



The United Nations
University

UNU/IIST

International Institute for
Software Technology

On The Design of Hybrid Control Systems Using I/O Automata Models

Dang Van Hung and Wang Ji

November 1994

UNU/IIST

UNU/IIST enables developing countries to attain self-reliance in software technology by: (i) their own development of high integrity computing systems, (ii) highest level post-graduate university teaching, (iii) international level research, and, through the above, (iv) use of as sophisticated software as reasonable.

UNU/IIST contributes through: (a) advanced, joint industry-university advanced development projects in which rigorous techniques supported by semantics-based tools are applied in case studies to large scale software developments, (b) own and joint university and academy institute research in which new techniques for application domain and computing platform modelling, requirements capture, software engineering and programming are being investigated, (c) advanced, post-graduate and post-doctoral level courses which typically teach Design Calculi oriented software development techniques, (d) events [panels, task forces, workshops and symposia], and (e) dissemination.

Application-wise, the advanced development projects presently focus on software to support large-scale infrastructure systems such as transport systems (railways, airlines, air traffic, etc.), manufacturing industries, telecommunications, etc., and are thus aligned with UN and International Aid System concerns. UNU/IIST is a leading research centre in the area of Duration Calculi, i.e. techniques applicable to *real-time, reactive, hybrid & safety critical systems*. The research projects parallel and support the advanced development projects.

At present, the technical focus of UNU/IIST in all of the above is on applying, teaching, researching, and disseminating Design Calculi oriented techniques and tools for trustworthy software development. UNU/IIST currently emphasises techniques that permit proper development steps and interfaces. UNU/IIST also endeavours to promulgate sound project and product management principles.

UNU/IIST's primary dissemination strategy is to act as a clearing house for reports from research and technology centres in industrial countries to industries and academic institutions in developing countries. At present more than 200 institutions worldwide contribute to UNU/IIST's report collection while UNU/IIST at the same time subscribes to more than 125 international scientific and technical journals. Information on reports received (and produced) and on journal articles is to be disseminated regularly to developing country centres — which are then free to order a reasonable number of report and article copies from UNU/IIST.

Dines Bjørner, Director — 1.7.1992–31.6.1997

UNU/IIST Reports are either *R*esearch, *T*echnical, *C*ompendia or *A*dministrative reports:

\mathcal{R} Research Report • \mathcal{T} Technical Report • \mathcal{C} Compendium • \mathcal{A} Administrative Report



The United Nations
University

UNU/IIST

**International Institute for
Software Technology**

P.O. Box 3058
Macau

On The Design of Hybrid Control Systems Using I/O Automata Models

Dang Van Hung and Wang Ji

Abstract

The present paper gives a systematic way to refine specifications of hybrid control systems written in DC (Duration Calculus) into their automata models. The iteration DC formulas are introduced together with rules for transforming, refining DC formulas into the forms that describe the systems in more detail. Then, the plant automata derived from DC formulas are defined, which allow the designers to derive a specification of control automata more easily. A necessary and sufficient condition for a plant automaton and a requirement to have a control automaton is given. Some simple examples are presented to illustrate the method.

Dang Van Hung is from the Institute of information Technology of National Center for Natural Science and Technology of Vietnam, where he is a researcher. He is a Fellow of UNU/IIST from April 1994 to July 1995. His research interest is in Formal Technique of Programming, Concurrent and Distributed systems. E-mail: dvh@iist.unu.edu.

Wang Ji is a fellow of UNU/IIST, on leave from Department of Computer Science, Changsha Institute of Technology, where he is a PhD student. His current research interests focus on Specification and Refinement of Real-Time and Hybrid Systems. Email: wj@iist.unu.edu

Contents

1	Introduction	1
2	Deriving decision makers from plant automata	5
3	Representing the behavior of real-time automata	12
3.1	Duration calculus with star	12
3.2	Representing the behavior of real-time automata	15
4	Refinement of DC Formulas into I/O real-time Automata	16

1 Introduction

In this paper, we shall deal with hybrid control systems which consist of continuous plants controlled by decision makers via controllers. Decision makers (control programs) are implemented on a sequential automaton. The control program reads an 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 sequence consisting of a read of the data followed by a “compute and impose the next control” constitutes the control cycle. How and when to make these control law changes is the business of the control program. 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. We use the traditional stepwise refinement approach to develop a systematic way for dealing with the problem. Started with a top level specification and a description of the components of a system, armed with some physical laws and control laws, we then step by step derive the specifications of the system in more and more detail until 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, can be determined directly. From the resulting plant automaton, it is easy to determine which control laws are needed to force the plants in which situations to ensure that the plant automaton has the allowable behavior. 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 directly and the control program is designed from the specification. Under certain conditions of the plant automaton, the process of deriving a decision maker can be fully automated.

The model of hybrid systems used in this paper is taken from [2, 11, 12]. It consists of three distinct levels (see Fig. 1). The decision maker is modeled as an I/O automaton. The decision maker receives, manipulates and outputs events represented by symbols. The plant is a continuous-state system typically modeled by differential/difference equations and it is the system to be controlled by the decision maker through controllers. A controller while controlling the plant, receives, manipulates and outputs signals represented by digitals. The decision maker and the controllers communicate via the interface that translates signals into symbols for the decision maker to use, and the decision maker outputs symbols into command signals for the controller input.

The plant, the controllers and the interface are taken together to result in a so-called discrete state system DES plant. As mentioned in [12, 2], it is advantageous to view a hybrid control system in this way because it allows it to be modeled as two interacting discrete event systems which are more easily analyzed than the system in its original form. A DES plant model is an I/O automaton similar to the decision maker. Our approach is to derive the DES plant from the specification of a system, and then to determine the behaviors of the plant automaton that satisfy the requirements. If these satisfy certain conditions, a feasible specification of a decision

maker is derived directly.

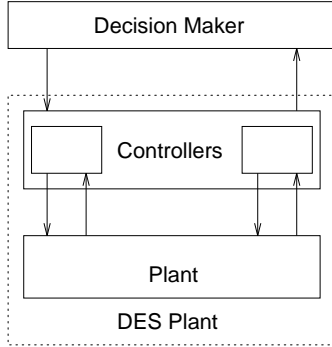


Fig. 1

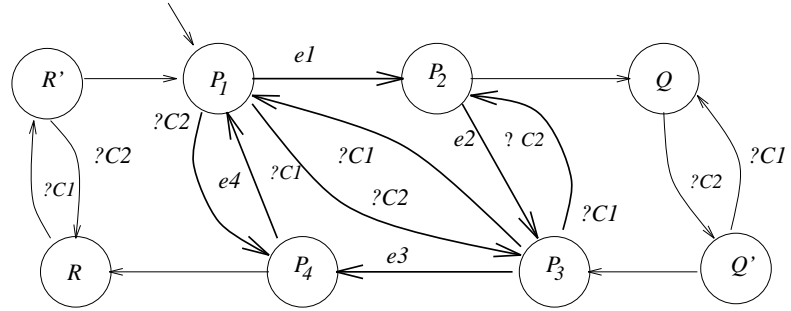


Fig. 2

To illustrate our approach, let consider the *water container example* which is taken from [5]. Consider a water level decision maker that opens and closes a valve regulating the outflow of water from a container. The container has an input vent, through which water flows in at constant rate a . When the valve is fully open, there is an outflow of $c > a$, leading to overall water level decrease of $-b = c - a$ per time unit. 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, say. Let $w(t)$ be the water level at time t .

One description of the system can be as the automaton depicted in Fig. 2, where

$D1$:	the valve is closed	$D2$:	the valve is open
$P1$:	$68 \leq w \leq 76 - \varepsilon \wedge D1$	$P2$:	$76 - \varepsilon < w \leq 76 \wedge D1$
$P3$:	$68 + \varepsilon < w \leq 76 \wedge D2$	$P4$:	$68 \leq w \leq 68 + \varepsilon \wedge D2$
Q :	$w > 76 \wedge D1$	R :	$w < 68 \wedge D2$
Q' :	$w > 76 \wedge D2$	R' :	$w < 68 \wedge D1$
$e2$:	open the valve	$e4$:	close the valve
$?C1$:	receiving the command 'open the valve'	$?C2$:	receiving the command 'close the valve'
$e1, e3$:	doing nothing, time elapsing		

The behaviors that satisfy the requirement $68 \leq w \leq 76$ are represented by the subgraph (indicated by the thick arrows) containing the vertices $P1, P2, P3, P4$. Also, the behaviors represented by the graph B with vertices $P1, P2, P3, P4$ and with edges $e1, e2, e3, e4$ satisfy the requirement. These behaviors are depicted in Fig. 3 (the lower line). 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

$P4$. So the decision maker needs to know when the plant automaton enters $P2$ and $P4$, and needs to know how long it will stay there. From a, b and ε , the decision maker can estimate the time $t1, t2$ that the plant can stay in $P2$ and $P4$ resp. from when it enters the states; so, the plant needs to inform the decision maker on entering the states. This means that 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 $P4$ resp.. Thus, the behavior of the decision maker needs to be as described by the upper line in Fig. 3. The automaton achieving this behavior is depicted in Fig. 4a that behaves as follows. In the state S , when the transition f is ready (there is an input, and the input is $P2$), the transition has to occur within $t1$ time units and gives the command ‘open the valve’ on entering the state S' . In the state S' , when the transition f' is ready (there is an input, and the input is $P4$), the transition has to occur within $t2$ time units and gives the command ‘close the valve’ on entering the state S . The interaction between the two parts to achieve this is shown in Fig. 3. 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. 4b, 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 decision maker to give the commands, but it takes no time for the controller to implement the commands. Thus, the roles of the the automaton of the decision maker and of the DES plant automaton are not symmetrical. The receipt of a signal by the decision maker and the send of a signal by the controller to the decision maker are not modeled by events in the model.

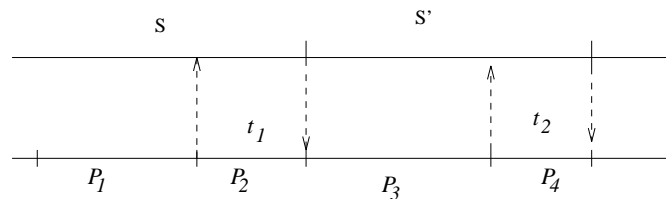


Fig. 3

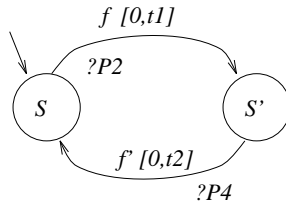


Fig. 4a

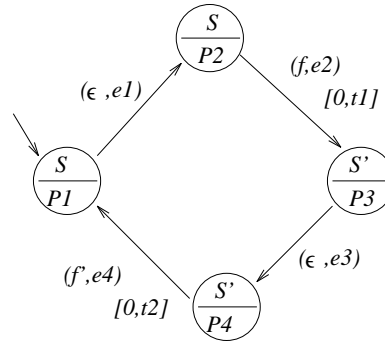


Fig. 4b

From the example we note the following:

1. From a description of the plant behavior in the form of the plant automaton, from the set of the behaviors of the plant automaton that satisfy the requirement, a specification of a real-time I/O automaton of the decision maker can be derived such that if its behavior satisfies the specification, the plant controlled by it will satisfy the requirement.
2. The DES plant may give no outputs, and transitions in the DES plant may have no inputs (for example, the plant controller gives no output when it enters either P_1 or P_3 , and transitions from P_1 to P_2 and from P_3 to P_4 need no input). 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 are distinguishable from those of the behaviors that does not satisfy the requirement.

Hence, a possible approach to the design of the 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 next sections, we shall elaborate this approach in a formal model of the hybrid control systems. In the next section, we present an I/O automata model of hybrid control systems and give a necessary and sufficient condition for a DES plant I/O automaton and a requirement to have a decision maker. We also give an algorithm for deriving a desired decision maker if it exists. The third section is devoted to how to represent the behaviors of the DES plant automaton by using the duration calculus (DC). Section 4 presents our method for deriving a DES plant automaton from a domain and a requirement.

2 Deriving decision makers from plant automata

We shall model the decision makers and DES plant automata by what we call real-time I/O automata. We add the input and output functions to real-time automata (see, e.g. [1]) to obtain real-time I/O automata. So, real-time I/O automata are I/O automata with real-time constraints on the occurrences of transitions.

Definition 1. A *Real-time I/O automaton* 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, $d : E \rightarrow \{[a, b] : a, b \in \mathbf{R}^+ \cup \{\infty\}, a \leq b\}$ is the delay-time function of transitions.
4. $\phi : S \rightarrow O \cup \{\epsilon\}$ is an output function.

The I/O 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 also not the case that $(s, \epsilon, s') \in E \wedge (s, a, s') \in E$ 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 $w = a_0a_1 \dots \in I^\omega = I^* \cup I^\infty$. A behavior σ_w of A corresponding to w is $\sigma_w = (e_0, t_0)(e_1, t_1) \dots$, 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 $b_0b_1 \dots = w$. 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 $out(\sigma) = \phi(s_1)\phi(s_2) \dots$. A is considered as a mapping $A(w) = \{out(\sigma_w) | \sigma_w \text{ is a behavior corresponding to } w\}$. It is our intention to use output of a behavior to represent the sequence of the commands given by the decision maker.

Sometimes we have to consider real-time constraints on the mapping. For this purpose, we define another kind of the mapping.

Signal $signal(\sigma)$ of the behavior σ is defined by $signal(\sigma) = (\phi(s_0), t_0)(\phi(s_1), t_1) \dots$.

By convention, $(\epsilon, t) = \epsilon$. An input with real-time constraints is $w_t = (a_0, t_0)(a_1, t_1) \dots$, where $a_0a_1 \dots \in I^\omega$. 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 $\sigma_{w_t} = (e_0, t'_0)(e_1, t'_1) \dots$, where $(e_0, t'_0)(e_1, t'_1) \dots$ is a behavior corresponding to $a_0a_1 \dots$, and $t'_i = t_i$. A is considered as a mapping $A_t(w_t) = \{out(\sigma_{w_t}) | \sigma_{w_t} \text{ is a behavior corresponding to } w_t\}$. 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 $tr(\sigma) = e_0e_1 \dots$

Since we are interested in the behaviors that satisfy the requirements throughout the life time of the system, we introduce the notion of infinite behavior. Let $Behaviors(A)$ denote the set of all behaviors of A . For a $\sigma \in Behaviors(A)$, $\sigma = (e_0, t_0)(e_1, t_1) \dots$ 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 I/O automata, we can assume that every behavior of A can be extended to an infinite behavior.

Our intention is to use real-time I/O 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. With this in mind, we define the parallel composition of two real-time I/O automata as follows.

Definition 2. Let $A = (S, I, O, E, d, s_0, \phi)$, $A' = (S', O, I, E', d', s'_0, \phi')$ be Real-time I/O 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 $A \times A' \triangleq (S \times S', E'', D, (s_0, s'_0))$, where

1. $S \times S'$ is the set of states of $A \times A'$, (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 $e = (s_1, \phi'(s'_1), s_2) \in E$, $e' = (s'_1, \phi(s_2), s'_2) \in E'$, 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 I/O automata is not symmetrical, and by our definition, the automaton A' , when it sends a signal to the automaton A , must wait for the command from A , which means that in a state with nonempty output, only transitions with nonempty input can occur.

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

$$\begin{aligned}\sigma_A &= (e_1, t_1)(e_2, t_2) \dots, \\ \sigma_{A'} &= (e'_1, t_1)(e'_2, t_2) \dots\end{aligned}$$

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 $w \in out(\sigma_1)$, then there exists a behavior σ of $A \times A'$ satisfying $\sigma_A = \sigma_1$ and $\sigma_{A'} = \sigma_2$.

A Hybrid System is a parallel composition of 2 real-time I/O automata, decision maker and DES plant automaton. The DES plant automaton describes the possible behavior of the plant under effect of physical laws, and decision maker is what we do to make the plant to behave as we want (the desired behaviors of the real system, as mentioned in the introduction).

Decision Maker: A real-time I/O 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.

DES plant automaton: A real-time I/O automaton $A' = (S', O, I, E', d', s'_0, \phi')$.

Problem of designing a decision maker from a DES plant automaton and requirement: Given a DES plant automaton A' that represents the physical descriptions and control laws of the system. Given a set of behaviors B of A' . B is the set of the desired behaviors containing only infinite behaviors. Find an 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. Informally, we require that the behavior of A' controlled by A should satisfy the requirements, and A should be a nontrivial solution such that the system is deadlock-free. Such an I/O real-time automaton A is called a decision maker of A' w.r.t. B .

From Proposition 1 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 the output of it. In other words, the mapping A_t defined by decision maker needs to satisfy the condition that if $A_t(\text{signal}(\sigma))$ is defined and if σ is infinite, then $\sigma \in B$ and $A_t(\text{signal}(\sigma)) = \text{in}(\sigma)$, and that $A_t(\text{signal}(\sigma))$ is undefined for those σ that cannot be extended to a behavior in B . However, the decision maker needs to satisfy some additional conditions, such as deadlock-freedom, etc.. This is formulated in the following theorem.

Theorem 1. A real-time I/O automaton A with no internal transitions is a solution of the problem of designing a decision maker if and only if for all $\sigma \in \text{Behaviors}(A)$, the following holds: for any behavior $\sigma' \in \text{Behaviors}(A')$ such that σ corresponds to $\text{signal}(\sigma')$ and σ' corresponds to $\text{out}(\sigma)$, σ' can be extended to a behavior σ'' in B such that σ can be extended to a behavior σ_1 corresponding to $\text{signal}(\sigma'')$ and σ'' is a behavior corresponding to $\text{out}(\sigma_1)$.

Proof. (\Leftarrow) We have to prove:

1. $\text{Behaviors}(A \times A') \neq \emptyset$
2. \forall infinite $\sigma \in \text{Behaviors}(A \times A') : \sigma_{A'} \in B$ (The behavior of A' controlled by A satisfies the requirements)
3. $\forall \sigma \in \text{Behaviors}(A \times A') : \sigma$ can be extended to an infinite behavior (deadlock freedom).

Item 1 is obvious from the assumption and the fact that $\epsilon \in \text{Behaviors}(A)$ corresponding to $\epsilon = \text{signal}(\epsilon)$. To prove item 2, let $\sigma \in \text{Behaviors}(A \times A')$ and σ is infinite. By Proposition 1, σ_A is a behavior of A corresponding to $\text{signal}(\sigma_{A'})$. (Thus, $A_t(\text{signal}(\sigma_{A'}))$ is defined.) Since $\sigma_{A'}$ is infinite, $\sigma_{A'}$ must be in B (It can be seen that if σ is infinite then so is $\sigma_{A'}$). Now we prove

item 3. Let $\sigma \in Behaviors(A \times A')$. By Proposition 1, σ_A is a behavior of A corresponding to $signal(\sigma_{A'})$ and $\sigma_{A'}$ is a behavior of A' corresponding to $out(\sigma_A)$. By the assumption, there exists an extension $\sigma' \in B$ of $\sigma_{A'}$ such that σ_A can be extended to a behavior σ'' of A corresponding to $signal(\sigma')$ and σ' corresponds to $out(\sigma)$. By Proposition 1, there exists a behavior of $A \times A'$ such that the projection of it on A' is σ' and the projection of it on A is σ'' . It is obvious that this behavior of $A \times A'$ is an infinite extension of σ .

(\Rightarrow) Obvious from Proposition 1 and the definition of a decision maker w.r.t. B .

Theorem 1 gives a functional and behavioral specification of a decision maker from B and the DES plant automaton. Now, 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 2. 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.

Proof. Suppose that there exists a real-time I/O automaton A which is a decision maker of A' w.r.t. the set of requirement B . By definition, the set

$$B' = \{\sigma_{A'} \mid \sigma \in Behaviors(A \times A') \text{ and } \sigma \text{ is infinite}\} \subseteq B$$

is not empty. Let $\sigma, \sigma' \in Behaviors(A')$, $(in(\sigma), signal(\sigma)) = (in(\sigma'), signal(\sigma'))$, and let $\sigma_1 \in B'$ be such that σ is a prefix of σ_1 . Then, by the definition of B' , there is a behavior σ'' of $A \times A'$ such that $\sigma''_{A'} = \sigma_1$. By Proposition 1, σ''_A is a behavior corresponding to $signal(\sigma_1)$ and σ_1 is a behavior of A' corresponding to $out(\sigma''_A)$. Thus, there is a prefix σ_2 of σ''_A such that σ_2 is a behavior of A corresponding to $signal(\sigma)$ and σ corresponds to $out(\sigma_2)$. Since $(in(\sigma), signal(\sigma)) = (in(\sigma'), signal(\sigma'))$, by Theorem 1, σ' can be extended to a behavior in B and σ_2 can be extended such that they are the projections on A' and A resp. of an infinite behavior of $A \times A'$. This means that, by Proposition 1, $\sigma' \in B'$.

Thus, if we want to derive a specification of a decision maker, first we have to verify that the plant automaton and the requirement satisfy the necessary condition in Theorem 2.

We will consider a special case in which Theorem 2 can be strengthened. Namely, 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, as in the example mentioned in the introduction of the paper, the condition in Theorem 2 becomes a sufficient condition as well, and there is an algorithm for deriving a decision maker.

Theorem 3. Let A' be a plant automaton, 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 $\sigma, \sigma' \in \text{Behaviors}(A')$, if $(\text{in}(\sigma), \text{signal}(\sigma)) = (\text{in}(\sigma'), \text{signal}(\sigma'))$ then σ can be extended to an infinite behavior in $\text{Behaviors}(A'')$ iff σ' can.

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

Proof. Let $A' = (S', I', O', E', d, s'_0)$. Since A'' can be considered as a finite automaton as well, there is a regular expression R 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). Let $\varphi : E \rightarrow O' \times I' \times d(E)$ be a letter substitution homomorphism defined as follows. For $e = (s, a, s') \in E$,

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

Then, $\varphi(R)$ is a regular expression as well. Hence, there exists a deterministic automaton A_1 such that its behaviors are presented by the regular expression $\varphi(R)$. Let G be a graph representation of A_1 . The edges of G represent the transitions and are labeled by elements in $\varphi(E)$. The vertices of G are states of A_1 . We will construct a decision maker A from G such that each edge of G labeled by $(m, a, d(e))$ corresponds to a transition of A with the input m , the delay time $d(e)$ and leading to a state that outputs a . In order to do so, first, we have to modify 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 $\text{difference}(s)$ be the number of incoming edges labeled by triples having different middle components. If $\text{difference}(s) \neq 0$, 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 . The modifying procedure is shown in Fig. 5.

Let \mathcal{M} be the multi-set consisting $\text{difference}(s)$ for each vertex s . Each time we modify G , we replace one element of the multi-set by two smaller elements, keeping the others the same. Since the set of multi-sets on the natural numbers with this kind of ordering is a well-founded

ordering, our procedure terminates after a finite number of steps (see [8]).

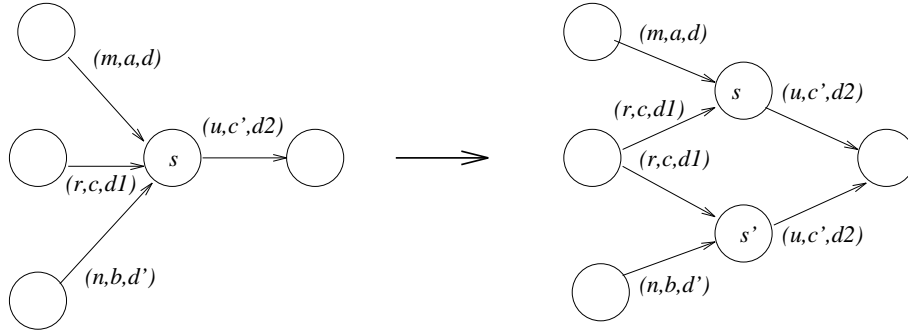


Fig. 5

It is obvious that the automaton represented by the resulting graph G (after it is modified) recognizes the same language represented by $\varphi(R)$, and the automaton represented by G is still deterministic. Now, let A be the real-time I/O automaton defined from G as: 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 . 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 . Now, we prove that A is a decision maker of A' w.r.t. B . For simplicity, we identify the transitions of A with edges of G . We verify that the automaton A satisfies the condition of Theorem 1. Let $\sigma = (f_1, t_1)(f_2, t_2) \dots$ be a behavior of A . Then, if f_i is labeled by (m_i, a_i, d_i) then $t_i \in d_i$. Thus, by the construction of G , there is a sequence $x = e_1 e_2 \dots$ of transitions in A'' such that $\varphi(x) = \text{label}(f_1 f_2 \dots)$, where label is a homomorphism replacing an edge of G by its label. Let $\sigma' = (e_1, t'_1)(e_2, t'_2) \dots$, where $t'_i \in d(e_i)$ is such that if e_i corresponds to f_j in the correspondence $\varphi(x) = \text{label}(f_1 f_2 \dots)$, then $t'_i = t_j$ (such a t'_i exists since the delay time of f_j is $d(e_i)$ by the definition of G). Hence, σ' is a behavior of A'' . Also from the definition of G , it follows that σ corresponds to $\text{signal}(\sigma')$ and σ' is corresponding to $\text{out}(\sigma)$. By the assumption on A'' , σ' can be extended to an infinite behavior of A'' . Now, let σ'' be any behavior of A' such that σ corresponds to $\text{signal}(\sigma'')$ and σ'' corresponds to $\text{out}(\sigma)$. This means that $(\text{in}(\sigma''), \text{signal}(\sigma'')) = (\text{in}(\sigma'), \text{signal}(\sigma'))$. By the assumption on A'' , σ'' can be extended to an infinite behavior $\sigma_1 \in \text{Behaviors}(A'')$. Thus, $\varphi(\text{trace}(\sigma_1))$ is a trace of a behavior of A having $f_1 f_2 \dots$ as a prefix of it. By the determinism of A and definition of delay time of transitions of A , it can be seen easily that σ can be extended to a behavior σ_2 of A such that σ_2 corresponds to $\text{signal}(\sigma_1)$ and σ_1 corresponds to $\text{out}(\sigma_2)$. The condition of Theorem 1 is fulfilled by A . The proof is completed.

Theorem 3 gives us an algorithm to construct a decision maker from a plant automaton that satisfies the assumption of the theorem. The algorithm is obvious from the proof of the theorem and is illustrated by the next example.

Example: Gas burner

Consider an example of gas burner taken from [3]. The behavior of a gas burner can be represented by the real-time I/O 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 of the system (Close the gas valve), ig : Ignit the burner.

The output function is defined as follows. $\phi(s) = s$, $s \in \{Rec, OnReq, OffReq, leak\}$; otherwise, $\phi(s) = \epsilon$. Here, we have reduced the diagram so that all the infinite behaviors of it 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 $R = (e1(e2e3e4e5)^*e2(e7 + e3e6)e8)^*$. The behaviors of the system (the plant automaton) that do not satisfy the requirement are those with the trace in $(e1(e2e3e4e5)^*e2(e7 + e3e6)e8)^*$ but with the delay of $e8$ greater than 1, and/or the delay of $e1$ different from 30.

Fig. 6

By induction on the length of the behaviors, it can be shown that

$$(in(\sigma), signal(\sigma)) = (in(\sigma'), signal(\sigma')) \Rightarrow (\sigma \in Behaviors(A'') \Leftrightarrow \sigma' \in Behaviors(A'')),$$

which means that A'' satisfies the condition in Theorem 3. The homomorphism φ is defined by:

$$\begin{array}{llll} \varphi(e1) \hat{=} f1 \hat{=} (Rec, C2, [30, 30]) & \varphi(e5) \hat{=} f5 \hat{=} (OffReq, off, \epsilon) \\ \varphi(e3) \hat{=} f3 \hat{=} (OnReq, ig, \epsilon) & \varphi(e7) \hat{=} f7 \hat{=} (OnReq, ig, \epsilon) \\ \varphi(e8) \hat{=} f8 \hat{=} (leak, C1, [0, 1]) & \varphi(e2) \hat{=} \varphi(e4) \hat{=} \varphi(e6) \hat{=} \epsilon \end{array}$$

Hence, since $f3 = f7$, $\varphi(R) = (f1(f3f5)^*(f7 + f3)f8)^* = (f1(f7f5)^*f7f8)^*$. 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 need not to modify the graph, and it is a state diagram of the decision maker as well. The real-time I/O automaton representing the decision maker constructed according to the algorithm is shown in Fig. 7b. Here, we should note that the necessary condition in Theorem 2 would be violated if in the definition of the gas burner diagram, we wrote $\phi(s) = s$, $s \in \{Rec, OnReq, OffReq\}$ instead, and in this case, a behavior satisfying the requirement would have the same signal and input with a behavior violating the requirement.

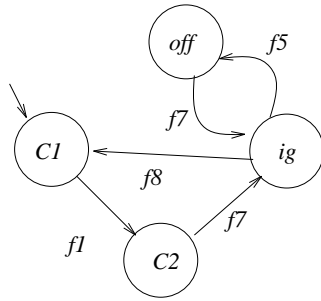


Fig. 7a

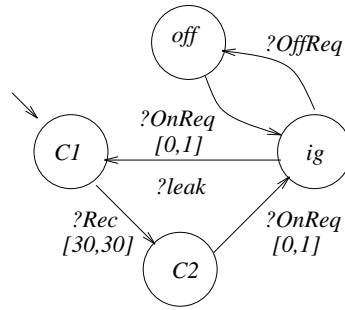


Fig. 7b

3 Representing the behavior of real-time automata

3.1 Duration calculus with star

We present an extension of duration calculus (DC*) that can represent the behaviors of real-time automata. Readers who are familiar with the duration calculus (DC) will see that our extension is to add the closure of the modality ‘;’ to DC.

Syntax of DC:*

The formulas of DC* are constructed from the following alphabet of symbols:

- Some (possibly infinitely many) state variables $V = \{X, Y, \dots\}$
- The special symbol ℓ , which stands for the length of an interval
- Some (possibly infinitely many) n -ary function names f_i^n , $i, n \geq 0$
- Some (possibly infinitely many) n -ary predicate names A_i^n , $i, n \geq 0$
- The special symbol *true*
- The connectives \vee, \neg .

Furthermore, we will use the auxiliary symbol f together with brackets and commas.

State expressions: The set of state expressions is generated by

1. 0,1 and every state variable are state expressions
2. if P and Q are state expressions, so are $(\neg P)$ and $(P \wedge Q)$, where \neg and \wedge are boolean operators on states and semantically different from the connectives on formulas.

Terms: The set of terms is generated by

1. if P is a state expression then $\int P$ is a term
2. ℓ is a term
3. if r_1, \dots, r_n are terms and f_i^n is an n -ary function name then $f_i^n(r_1, \dots, r_n)$ is a term

Formulas: the set of formulas is generated by

1. if A_i^n is an n -ary predicate name, and r_1, \dots, r_n are n terms then $A_i^n(r_1, \dots, r_n)$ is a formula,
2. *true* is a formula
3. if D_1 and D_2 are formulas, then so are $\neg D_1$, $D_1 \vee D_2$, $D_1; D_2$, and D_1^+ .

*Semantics of DC**

Assume that each n -ary function name f_i^n is associated with a total function from \mathbf{R}^n to \mathbf{R} which is denoted by f_i^n also, and each n -ary predicate name A_i^n is associated with a total function from \mathbf{R}^n to $\{tt, ff\}$ which is also denoted by A_i^n . An interpretation I is a function $I \in (V \rightarrow (time \rightarrow \{0, 1\}))$, for which each $I(X)$, $X \in V$ has at most finitely many discontinuity points in any interval $[a, b]$. We shall use the abbreviation $X_I \hat{=} I(X)$.

Semantics of state expressions: The semantics of a state expression P in an interpretation I is a function $I_P \in Time \rightarrow \{0, 1\}$ defined inductively on the structure of state expressions by:

$$\begin{array}{ll}
 I_0(t) \hat{=} 0 & I_1(t) \hat{=} 1 \\
 I_X(t) \hat{=} X_I(t) & I_{(\neg P)}(t) \hat{=} 1 - I_P(t) \\
 I_{(P \vee Q)}(t) \hat{=} \begin{cases} 0 & \text{if } I_P(t) = 0 \text{ and } I_Q(t) = 0 \\ 1 & \text{otherwise} \end{cases}
 \end{array}$$

Semantics of terms: The semantics of a term r in an interpretation I is a function $I_r \in intv \rightarrow \{0, 1\}$ (*intv* stands for the set of closed intervals on \mathbf{R}^+) defined inductively on the structure of terms by:

$$\begin{array}{ll}
 I_{\int P}([a, b]) \hat{=} \int_a^b I_P(t) dt & I_{\ell}([a, b]) \hat{=} b - a \\
 I_{f_i^n(r_1, \dots, r_n)}([a, b]) \hat{=} f_i^n(I_{r_1}([a, b]), \dots, I_{r_n}([a, b]))
 \end{array}$$

Semantics of formulas: The semantics of a formula D in an interpretation I is a function $I_D \in \text{intv} \rightarrow \{tt, ff\}$ defined inductively on the structure of formulas below.

Using the abbreviations $I, [a, b] \models D \triangleq I_D([a, b]) = tt$ and $I, [a, b] \not\models D \triangleq I_D([a, b]) = ff$, I_D is defined by:

$$\begin{aligned}
I, [a, b] \models A_i^n(r_1, \dots, r_n) &\text{ iff } A_i^n(I_{r_1}([a, b]), \dots, I_{r_n}([a, b])) = tt \\
I, [a, b] \models true & \\
I, [a, b] \models (\neg D) &\text{ iff } I, [a, b] \not\models D \\
I, [a, b] \models (D_1 \vee D_2) &\text{ iff } I, [a, b] \models D_1 \text{ or } I, [a, b] \models D_2 \\
I, [a, b] \models (D_1; D_2) &\text{ iff } I, [a, m] \models D_1 \text{ and } I, [m, b] \models D_2 \text{ for some } m \in [a, b] \\
I, [a, b] \models D_1^+ &\text{ iff } I, [m_i, m_{i+1}] \models D_1, i = 0, 1, \dots, n \\
&\text{ for some } a = m_0 < m_1 < \dots < m_{n+1} = b, n \geq 0
\end{aligned}$$

Abbreviations: For a DC* formulas D, D_1 , a state expression P , we use the following abbreviations:

$$\begin{aligned}
\Diamond D &\triangleq (true; D); true & \Box D &\triangleq (\neg(\Diamond(\neg D))) \\
[P] &\triangleq (\int P = \ell \wedge \ell > 0) & [\] &\triangleq (\ell = 0) \\
D^* &\triangleq D^+ \vee [\] & D \Rightarrow D_1 &\triangleq ((\neg D) \vee D_1) \\
D \wedge D_1 &\triangleq \neg((\neg D) \vee (\neg D_1))
\end{aligned}$$

The readers are referred to [3, 4] for the set of axioms and rules of DC which are the same as in DC*. In the following we list some axioms and theorems in the calculus that we are going to use in the sequel.

1. (Monotonicity) If $D_1 \Rightarrow A_1$ and $D_2 \Rightarrow A_2$ then $D_1; D_2 \Rightarrow A_1; A_2$
2. (Monotonicity for star) If $D \Rightarrow A$ then $D^* \Rightarrow A^*$
3. (Associativity) $(D_1; D_2); D_3 \Leftrightarrow D_1; (D_2; D_3)$
4. $P \Rightarrow Q$ then $[P] \Rightarrow [Q]$
5. $([P] \wedge [Q]) \Rightarrow [P \wedge Q]$
6. $[P] \wedge \ell = x + y \Rightarrow ([P] \wedge \ell = x); ([P] \wedge \ell = y)$
7. $[P] \Leftrightarrow [P]; [P]$
8. (Induction rules) Let X be a formula letter occurring in the formula $D(X)$ and let P be a state. Then
If $D([\])$ holds and $D(X \vee X; [P]) \wedge D(X \vee X; [\neg P])$ are provable from $D(X)$, then $D(true)$ holds,
If $D([\])$ holds and $D(X \vee [\neg P]; X) \wedge D(X \vee [\neg P]; X)$ are provable from $D(X)$, then $D(true)$ holds.
9. D^+ holds iff there is $n > 0$ such that D^n holds, where D^n is defined by
 $D^1 \triangleq D, D^{m+1} \triangleq D; D^m$ for all natural $m > 1$
10. $D^*; D^+ \Rightarrow D^+$, and $D^+ \Rightarrow D^*$.

Theorem 4. Let $P_i, i = 1, 2, \dots, n$ be state expressions. Let $D \triangleq [\bigvee_{i=1}^n P_i]$. Then

$$D \Leftrightarrow \left(\bigvee_{i=1}^n [P_i] \right)^+$$

Proof. $(\bigvee_{i=1}^n [P_i])^+ \Rightarrow D$ is obvious from the rules listed above. Now, we prove $D \Rightarrow (\bigvee_{i=1}^n [P_i])^+$. First, we assume that $n = 2$. Let $[P_1 \vee P_2]$ be *tt*. We will use the induction rule. Let $H(X) \triangleq X \Rightarrow (([P_1] \vee [P_2])^+ \vee [\])$. Clearly, $H([\])$ holds. Furthermore,

$$\begin{aligned} H(X) &\Rightarrow (X; [P_1] \Rightarrow (([P_1] \vee [P_2])^+; [P_1])) && \text{(monotonicity)} \\ &\Rightarrow (X; [P_1] \Rightarrow ([P_1] \vee [P_2])^+) && \text{(by the rules 9, 10)} \\ &\Rightarrow H(X; [P_1]), \end{aligned}$$

$$\begin{aligned} H(X) &\Rightarrow (X; [\neg P_1] \Rightarrow (([P_1] \vee [P_2])^+; [\neg P_1])) && \text{(monotonicity)} \\ &\Rightarrow (X; [\neg P_1] \Rightarrow ([P_1] \vee [P_2])^+; ([\neg P_1] \wedge [P_1 \vee P_2])) \\ &\text{(by the rule 7 and assumption)} \\ &\Rightarrow (X; [\neg P_1] \Rightarrow ([P_1] \vee [P_2])^+; [P_2]) && \text{(by the rule 5)} \\ &\Rightarrow (X; [\neg P_1] \Rightarrow ([P_1] \vee [P_2])^+) && \text{(by the rules 9, 10)} \\ &\Rightarrow H(X; [\neg P_1]) \end{aligned}$$

By the induction rule, $H(\text{true})$ holds, which completes the proof for $n = 2$. The case that $n > 2$ follows directly from this case and the above rules.

3.2 Representing the behavior of real-time automata

DC* formulas can be used to describe the behavior of finite real-time automata in the same way that the regular expressions represent the behavior of finite automata. For example, the real-time automaton in Fig. 8 is represented by the following formula

$$D = ([N] \wedge \ell \geq 30; [leak] \wedge \ell \leq 1)^*; ([N] \vee [\])$$

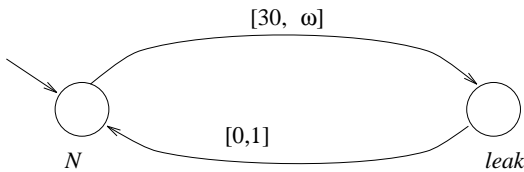


Fig. 8

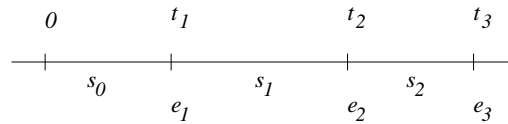


Fig. 9

This is formalized in the following definition:

Definition 3. Let A be a real-time automaton with a set S of states which satisfies the condition that every behavior of it can be extended to an infinite behavior. Each infinitely timed sequence of consecutive transitions $\sigma = (e_1, t_1)(e_2, t_2) \dots$ of A defines an interpretation I^σ of state functions in S in the natural way (see Fig. 9). Namely, for a state $s \in S$,

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

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

Definition 4. Simple DC* formulas are the DC* formula generated by the following syntax:

$$D ::= [\] \mid [P] \mid [P] \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 $([s] \wedge a \leq \ell \leq b)$ 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. [6]) we have the following theorem:

Theorem 5. 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 , 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 .

4 Refinement of DC Formulas into I/O real-time Automata

In this section we present a systematic method for refining DC formulas into automata models through a simple example. 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' that satisfies the condition of Theorem 3. In fact, we only need to find a finite sub-automaton A'' of A that represents a subset of behaviors of A' satisfying the requirement. Then, the input and output functions are added to A'' to make it satisfy the condition of Theorem 3. Starting from $D_1 = D$, by stepwise refinements, we find DC* formulas D_2, D_3, \dots, D_n such that $D_i \Rightarrow D_{i-1}$, $i = 2, \dots, n$, and D_n is a simple DC* formula. By Theorem 5, A'' is derived directly from D_n .

We illustrate our method through the Gas Burner Example also. A physical description of the system may be as follows.

1. States:

- Non-leak (N): $OnReq, OffReq, Burn, Idle, Rec$ (N is called abstract state),
- Leak (L),

2. Events:

$$\begin{array}{llll}
 On \cong (Idle, OnReq) & \text{(heat request)} & Off \cong (Burn, OffReq) & \text{(off request)} \\
 R \cong (L, Rec) & \text{(recover)} & IgOK \cong (OnReq, Burn) & \text{(ignition OK)} \\
 Igfl \cong (OnReq, L) & \text{(ignition failure)} & Ffl \cong (Burn, L) & \text{(flame failure)}
 \end{array}$$

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

The requirement written in DC is as $D_1 \triangleq \ell \geq 60 \Rightarrow \int L \leq 0.05 * \ell$, which means that the duration of $leak$ is no more than 5% of the interval of observation time if the interval of observation time is at least one minute. Now, we consider a design of the system:

$$D_2 = (\Box [leak] \Rightarrow \ell \leq 1) \wedge (\Box [leak]; [N]; [leak] \Rightarrow \ell \geq 30)$$

We have to derive a plant automaton that has duration formulas built from physical states of the system and has behaviors satisfying the requirement.

It has been proved that $D_2 \Rightarrow D_1$ (see [3]). An abstract description of the system is $[N \vee L]$, which is $([N] \vee [\]); ([leak]; [N])^*; ([leak] \vee [\])$ by Theorem 4. Those behaviors of the system that satisfy D_2 are represented by $([N] \vee [\]); ([leak] \wedge \ell \leq 1; [N] \wedge \ell \geq 30)^*; ([leak] \wedge \ell \leq 1 \vee [leak] \wedge \ell \leq 1; [N])$. A real-time automaton derived from this formula is depicted in Fig. 8.

Since $[N] \wedge \ell \geq 30$ is not a state of the plant automaton, we need to refine the formula. By the description of the system and by Theorem 4,

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

From the description of events and noticing that $true; [s]; [s']; true$ iff (s, s') is an event, it follows:

$$\begin{aligned}
 [N] \wedge \ell \geq 30 &= ([Rec]; ([Idle]; [OnReq]; [OffReq]; [Burn])^*; \\
 & \quad ([Idle]; [OnReq]; [OffReq] \vee [Idle]; [OnReq] \vee [Idle] \vee [\])) \wedge \ell \geq 30.
 \end{aligned}$$

Let

$$\begin{aligned}
 D_3 \triangleq & [Rec] \wedge \ell \geq 30; ([Idle]; [OnReq]; [OffReq]; [Burn])^*; \\
 & ([Idle]; [OnReq]; [OffReq] \vee [Idle]; [OnReq] \vee [Idle] \vee [\]).
 \end{aligned}$$

By the proof rules of DC, $D_3 \Rightarrow [N] \wedge \ell \geq 30$. D_3 a simple DC* formula. An automaton having the behavior represented by this formula has the state diagram illustrated as Fig. 10.

Having replaced $\lceil N \rceil \wedge \ell \geq 30$ by this automaton, with the changes that the transition from *leak* to *N* is replaced by the one from *leak* to *Rec*, and the transition from *N* to *leak* is replaced by the ones from *OnReq* to *leak* and *burn* to *leak* (the only possible replacements from the description of events), we get the automaton with the state diagram in Fig 6. After incorporating the input and output functions as in Fig. 6, we obtain a real time I/O automaton satisfying the condition of Theorem 3 as shown in the previous example.

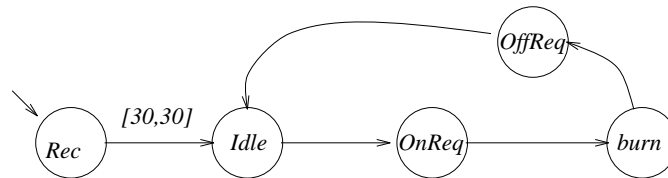


Fig. 10

Acknowledgment:

The authors would like to thank Prof. Zhou Chaochen, Chris George and Dr. Xu Qiwen for their comments and criticism.

References

- [1] L. Alur and D. L. Dill, A theory of timed automata, *Theoretical Computer Science*, 126, 1994, 183-235.
- [2] P. J. Antsaklis, J. A. Stiver, M. Lemmon, Hybrid system modeling and autonomous control systems, LNCS 736, 366-392.
- [3] Z. Chaochen, C. A. R. Hoare, A. P. Ravn, A calculus of durations, *Information Processing Letters*, 40, 269-276.
- [4] Z. Chaochen, A. P. Ravn, M. R. Hansen, Extended Duration Calculus for Hybrid Real-time Systems, LNCS 736, 36-59.
- [5] T. A. Henzinger, Z. Manna, A. Pnueli, Towards refining temporal specifications into hybrid systems, LNCS, 736, 1993, 60-76.
- [6] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Reading, MA, Addison-Wesley, 1979.
- [7] R. Kurki-Suonio, Hybrid Models with Fairness and Distributed Clocks, LNCS 736, 1993, 103-120.
- [8] Z. Manna, *Logics of Programming*, Nord-Holland, 1985.
- [9] D. Murphy, Time and duration in non-interleaving concurrency, *Fundamenta Informaticae*, 19, 1993, 403-416.
- [10] A. Nerode, W. Kohn, Models for hybrid systems: automata, topologies, controllability, observability, LNCS 736, 317-356.
- [11] X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, An approach to the description and analysis of hybrid systems, LNCS 736, 149-178.
- [12] C. Zong Ji et al. An Abstraction of Hybrid Control Systems, *UNU/IIST Technical Report*, 1994.