Effective control of logical discrete event systems in a trace theory setting using the reflection operator

Smedinga, Rein

Published in:
11th international conference on Analysis and optimalization of systems, discrete event systems, Sophia-Antipolis, June 15-17, 1994

Effective control of logical discrete event systems in a trace theory setting using the reflection operator*

Rein Smedinga

department of computing science, University of Groningen
p.o.box 800, 9700 AV Groningen, the Netherlands
tel. +31 50 633937 , fax +31 50 633800, E-mail: rein@cs.rug.nl

Logical discrete event systems can be modelled using trace theory. In this paper we present an effective algorithm to find a controller using an operator (the reflection) that leads to systems that go beyond our scope.
We define a discrete event system (DES) to be a triple, see [Sme93b, Sme93c]:

$$P = \langle \mathbf{a}P, \mathbf{b}P, \mathbf{t}P \rangle$$

with $\mathbf{a}P$ the alphabet (set of events), $\mathbf{b}P \subseteq (\mathbf{a}P)^*$ the behaviour set, and $\mathbf{t}P \subseteq (\mathbf{a}P)^*$ the task set. $\mathbf{a}P$ is a finite set of symbols, $\mathbf{b}P$ and $\mathbf{t}P$ are possibly infinite sets of strings of symbols (traces). For any $x \in \mathbf{t}P$ we assume that, after x, P may stop without performing another event, while after $x \in \mathbf{b}P \backslash \mathbf{t}P$ the system will eventually perform another event or it deadlocks.
We call a DES *realistic*, if the behaviour is prefix-closed: $\mathbf{b}P = \mathbf{pref}(\mathbf{b}P)$ and each completed task is a behaviour: $\mathbf{t}P \subseteq \mathbf{b}P$. An unrealistic DES goes beyond our scope of a discrete event system. Nevertheless, it will play a crucial role in the remainder of this paper.

The restriction of a trace x to some alphabet A, $x\lceil A$, is defined by $\epsilon\lceil A = \epsilon$, and $xa\lceil A = x\lceil A$, if $a \notin A$, and $xa\lceil A = (x\lceil A)a$, otherwise. Here ϵ denotes the empty string. Alphabet restriction can easily be extended to work on trace sets: $T\lceil A = \{x\lceil A \mid x \in T\}$, and on DESs: $P\lceil A = \langle \mathbf{a}P \cap A, \mathbf{b}P\lceil A, \mathbf{t}P\lceil A \rangle$.

For DESs P and R we define the interaction, see [Sme93b, Sme93c], by:

$$P \| R = \langle \mathbf{a}P \cup \mathbf{a}R,\ \{x \mid x\lceil \mathbf{a}P \in \mathbf{b}P \ \wedge\ x\lceil \mathbf{a}R \in \mathbf{b}R\},\ \{x \mid x\lceil \mathbf{a}P \in \mathbf{t}P \ \wedge\ x\lceil \mathbf{a}R \in \mathbf{t}R\} \rangle$$

For interactions of systems the common events can be seen as internal events. Such events need no longer be visible outside the interaction. Therefore, we introduce the external interaction operator that deletes the common events:

$$P \rceil\lceil R = (P \| R)\lceil (\mathbf{a}P \div \mathbf{a}R)$$

* In Guy Cohen and Jean-Pierre Quadrat, editors, 11th International Conference on Analysis and optimalization of Systems, Discrete event systems, Sophia-Antipolis, june 15-17, 1994, number 199 in Lecture notes in Control and Information sciences, pages 66-72. Springer Verlag, 1994.

For systems with equal alphabets we say P is a *subsystem* of R if:

$$P \subseteq R \;\Leftrightarrow\; \mathbf{a}P = \mathbf{a}R \;\wedge\; \mathbf{b}P \subseteq \mathbf{b}R \;\wedge\; \mathbf{t}P \subseteq \mathbf{t}R$$

For DESs P and R with equal alphabets ($\mathbf{a}P = \mathbf{a}R$) we define the difference by

$$P \backslash R = \langle \mathbf{a}P, \mathbf{b}P \backslash \mathbf{b}R, \mathbf{t}P \backslash \mathbf{t}R \rangle$$

From Verhoef [T.V90, T.V91] we have the following definition of the reflection of some DES (in fact the complementary system):

$$\sim P = \langle \mathbf{a}P,\ (\mathbf{a}P)^* \backslash \mathbf{b}P,\ (\mathbf{a}P)^* \backslash \mathbf{t}P \rangle$$

If P and R are realistic, it can be shown that $P \| R$, $P]\![R$, and $P \lceil A$ are also realistic. Notice that, if P is realistic, $\sim P$ need not be. This is why we need unrealistic DESs as well. The *realistic interior* of some DES P is defined by

$$\begin{aligned} \mathbf{real}(P) = \langle \mathbf{a}P,\ & \{x \mid x \in \mathbf{b}P \;\wedge\; (\forall y : y \in \mathbf{pref}(x) : y \in \mathbf{b}P)\}, \\ & \{x \mid x \in \mathbf{t}P \;\wedge\; (\forall y : y \in \mathbf{pref}(x) : y \in \mathbf{b}P)\} \rangle \end{aligned}$$

$\mathbf{real}(P)$ is the greatest realistic subsystem of P.

In the sequel we will use $a, b, \ldots$ to denote events, $x, y, \ldots$ to denote strings, and $P, R, \ldots$ to denote DESs. $|A|$ denotes the number of elements in a set A.

1 A control problem

Assume systems P, $L_{\min}$, and $L_{\max}$ are given with $L_{\min} \subseteq L_{\max}$. Our control problem is finding a system R such that $L_{\min} \subseteq P]\![R \subseteq L_{\max}$.
In this formulation $L_{\min}$ and $L_{\max}$ describe minimal and maximal wanted behaviours of the interaction. Mostly, $L_{\min}$ describes the minimal acceptable behaviour and $L_{\max}$ the legal or admissible behaviour. Notice that $\mathbf{a}L_{\min} = \mathbf{a}L_{\max}$ and R should be such that $\mathbf{a}R = \mathbf{a}P \div \mathbf{a}L_{\min}$. Events from $\mathbf{a}R$ are used to control the order of the remaining events. Earlier versions of this control problem (formulated using trace structures instead of DESs) can be found in [Sme89]. From [Sme92, Sme93c] we know that $F(P, L) = \sim(P]\![\sim L)$ may lead to a solution:

Theorem 1. *The control problem has a solution if and only if*

$$L_{\min} \subseteq P]\![F(P, L_{\max})$$

and, if it is solvable, the greatest solution (with respect to $\subseteq$) is $F(P, L_{\max})$.
If P, $L_{\min}$, and $L_{\max}$ are realistic, the greatest possible realistic solution equals $\mathbf{real}(F(P, L_{\max}))$.

2 State graphs

In order to have algorithms to compute solutions for our control problem effectively, we introduce so-called state graphs for our DESs and construct a controller, according to theorem 1, in terms of algorithms on these state graphs.

It is well-known that we can associate with a trace structure (language) a (finite) state automaton. Each path in the automaton, starting in the initial state and ending in a final state corresponds to a trace in the trace set. If the trace structure is regular, the number of needed states is finite. A DES is in fact a pair of trace structures, so we could use two automatons to represent one DES. However, in this way we lose the correspondence between behaviour and task. Instead, we use a more general automaton, called a state graph here, containing two kinds of final states:

Definition 2. A state graph is a tuple (A, Q, δ, q, B, T) with A the alphabet, a finite set of labels; Q the state set; $\delta: Q \times A \to Q$ the state transition function; $q \in Q$ the initial state; $B \subseteq Q$ the behaviour state set; and $T \subseteq Q$ the task state set. δ is a total function.

The state set Q need not be a finite set. Because we deal with paths in the graph, we extend δ to $\delta^*: Q \times A^* \to Q$, by: $\delta^*(p, \epsilon) = p$, and $\delta^*(p, xa) = \delta(\delta^*(p, x), a)$.

For a DES P we can construct a state graph using (extended) Nerode equivalence with equivalence classes:

$$[x]_P = \{y \mid (\forall z :: xz \in \mathbf{b}P \Leftrightarrow yz \in \mathbf{b}P \ \wedge \ xz \in \mathbf{t}P \Leftrightarrow yz \in \mathbf{t}P)\}$$

Given some state graph $G = (A, Q, \delta, q, B, T)$ the corresponding DES equals

$$\mathbf{des}(G) = \langle A, \ \{x \mid x \in A^* \ \wedge \ \delta^*(q, x) \in B\}, \ \{x \mid x \in A^* \ \wedge \ \delta^*(q, x) \in T\}\rangle$$

and given some system P a possible state graph is:

$$\mathbf{sg}(P) = (\mathbf{a}P, \{[x]_P \mid x \in (\mathbf{a}P)^*\}, \delta, [\epsilon]_P, \{[x]_P \mid x \in \mathbf{b}P\}, \{[x]_P \mid x \in \mathbf{t}P\})$$

with δ defined by $\delta([x]_P, a) = [xa]_P$.
If the behaviour and the task set of a system are regular sets, the number of equivalence classes is finite and the resulting state graph has only a finite number of states. State graphs can be displayed as is shown in figure 1.

We have $\mathbf{des}(\mathbf{sg}(P)) = P$, but, in general, $\mathbf{sg}(\mathbf{des}(G)) \neq G$, because more state graphs exist that correspond to the same DES.

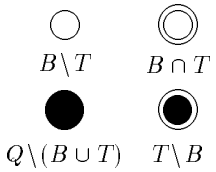

Fig. 1. Displaying of different states

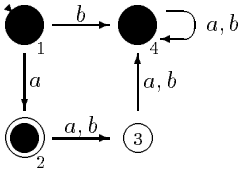

Fig. 2. System from example 1

Example 1. Consider the system $P = \langle \{a,b\}, \{aa, ab\}, \{a\} \rangle$ The following equivalence classes can be found: $p_1 = [\epsilon] = \{\epsilon\}$, $p_2 = [a] = \{a\}$, $p_3 = [aa] = \{aa, ab\}$, and $p_4 = [b] = \{a,b\}^* \backslash (p_1 \cup p_2 \cup p_3)$. In figure 2 the corresponding graph is shown. $q = [\epsilon]$ is denoted using an extra small arrow.

For the operator $]\![$ we need a nondeterministic graph (nd-graph), because we may have to delete labels. This results in graphs in which δ is no longer a function.

Definition 3. An nd-graph is a tuple $(A, Q, \gamma, q, B, T)_{\mathrm{nd}}$ with A, Q, q, B, and T as in definition 2 and $\gamma : Q \times (A \cup \{\epsilon\}) \to \mathbf{2}^Q$ the state transition map. Again γ is supposed to be total.

The DES $\mathbf{des}(G_{\mathrm{nd}})$ corresponding to an nd-graph G_{nd} is given by

$$\langle A,\ \{x \mid x \in A^* \ \wedge\ \gamma^*(q,x) \cap B \neq \emptyset\},\ \{x \mid x \in A^* \ \wedge\ \gamma^*(q,x) \cap T \neq \emptyset\} \rangle$$

where $\gamma^*(Q', x)$ represents the set of all states, reachable from a state in $Q' \subseteq Q$ via a path x including zero or more ϵ-transitions. γ^* is the extension of γ.

Example 2. In figure 5 an nd-graph is given for $P = \langle \{a,b\}, \{\epsilon, a, b, ab\}, \{a, b, ab\} \rangle$

In the remainder of this paper we use state graphs $G_i = (A_i, Q_i, \delta_i, q_i, B_i, T_i)$ $(i = , 1, 2)$ and nd-graph $G_{\mathrm{nd}} = (A, Q, \gamma, q, B, T)_{\mathrm{nd}}$. We give algorithms on state graphs for all operators on DESs. We will use the same operator symbol for DESs as well as for state graphs.

3 Algorithms on state graphs

Algorithm 1. $G_1 \| G_2 = (A_1 \cup A_2, Q_1 \times Q_2, \delta, (q_1, q_2), B_1 \times B_2, T_1 \times T_2)$

where $\delta((p_1,p_2),a) = \begin{cases} (\delta_1(p_1,a), p_2) & \text{if } a \in A_1 \backslash A_2 \\ (p_1, \delta_2(p_2,a)) & \text{if } a \in A_2 \backslash A_1 \\ (\delta_1(p_1,a), \delta(p_2,a)) & \text{if } a \in A_1 \cap A_2 \end{cases}$

Complexity: $\mathcal{O}(|Q_1| \cdot |Q_2| \cdot |\mathbf{a}P \cup \mathbf{a}R|)$

Property: $\mathbf{des}(\mathbf{sg}(P_1) \| \mathbf{sg}(P_2)) = P_1 \| P_2$

Example 3. Consider the graphs as given in figure 3. According to the previous property the graph for the interaction is as given in figure 4. Notice that this graph is not minimal: all non-behaviour states can be replaced by one non-behaviour state.

Algorithm 2. $\mathbf{det}(G_{\mathrm{nd}}) = (A, \mathbf{2}^Q, \delta, \overline{q}, \overline{B}, \overline{T})$

where (for $r \in \mathbf{2}^Q$): $\overline{B} = \{r \mid r \cap B \neq \emptyset\}$ $\quad \overline{q} = \gamma^*(q, \epsilon)$

$\overline{T} = \{r \mid r \cap T \neq \emptyset\}$ $\quad \delta(r,a) = \bigcup_{p \in r} \gamma^*(p,a)$

Property: $\mathbf{des}(\mathbf{det}(G_{\mathrm{nd}})) = \mathbf{des}(G_{\mathrm{nd}})$

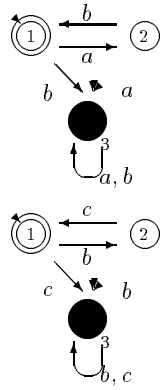

Fig. 3. State graphs G_1 (upper diagram) and G_2 for example 3

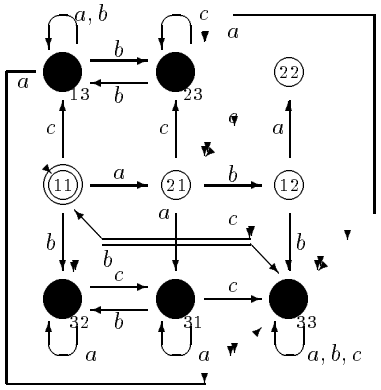

Fig. 4. Interaction of G_1 and G_2

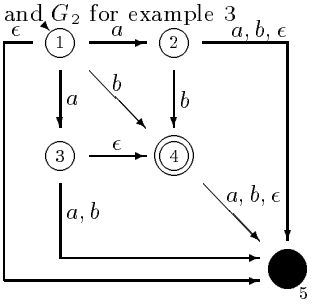

Fig. 5. An nd-graph representing the system from example 2

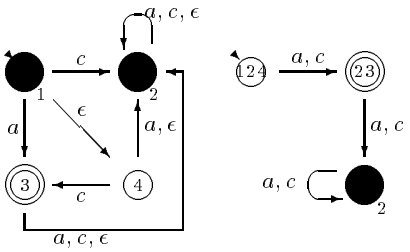

Fig. 6. An nd-graph and its deterministic equivalence for example 4

The above construction is a generalization of the well-known construction to find the deterministic equivalent of a nondeterministic automaton. Apart from unreachable states, each state in $\mathbf{det}(G_{\mathrm{nd}})$ is the set of states that can be reached from another set by doing zero or more ϵ-moves, followed by one normal move, followed by zero or more ϵ-moves.

Example 4. We can use the above construction on the graph of figure 6 (left) to get a deterministic graph. We find: $\overline{q} = \gamma^*(q,\epsilon) = \{p_1,p_2,p_4\}, \gamma^*(\{p_1,p_2,p_4\},a) = \{p_2,p_3\}$, $\gamma^*(\{p_1,p_2,p_4\},c) = \{p_2,p_3\}$, $\gamma^*(\{p_2,p_3\},a) = \{p_2\}$, $\gamma^*(\{p_2,p_3\},c) = \{p_2\}$, $\gamma^*(\{p_2\},a) = \{p_2\}$, and $\gamma^*(\{p_2\},c) = \{p_2\}$, which leads to the graph of figure 6 (right). Notice that we only have examined the reachable states. The behaviour and task states are $\overline{B} = \{\{p_1,p_2,p_4\},\{p_2,p_3\}\}$ and $\overline{T} = \{\{p_2,p_3\}\}$. It can easily be checked that this graph also represents the same system.

Algorithm 3. $G\lceil A_1 = \mathbf{det}(G_{\mathrm{nd}})$ with $G_{\mathrm{nd}} = (A \cap A_1, Q, \gamma, q, B, T)_{\mathrm{nd}}$
where: $\gamma(p,\epsilon) = \bigcup_{a \in A \setminus A_1} \delta(p,a)$ $\quad \gamma(p,a) = \{\delta(p,a)\}$ (for $a \in A \cap A_1$)
Complexity: $\mathcal{O}(|A| \cdot |Q|)$ (for computing G_{nd})
Property: $\mathbf{des}(\mathbf{sg}(P)\lceil A) = P\lceil A$

We also have: $\mathbf{des}\big((\mathbf{sg}(P)\|\mathbf{sg}(R))\lceil(\mathbf{a}P \div \mathbf{a}R)\big) = P]\lceil R$, that shows a way to get a graph for $P]\lceil R$:

Algorithm 4. $G_1 \lceil G_2 = (G_1 \| G_2) \lceil (A_1 \div A_2)$

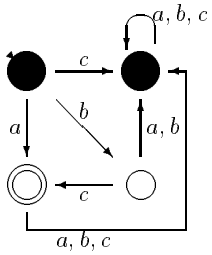

Fig. 7. $\mathbf{sg}(P)$ for example 5

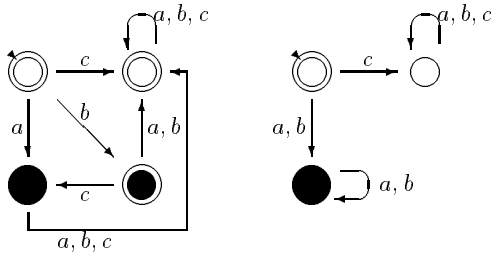

Fig. 8. Graph $\sim\mathbf{sg}(P)$ (left) and its realistic interior for example 6.

Example 5. Consider the system $P = \langle \{a,b,c\}, \{a,b,bc\}, \{a,bc\} \rangle$. Its corresponding graph $\mathbf{sg}(P)$ is given in figure 7. The graph $\mathbf{sg}(P) \lceil \{a,c\}$ is to be found in figure 6, constructed using algorithm 3. Making this graph deterministic leads to a graph representing the system $\langle \{a,c\}, \{\epsilon, a, c\}, \{a,c\} \rangle$ which is equal to $P \lceil \{a,c\}$.

The reflection operator of a graph is simply the graphs complement, i.e., interchange the types of all states:

Algorithm 5. $\sim G = (A, Q, \delta, q, Q \backslash B, Q \backslash T)$
Complexity: $\mathcal{O}(|Q|)$
Property: $\mathbf{des}(\sim\mathbf{sg}(P)) = \sim P$

Finding the DES-interior means finding all states reachable from q with paths not going through non-behaviour states:

Algorithm 6. $\mathbf{real}(G) = (A, \{q\}, \mathbf{1}, q, \emptyset, \emptyset)$ if $q \notin B$
$= (A, B \cup \{\overline{q}\}, \overline{\delta}, q, B, F \cap B)$ otherwise
where $\overline{q}$ is a fresh state ($\notin B$) and $\overline{\delta}(p,a) = \delta(p,a)$ if $\delta(p,a) \in B$, $\overline{\delta}(p,a) = \overline{q}$ otherwise, and $\overline{\delta}(\overline{q}, a) = \overline{q}$ for all $a \in A$.
Complexity: $\mathcal{O}(|A| \cdot |Q|)$
Property: $\mathbf{des}(\mathbf{real}(\mathbf{sg}(P))) = \mathbf{real}(P)$

The graph $(A, \{q\}, \mathbf{1}, q, \emptyset, \emptyset)$ is a representation for the empty system:

$$\mathbf{des}((A, \{q\}, \mathbf{1}, q, \emptyset, \emptyset)) = \langle A, \emptyset, \emptyset \rangle$$

Algorithm 7. $G_1 \backslash G_2 = (A, Q_1 \times Q_2, \delta, (q_1, q_2), B_1 \times Q_2 \backslash B_2, T_1 \times Q_2 \backslash T_2)$
with $\delta((p_1,p_2),a) = (\delta_1(p_1,a), \delta_2(p_2,a))$.
Complexity: $\mathcal{O}(|A) \cdot |Q_1| \cdot |Q_2|)$
Property: $\mathbf{des}(\mathbf{sg}(P) \backslash \mathbf{sg}(R)) = P \backslash R$

Example 6. Computing $\sim\mathbf{sg}(P)$ for the system displayed in figure 7 leads to the system as displayed in figure 8 (left). In figure 8 (right) the realistic part of that system can be found.

Example 7. Reconsider the systems as displayed in figure 3. Computing $G_1 \backslash G_2$ leads to the same graph as in figure 4 but with other state types: $B = \{p_{13}\}$ and $T = \{p_{12}, p_{13}\}$.

Because we deal with paths in a graph starting in the initial state it has no effect if states ared added that cannot be reached from the initial state. Such states can, if present, also easily be eliminated. Algorithms can be made more efficient if only the states, reachable from the initial state, are really computed. Moreover, we can use an extension of the standard algorithm on automatons (see [HU79]) to minimize state graphs. Also, we can easily extend the operators to work on nd-graphs as well. Therefore we can do without the operator **det** from algorithm 2.

4 Effectively computable

If P and L are regular, i.e., can be displayed using finite state graphs, we see from the algorithms and properties above, that $\mathbf{real}(F(P, L))$ can be computed in polynomial time. Moreover, to test the condition for having a solution, we can compute the graph equivalence of $L_{\min} \backslash (P \,]\![\, \mathbf{real}(F(P, L_{\max})))$. If $L_{\min} \subseteq (P \,]\![\, \mathbf{real}(F(P, L_{\max})))$, this results in an empty graph, i.e., $B = \emptyset$ and $T = \emptyset$.

Theorem 4. *Let $G_{\min}$, $G_{\max}$, and G_P be state graphs for $L_{\min}$, $L_{\max}$, and P, respectively. Then is*

$$G_R := \mathbf{real}(\sim(G_P \,]\![\, \sim G_{\max}))$$

a state graph for $\mathbf{real}(F(P, L_{\max}))$. It can be computed in polynomial time. If computation of $G_{\min} \backslash (G_P \,]\![\, G_R)$ results in an empty graph, the control problem is solvable and G_R represents the largest possible solution. Also this last computation can be done in polynomial time.

5 Conclusions

We have shown that a trace theory based approach leads to an elegant definition of a logical discrete event system and gives a nice algorithm to find a solution for a control problem, where we, temporally, go beyond the scope of a DES. The algorithm can be translated to work on state graphs, leading to an effectively computable solution if the systems itself are regular. Proofs of all properties can be found in Smedinga [Sme92].

References

[BKS93] S. Balemi, P. Kozák, and R. Smedinga, editors. *Discrete Event Systems: Modeling and Control*, volume 13 of *Progress in Systems and Control Theory*. Birkhäuser Verlag, Basel, Switzerland, 1993. (Proceedings of the Joint Workshop on Discrete Event Systems (WODES'92), August 26–28, 1992, Prague, Czechoslovakia).

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages, and computation.* Addison Wesley, 1979.

[Sme89] R. Smedinga. *Control of discrete events.* PhD thesis, University of Groningen, 1989.

[Sme92] R. Smedinga. The reflection operator in discrete event systems. Technical Report CS9201, Department of computing science, University of Groningen, 1992.

[Sme93a] R. Smedinga. Discrete event systems. course-notes, second version, Department of computing science, University of Groningen, 1993.

[Sme93b] R. Smedinga. Locked discrete event systems: how to model and how to unlock. *Journal on Discrete Event Dynamic Systems, theory and applications*, 2(3/4), 1993.

[Sme93c] R. Smedinga. *An Overview of Results in Discrete Event Systems using a Trace Theory Based Setting*, pages 43–56. Volume 13 of Balemi et al. [BKS93], 1993.

[T.V90] T.Verhoeff. Solving a control problem. Internal report, Eindhoven University, 1990.

[T.V91] T.Verhoeff. Factorization in process domains. Internal report, Eindhoven University, 1991.

This article was processed using the LaTeX macro package with LMAMULT style