By the description of the system and by Theorem 4,

$$\begin{aligned}\lceil N \rceil \wedge \ell \geq 30 &\Leftrightarrow \lceil OnReq \vee OffReq \vee Burn \vee Idle \vee Rec \rceil \wedge \ell \geq 30 \\ &\Leftrightarrow (\lceil Idle \rceil \vee \lceil OnReq \rceil \vee \lceil OffReq \rceil \vee \lceil Rec \rceil \vee \lceil Burn \rceil)^{+} \wedge \ell \geq 30.\end{aligned}$$

From the description of events and noticing that for $s \neq s'$, $true; \lceil s \rceil; \lceil s' \rceil; true$ iff (s, s') is an event, it follows:

$$\begin{aligned}\lceil N \rceil \wedge \ell \geq 30 \Leftrightarrow\ & (\lceil Rec \rceil; (\lceil Idle \rceil; \lceil OnReq \rceil; \lceil Burn \rceil; \lceil OffReq \rceil)^{*}; \\ & (\lceil Idle \rceil; \lceil OnReq \rceil; \lceil Burn \rceil \vee \lceil Idle \rceil; \lceil OnReq \rceil \vee \lceil Idle \rceil \vee \lceil\ \rceil)) \\ & \wedge \ell \geq 30.\end{aligned}$$

Let

$$\begin{aligned}D_3 \mathrel{\widehat{=}}\ & \lceil Rec \rceil \wedge \ell = 30; (\lceil Idle \rceil; \lceil OnReq \rceil; \lceil Burn \rceil; \lceil OffReq \rceil)^{*}; \\ & (\lceil Idle \rceil; \lceil OnReq \rceil; \lceil Burn \rceil \vee \lceil Idle \rceil; \lceil OnReq \rceil \vee \lceil Idle \rceil \vee \lceil\ \rceil).\end{aligned}$$

We have $D_3 \Rightarrow \lceil N \rceil \wedge \ell \geq 30$. D_3 is a simple DC^{*} formula. An automaton having the behavior represented by this formula has the state diagram illustrated as Fig. 5. Therefore, we get an automaton with the state diagram in Fig. 6.

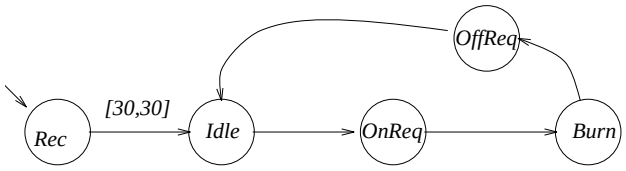


Fig. 5

3 Deriving Decision Maker from Plant Automata

In this section, we present a method to derive a decision maker from the plant automata w.r.t. the requirements. In our approach, a hybrid system is modeled as a parallel composition of 2 real-time communicating automata, decision maker and DES plant automaton. The DES plant automaton describes the possible behavior of the plant under effects of physical laws, and decision maker is to make the plant behave as we desire. In our context, a decision maker is a real-time communicating automaton $A = (S, I, O, E, d, s_0, \phi)$, in which there are no internal transitions and $\forall s \in S \bullet \phi(s) \neq \epsilon$. O is the alphabet of control law names; A DES plant automaton is a real-time communicating automaton $A' = (S', O, I, E', d', s'_0, \phi')$. A hybrid system is in fact the parallel composition of its decision maker and plant automaton, $A \times A'$. Therefore, the development

of hybrid systems can be specified as follows. *Given a DES plant automaton A' that represents the physical descriptions and control laws of the system and a set of behaviors B of A', where B is the set of the desired behaviors containing only infinite behaviors, find a decision maker A such that the projection of the set of infinite behaviors of $A \times A'$ on A' is a nonempty subset of B, and such that every behavior of $A \times A'$ can be extended to an infinite behavior.* Such a real-time communicating automaton A is called a decision maker of A' w.r.t. B.

From Proposition 3 we can derive directly a functional specification of a decision maker from a DES plant automaton and requirement. That is, every behavior of the decision maker corresponds only to the signal of a behavior of the DES plant that satisfies the requirement which in turn corresponds to its output. However, the decision maker needs to satisfy some additional conditions, such as deadlock-freedom, etc.. This is formulated in the following theorem.

Theorem 3. *A real-time communicating automaton A with no internal transitions is a solution of the problem of designing a decision maker if and only if for all $\sigma \in Behaviors(A)$, the following holds: for any behavior $\sigma' \in Behaviors(A')$ such that σ corresponds to $signal(\sigma')$ and σ' corresponds to $out(\sigma)$, σ' can be extended to a behavior σ'' in B such that σ can be extended to a behavior σ_1 corresponding to $signal(\sigma'')$ and σ'' is a behavior corresponding to $out(\sigma_1)$.* □

Next, we consider a necessary condition on A' and B for the existence of an automaton A satisfying the conditions of the theorem, which says that the $signal(\sigma)$ of behavior σ of A' needs to carry enough information for designing decision makers.

Theorem 4. *A necessary condition for a DES plant automaton A' and a set B of the infinite behaviors of A' to have a decision maker is that there exists a nonempty subset B' of B such that for all $\sigma, \sigma' \in Behaviors(A')$, if $(in(\sigma), signal(\sigma)) = (in(\sigma'), signal(\sigma'))$ (component-wise) then σ can be extended to a behavior in B' iff σ' can.* □

When we have a subset of the behaviors of the plant automaton satisfying the requirement which is represented by a finite subgraph of the plant automaton, the condition in Theorem 9 becomes a sufficient condition as well, and there is an algorithm for deriving a decision maker.

Theorem 5. *Let A' be a plant automaton, and B be a set of its infinite behaviors. Assume that there is a sub-automaton A'' of A' (its graph representation is a subgraph in the graph representation of A') such that*

1. *A'' has a finite set of states,*
2. *All infinite behaviors of A'' are in B, and*
3. *For all $\sigma, \sigma' \in Behaviors(A')$, if $(in(\sigma), signal(\sigma)) = (in(\sigma'), signal(\sigma'))$ then σ can be extended to an infinite behavior in $Behaviors(A'')$ iff σ' can.*

Then, there is a decision maker A of A'' w.r.t. B. □

From Theorem 10, we can extract an algorithm to construct a decision maker from a plant automaton that satisfies the assumptions of the theorem.

Algorithm.

1. Let $\varphi : E \to O' \times I' \times d(E)$ be a letter substitution homomorphism defined as follows. For $e = (s, a, s') \in E$, where E is the transition set of A'',

$$\varphi(e) \mathrel{\widehat{=}} \begin{cases} (\phi'(s), a, d(e)) & \text{if } \phi'(s) \neq \epsilon \\ \epsilon & \text{otherwise} \end{cases}$$

2. Let R be a regular expression over a finite set of the transitions E of A'' which represents the set of all the traces of the behavior of A'' (prefix-closed set). Constructing a deterministic automaton A_1 in a standard way, such that its behaviors are presented by the regular expression $\varphi(R)$. Let G be a graph representation of A_1.
3. Modifying G such that for any vertex s of G, the incoming edges are labeled by triples having the same middle component. For any vertex s of G, let $difference(s)$ be the number of incoming edges labeled by triples having different middle components. If $difference(s) \neq 0$, let $E_s(a)$ be the set of the incoming edges of s labeled by triples having a as the middle component. For $a \neq b$ such that $E_s(a) \neq \emptyset$ and $E_s(b) \neq \emptyset$, add a new vertex s' and change the edges in $E_s(b)$ to point to s'. For an incoming edge f of s that is neither in $E_s(b)$ nor $E_s(a)$, add a new edge f' pointing to s' with the same label and the same beginning vertex as f. For an outgoing edge f of s, add a new edge f' leaving s' with the same label and the same ending vertex as f.
4. Deriving the decision maker A from the modified G:
   (a) Each state of A is a vertex s of G, with output a, where a is the middle component of any incoming edge of s.
   (b) Each transition of A from s to s' is an edge of G from s to s' labeled by a triple $(m, a, d(e))$, which has delay time $d(e)$ and input m.

Gas Burner Example (Continued). The behavior of a gas burner can be represented by the real-time communicating automaton A'' in Fig. 6, with the states *Rec, Idle, OnReq, Burn, OffReq, Leak* and with inputs $C1$, $C2$, *ig*, *off*, with the following meaning. (1) $C1$: Close the gas valve for recovering from *Leak*; (2) $C2$: Give the green light (to show ready state); (3) *off*: Turn off the system (Close the gas valve); (4) *ig*: Ignite the burner. The output function is defined as follows.

$$\phi(s) = \begin{cases} s, & s \in \{Rec, OnReq, OffReq, Leak\} \\ \epsilon & otherwise \end{cases}$$

Here, we have reduced the diagram so that all its infinite behaviors are desired behaviors, and every behavior can be extended to an infinite behavior. The traces of the system are represented by the prefix-closure of the language generated by the regular expression $R = (e1(e2e3e4e5)^*e2(e7 + e3e6)e8)^*$.

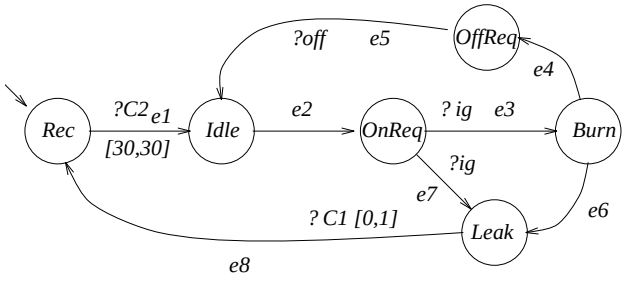


Fig. 6. Gas Burner Example: Plant Automaton

By induction on the length of the behaviors, it can be shown that A'' satisfies the condition in Theorem 10. The homomorphism φ is defined by:

$$\varphi(e1) \mathrel{\widehat{=}} f1 \mathrel{\widehat{=}} (Rec, C2, [30, 30]) \quad \varphi(e5) \mathrel{\widehat{=}} f5 \mathrel{\widehat{=}} (OffReq, off, \epsilon)$$
$$\varphi(e3) \mathrel{\widehat{=}} f3 \mathrel{\widehat{=}} (OnReq, ig, \epsilon) \quad \varphi(e7) \mathrel{\widehat{=}} f7 \mathrel{\widehat{=}} (OnReq, ig, \epsilon)$$
$$\varphi(e8) \mathrel{\widehat{=}} f8 \mathrel{\widehat{=}} (Leak, C1, [0, 1]) \quad \varphi(e2) \mathrel{\widehat{=}} \varphi(e4) \mathrel{\widehat{=}} \varphi(e6) \mathrel{\widehat{=}} \epsilon$$

Hence, since $f_3 = f_7$, $\varphi(R) = (f_1(f_3f_5)^*(f_7 + f_3)f_8)^* = (f_1(f_7f_5)^*f_7f_8)^*$. From this expression, we can construct an automaton as shown in Fig. 7a to recognize the prefix-closure of $\varphi(R)$. Here, the label of each incoming edge of a state has the same middle component. Thus, we do not need to modify the graph. The real-time communicating automaton representing the decision maker constructed according to the algorithm is shown in Fig. 7b.

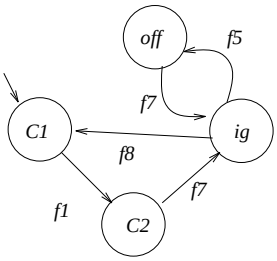


Fig. 7a

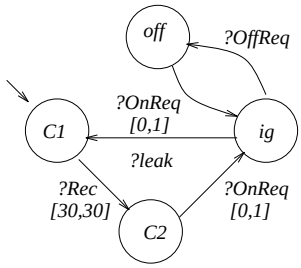


Fig. 7b

4 Conclusion

We have given a template of the refinement of hybrid system specifications written in *DC* into automata models: By using real-time communicating automata and their parallel composition, we formalize the conceptual model for hybrid control systems; *DC* formulas are extended with iteration form and a technique

is presented for deriving plant automata from *DC* formulas; An algorithm is presented to synthesize a specification of control automata from plant automata with respect to requirements.

ProCoS presents the significant results on how to develop an embedded system from a *DC* specification [6, 10, 11]. Requirements captured in Duration Calculus can be systematically refined into CSP like implementations. Here, we confine ourselves to a more restricted class of hybrid systems and pursue more automatic techniques. We share the dual language framework with Hybrid Temporal Logic and Phase Transition System [7], but we pay emphasis on design rather than verification. The backbone of our approach presented in this paper is to accommodate design calculation to a set of pragmatic techniques for the development of hybrid systems.

Acknowledgment: The authors thank Prof. Zhou Chaochen, Chris George and Dr. Xu Qiwen for their comments and criticism.

References

1. R.Alur, T. Henzinger and E. Sontag (Eds.), Hybrid Systems, Lecture Notes in Computer Science 1066, Springer Verlag, 1996.
2. R. Alur and D. L. Dill, A Theory of Timed Automata, *Theoretical Computer Science*, 126, 183-235, 1994.
3. P. J. Antsaklis, J. A. Stiver, M. Lemmon, Hybrid System Modeling and Autonomous Control Systems, LNCS 736, Springer Verlag, 366-392, 1993.
4. Chen Zongji, Wang Ji, Yu Xinyao and Zhou Chaochen, An Abstraction of Hybrid Control Systems, *IEEE Intelligent Control and Instrumentation*, Singapore, 1995.
5. R.L. Grossman, A. Nerode, A.P. Ravn and H. Rischel (Eds.), Hybrid Systems, Lecture Notes in Computer Science 736, Springer Verlag, 1993.
6. J. He, C. A. R. Hoare, M. Franzle, M. Muller- Olm, E.- R. Olderog, M. Schenke, M. R. Hansen, A. P. Ravn, and H. Rischel, Provably Correct Systems. In Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863, pages 288-335, Springer Verlag, 1994.
7. T. A. Henzinger, Z. Manna, A. Pnueli, Towards Refining Temporal Specifications into Hybrid Systems, LNCS 736, Springer Verlag, 60-76, 1993.
8. J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Reading, MA, Addison-Wesley, 1979.
9. X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, An Approach to the Description and Analysis of Hybrid Systems, LNCS 736, Springer Verlag, 149-178, 1993.
10. A. Ravn, Design of Embedded Real-Time Computing Systems. Department of Computer Science, Technical University of Denmark, 1994.
11. A. Ravn, H. Rischel and K.M. Hansen, Specifying and Verifying Requirements of Real-time Systems, IEEE Trans. on Software Engineering, 1993.
12. Zhou Chaochen, C. A. R. Hoare, A. P. Ravn, A Calculus of Durations, *Information Processing Letters*, 40, 269-276, 1991.
13. Zhou Chaochen, A. P. Ravn, M. R. Hansen, Extended Duration Calculus for Hybrid Real-time Systems, LNCS 736, Springer Verlag, 36-59, 1993.