ON THE SUPREMAL L-CONTROLLABLE SUBLANGUAGE OF A NON PREFIX-CLOSED LANGUAGE

Roberto M. Ziller and José E. R. Cury
LCMI - EEL - UFSC
88040-900 - Florianópolis - SC -- Brazil
e-mail: ziller@lcmi.ufsc.br

ABSTRACT

In the Ramadge-Wonham framework for Discrete Event Systems modeling, supervisor synthesis problems are formulated and solved using languages and automata theory. Computing solutions to a class of synthesis problems amounts to find the *supremal* L_m-*closed and L-controllable sublanguage* of the language that represents the target system behavior. The computation of the *supremal L-controllable sublanguage* of a given language is therefore a fundamental issue in supervisor synthesis. It has been extensively discussed in the literature; formulas and algorithms to calculate this language in the particular case of prefix-closed specifications have been found, and an iterative operator is given in [WR87] for the general case where the languages involved need not be prefix-closed. This paper presents a formula for the general case and outlines its computer implementation.

Keywords: Discrete Event Systems, Supervisory Control, Language and Automata Theory

1 INTRODUCTION

In the last ten years, the Ramadge-Wonham framework for Discrete Event Systems modeling, analysis and controller (supervisor) synthesis has advanced to an important position among the various models suggested in this research field.

In the basic RW-model, both the system to be controlled and the desired closed-loop behavior are specified through the use of languages. Solving a synthesis problem amounts to find a controller - called *supervisor* - that restricts the physically possible behavior of the system to be controlled - called *plant* - to the desired one. Necessary and sufficient conditions for the existence of such a supervisor were presented in [RW87], and the solution of the basic synthesis problem is stated in terms of the *supremal* $L_m(G)$*-closed and* $L(G)$*-controllable sublanguage* of the language representing the target behavior, where $L_m(G)$ and $L(G)$ are languages associated with an automaton G representing the plant.

As shown in [Zill93] and [ZC94], the computation of this language can be done in two steps, first taking the supremal $L_m(G)$-closed sublanguage of the target language and then taking the supremal $L(G)$-controllable sublanguage - from this point on abbreviated by supC - of the result.

The computation of supC is hence a fundamental issue in supervisor synthesis. It has been discussed in [WR87], [Rudi88], [LVW88], [BGK+90] and [KGM91]. The methods to find supC can be divided into two classes, namely those applicable only when the languages involved are prefix-closed ([LVW88], [BGK+90], [KGM91]) and those that apply to both prefix-closed and non prefix-closed languages ([WR87], [Rudi88]). For prefix-closed languages, [LVW88] and [KGM91] present algorithms and [BGK+90] gives a formula for supC; for the general case when the languages are not necessarily prefix-closed, [WR87] presents an operator which leads to an iterative algorithm implemented in [Rudi88]. As far as the authors know, no formula in the sense of an expression for supC like the one presented in [BGK+90] is known for the general case.

In this paper we modify the operator Ω given in [WR87] so that it will always converge at the first iteration, thus obtaining a formula for the supremal $L(G)$-controllable sublanguage which is valid in the general case.

The paper is organized as follows: sections 2 and 3 recall the results needed from RW-theory; section 4 recalls the Ω operator from [WR87] and presents the new formula for supC, which we call Ω^*; section 5 outlines an algorithm to compute it.

2 PRELIMINARIES

In the Ramadge-Wonham framework, Discrete Event Systems are modeled as generators of formal languages. A detailed description of this model is given in [RW87] and related articles, the reader being referred to these sources for background knowledge. In the present paper, we only recall some facts needed to present our contribution.

System behavior is represented by a 5-tuple $G = \langle \Sigma, Q, \delta, q_0, Q_m \rangle$ called *generator,* where Σ is a set of event labels, also called the *event alphabet*, Q is a set of states, and $\delta : \Sigma \times Q \to Q$ is a (generally partial) transition function defined at each $q \in Q$ for a subset of the $\sigma \in \Sigma$ so that $q' = \delta(\sigma, q)$ represents the state transition $q \xrightarrow{\sigma} q'$, meaning that the occurrence of event σ takes the system from state q to state q'. $q_0 \in Q$ is the *initial state* and $Q_m \subseteq Q$ is a set of *marker states.* These are used to mark the termination of certain event sequences, representing the completion of a task by the system.

The set of all strings that can be formed by any number of symbols from Σ, plus the empty string ε is denoted by Σ^*. The transition function is extended to process strings from Σ^* in the following natural way:

$$\delta : \Sigma^* \times Q \to Q = \begin{cases} \delta(\varepsilon, q) = q \\ \delta(s\sigma, q) = \delta(\sigma, \delta(s, q)). \end{cases}$$

A state $q \in Q$ is said to be *accessible* iff there exists some string $s \in \Sigma^*$ such that $\delta(s, q_0) = q$; state $q \in Q$ is *coaccessible* iff there exists some string $s \in \Sigma^*$ such that $\delta(s, q) \in Q_m$. A generator G is (co)accessible iff all its states are (co)accessible; G is *trim* iff it is accessible and coaccessible.

Each generator G has two associated *languages:* $L(G)$, the language *generated* by G, and $L_m(G)$, the language *marked* by G. These are sets of *words* formed with symbols of Σ. $L(G)$ represents the physically possible behavior of the system, while $L_m(G)$ stands for the tasks it is able to complete.

The alphabet Σ is partitioned into *controllable* and *uncontrollable* events according to $\Sigma = \Sigma_c \bigcup \Sigma_u$ and $\Sigma_c \bigcap \Sigma_u = \varnothing$. Control action is performed by an external agent called *supervisor* which observes the events generated by the plant and applies a *control input* $\gamma \subseteq \Sigma$ to the system in response to them. The events in γ are those specified to be enabled by the supervisor.

This control action restricts the system generated and marked languages. The languages representing the physically possible behavior and the tasks the system may complete under supervision are denoted by $L(S/G)$ and $L_c(S/G)$, respectively.

The prefix-closure $\overline{K}$ of a language K is the set of all prefixes (initial segments) of strings in K. K is said to be *prefix-closed* iff $K = \overline{K}$. $L(G)$ is always prefix-closed, while $L_m(G)$ does not need to be.

Given two arbitrary languages $K, L \subseteq \Sigma^*$ and a partition $\Sigma = \Sigma_c \bigcup \Sigma_u$, K is said to be *L-closed* iff $K = \overline{K} \bigcap L$; K said to be *L-controllable* iff $\overline{K}\Sigma_u \bigcap L \subseteq \overline{K}$.

A supervisor is said to be *complete* iff it is always able to follow the events generated by the plant; *non rejecting* iff the closed-loop system never enters a state from which it is no longer able to recognize any completed task as such; and *non blocking* iff the closed-loop system is never kept from terminating a task (i.e., kept from reaching a marker state). A *proper* supervisor is a supervisor that is complete, non rejecting and non blocking.

3 SUPERVISOR SYNTHESIS PROBLEMS

The languages presented in section 2 allow the formulation of several abstract supervisor synthesis problems. In the approach chosen in [RW87] the system to be controlled is represented by a plant G. The target behavior is specified as a language $E \subseteq L_m(G)$, representing the tasks desired to be executable under supervision, and the problem objective is to find (if possible) a proper supervisor

S for *G* such that the closed-loop behavior satisfies the equality $L_c(S/G)=E$.

As shown in [RW87], there exists a proper supervisor such that $L_c(S/G)=E$ iff E is both $L_m(G)$-closed and $L(G)$-controllable. These conditions are not always satisfied, since the desired behavior for a system is not necessarily related to that what can actually be achieved. Therefore, the abstract supervisor synthesis problem is formulated in the following way:

Supervisory Control Problem (SCP): Given a plant *G*, a target language $E \subseteq L_m(G)$ and a minimal acceptable language $A \subseteq E$, construct a proper supervisor for *G* such that

$$A \subseteq L_c(S/G) \subseteq E.$$

The language *E* is interpreted as the desired behavior under supervision, while *A* stands for the minimal closed-loop behavior that is still acceptable. Now, if it is not possible to construct a supervisor such that $L_c(S/G)=E$ (because E is not $L_m(G)$-closed and $L(G)$-controllable), one can try to find a more restrictive solution, so that $L_c(S/G) \subset E$ which would be accepted provided $L_c(S/G) \supseteq A$. It is desirable to find an optimal solution in the sense of restricting the closed-loop behavior only as much as necessary to satisfy the supervisor existence conditions. Such an optimal solution exists. It is shown in [RW87] that the class $C(E)$ of all $L(G)$-controllable sublanguages of *E* and the class $F(E)$ of all $L_m(G)$-closed sublanguages of *E* are both non empty and closed under arbitrary unions. Consequently, these classes contain the supremal elements $\sup C(E)$ and $\sup F(E)$, called *supremal* $L(G)$*-controllable sublanguage of E* and *supremal* $L_m(G)$*-closed sublanguage of E*, respectively. The same applies to the class $CF(E)=C(E)\cap F(E)$, so there also exists $\sup CF(E)$, the *supremal* $L(G)$*-controllable and* $L_m(G)$*-closed sublanguage of E.*

The solution to SCP is then given by the following theorem ([RW87], theorem 7.1, part (ii)):

Theorem 3.1: SCP is solvable iff $\sup CF(E) \supseteq A$. ♦

It is clear from theorem 3.1 that an algorithm for $\sup CF(E)$ is needed in order to compute solutions to SCP in the general case. The following lemma states that $\sup CF(E)$ can be computed by first obtaining $\sup F(E)$ and then by applying an algorithm for supC to the result. From this point on, we abbreviate $L(G)$ by L and $L_m(G)$ by L_m where no confusion is possible.

Lemma 3.1: Given a language $E \subseteq L_m$, $\sup CF(E) = \sup C[\sup F(E)]$.

Proof: See [Zill93] (proposition 5.6) or [ZC94] (lemma 3). ♦

A formula for $\sup F(E)$ is given in [ZC94]; from this point on we concentrate on the computation of supC. In the next section we recall the Ω operator for supC introduced in [WR87] and present our extension to it.

4 FROM OPERATOR TO FORMULA

The operator defined in [WR87] (with some changes for notation uniformity) is given by

Definition 4.1: Let $\Omega\!:\!\Sigma^* \to \Sigma^*$ be the operator
$\Omega(K) = E \cap \mathbf{sup}\left[T\!:\!T \subseteq \Sigma^* \wedge T=\overline{T} \wedge T\Sigma_u \cap L(G) \subseteq \overline{K}\right]$,
where

$E \subseteq L(G)$ is an arbitrary (not necessarily prefix-closed) target language,

Σ_u is the set of uncontrollable events and

$K \subseteq \Sigma^*$. ♦

It is shown in [WR87] that the sequence of languages K_j given by

$K_0 = E, K_{j+1} = \Omega(K_j), j \geq 0$

converges to $\sup C(E)$ in a finite number of iterations.

We define the new operator Ω^* as follows:

Definition 4.2: Let $\Omega^*\!:\!\Sigma^* \to \Sigma^*$ be the operator
$\Omega^*(K) = E \cap \mathbf{sup}\left\{T\!:\!T \subseteq \Sigma^* \wedge T=\overline{T} \wedge T\Sigma_u^* \cap L(G) \subseteq \overline{K}\right\}$,
where

$E \subseteq L(G)$ is an arbitrary (not necessarily prefix-closed) target language,

Σ_u^* is the set of all words that can be formed with symbols from Σ_u and

$K \subseteq \Sigma^*$. ♦

The following equivalent formulation will be useful:

Lemma 4.1:
$\Omega^*(K) = \left\{t\!:\!t \in E \wedge \bar{t}\Sigma_u^* \cap L(G) \subseteq \overline{K}\right\}$.

Proof: Immediate from definition 4.2. ♦

We also need the following lemma to extend the definition of controllability:

Lemma 4.2: Let K be an arbitrary (not necessarily prefix-closed) sublanguage of $L(G)$. K is controllable if and only if $\overline{K}\Sigma_u^* \cap L(G) \subseteq \overline{K}$.

Proof:

(If): Assume that $\overline{K}\Sigma_u^* \cap L(G) \subseteq \overline{K}$. Since $\Sigma_u \subseteq \Sigma_u^*$, $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}\Sigma_u^* \cap L(G) \subseteq \overline{K}$.

(Only if): Let Σ_u^n be the set of all words of length n formed with symbols of Σ_u. It will be shown by induction that

$K\Sigma_u^n \cap L(G) \subseteq \overline{K}$ for $n = 0,1,\ldots\infty$. Since K is controllable by hypothesis, the above expression is valid for $n = 1$. Assume that the inductive hypothesis $\overline{K}\Sigma_u^i \cap L(G) \subseteq \overline{K}$ is valid for $n = i$. Then

$$\begin{aligned}
\overline{K}\Sigma_u^{i+1} \cap L(G) &\subseteq \overline{K}\Sigma_u^i\Sigma_u \cap L(G)\\
&\subseteq \overline{K}\Sigma_u^i\Sigma_u \cap L(G)\Sigma_u \cap L(G)\\
&\subseteq \left[\overline{K}\Sigma_u^i \cap L(G)\right]\Sigma_u \cap L(G)\\
&\subseteq \overline{K}\Sigma_u \cap L(G)\\
&\subseteq \overline{K}, \qquad \text{so}
\end{aligned}$$

$\overline{K}\Sigma_u^* \cap L(G) \subseteq \overline{K}$. ♦

The following proposition gives our main result:

Proposition 4.1: The supremal $L(G)$-controllable sublanguage of a given arbitrary language $E \subseteq L(G)$ is given by

$$\sup C(E) = \Omega^*(E) = \left\{t : t \in E \wedge \overline{t}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right\}.$$

Proof:

($\supseteq$): Let the right member of the former expression be denoted by Z. Since $Z \subseteq E$ by definition, showing that $\sup C(E) \supseteq Z$ amounts to show that Z is controllable. Suppose the contrary. Then, according to lemma 4.2,

$\exists u \in \overline{Z} \wedge \exists v \in \Sigma_u^*$ such that

$u \in \overline{Z} \wedge uv \in L(G) \wedge \neg(uv \in \overline{Z})$.

Now $\overline{Z} = \left\{t : t \in \overline{E} \wedge \overline{t}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right\}$, so the above statement becomes

$$\begin{aligned}
&u \in \overline{Z} \wedge uv \in L(G) \wedge\\
&\quad \wedge\neg\left\{uv \in \left\{t : t \in \overline{E} \wedge \overline{t}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right\}\right\}\\
&\quad \wedge\neg\left\{uv \in \overline{E} \wedge \overline{uv}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right\}\\
&\quad \wedge\left\{\neg(uv \in \overline{E}) \vee \neg\left[\overline{uv}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right]\right\}\\
&\left[u \in \overline{Z} \wedge uv \in L(G) \wedge (uv \notin \overline{E})\right] \vee\\
&\quad \left\{u \in \overline{Z} \wedge uv \in L(G) \wedge \neg\left[\overline{uv}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right]\right\}.
\end{aligned}$$

The first term of this expression gives

$$\begin{aligned}
&u \in \left\{t : t \in \overline{E} \wedge \overline{t}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right\} \wedge uv \in L(G) \wedge uv \notin \overline{E}\\
&u \in \overline{E} \wedge \overline{u}\Sigma_u^* \cap L(G) \subseteq \overline{E} \wedge uv \in L(G) \wedge uv \notin \overline{E}\\
&u \in \overline{E} \wedge u\Sigma_u^* \cap L(G) \subseteq \overline{E} \wedge uv \in L(G) \wedge uv \notin \overline{E}\\
&u \in \overline{E} \wedge uv \cap L(G) \subseteq \overline{E} \wedge uv \in L(G) \wedge uv \notin \overline{E}\\
&u \in \overline{E} \wedge uv \in \overline{E} \wedge uv \in L(G) \wedge uv \notin \overline{E},
\end{aligned}$$

a contradiction. The second term gives

$$\begin{aligned}
&u \in \left\{t : t \in \overline{E} \wedge \overline{t}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right\} \wedge\\
&\qquad \wedge uv \in L(G) \wedge \neg\left[\overline{uv}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right]\\
&u \in \overline{E} \wedge \overline{u}\Sigma_u^* \cap L(G) \subseteq \overline{E} \wedge\\
&\qquad \wedge uv \in L(G) \wedge \neg\left[\overline{uv}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right]\\
&u \in \overline{E} \wedge \overline{u}\overline{v} \cap L(G) \subseteq \overline{E} \wedge\\
&\qquad \wedge uv \in L(G) \wedge \neg\left[\overline{uv}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right]\\
&u \in \overline{E} \wedge \overline{uv} \cap L(G) \subseteq \overline{E} \wedge\\
&\qquad \wedge uv \in L(G) \wedge \neg\left[\overline{uv}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right],
\end{aligned}$$

again a contradiction.

($\subseteq$): Since $\sup C(E) \subseteq E$ by definition, it is sufficient to show that $\sup C(E) \subseteq \left\{\overline{t}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right\}$.

Suppose the contrary. Then $\exists u \in \Sigma^*$ such that

$u \in \sup C(E) \wedge \neg(u \in \left\{t : \overline{t}\Sigma_u^* \cap L(G) \subseteq \overline{E}\right\})$

$u \in \sup C(E) \wedge (\overline{u}\Sigma_u^* \cap L(G) \not\subseteq \overline{E})$.

By lemma 4.2,

$u \in \sup C(E) \Rightarrow \overline{u}\Sigma_u^* \cap L(G) \subseteq \overline{E}$. Therefore the above is also a contradiction, and the proof is complete. ♦

5 COMPUTER IMPLEMENTATION

In this section we develop an algorithm to compute supC based on the formula given in proposition 4.1. It is clear that any implementation of an algorithm for supC (as the one in [Rudi88]) will compute Ω^*. The implementation suggested below differs from the one in [Rudi88] in that it does the computation in one pass, without the need of an auxiliary algorithm for the trim component of the generator.

Given two generators $G = \langle \Sigma, Q, \delta, q_0, Q_m \rangle$ and $H = \langle \Sigma, X, \xi, x_0, X_m \rangle$, H is said to *refine* G if $L(H) \subseteq L(G)$ and for all $s, t \in L(G)$,

$\xi(s, x_0) = \xi(t, x_0) \Rightarrow \delta(s, x_0) = \delta(t, x_0)$.

If *H* refines *G* then it is easy to show that there exists a unique function $h\colon X \to Q$ satisfying

$$h[\xi(s, x_0)] = \delta(s, q_0) \qquad (5.1).$$

The following lemma will support our algorithmic implementation:

Lemma 5.1: Let $G = \langle \Sigma, Q, \delta, q_0, Q_m \rangle$ and $H = \langle \Sigma, X, \xi, x_0, X_m \rangle$ be trim generators such that

i. *G* represents a plant;

ii. $L_m(H) = E$ is a target language;

iii. *H* refines *G*.

Let $h\colon X \to Q$ be the unique map satisfying (5.1). Then $w \in \sup C(E)$ iff $w \in E$ and $(\forall u \in \overline{w}) x \in \xi(u, x_0) \Rightarrow \Sigma^*[h(x)] \cap \Sigma_u^* \subseteq \Sigma^*(x)$, where $\Sigma^*(q)$ is the set of all strings s for which $\delta(s, q)$ is defined.

Proof: From proposition 4.1 we have that $w \in \Omega^*(E)$ iff

$w \in E \wedge \overline{w}\Sigma_u^* \cap L(G) \subseteq \overline{E}$

$w \in E \wedge (\forall u \in \overline{w}) u\Sigma_u^* \cap L(G) \subseteq \overline{E}$

$w \in E \wedge (\forall u \in \overline{w})(\forall v \in \Sigma_u^*) uv \in L(G) \Rightarrow uv \in \overline{E}$

$w \in E \wedge (\forall u \in \overline{w})(\forall v \in \Sigma^*[\delta(u, q_0)] \cap \Sigma_u^*) uv \in \overline{E}$

$w \in E \wedge (\forall u \in \overline{w}) \Sigma^*[\delta(u, q_0)] \cap \Sigma_u^* \subseteq \Sigma^*[\xi(u, x_0)]$

$w \in E \wedge$

$\wedge (\forall u \in \overline{w}) x = \xi(u, x_0) \Rightarrow \Sigma^*[h(x)] \cap \Sigma_u^* \subseteq \Sigma^*[x]$.

♦

The condition $w \in E$ implies $\overline{w} \subseteq \overline{E}$ and so each state *x* in the statement of lemma 5.1 has to be coaccessible. The condition

$$\Sigma^*[h(x)] \cap \Sigma_u^* \subseteq \Sigma^*[x] \qquad (5.2)$$

means that for an arbitrary state *x* of *H*, every path of uncontrollable events starting from the corresponding state $h(x)$ in *G* must also be defined starting from *x* in *H.* In view of these facts it is possible to construct a generator for $\sup C(E)$ by removing those states from *H* that fail to satisfy condition (5.2) and also those states which become not coaccessible due to the former operation. (Initially, all states are coaccessible, since *H* is trim by hypothesis).

The algorithm outlined below uses a list of *bad states*, which is a dynamic list containing the states to be removed from *H* in order to construct a generator for $\sup C(E)$.

Algorithm 5.1:

Given generators *G* and *H* as in lemma 5.1, the following steps transform *H* into a generator for $\sup C(E)$:

1. create a dynamic list of states called *badlist* and initialize it with the states that fail to satisfy

   $$\Sigma(h(x)) \cap \Sigma_u \subseteq \Sigma(x);$$

2. for each state x_i in *badlist:*
   - add the states $x_j\colon (\exists \sigma_u \in \Sigma_u) x_i = \xi(\sigma_u, x_j)$ to *badlist;*
   - add the states $x_k\colon (\exists \sigma_c \in \Sigma_c) x_i = \xi(\sigma_c, x_k) \wedge x_k \notin X_m \wedge |\Sigma(x_k)| = 1$ to *badlist;*
   - remove x_i from *badlist* and from the generator, together with its incoming and outgoing transitions;
3. take the accessible component of the resulting generator. ♦

Step 1 initializes the list of states to be removed from *H* with those states in which an uncontrollable event that is physically possible is not defined. Step 2 both processes and increases this list. For each state x_i already in the list:

states x_j that have an outgoing event $\sigma_u \in \Sigma_u$ leading into x_i are added to the list, implementing condition (5.2);

states x_k that have an outgoing event $\sigma_c \in \Sigma_c$ leading into x_i are added to the list iff x_k is not a marker state and σ_c is the only outgoing event from x_k.

The reader may verify that the second part of step 2 preserves coaccessibility. Step 3 just removes any states left inaccessible by the former operations.

6 CONCLUSION

The formula and the algorithm presented are valid in the general case of non prefix-closed languages. The algorithm preserves coaccessibility at each step, thus avoiding the use of an auxiliary algorithm to trim the generators obtained as intermediate results.

REFERENCES

[BGK+90] Brandt, R. D.; Garg, V.; Kumar, R.; Lin, F.; Marcus, S. I.; Wonham, W. M.: "Formulas for Calculating Supremal Controllable and Normal Sublanguages". Systems & Control Letters, 15(2):111-117 (1990)

[KGM91] Kumar, R.; Garg, V.; Marcus, S. I.: "On Controllability and Normality of Discrete Event Dynamical Systems". *Systems & Control Letters*, 17:157-168 (1991)

[LVW88] Lin, F.; Vaz, A. F.; Wonham, W. M.: "Supervisor Specification and Synthesis for Discrete Event Systems". *Int. J. Control,* 48(1):321-332 (1988)

[Rudi88] Rudie, Karen G.: "Software for the Control of Discrete Event Systems: A Complexity Study". *M. A. Sc. Thesis,* Department of Electrical Engineering, University of Toronto, Toronto, Canada (1988)

[RW87] Ramadge, P. J.; Wonham, W. M.: "Supervisory Control of a Class of Discrete Event Processes". *SIAM J. Control and Optimization,* 25(1):206-230 (1987)

[WR87] Wonham, W. M.; Ramadge, P. J.: "On the Supremal Controllable Sublanguage of a Given Language". *SIAM J. Control and Optimization,* 25(3):637-659 (1987)

[Zill93] Ziller, Roberto M.: "A Abordagem Ramadge-Wonham no Controle de Sistemas a Eventos Discretos: Contribuições à Teoria". *Dissertação de Mestrado,* Departamento de Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis - SC, Brasil (1993)

[ZC94] Ziller, Roberto M.; Cury, José E. R.: "On the Supremal Lm-closed and the Supremal Lm-closed and L-controllable Sublanguages of a Given Language". *Lecture Notes in Control and Information Science 94,* Springer Verlag (to appear)

Note 1: This paper was published in the annals of the 10.º Congresso Brasileiro de Automática / 6.º Congresso Latino-Americano de Controle Automático (Rio de Janeiro, Brazil, September 19-23, 1994), vol.1, pp. 260-265 (1994).

Note 2: While discussing algorithm 5.1 with the Discrete Subgroup of the University of Toronto, it has been found that the resulting generator may not be trim if the generator specifying the desired behavior has one or more second order bad states contained in a non-marker state loop. In order to assure the result is correct, step 3 should check for states that are either not accessible or not coaccessible. If there are any such states, they should be added to the badlist and processing return to step 2.

Note 3: Line 4 of paragraph 2 in section 5 should read: $\xi(s, x_0) = \xi(t, x_0) \Rightarrow \delta(s, q_0) = \delta(t, q_0)$.