Discrete Event Dynamic Systems: Theory and Applications 3, (1993): 71–100

Recursive Computation of Limited Lookahead Supervisory Controls for Discrete Event Systems*

SHENG-LUEN CHUNG AND STÉPHANE LAFORTUNE
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122

FENG LIN
Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202

Received March 31, 1992; Revised November 23, 1992

Abstract. We continue the study of limited lookahead policies in supervisory control of discrete event systems undertaken in a previous paper. On-line control of discrete event systems using limited lookahead policies requires, after the execution of each event, the calculation of the supremal controllable sublanguage of a given language with respect to another larger language. These two languages are finite and represented by their tree generators, where one tree is a subtree of the other. These trees change dynamically from step to step, where one step is the execution of one event by the system. We show in this paper how to perform this calculation in a recursive manner, in the sense that the calculation for a new pair of trees can make use of the calculation for the preceding pair, thus substantially reducing the amount of computation that has to be done on-line. In order to make such a recursive procedure possible from step to step, we show how the calculation for a single step (i.e., for a given pair of trees) can itself be performed recursively by means of a backward dynamic programming algorithm on the vertices of the larger tree. These two nested recursive procedures are also extended to the limited lookahead version of the "supervisory control problem with tolerance."

Key Words: supervisory control, limited lookahead policies, dynamic programming

1. Introduction

In a recent paper [Chung et al. 1992a], we introduced a supervisory control scheme for discrete event systems based on "limited lookahead control policies." Instead of attempting to calculate off-line the complete control policy for the entire set of possible behaviors of the discrete event system under control, the strategy with limited lookahead policies (LLPs) is to calculate on-line the next control action on the basis of an N-step ahead projection of the behavior of the system (represented as an N-level tree). This procedure is repeated after the execution of each event. This approach is motivated by the fact that if the discrete event system is complex and has a large number of states, or if it is time-varying, then it may be difficult if not impossible to build the automaton models of the system and of the legal behavior that are necessary for the calculation of the complete control policy. In essence, the computational complexity is broken from one off-line problem into the repetitive solution of similar but smaller problems on-line. The precise formulation and the optimality properties of this "LLP scheme" can be found in Chung et al. [1992a].

*Research supported in part by the National Science Foundation under grants ECS-9057967 and ECS-9008947. The first two authors also acknowledge support from GE and DEC.

In this paper, we continue our study of the LLP scheme and focus our attention on the on-line calculation of the control actions. In the case of the standard supervisory control problem, namely that of maintaining the behavior of the system inside the legal language with minimally restrictive controls (cf. Ramadge and Wonham [1987] and Wonham and Ramadge [1987]), the LLP scheme requires, after the execution of each event, the calculation of the supremal controllable sublanguage of a given (not necessarily closed) language with respect to another larger closed language. These two languages are finite and represented by their tree generators, where one tree is a subtree of the other. These trees change dynamically from step to step, where one step is the execution of one event by the system. The main contribution of this paper is to show how to perform this calculation in a recursive manner, in the sense that the calculation for a new pair of trees can make use of the calculation for the preceding pair, thus substantially reducing the amount of computation that has to be done on-line.

In order to make such a recursive procedure possible from step to step, we show how the calculation for a single step (i.e., for a given pair of trees) can itself be performed recursively by means of a backward dynamic programming algorithm on the vertices of the larger tree. For this purpose, we first "reformulate" in Section 2 the problem of finding the supremal controllable sublanguage as an optimal control problem with a $0/\infty$ cost structure. The purpose of this reformulation is to associate the supremal controllable sublanguage with the least-restrictive optimal policies that solve the optimal control problem. We then show in Section 3 how standard dynamic programming techniques can be applied to recursively calculate least-restrictive optimal policies when the languages of interest are finite and represented by their tree generators. The special feature of this dynamic programming problem is that the cost function must account for the fact that the languages of interest need not be closed. We also prove in Section 3 several properties of the dynamic programming solution that constitute the basis of the recursive (step-to-step) procedure that we propose in Section 4 for the calculation of the control actions in the LLP scheme. These two nested recursive procedures are also extended in Section 4 to include the limited lookahead version of the "supervisory control problem with tolerance" of Lafortune and Lin [1991], where the calculation of the control action at each step involves additional operations.

2. Optimal Control Formulation of the Supremal Controllable Sublanguage

We first formulate the problem of finding the supremal controllable sublanguage of a given language as an optimal control problem with a $0/\infty$ cost structure. In this context, we will show that the closed-loop behavior supervised by the least-restrictive optimal policies that solve the optimal control problem corresponds to the supremal controllable sublanguage. Kumar and Garg [1991] also investigated the issue of formulating a class of supervisory control problems as optimal control problems; they considered constraints that are represented by closed languages and solved these problems using network flow techniques. In contrast, our formulation does not require the constraint to be represented by a closed language. Furthermore, we use dynamic programming to solve the associated optimal control problems, as described in Section 3. (The more general issue of optimally controlling a discrete event system under a cost structure not restricted to $0/\infty$ has been addressed in

several references, among them Passino and Antsaklis [1989], Brave and Heymann [1990], and Sengupta and Lafortune [1992].)

Given the closed language $L(P) \subseteq \Sigma^*$ generated by a discrete event process P with event set Σ, define a control policy g as

$$g\colon L(P) \to 2^{\Sigma} \tag{1}$$

such that $(\forall s \in L(P))g(s) \subseteq \Sigma_{L(P)}(s)$, where $\Sigma_{L(P)}(s) = \{\sigma \in \Sigma\colon s\sigma \in L(P)\}$ is the active set of $L(P)$ at s. Let $\mathcal{G}$ be the set of all control policies for $L(P)$. The resultant language when P is controlled by g is denoted as $L(P, g)$ and is defined as follows:

the empty trace $\epsilon \in L(P, g)$,

$(\forall s \in L(P, g))s\sigma \in L(P, g) \Leftrightarrow \sigma \in g(s)$.

Let $L_m(P) \subseteq L(P)$ be the language marked by P and consider the language $L \subseteq L_m(P)$ representing the "legal" behavior; i.e., L is the language whose supremal controllable sublanguage we wish to calculate. The supremal controllable operation, which is denoted by the superscript $\uparrow$, is with respect to $L(P)$ and to $\Sigma_u \subseteq \Sigma$, the set of uncontrollable events. As usual, we define

$$L_m(P, g) := L(P, g) \cap L_m(P).$$

Observe that we do *not* require that $\overline{L_m(P)} = L(P)$.

The control cost associated with policy g and with respect to L is

$$J(g) = \sum_{s \in L(P, g)} c(s, g(s)),$$

where

$$c(s, g(s)) = \begin{cases} 0 & \text{if } s \notin L(P) - \bar{L} \wedge \Sigma_u \cap \Sigma_{L(P)}(s) \subseteq g(s) \\ & \quad \wedge (s \in (\bar{L} - L) \Rightarrow g(s) \neq \emptyset), \\ \infty & \text{otherwise.} \end{cases}$$

We define

$$J^* := \inf_{g \in \mathcal{G}} J(g).$$

DEFINITION 2.1. (i) A control policy g is *optimal* if $J(g) = 0$.

(ii) A control policy g is *least-restrictive optimal* if

$$J(g) = 0 \quad \text{and} \quad (\forall g' \in \mathcal{G})J(g') = 0 \Rightarrow L(P, g') \subseteq L(P, g).$$

Therefore, all least-restrictive optimal policies generate the same language. We show in this section that $\overline{L^\uparrow}$ is the language generated by least-restrictive optimal policies (cf. Corollary 2.2). We also show that $L^\uparrow \neq \emptyset$ is a sufficient condition to guarantee the existence of a least-restrictive optimal policy (cf. Theorem 2.1).

Before introducing the necessary assumptions about L and $L_m(P)$, we first recall the definition of *livelock-free* languages from Lafortune and Lin [1991].

DEFINITION 2.2. A language $L \subseteq \Sigma^*$ is said to be *livelock-free* if

$$(\forall s \in \bar{L})(\exists n \in \mathbb{N})(\forall t \in \Sigma^*)|t| \geq n \wedge st \in \bar{L} \Rightarrow (\exists u \in \Sigma^*)(s \leq u \leq st \wedge u \in L),$$

where $\mathbb{N}$ is the set of natural numbers, $|t|$ is the length of t, and $s \leq u$ denotes that s is a prefix of u.

If a language L is regular, then it will be livelock-free if and only if each directed cycle in the directed graph representation of any finite-state generator of L touches at least one marked state.

ASSUMPTION 2.1. (i) $L = \bar{L} \cap L_m(P)$, and (ii) $L_m(P)$ is livelock-free.

The following result will be used in subsequent proofs.

LEMMA 2.1. Let $L(P, g) \subseteq \bar{L}$. Then $L(P, g) \cap (\bar{L} - L) = L(P, g) - L_m(P, g)$.

Proof.

$$\begin{aligned}
L(P, g) \cap (\bar{L} - L) &= L(P, g) \cap \bar{L} \cap L^c \\
&= L(P, g) \cap \bar{L} \cap (\bar{L} \cap L_m(P))^c \qquad [\text{by } L = \bar{L} \cap L_m(P)] \\
&= L(P, g) \cap \bar{L} \cap [(\bar{L})^c \cup L_m(P)^c] \\
&= L(P, g) \cap \bar{L} \cap L_m(P)^c \qquad [\text{by } \bar{L} \cap (\bar{L})^c = \emptyset] \\
&= L(P, g) \cap L_m(P)^c \qquad [\text{by hypothesis } L(P, g) \subseteq \bar{L}] \\
&= L(P, g) - L_m(P) = L(P, g) - (L(P, g) \cap L_m(P)) \\
&= L(P, g) - L_m(P, g).
\end{aligned}$$

Q.E.D.

The next lemma shows that a control policy g is optimal if and only if the associated closed-loop behavior (i) does not violate the legal constraint $L(P, g) \subseteq \bar{L}$, (ii) is controllable (i.e., g does not disable uncontrollable events), and (iii) is nonblocking.

LEMMA 2.2.

$$J(g) = 0 \Leftrightarrow \begin{cases} L(P, g) \subseteq \bar{L} \wedge \\ L(P, g)^{\uparrow} = L(P, g) \wedge \\ L(P, g) = \overline{L_m(P, g)}. \end{cases}$$

Proof. From the definition of $J(g)$,

$$J(g) = 0 \Leftrightarrow (\forall s \in L(P, g))c(s, g(s)) = 0$$

$$\Leftrightarrow (\forall s \in L(P, g)) \begin{cases} s \notin L(P) - \bar{L} \wedge \\ \Sigma_u \cap \Sigma_{L(P)}(s) \subseteq g(s) \wedge \\ (s \in (\bar{L} - L) \Rightarrow g(s) \neq \emptyset). \end{cases}$$

We now prove each direction of the lemma.

($\Rightarrow$)

$$(\forall s \in L(P, g))s \notin L(P) - \bar{L} \wedge \Sigma_u \cap \Sigma_{L(P)} \subseteq g(s) \wedge (s \in (\bar{L} - L) \Rightarrow g(s) \neq \emptyset) \quad (2)$$

implies

(i) $L(P, g) \subseteq \bar{L}$ (immediately),
(ii) $L(P, g) = L(P, g)^{\uparrow}$ (by definition of controllability),
(iii) $L(P, g) = \overline{L_m(P, g)}$, as shown below.

1. $\overline{L_m(P, g)} \subseteq L(P, g)$ by definition.
2. $L(P, g) \subseteq \overline{L_m(P, g)}$ by a proof by contradiction.
   Otherwise, suppose $(\exists s \in \Sigma^*)s \in L(P, g)$ and $s \notin \overline{L_m(P, g)}$. We now show by induction on n that

$$(\forall n \in \mathbb{N})(\exists t \in \Sigma^*)(|t| = n \wedge \{s\}\overline{\{t\}} \subseteq L(P, g) - \overline{L_m(P, g)}). \quad (3)$$

Induction base. Take $n = 0$. Because $s \in L(P, g) \wedge s \notin \overline{L_m(P, g)} \Rightarrow s \in L(P, g) - \overline{L_m(P, g)}$. Thus we have that

$$(\exists t \in \Sigma^*)(|t| = 0 \wedge \{s\}\overline{\{t\}} \subseteq L(P, g) - \overline{L_m(P, g)}).$$

Induction hypothesis. Assume that for $n = k$, we have

$$(\exists t \in \Sigma^*)(|t| = k \wedge \{s\}\overline{\{t\}} \subseteq L(P, g) - \overline{L_m(P, g)}).$$

Induction step. We must show that for $n = k + 1$

$$(\exists t \in \Sigma^*)(|t| = k + 1 \wedge \{s\}\overline{\{t\}} \subseteq L(P, g) - \overline{L_m(P, g)}).$$

By the induction hypothesis,

$$(\exists t = \sigma_1\sigma_2 \ldots \sigma_k \in \Sigma^*)\{s\}\overline{\{t\}} \subseteq L(P, g) - \overline{L_m(P, g)}.$$

By (i), $L(P, g) \subseteq \bar{L} \subseteq \overline{L_m(P)}$, so $s' := s\sigma_1\sigma_2 \ldots \sigma_k \in \overline{L_m(P)}$. Also, $s' \in L(P, g) - \overline{L_m(P, g)} \Rightarrow s' \in L(P, g) - L_m(P, g)$. Then, by Lemma 2.1, $s' \in L(P, g) \cap (\bar{L} - L)$. Since $s' \in \bar{L} - L$, $g(s') \neq \emptyset$ by Equation (2). As such, $(\exists\sigma_{k+1} \in \Sigma)s'\sigma_{k+1} \in L(P, g)$. On the other hand, $s' \notin \overline{L_m(P, g)} \Rightarrow s'\sigma_{k+1} \notin \overline{L_m(P, g)}$. Therefore, $s'\sigma_{k+1} = s\sigma_1\sigma_2 \ldots \sigma_k\sigma_{k+1} \in L(P, g) - \overline{L_m(P, g)}$, which, together with the induction hypothesis, implies that

$$(\exists t \in \Sigma^*)(|t| = k + 1 \wedge \{s\}\overline{\{t\}} \subseteq L(P, g) - \overline{L_m(P, g)}).$$

This completes the proof of Equation (3). Meanwhile, $L(P, g) - \overline{L_m(P, g)} \subseteq L(P, g) - L_m(P, g) = L(P, g) - L_m(P)$. Therefore, using (3), we have

$$(\forall n \in \mathbb{N})(\exists t \in \Sigma^*)(|t| = n \wedge \{s\}\overline{\{t\}} \subseteq L(P, g) - L_m(P)).$$

Thus, by (i),

$$(\exists s \in \overline{L_m(P)})(\forall n \in \mathbb{N})(\exists t \in \Sigma^*)(|t| \geq n \wedge st \in \overline{L_m(P)} \wedge (\forall u \in \Sigma^*)s \leq u \leq st \Rightarrow u \notin L_m(P)).$$

This contradicts the assumption that $L_m(P)$ is livelock-free and completes the proof of (iii).

($\Leftarrow$) Clearly,

$$L(P, g) \subseteq \bar{L} \Rightarrow (\forall s \in L(P, g))s \notin L(P) - \bar{L}$$

and

$$L(P, g) = L(P, g)^{\uparrow} \Rightarrow (\forall s \in L(P, g))\Sigma_u \cap \Sigma_{L(P)}(s) \subseteq g(s).$$

Also

$$L(P, g) \subseteq \bar{L} \wedge L(P, g) = \overline{L_m(P, g)} \Rightarrow [(\forall s \in L(P, g))s \in \bar{L} - L \Rightarrow g(s) \neq \emptyset].$$

Otherwise,

$$\neg[(\forall s \in L(P, g))s \in \bar{L} - L \Rightarrow g(s) \neq \emptyset]$$

$$\Rightarrow (\exists s \in L(P, g) \cap (\bar{L} - L))g(s) = \emptyset$$

$$\Rightarrow (\exists s \in L(P, g) - L_m(P, g))g(s) = \emptyset \quad \text{(by Lemma 2.1)}$$

$$\Rightarrow \{s\}\Sigma^* \cap L(P, g) = \{s\} \wedge s \notin L_m(P, g)$$

$$\Rightarrow \{s\}\Sigma^* \cap L_m(P, g) = \{s\}\Sigma^* \cap L(P, g) \cap L_m(P) = \emptyset$$

$$\Rightarrow s \notin \overline{L_m(P, g)}.$$

But $s \in L(P, g)$ implies $\overline{L_m(P, g)} \subset L(P, g)$, which leads to a contradiction. Q.E.D.

The next example shows why the livelock-free condition in Assumption 2.1(ii) is necessary for Lemma 2.2.

EXAMPLE 2.1. Let

$$L_m(P) = \epsilon + a(bc)^*d + a(bc)^*de,$$

$$L(P) = \overline{L_m(P)},$$

$$L = \epsilon + a(bc)^*d,$$

$$\Sigma = \{a, b, c, d, e\},$$

$$\Sigma_u = \{e\}.$$

Consider the following control policy g:

$$g(s) = \begin{cases} \{a\} & \text{if } s = \epsilon, \\ \{b\} & \text{if } s = a(bc)^*, \\ \{c\} & \text{if } s = a(bc)^*b, \\ \{e\} & \text{if } s = a(bc)^*d, \\ \emptyset & \text{if } s = a(bc)^*de. \end{cases}$$

Then $L(P, g) = a(bc)^* \neq \overline{L_m(P, g)} = \{\epsilon\}$. However, $J(g) = \Sigma_{s\in L(P,g)} c(s, g(s)) = 0$.

The following corollary of Lemma 2.2 will be used in the proof of Theorem 2.1.

COROLLARY 2.1. Let $\emptyset \subset M = M^\uparrow \subseteq L$ and $\overline{\bar{M} \cap L_m(P)} = \bar{M}$. Then there exists a g such that $J(g) = 0$ and $L(P, g) = \bar{M}$.

Proof. There exists a g such that $L(P, g) = \bar{M}$ since $\emptyset \subset \bar{M} \subseteq L(P)$. But

$$L(P, g) = \bar{M} \subseteq \bar{L},$$

$$L(P, g) = \bar{M} = (\bar{M})^\uparrow = L(P, g)^\uparrow,$$

$$\overline{L(P, g) \cap L_m(P)} = \overline{\bar{M} \cap L_m(P)} = \bar{M} = L(P, g).$$

Therefore, by Lemma 2.2, $J(g) = 0$. Q.E.D.

Given a policy $g \in \mathcal{G}$, define its restricted version g^R, also in $\mathcal{G}$, by

$$g^R(s) := \begin{cases} g(s) & \text{if } s \in L(P, g), \\ \emptyset & \text{otherwise.} \end{cases}$$

Also define the disjunction of two policies $g_1, g_2 \in \mathcal{G}$ by

$$g_{1\vee 2}(s) := g_1^R(s) \cup g_2^R(s).$$

We present two results about the disjunction of two policies.

LEMMA 2.3. (i) $L(P, g^R) = L(P, g)$.
(ii) $L(P, g_{1\vee 2}) = L(P, g_1) \cup L(P, g_2)$.

Proof. (i) ($\supseteq$) If otherwise, then

$$s\sigma \in L(P, g) \wedge s \in L(P, g^R) \wedge s\sigma \notin L(P, g^R) \Rightarrow s \in L(P, g) \wedge g(s) \neq g^R(s),$$

which leads to a contradiction.
($\subseteq$) Following the definition, $(\forall s \in L(P))g^R(s) \subseteq g(s) \Rightarrow L(P, g^R) \subseteq L(P, g)$.
(ii) ($\supseteq$) Follows from (i) and the definition of $g_{1\vee 2}$.
($\subseteq$) By contradiction,

$$s\sigma \in L(P, g_{1\vee 2})) \wedge s \in L(P, g_1) \cup L(P, g_2) \wedge s\sigma \notin L(P, g_1) \cup L(P, g_2)$$

$$\Rightarrow \sigma \notin g_1^R(s) \wedge \sigma \notin g_2^R(s) \quad \text{(because of the use of } g^R\text{)}$$

$$\Rightarrow \sigma \notin g_{1\vee 2}(s)$$

which contradicts $s\sigma \in L(P, g_{1\vee 2})$. Q.E.D.

LEMMA 2.4. If $J(g_1) = 0$ and $J(g_2) = 0$, then $J(g_{1\vee 2}) = 0$.

Proof. By contradiction, let $J(g_{1\vee 2}) = \infty$. Then

$$(\exists s_k \in L(P, g_{1\vee 2}))c(s_k, g_{1\vee 2}(s_k)) = \infty.$$

Assume that k denotes the length of s and take the smallest k satisfying the above condition. Recall that

$$L(P, g_{1\vee 2}) = L(P, g_1) \cup L(P, g_2).$$

Without loss of generality, let $s_k \in L(P, g_1)$, which implies that $g_1^R(s_k) = g_1(s_k)$, and consider the three cases for which $c(s_k, g_{1\vee 2}(s_k)) = \infty$.

1. If $s_k \notin \bar{L}$, then $J(g_1) = \infty$.
2. If $\Sigma_u \cap \Sigma_{L(P)}(s_k) \not\subseteq g_1^R(s_k) \cup g_2^R(s_k)$, then $\Sigma_u \cap \Sigma_{L(P)}(s_k) \not\subseteq g_1^R(s_k) = g_1(s_k)$ and we have that $J(g_1) = \infty$.
3. If blocking occurs at s_k, i.e., $s_k \in \bar{L} - L \wedge (g_1^R(s_k) \cup g_2^R(s_k)) = \emptyset$, then we have $s_k \in \bar{L} - L \wedge g_1(s_k) = \emptyset$. Hence $J(g_1) = \infty$.

In all cases, we have the desired contradiction. Q.E.D.

Define

$$g^* := \begin{cases} \bigcup \{g^R : J(g) = 0\} & \text{if this set is not empty,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It follows that if g^* is defined, then g^* is a least-restrictive optimal policy, since $J(g^*) = 0$ by Lemma 2.4 and since

$$(\forall g' \in \mathcal{G}) J(g') = 0 \Rightarrow g'^R \subseteq g^*$$

$$\Rightarrow L(P, g') = L(P, g'^R) \subseteq L(P, g^*).$$

We are now ready to state the main result of this section.

THEOREM 2.1. (i) If g^* is defined, then $L(P, g^*) = \overline{L^\uparrow}$.
(ii) If $L^\uparrow \neq \emptyset$, then g^* is defined.

Proof. (i) Assume that g^* is defined. We prove that $L(P, g^*) = \overline{L^\uparrow}$.
($\subseteq$) By Lemma 2.4, $J(g^*) = 0$. By Lemma 2.2,

$$L(P, g^*) \subseteq \bar{L} \wedge L(P, g^*) = L(P, g^*)^\uparrow \wedge L(P, g^*) = \overline{L_m(P, g^*)}$$

$$\Rightarrow \begin{cases} L(P, g^*) \cap L_m(P) \subseteq \bar{L} \cap L_m(P) = L \wedge \\ \qquad L(P, g^*) = \underline{L(P, g^*)^\uparrow} \wedge \\ \qquad L(P, g^*) = \overline{L(P, g^*) \cap L_m(P)}. \end{cases}$$

Note that

$$(L(P, g^*) \cap L_m(P))^\uparrow = L(P, g^*) \cap L_m(P).$$

This is because

$$\overline{[L(P, g^*) \cap L_m(P)]}\Sigma_u^* \cap L(P) = L(P, g^*)\Sigma_u^* \cap L(P)$$

$$\subseteq L(P, g^*) \quad \text{(by definition of controllability)}$$

$$= \overline{L(P, g^*) \cap L_m(P)}.$$

Therefore,

$$L(P, g^*) \cap L_m(P) \subseteq L^\uparrow \wedge L(P, g^*) = \overline{L(P, g^*) \cap L_m(P)}$$

$$\Rightarrow L(P, g^*) \subseteq \overline{L^\uparrow}.$$

($\supseteq$) Let g be the control policy synthesizing $\overline{L^\uparrow}$; that is, $L(P, g) = \overline{L^\uparrow}$. By Lemma 2.2 and Assumption 2.1(i), $J(g) = 0$. By definition of g^*, for all $s \in L(P, g)$, $g(s) \subseteq g^*(s)$. Therefore

$$L(P, g) = \overline{L^\uparrow} \subseteq L(P, g^*).$$

(ii) Because $\emptyset \subset L^\uparrow = (L^\uparrow)^\uparrow \subseteq L$ and $\overline{\overline{L^\uparrow} \cap L_m(P)} = \overline{L^\uparrow}$, by Corollary 2.1, there exists a g such that $J(g) = 0$. Therefore, g^* is defined. Q.E.D.

We conclude this section with the following result.

COROLLARY 2.2. Let g be a least-restrictive optimal policy in $\mathcal{G}$. Then $L(P, g) = \overline{L^\uparrow}$. Moreover, if $\overline{L_m(P)} = L(P)$, then $L_m(P, g) = L^\uparrow$.

Proof. The first statement follows from the definition of least-restrictive optimal policies. The second statement is true because when $\overline{L_m(P)} = L(P)$, Assumption 2.1(i) implies that $L^\uparrow = \overline{L^\uparrow} \cap L_m(P)$ by Proposition 6.1(2) in Wonham and Ramadge [1988]. Q.E.D.

3. Solution by Finite Horizon Dynamic Programming

3.1. Calculation of a Least-Restrictive Optimal Policy

In this section, we develop a finite horizon dynamic programming solution of the optimal control problem of the previous section for the case when $L(P)$ is finite. This solution will be applicable to the case of supervisory control with limited lookahead policies. Tsitsiklis [1989] also discussed dynamic programming solutions of supervisory control problems for the class of closed legal languages L. Our approach is different because it is geared to limited lookahead policies and it does not require L to be closed.

If $L(P)$ is a finite language, then we can represent it with a finite generator P whose directed graph representation is a tree, i.e.,

$$P = (X, \Sigma, x_0, \delta, X_m)$$

and the transition function δ satisfies the condition

$$\delta(s_1, x_0) = \delta(s_2, x_0) \Leftrightarrow s_1 = s_2.$$

In this case, we can rename each state by its (unique) associated trace, i.e.,

$$(\forall s \in L(P))\delta(s, x_0) = x \Rightarrow x := s.$$

We now introduce the following new assumption, which is stronger than Assumption 2.1.

ASSUMPTION 3.1. (i) $L(P)$ is finite and is represented by P, a finite tree generator.
(ii) $L_m(P) = L \subseteq L(P)$.

Using the above renaming of the state space, we define the two subsets

$$X_{\text{illegal}} = L(P) - \bar{L},$$

$$X_{\text{transient}} = \bar{L} - L.$$

Also, by Assumption 3.1(ii), $X_m = L$. Observe that

$$X = X_{\text{illegal}} \dot{\cup} X_{\text{transient}} \dot{\cup} X_m,$$

where $\dot{\cup}$ denotes disjoint union.

Using the notation of this section, the incremental cost function c of the preceding section is rewritten as

$$c(x, g(x)) = \begin{cases} 0 & \text{if } (g(x) \supseteq \Sigma_u \cap \Sigma_{L(P)}(x)) \wedge (x \in X_m \cup X_{\text{transient}}) \\ & \quad \wedge (x \in X_{\text{transient}} \Rightarrow g(x) \neq \emptyset) \\ \infty & \text{otherwise.} \end{cases} \tag{4}$$

Define the ith layer states as

$$X(i) := \{s \,|\, \delta(s, x_0)! \wedge |s| = i\},$$

where $\delta(s, x_0)!$ denotes that $\delta(s, x_0)$ is defined. Also, for a given control policy $g \in \mathcal{G}$, define recursively the functions

$$\forall x_N \in X(N) \quad V^g(x_N) := \begin{cases} 0 & \text{if } x \in L, \\ \infty & \text{otherwise,} \end{cases}$$

$$\forall x_k \in X(k),\ 0 \leq k \leq N-1 \quad V^g(x_k) := c(x_k, g(x_k)) + \sum_{\sigma \in g(x_k)} V^g(x_k\sigma),$$

where N is the length of the longest trace in $L(P)$. We also need to define the subtree generator P/x and its associated control policy g/x: $\Sigma^* \to 2^\Sigma$ as

$$P/x := \begin{cases} (X, \Sigma, x, \delta, X_m) & \text{if } x \in X, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$(g/x)(t) := \begin{cases} g(xt) & \text{if } xt \in L(P), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The next three results serve to establish, in the usual manner (see, e.g., Kumar and Varaiya [1986], Chap. 6), the main dynamic programming result (Theorem 3.1) at the end of this section.

LEMMA 3.1.

$$\forall x_k \in X(k),\ 0 \le k \le N,\quad V^g(x_k) = \sum_{t \in L(P/x_k,\, g/x_k)} c(x_k t,\ g(x_k t)).$$

Proof. We proceed by induction on k.

1. *Induction base.* Take $k = N$. Then

$$\sum_{t \in L(P/x_N,\, g/x_N)} c(x_N t,\ g(x_N t)) = c(x_N,\ g(x_N))$$

$$= \begin{cases} 0 & \text{if } x_N \in X_m \quad \text{[by definition of } cx,\ g(x)\text{]}, \\ \infty & \text{otherwise}, \end{cases}$$

$$= \begin{cases} 0 & \text{if } x_N \in L \quad \text{(by Assumption 3.1)}, \\ \infty & \text{otherwise}, \end{cases}$$

$$= V^g(x_N).$$

2. *Induction hypothesis.* Assume that for all k, $j \le k \le N$, we have

$$V^g(x_k) = \sum_{t \in L(P/x_k,\, g/x_k)} c(x_k t,\ g(x_k t)).$$

3. *Induction step.* We must show that

$$V^g(x_{j-1}) = \sum_{t \in L(P/x_{j-1},\, g/x_{j-1})} c(x_{j-1} t,\ g(x_{j-1} t)).$$

By definition,

$$V^g(x_{j-1}) = c(x_{j-1},\ g(x_{j-1})) + \sum_{\sigma \in g(x_{j-1})} V^g(x_{j-1}\sigma).$$

But $x_{j-1}\sigma \in X(j)$ and thus by the induction hypothesis,

$$V^g(x_{j-1}) = c(x_{j-1}, g(x_{j-1})) + \sum_{\sigma \in g(x_{j-1})} \left[\sum_{t \in L(P/x_{j-1}\sigma, g/x_{j-1}\sigma)} c(x_{j-1}\sigma t, g(x_{j-1}\sigma t)) \right]$$

$$= \sum_{t \in L(P/x_{j-1}, g/x_{j-1})} c(x_{j-1}t, g(x_{j-1}t)).$$

Q.E.D.

Observe that $V^g(x_0) = J(g)$.

LEMMA 3.2. Let $W : X \to \{0, \infty\}$ be such that

$$W(x_N) \leq c(x_N, \emptyset) \qquad \forall x_N \in X_N, \tag{5}$$

$$W(x_k) \leq c(x_k, \gamma) + \sum_{\sigma \in \gamma} W(x_k\sigma) \qquad \forall \gamma \subseteq \Sigma_{L(P)}(x_k), \quad \forall x_k \in X_k, \ 0 \leq k \leq N-1. \tag{6}$$

Then, $\forall g \in \mathcal{G}$ and $\forall x_k \in L(P, g)$, $W(x_k) \leq V^g(x_k)$. In particular, $W(x_0) \leq J^*$.

Proof. We proceed by induction on k.

1. *Induction base.* Take $k = N$. Then, for all $g \in \mathcal{G}$ and all $x_N \in X_N$,

$$V^g(x_N) = c(x_N, \emptyset) \geq W(x_N).$$

2. *Induction hypothesis.* Assume that $W(x_k) \leq V^g(x_k)$ $\forall g \in \mathcal{G}$, $\forall x_k \in \bigcup_{k=j}^{N} X_k$.
3. *Induction step.* We must show that $W(x_{j-1}) \leq V^g(x_{j-1})$ $\forall g \in \mathcal{G}$, $\forall x_{j-1} \in X_{j-1}$. But

$$W(x_{j-1}) \leq c(x_{j-1}, g(x_{j-1})) + \sum_{\sigma \in g(x_{j-1})} W(x_{j-1}\sigma)$$

$$\leq c(x_{j-1}, g(x_{j-1})) + \sum_{\sigma \in g(x_{j-1})} V^g(x_{j-1}\sigma) \qquad \text{(by the induction hypothesis)}$$

$$= V^g(x_{j-1}).$$

Q.E.D.

For all $x_N \in X(N)$ and all $x_{N-i} \in X(N-i)$, $1 \leq i \leq N$, define recursively the "value" or "(optimal) cost-to-go" function $V : X \to \{0, \infty\}$ by

$$V(x_N) = c(x_N, \emptyset), \tag{7}$$

$$g^*_{dp}(x_{N-i}) = \begin{cases} \Sigma_{L(P)}(x_{N-i}) & \text{if } (\forall \gamma \subseteq \Sigma_{L(P)}(x_{N-i}))c(x_{N-i}, \gamma) + \sum_{\sigma \in \gamma} V(x_{N-i}\sigma) = \infty, \\ \bigcup \{\gamma \subseteq \Sigma_{L(P)}(x_{N-i}) : c(x_{N-i}, \gamma) + \sum_{\sigma \in \gamma} V(x_{N-i}\sigma) = 0\} & \text{otherwise,} \end{cases} \tag{8}$$

$$V(x_{N-i}) = c(x_{N-i}, g^*_{dp}(x_{N-i})) + \sum_{\sigma \in g^*_{dp}(x_{N-i})} V(x_{N-i}\sigma). \tag{9}$$

LEMMA 3.3. The function V defined in (7)–(9) satisfies (5) and (6).

Proof. Take $i = 0$. Then since $\Sigma_{L(P)}(x_N) = \emptyset$ for all $x_N \in X_N$, $\gamma \subseteq \Sigma_{L(P)}(x_N)$ implies that $\gamma = \emptyset$. Thus

$$V(x_N) = c(x_N, \emptyset) = c(x_N, \gamma),$$

and (5) is satisfied.

Take $1 \leq i \leq N$. Then for all $x_{N-i} \in X(N - i)$, by (9),

$$V(x_{N-i}) = c(x_{N-i}, g^*_{dp}(x_{N-i})) + \sum_{\sigma \in g^*_{dp}(x_{N-i})} V(x_{N-i}\sigma).$$

1. If $V(x_{N-i}) = 0$, then

$$V(x_{N-i}) \leq c(x_{N-i}, \gamma) + \sum_{\sigma \in \gamma} V(x_{N-i}\sigma) \quad \forall \gamma \subseteq \Sigma_{L(P)}(x_{N-i}).$$

2. Otherwise, if $V(x_{N-i}) = \infty$, then by (8) and (9),

$$(\forall \gamma \subseteq \Sigma_{L(P)}(x_{N-i}))c(x_{N-i}, \gamma) + \sum_{\sigma \in \gamma} V(x_{N-i}, \gamma) = \infty$$

$$\Rightarrow V(x_{N-i}) \leq c(x_{N-i}, \gamma) + \sum_{\sigma \in \gamma} V(x_{N-i}\sigma) \quad \forall \gamma \subseteq \Sigma_{L(P)}(x_{N-i}).$$

Therefore (6) is satisfied. Q.E.D.

We now state the main dynamic programming result for calculating a least-restrictive optimal policy.

THEOREM 3.1. If g^*_{dp} satisfies (8) and if $V(x_0) = 0$ in (9), then (i) g^*_{dp} is a least-restrictive optimal policy and (ii) $L(P, g^*_{dp}) = \overline{L^\uparrow}$ and $L_m(P, g^*_{dp}) = L^\uparrow$.

Proof. (i) We first need to show that g^*_{dp} is optimal; i.e.,

$$J(g^*_{dp}) = \sum_{t \in L(P, g^*_{dp})} c(t, g^*_{dp}(t)) = 0.$$

By (7)–(9),

$$V(x) = V^{g^*_{dp}}(x) \Rightarrow V(x_0) = V^{g^*_{dp}}(x_0) = J(g^*_{dp}) \quad \text{(by Lemma 3.1)}$$

$$\Rightarrow J(g^*_{dp}) = 0 \quad [\text{since } V(x_0) = 0].$$

Therefore, g^*_{dp} is optimal.

If $J(g) = 0$, then we claim that $g^R(x) \subseteq g_{dp}^{*R}(x)\ \forall x \in L(P)$. Otherwise, let $x \in L(P, g_{dp}^*) \cap L(P, g)$ be the shortest trace such that

$$g^R(x) \not\subseteq g_{dp}^{*R}(x).$$

Without loss of generality, let $g(x) = g_{dp}^*(x) \cup \{\sigma\}$. Then

$$J(g) = 0 \Rightarrow V^g(x) = 0 \Rightarrow c(x, g(x)) = 0.$$

By Lemma 3.3, $V(x)$ defined by (7) and (9) satisfies inequalities (5) and (6) in the statement of Lemma 3.2. Therefore, since $x\sigma \in L(P, g)$, $V(x\sigma) \leq V^g(x\sigma)$ by Lemma 3.2. Thus,

$$\begin{aligned} c(x, g(x)) + \sum_{\sigma' \in g(x)} V(x\sigma') &= c(x, g(x)) + \sum_{\sigma' \in g_{dp}^*(x)} V(x\sigma') + V(x\sigma) \\ &\leq 0 + 0 + V^g(x\sigma) \qquad \text{(from the optimality of } g_{dp}^*) \\ &= 0 + 0 + 0 = 0 \qquad [\text{because } J(g) = 0 \Rightarrow V^g(x\sigma) = 0]. \end{aligned}$$

But this implies that $g(x) \subseteq g_{dp}^*(x)$ by (8), which leads to a contradiction. Thus the above claim is true and it follows that $L(P, g) \subseteq L(P, g_{dp}^*)$; i.e., g_{dp}^* is a least-restrictive optimal policy.

(ii) The fact that $L(P, g_{dp}^*) = \overline{L^\uparrow}$ follows from Corollary 2.2. Moreover, even if $\overline{L_m(P)} \neq L(P)$, $L_m(P, g_{dp}^*) = L^\uparrow$ because it is easily verified that $L^\uparrow = \overline{L^\uparrow} \cap L$. Q.E.D.

The following example illustrates the recursive algorithm (7)–(9).

EXAMPLE 3.1. Let

$$L(P) = \overline{\epsilon + abc(\epsilon + d + e)},$$

$$L = \epsilon + abc(\epsilon + d),$$

$$\Sigma = \{a, b, c, d, e\},$$

$$\Sigma_u = \{b, e\}.$$

P is depicted in Figure 1. By (7)–(9), we have

$$V(5) = 0, \quad g_{dp}^*(4) = \{d, e\}, \quad g_{dp}^*(3) = \{c\}, \quad g_{dp}^*(2) = \{b\}, \quad g_{dp}^*(1) = \emptyset,$$

$$V(6) = \infty, \quad V(4) = \infty, \quad V(3) = \infty, \quad V(2) = \infty, \quad V(1) = 0.$$

Consequently, $L(P, g_{dp}^*) = \{\epsilon\} = \overline{L^\uparrow}$.

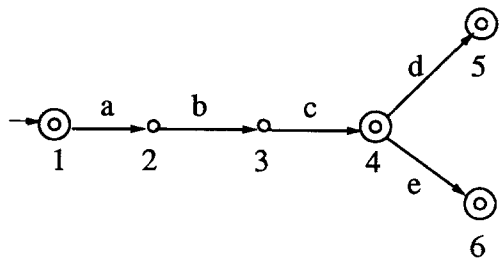

Figure 1. Example 3.1.

3.2. Properties of the Solution

In this section, we prove special properties of the dynamic programming recursive algorithm (7)–(9) developed above for the solution of the supremal controllable sublanguage problem. These properties will be used in Section 4 in the context of the recursive computation of limited lookahead supervisory controls.

The first result concerns Equation (9) and demonstrates that the optimal cost-to-go from a given state, given by the cost-to-go function V, can be determined solely from the optimal costs-to-go from the children of that state, without explicit reference to the associated control action at that state, i.e., to Equation (8).

THEOREM 3.2.

$$V(x) = 0 \Leftrightarrow \begin{cases} x \in X_m \cup X_{\text{transient}} \wedge \\ (\nexists \sigma_u \in \Sigma_u \cap \Sigma_{L(P)}(x))V(x\sigma_u) = \infty \wedge \\ (x \in X_{\text{transient}} \Rightarrow (\exists \sigma \in \Sigma_{L(P)}(x))V(x\sigma) = 0). \end{cases}$$

Proof.

($\Rightarrow$)

$$V(x) = 0 \Rightarrow c(x, g^*_{dp}(x)) + \sum_{\sigma \in g^*_{dp}(x)} V(x\sigma) = 0$$

$$\Rightarrow c(x, g^*_{dp}(x)) = 0 \wedge \sum_{\sigma \in g^*_{dp}(x)} V(x\sigma) = 0$$

$$\Rightarrow \begin{cases} x \in X_m \cup X_{\text{transient}} \wedge g^*_{dp}(x) \supseteq \Sigma_u \cap \Sigma_{L(P)}(x) \wedge \\ (x \in X_{\text{transient}} \Rightarrow g^*_{dp}(x) \neq \emptyset) \wedge \\ \sum_{\sigma \in g^*_{dp}(x)} V(x\sigma) = 0, \end{cases}$$

$$\Rightarrow \begin{cases} x \in X_m \cup X_{\text{transient}} \wedge \\ \sum_{\sigma \in \Sigma_u \cap \Sigma_{L(P)}(x)} V(x\sigma) = 0 \wedge \\ (x \in X_{\text{transient}} \Rightarrow g^*_{dp}(x) \neq \emptyset \wedge \sum_{\sigma \in g^*_{dp}(x)} V(x\sigma) = 0), \end{cases}$$

$$\Rightarrow \begin{cases} x \in X_m \cup X_{\text{transient}} \wedge \\ (\exists \sigma_u \in \Sigma_u \cap \Sigma_{L(P)}(x)) V(x\sigma_u) = \infty \wedge \\ (x \in X_{\text{transient}} \Rightarrow (\exists \sigma \in \Sigma_{L(P)}(x)) V(x\sigma) = 0). \end{cases}$$

($\Leftarrow$) We prove by contradiction.

1. If $\neg [x \in X_m \cup X_{\text{transient}}]$, then $x \in X_{\text{illegal}} \Rightarrow c(x, g^*_{dp}(x)) = \infty \Rightarrow V(x) = \infty$.
2. If $\neg [(\exists \sigma_u \in \Sigma_u \cap \Sigma_{L(P)}(x)) V(x\sigma_u) = \infty]$, then $(\exists \sigma_u \in \Sigma_u \cap \Sigma_{L(P)}(x)) V(x\sigma_u) = \infty$.
   (a) If $\Sigma_{\sigma \in g^*_{dp}(x)} V(x\sigma) = \infty$, then $V(x) = \infty$.
   (b) If $\Sigma_{\sigma \in g^*_{dp}(x)} V(x\sigma) = 0$, then $(\exists \sigma_u \in \Sigma_u \cap \Sigma_{L(P)}(x)) \sigma_u \notin g^*_{dp}(x)$. Therefore, $c(x, g^*_{dp}(x)) = \infty$ and $V(x) = \infty$.
3. If $\neg [x \in X_{\text{transient}} \Rightarrow (\exists \sigma \in \Sigma_{L(P)}(x)) V(x\sigma) = 0]$, then

$$x \in X_{\text{transient}} \wedge (\forall \sigma \in \Sigma_{L(P)}(x)) V(x\sigma) = \infty.$$

   (a) If $\Sigma_{\sigma \in g^*_{dp}(x)} V(x\sigma) = \infty$, then $V(x) = \infty$.
   (b) If $\Sigma_{\sigma \in g^*_{dp}(x)} V(x\sigma) = 0$, then $g^*_{dp}(x) \cap \Sigma_{L(P)}(x) = \emptyset$. Therefore, $c(x, g^*_{dp}(x)) = \infty$ and $V(x) = \infty$. Q.E.D.

The second result uses Theorems 3.1 and 3.2 to relate g^*_{dp} with the nonemptiness of $L^\uparrow$.

THEOREM 3.3. The following three statements are equivalent.
(i) $L^\uparrow \neq \emptyset$.
(ii) $V(x_0) = 0$.
(iii) g^*_{dp} is a least-restrictive optimal policy.

Proof. (i) $\Rightarrow$ (ii)

$$L^\uparrow \neq \emptyset \Rightarrow g^* \text{ is defined} \quad [\text{by Theorem 2.1(ii)}]$$

$$\Rightarrow J(g^*) = \sum_{x \in L(P, g^*)} c(x, g^*(x)) = 0 \quad (\text{by definition of } g^*)$$

$$\Rightarrow J(g^*) = \sum_{x \in \overline{L^\uparrow}} c(x, g^*(x)) = 0 \quad [\text{by Theorem 2.1(i)}]$$

$$\Rightarrow V^{g^*}(x_0) = 0 \quad (\text{by Lemma 3.1}).$$

But by Lemmas 3.3 and 3.2

$$V(x_0) \leq V^{g^*}(x_0) \Rightarrow V(x_0) = 0.$$

(ii) $\Rightarrow$ (iii) Follows by Theorem 3.1(i).

(iii) $\Rightarrow$ (i) By Corollary 2.2, (iii) implies that $L(P, g^*_{dp}) = \overline{L^\uparrow}$. But, by definition, $L(P, g^*_{dp}) \supseteq \{\epsilon\}$. Therefore, $L^\uparrow \neq \emptyset$. This completes the proof. Q.E.D.

This result complements Theorem 3.1 in that as long as $L^\uparrow \neq \emptyset$, we do not have to distinguish between g^* and g^*_{dp}. Therefore, we will use g^* to denote g^*_{dp} hereafter.

The third result shows how to reconstruct the least-restrictive optimal policy g^* from the cost-to-go function V.

THEOREM 3.4. If $V(x) = 0$, then $g^*(x) = \{\sigma \in \Sigma_{L(P)}(x)\colon V(x\sigma) = 0\}$.

Proof.

($\subseteq$)

$$V(x) = 0 \Rightarrow c(x, g^*(x)) + \sum_{\sigma \in g^*(x)} V(x\sigma) = 0 \quad \text{[by (9)]}$$

$$\Rightarrow \sum_{\sigma \in g^*(x)} V(x\sigma) = 0$$

$$\Rightarrow g^*(x) \subseteq \{\sigma \in \Sigma_{L(P)}(x) | V(x\sigma) = 0\}.$$

($\supseteq$) Prove by contradiction. Let $z \notin g^*(x)$, but $z \in \{\sigma \in \Sigma_{L(P)}(x)\colon V(x\sigma) = 0\}$. Let $g(x) = g^*(x) \cup \{z\}$. First,

$$V(x) = 0 \Rightarrow c(x, g^*(x)) = 0$$

$$\Rightarrow (g^*(x) \supseteq \Sigma_u \cap \Sigma_{L(P)}(x)) \wedge (x \in X_m \cup X_{\text{transient}}$$

$$\wedge (x \in X_{\text{transient}} \Rightarrow g^*(x) \neq \emptyset) \quad \text{[by definition of } c(x, g(x))]$$

$$\Rightarrow c(x, g(x)) = 0 \quad \text{(because } g(x) \supseteq g^*(x)).$$

Then

$$c(x, g(x)) + \sum_{\sigma' \in g(x)} V(x\sigma') = c(x, g(x)) + \sum_{\sigma \in g^*(x)} V(x\sigma) + V(xz)$$

$$= 0 + 0 + 0 = 0.$$

But, by (8), $g(x) \subseteq g^*(x)$, which leads to a contradiction. Q.E.D.

Let $V(X(k)) = 0$ denote that for all $x \in X(k)$, $V(x) = 0$. The next result shows that it may not be necessary to perform the dynamic programming recursion (7)–(9) all the way back to the root x_0 of the tree.

THEOREM 3.5. If $V(X(k)) = 0$, then $V(X(j)) = 0$, $0 \leq j \leq k$.

Proof. It suffices to prove that

$$V(X(j)) = 0 \Rightarrow V(X(j-1)) = 0, \qquad 0 \leq j \leq k.$$

$$V(X(j)) = 0 \Rightarrow \forall x \in X(j),\ x \in X_m \cup X_{\text{transient}} \qquad \text{[by definition of } c(x, g(x))\text{]}$$

$$\Rightarrow \forall x \in X(j-1),\ x \in X_m \cup X_{\text{transient}} \qquad (s\sigma \in \bar{L} \Rightarrow s \in \bar{L}). \tag{10}$$

Also, because $V(X(j)) = 0$, both of the following are true for all $x \in X(j-1)$:

$$(\exists \sigma_u \in \Sigma_u \cap \Sigma_{L(P)}(x))V(x\sigma_u) = \infty, \tag{11}$$

$$x \in X_{\text{transient}} \Rightarrow (\exists \sigma \in \Sigma_{L(P)}(x))V(x\sigma) = 0. \tag{12}$$

Therefore, by Theorem 3.2 (10)–(12) together imply that $V(X(j-1)) = 0$. Q.E.D.

More conclusions regarding potential computational savings can be drawn for the tree structure under consideration. The theorem below shows the relation between the calculation in a tree and that in another tree with similar structure. We will comment on the implications of this result in Section 4.

THEOREM 3.6. Let $L(P_1)$, L_1, $L(P_2)$, and L_2 be four languages over the same alphabet Σ, with uncontrollable events set $\Sigma_u \subseteq \Sigma$. Assume that $L(P_1)$ and $L(P_2)$ are closed and that $L(P_i)$ and L_i satisfy Assumption 3.1 for $i = 1, 2$, respectively. Let function V^i be defined according to (7)–(9) with $L(P_i)$ and L_i for $i = 1, 2$, respectively. Take $x_1 \in X_1$ and $x_2 \in X_2$. If

1. $\Sigma_{L(P_1)}(x_1) = \Sigma_{L(P_2)}(x_2)$.
2. $(\forall \sigma \in \Sigma_{L(P_1)}(x_1))V^1(x_1\sigma) =$ (or $\geq$, $\leq$, respectively) $V^2(x_2\sigma)$.
3. x_1 and x_2 belong to the same category (marked, transient, or illegal) in their respective state spaces,

then

$$V^1(x_1) = \text{(or } \geq, \leq, \text{ respectively) } V^2(x_2).$$

Proof. Follows from (7)–(9) and the definition of c in (4). Q.E.D.

4. Recursive Computation of Controls in LLP Scheme

4.1. Control Architecture

Consider a discrete event system G, generator of the closed language $L(G)$ and the marked language $L_m(G)$, that is to be controlled by means of LLPs, according to the on-line scheme presented in Chung et al. [1992a] and depicted in Figure 2. In the following, we assume some familiarity on the part of the reader with the results in Chung et al. [1992a] (especially Section 2). For convenience, we recall some notation that we will be using.

Remark 4.1. The postlanguage of language $K \subseteq \Sigma^*$ after trace $s \in \Sigma^*$ is the language

$$K/s := \{t \in \Sigma^*: st \in K\}.$$

The truncation of $K \subseteq \Sigma^*$ to the positive integer N is the language

$$K|_N := \{t \in K: |t| \leq N\}.$$

Let $K \subseteq L(G)/s|_N$. Then $(K)^{\uparrow/s|_N}$ denotes the supremal controllable sublanguage of K with respect to $L(G)/s|_N$ and Σ_u. Similarly, $K^{\downarrow/s|_N}$ denotes the infimal closed controllable superlanguage of K with respect to $L(G)/s|_N$ and Σ_u.

The overall on-line operation of an LLP supervisor is divided into a start-up phase and successive updating phases. In all cases, in order to determine the control action $\gamma^N(s)$ at the current trace $s \in L(G)$, the supervisor has to first generate the N-step lookahead

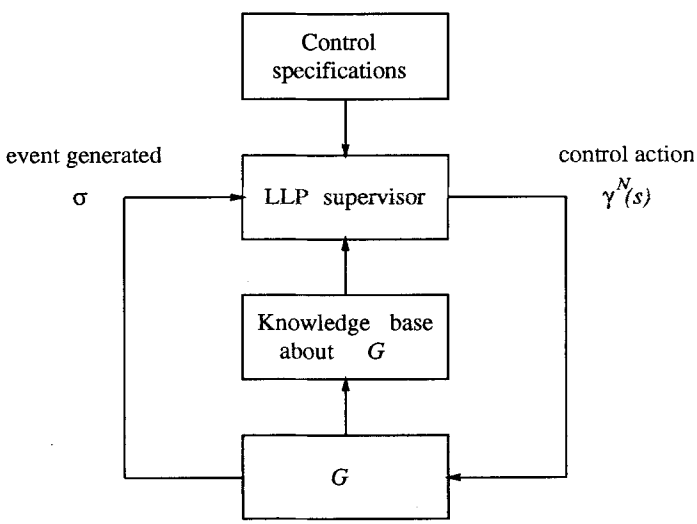

Figure 2. Limited lookahead supervisory control.

window, $L(G)/s\,|_N$, which is represented as an N-level tree, then classify all the traces in the window according to their legality properties, and finally perform the calculation of the control action $\gamma^N(s)$. The control action depends on the particular control problem under consideration and on the attitude adopted regarding the uncertainty on the behavior of the system beyond N steps. In the next two sections, we consider (i) the "standard" supervisory control problem (cf. Ramadge and Wonham [1987]) with either a conservative or an optimistic attitude, which is the problem studied in Chung et al. [1992a], and (ii) the supervisory control problem with tolerance (cf. Lafortune and Lin [1991]).

During the start-up phase, all the required calculations must be performed from scratch. In contrast, during the successive updating phases, many of the previous calculations can be reutilized from one step to the next; the only necessary additional calculations are those that reflect the new information gained with the one-step transition (event σ in Figure 2) of the system.

In this connection, in order to distinguish successive windows, we use a superscript $t = |s|$ (subscript in the case of g^*) to index all the relevant notation and the computational results of the current window rooted at trace s. In what follows we let $s = s'\alpha$. Then s' denotes the previous trace before the execution of $\alpha \in \gamma^N(s')$, where $|s| = t$ and $|s'| = t - 1$. In particular, concerning the dynamic programming algorithm presented in Section 3, the cost-to-go function V is now defined as

$$V^t\colon X^t \to \{0, \infty\},$$

where the states in X^t are labeled by the unique suffix of s that they correspond to. Therefore, $x \in X^t$ is the same state as $\alpha x \in X^{t-1}$. Also, in the following $x_0 = \epsilon$ where $X^t(0) = \{x_0\}$.

4.2. LLP Algorithm for the Standard Supervisory Control Problem

We now present an implementation of the LLP scheme of Chung et al. [1992a] for the standard supervisory control problem (cf. Ramadge and Wonham [1987]). In general, control policies of LLP supervisors are defined as

$$\gamma^N\colon L(G) \to 2^{\Sigma \cup \{\epsilon\}}. \tag{13}$$

The desired legal behavior for the closed-loop system $L(G, \gamma^N)$ is the language $\bar{K}$ where $K \subseteq L_m(G)$, $K \neq \emptyset$, and $K = \bar{K} \cap L_m(G)$. For the limited lookahead version of the standard supervisory control problem, the control action $\gamma^N(s)$ at trace $s \in L(G)$ is defined as

$$\gamma^N(s) := \overline{L^{\uparrow}}\,|_1 \cup \overline{\Sigma_u \cap \Sigma_{L(G)}(s)}, \tag{14}$$

where

$$L = L_m(P) = \begin{cases} \text{conservative:} & K/s|_N - (\bar{K}/s|_N - \bar{K}/s|_{N-1}) = K/s|_{N-1}, \\ \text{optimistic:} & K/s|_N \cup (\bar{K}/s|_N - \bar{K}/s|_{N-1}), \end{cases} \tag{15}$$

$$L(P) = L(G)/s|_N,$$

$\uparrow$ = supremal controllable sublanguage operation with respect to $L(P)$ and Σ_u.

This definition of $\gamma^N(s)$ is from Chung et al. [1992a], with the notation adapted to fit the problem addressed in Section 3, with uncontrolled behavior $L(P)$, desired behavior $L = L_m(P)$, and supremal controllable sublanguage operation in terms of $L(P)$ and Σ_u.

At any given window, we are only interested in the control action $\gamma^N(s)$ at the root s, and not in the complete control policy for the whole tree $L(P)$. According to Theorem 3.3, $L^{\uparrow} \neq \emptyset$ if and only if $V^t(x_0) = 0$. This in turn implies by Theorem 3.1 and by the definitions (1), (8), and (14) of g, g^*, and $\gamma^N(s)$, respectively, that

$$\gamma^N(s) = \begin{cases} g_t^*(x_0) \cup \{\epsilon\} & \text{if } V^t(x_0) = 0, \\ \overline{\Sigma_u \cap \Sigma_{L(G)}(s)} & \text{otherwise (i.e., } V^t(x_0) = \infty). \end{cases}$$

(Readers familiar with Chung et al. [1992a] will note that if $V^t(x_0) = \infty$, then we have a "run-time error" at trace s since $L^{\uparrow} = \emptyset$.)

The problem at hand is thus reduced to the calculation of $V^t(x_0)$ and $g_t^*(x_0)$ using the approach and results of Section 3. Theorems 3.2 and 3.4 show that in deriving g_t^*, instead of computing simultaneously for all the states in the window both the cost-to-go and the control action, we can compute independently the costs-to-go. Consequently, the efficiency of the algorithm relies on how these costs-to-go in the window are computed. In this regard, observe the fact that, as the system advances from one step to the next, the two succeeding pairs of L and $L(P)$ are similar in structure. Hence, we now focus on the issue of how to utilize the costs-to-go calculated from the previous step in calculating those at the current step. In particular, in addition to Theorem 3.5, Theorem 3.6 suggests another termination criterion. That is, during the dynamic programming recursion, if up to a certain level of the tree, for each state in this level, the newly calculated cost-to-go is identical to that calculated at the previous step, then so is the case for all states in the following backward levels.

Furthermore, the following result shows that given a fixed attitude, the cost-to-go associated with a trace is monotonic as the window moves forward.

THEOREM 4.1. Let $x \in X^t(j)$, $j \in \{0, \ldots, N-1\}$.

(1) If the attitude is conservative, then $V^t(x) \leq V^{t-1}(\alpha x)$.

(2) If the attitude is optimistic, then $V^t(x) \geq V^{t-1}(\alpha x)$.

Proof. We only prove (1). The other statement is proved similarly.

Let us prove by induction on j that for all $j \in \{0, \ldots, N-1\}$

$$(\forall x \in X^t(N-1-j))V^t(x) \leq V^{t-1}(\alpha x). \tag{16}$$

For $j = 0$, since $(\forall x \in X^t(N-1))V^{t-1}(\alpha x) = \infty$, $V^t(x) \leq V^{t-1}(\alpha x)$. Assume that (16) is true for $j \leq k$. Then for $x \in X^t(N-1-k-1)$, conditions 1, 2 and 3 of Theorem 3.6 hold. Therefore, by Theorem 3.6,

$$V^t(x) \leq V^{t-1}(\alpha x).$$

This completes the proof of (1). Q.E.D.

COROLLARY 4.1. Let $x \in X^t(j)$, $j \in \{0, \ldots, N-1\}$.

(1) If the attitude is conservative, then $V^{t-1}(\alpha x) = 0 \Rightarrow V^t(x) = 0$.

(2) If the attitude is optimistic, then $V^{t-1}(\alpha x) = \infty \Rightarrow V^t(x) = \infty$.

This result shows that if the conservative (or optimistic, respectively) attitude is adopted, once the cost-to-go from a trace becomes 0 (or ∞, respectively), the cost will remain at that value irrespective of the possible changes in the costs-to-go of the continuations of the trace.

In implementing the dynamic programming algorithm of Section 3.1, we can use the aforementioned results to minimize the number of new calculations required to determine the control action as the system advances one step in time. The following algorithm illustrates how this can be done.

LLP ALGORITHM 1.

(1) Initial condition: For all $x \in X^t(N)$, assign $V^t(x)$ according to (15):

1. If the conservative attitude is adopted, assign $V^t(x) = \infty$.
2. If the optimistic attitude is adopted, assign $V^t(x) = \infty$ if $x \in X_{\text{illegal}}$ and assign $V^t(x) = 0$ otherwise.

(2) Proceed recursively backwards over all $x \in X^t(j)$, initially from $j = N-1$.

1. At the jth level, compute $V^t(x)$ for all x in $X^t(j)$: First, by Corollary 4.1:
   (a) If the attitude is conservative and $V^{t-1}(\alpha x) = 0$, then $V^t(x) = 0$.
   (b) If the attitude is optimistic and $V^{t-1}(\alpha x) = \infty$, then $V^t(x) = \infty$.
   Otherwise, by Theorem 3.2:

$$V^t(x) = \begin{cases} 0 & \text{if } x \in X_m^t \cup X_{\text{transient}}^t \wedge \\ & (\exists \sigma_u \in \Sigma_u \cap \Sigma_{L(P)}(x))V^t(x\sigma_u) = \infty \wedge \\ & (x \in X_{\text{transient}}^t \Rightarrow (\exists \sigma \in \Sigma_{L(P)}(x))V^t(x\sigma) = 0), \\ \infty & \text{otherwise} \end{cases}$$

2. Terminate the recursion when $j = 0$ or when one of the following two termination criteria is satisfied.
   (a) TC1: By Theorem 3.5, if $V^t(X^t(j)) = 0$, then let $V^t(X^t(k)) = 0$ for $0 \leq k \leq j$.
   (b) TC2: By Theorem 3.6, if $V^t(x) = V^{t-1}(\alpha x)$ for all $x \in X^t(j)$, then $V^t(x) = V^{t-1}(\alpha x)$ for all $x \in \bigcup_{k=0}^{j-1} X^t(x)$.

(3) If $V^t(x_0) = 0$, return $g_t^*(x_0)$, the least-restrictive optimal control at the current trace s according to (cf. Theorem 3.4)

$$g_t^*(x_0) = \{\sigma \in \Sigma_{L(P)}(x_0): V^t(x_0\sigma) = 0\}.$$

Otherwise, return "run-time error."

EXAMPLE 4.1. Consider the system G shown in Figure 3. Assume that α_i is controllable, and β_i is uncontrollable, $i = 1, 2$. The legal behavior K is generated by the generator in Figure 4. Because the longest uncontrollable subtrace in $L(G)$ has length 2, according to Theorem 5.1 of Chung et al. [1992a] it is sufficient to construct γ^3_{optm} (i.e., a LLP supervisor with window size $N = 3$ and optimistic attitude) in order to get $L(G, \gamma^3_{\text{optm}}) = K^{\uparrow}$.

In the start-up phase, depicted in Figure 5a, termination criterion TC1 is satisfied at the third layer. The control action is $\gamma^3(\epsilon) = \{\epsilon, \alpha_1, \alpha_2\}$. Suppose that the system executes event α_1. In the next successive updating phase, depicted in Figure 5b, TC1 and TC2 are satisfied at the first layer. The control action is $\gamma^3(\alpha_1) = \{\epsilon, \beta_1, \alpha_2\}$.

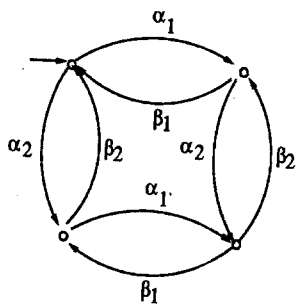

Figure 3. Example 4.1: G.

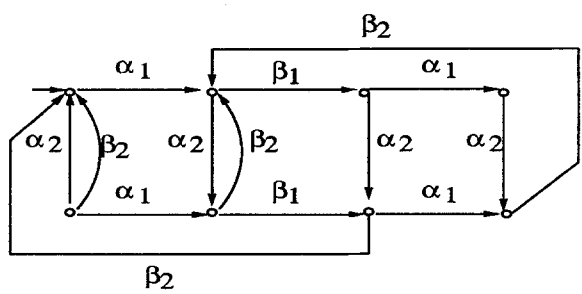

Figure 4. Example 4.1: Generator of $K = \bar{K}$.

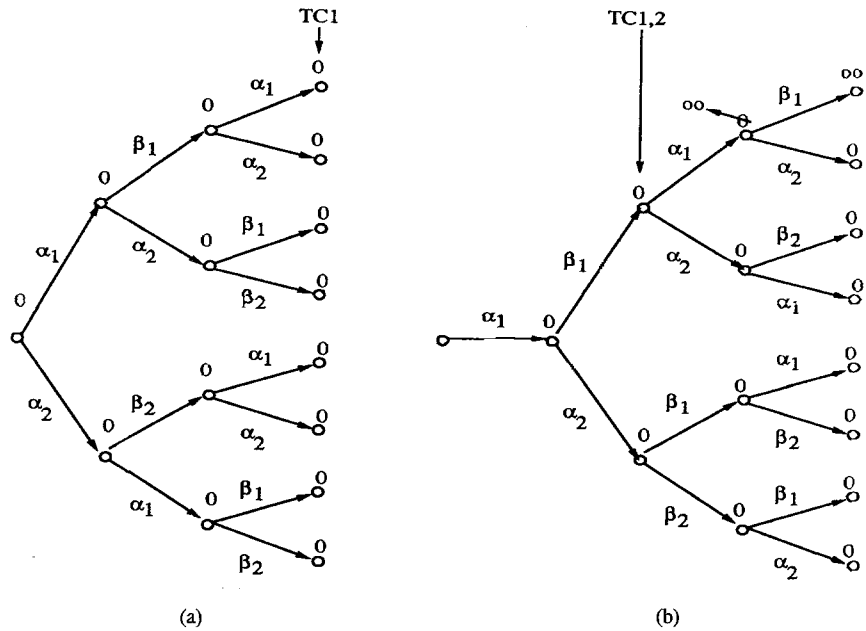

Figure 5. Example 4.1: LLP Algorithm 1: (a) start-up phase; (b) successive updating phase.

4.3. LLP Algorithm for Supervisory Control Problem with Tolerance

In applying LLPs to supervisory control problems with tolerance [Lafortune and Lin 1991], one possible policy (cf. Chung et al. [1992a]) is

$$\gamma^N(s) := \overline{f^N(s)}|_1 \cup \overline{\Sigma_u \cap \Sigma_{L(G)}(s)}, \tag{17}$$

where

$$f^N(s) := [B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}]^{\downarrow/s|_N}, \tag{18}$$

where $B_1 \subseteq L_m(G)$ and $B_2 \subseteq L(G)$ denote the desirable and tolerable behaviors, respectively. Such a policy guarantees that $L(G, \gamma^N)$ never goes beyond the tolerable behavior B_2. In the supervisory control problems considered in Lafortune and Lin [1991], the language B_1 and B_2 are assumed to satisfy certain assumptions. For the purpose of the analysis in this section, the only assumption required is that $B_2 = \overline{B_2}$.

We now construct a recursive scheme for the calculation of limited lookahead supervisory controls when the LLP control policy is defined by (17).

Let

$$B(s) := B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}. \tag{19}$$

Then because $L^{\downarrow} = \bar{L}\Sigma_u^* \cap L(G)$ (see Lafortune and Chen [1990], Lin and Wonham [1988]),

$$\gamma^N(s) = \overline{\overline{B(s)}\Sigma_u^* \cap L(G)/s|_N|_1} \cup \overline{\Sigma_u \cap \Sigma_{L(G)}(s)}$$

$$= \overline{B(s)}|_1 \cup \overline{\Sigma_u \cap \Sigma_{L(G)}(s)}. \tag{20}$$

Now the problem boils down to finding a recursive computation scheme for $\overline{B(s)}|_1$, that is

$$\overline{B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}|_1}.$$

$(B_2/s|_{N-1})^{\uparrow/s|_N}$ can be derived by using the LLP Algorithm 1 of Section 4.2, with the conservative attitude and with the constraint K being replaced by B_2. What remains to be tackled with are the set intersection operation and the one-step truncation of the prefix closure.

In order to avoid unnecessary computations and to reutilize previous computational results, we can take advantage of the following two facts. First, observe that

$$(\forall \sigma \in \Sigma_{L(G)}(s))\sigma \in \overline{B(s)}|_1 \Leftrightarrow (\exists t \in \Sigma^*)\sigma t \in B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}.$$

This fact implies that in order to derive $\overline{B(s)}|_1$ it is not necessary to enumerate all traces recognized by $B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}$. Rather, for every $\sigma \in \Sigma_{L(G)}(s)$, we can conclude that $\sigma \in \gamma^N(s)$ if there exists a trace $\sigma t \in B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}$. As such, we may as well use the depth-first search method in the subtree to find the first trace in $[B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}]/\sigma$. In this connection, the second fact, Theorem 4.2 relates searching results derived at successive steps as the system proceeds in time.

THEOREM 4.2. Given $B(s)$ defined in (19),

$$(\forall \sigma \in \Sigma)\ B(s)/\sigma \subseteq B(s\sigma).$$

Proof. By Theorem A.1(ii) in Chung et al. [1992a],

$$(\forall s \in \Sigma^*)K^{\uparrow}/s \subseteq (K/s)^{\uparrow/s}.$$

Therefore, by taking $s \equiv \sigma$, $K \equiv B_2/s|_{N-1}$, $\uparrow \equiv \uparrow/s|_N$, and $\sigma \in \Sigma$, we have

$$[(B_2/s|_{N-1})^{\uparrow/s|_N}]/\sigma \subseteq [(B_2/s|_{N-1})/\sigma]^{(\uparrow/s|_N)/\sigma} = (B_2/s\sigma|_{N-2})^{\uparrow/s\sigma|_{N-1}}$$

$$= (B_2/s\sigma|_{N-2})^{\uparrow/s\sigma|_N} \subseteq (B_2/s\sigma|_{N-1})^{\uparrow/s\sigma|_N}.$$

Hence, for all $\sigma \in \Sigma$,

$$B(s)/\sigma := [B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}]/\sigma$$

$$= (B_1/s|_{N-1})/\sigma \cap [(B_2/s|_{N-1})^{\uparrow/s|_N}]/\sigma$$

(by Lemma A.2 in Chung et al. [1992a])

$$\subseteq B_1/s\sigma|_{N-1} \cap (B_2/s\sigma|_{N-1})^{\uparrow/s\sigma|_N}$$

$$=: B(s\sigma).$$

This completes the proof. Q.E.D.

Theorem 4.2 implies that if, at s, a continuation path t, with length longer than one, has been verified to lead to a marked trace in $B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}$, then as the system advances one step transition by σ to $s\sigma$, the remaining continuation path t/σ should also lead to a marked trace in $B_1/s\sigma|_{N-1} \cap (B_2/s\sigma|_{N-1})^{\uparrow/s\sigma|_N}$, thus sparing the need for another search. In this regard, we need to introduce the notation $FB_1(s)$ for $s \in L(G)$ to remember whether or not the searched trace s is recognized in $B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N}$. Initially, $FB_1(s) = \textit{blank}$ for all $s \in L(G)$.

In addition to the above two facts, note that according to (17), uncontrollable events in the active set are eventually included in the control action $\gamma^N(s)$. As a result, in order to compute $\gamma^N(s)$, it is necessary to perform the depth-first search described above only for the controllable events in the active set.

In solving the control action defined by (20), the following algorithm illustrates how to incorporate the aforementioned ideas to minimize the required computation.

LLP ALGORITHM 2.

(1) Regarding $(B_2/s|_{N-1})^{\uparrow/s|_N}$:

1. Apply LLP Algorithm 1 in Section 4.2 with $K = B_2$, and the conservative attitude to obtain V^t: $X^t \to \{0, \infty\}$.
2. a. If $V^t(x_0) = \infty$ [i.e., $(B_2/s|_{N-1})^{\uparrow/s|_N} = \emptyset$ by Theorem 3.3], then

$$\gamma^N(s) = \overline{\Sigma_{L(G)}(s) \cap \Sigma_u}$$

and exit.

b. Otherwise, $(B_2/s|_{N-1})^{\uparrow/s|_N}$ is obtained by the reachable traces spanned by nodes with zero cost $V^t(\cdot)$ from the node x_0.

(2) For all $\sigma \in \Sigma_{L(G)}(s) \cap \Sigma_c$ satisfying $V^t(x_0\sigma) = 0$ and $FB_1(s\sigma) = \textit{blank}$:

1. Do depth-first search over $\overline{(B_1/s|_{N-1} \cap (B_2/s|_{N-1})^{\uparrow/s|_N})}/\sigma$. The goal is to find the first trace p such that $V^t(x_0\sigma p) = 0$, $(\forall t < p)V^t(x_0\sigma t) = 0$, and $s\sigma p \in B_1$.
2. If the goal is found, return the path-to-goal p. Then, for all nodes x along the path-to-goal set $FB_1(sx) = \textit{green}$. Otherwise, leave the searched subtrees as blank.

(3) Finally, in deciding the control action $\gamma^N(s)$:

1. If the depth-first search fails in step (2) and none of the next-step events is marked as *green*, then

$$\gamma^N(s) = \overline{\Sigma_{L(G)}(s) \cap \Sigma_u}.$$

2. Otherwise, the control action at the current trace s is obtained by

$$\gamma^N(s) = \overline{\{\sigma \in \Sigma_{L(G)}(s): FB_1(s\sigma) = green\}} \cup \overline{\Sigma_{L(G)}(s) \cap \Sigma_u}.$$

EXAMPLE 4.2. Consider the system G shown in Figure 6. Assume that α_i is controllable and β_i is uncontrollable, $i = 1, 2$. The desired behavior B_1 is generated by the generator in Figure 7. The tolerable behavior B_2 is the same as the legal behavior K in Example 4.1. (This is the same data as Example 2.3 in Lafortune and Lin [1991].)

To illustrate the LLP Algorithm 2, we choose the window size $N = 3$. In the start-up phase, depicted in Figure 8a, termination criterion TC1 is satisfied at the first layer. Both depth-first searches are successful at the first layer (we use double lines to indicate the "greening" effect). Accordingly, the control action is $\gamma^3(\epsilon) = \{\epsilon, \alpha_1, \alpha_2\}$. Suppose that the system executes event α_1. In the next successive updating phase, depicted in Figure 8b, termination criterion TC2 is satisfied at the second layer. No controllable event in the active set of α_1 is eligible for the depth-first search. The uncontrollable event β_1 is "greened" according to step (3). Then, the control action is $\gamma^3(\alpha_1) = \{\epsilon, \beta_1\}$.

5. Conclusion

We have presented in Section 4 recursive algorithms for the on-line computation of limited lookahead supervisory controls, in the context of the standard supervisory control problem (i.e., maintain the closed-loop behavior inside the legal language) and of the supervisory control problem with tolerance. These algorithms are based on an optimal control reformulation of the problem of finding the supremal controllable sublanguage of a given language (Section 2), a reformulation that then allows us to solve that problem using a recursive algorithm based on dynamic programming (Section 3). Overall, the algorithms for computing limited lookahead controls consist of two nested recursions. The (worst-case) complexity of these algorithms is linear in the cardinality of the state space X^t at each step. (For further discussion on computational complexity issues regarding the LLP scheme, the reader is referred to Chung et al. [1992a], Sections 6 and 7.)

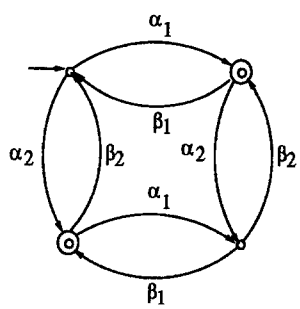

Figure 6. Example 4.2: G.

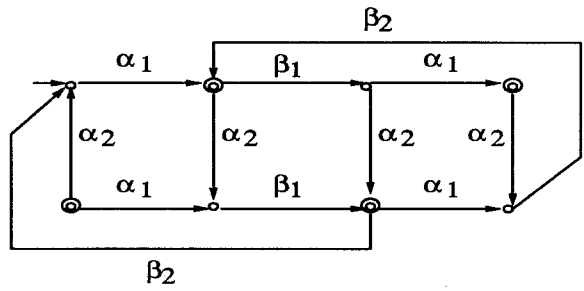

Figure 7. Example 4.2: Generator of B_1.

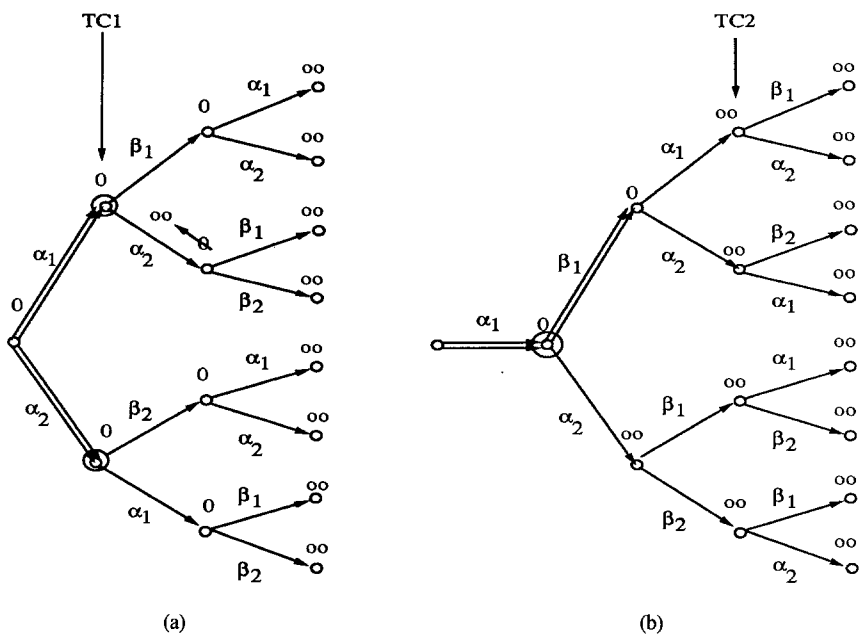

Figure 8. Example 4.2: LLP Algorithm 2: (a) start-up phase; (b) successive updating phase.

Although the main purpose of Sections 2 and 3 is to support the development of the algorithms in Section 4, these sections are also of independent interest since they complement or extend related work in the literature, in particular regarding the treatment of non-closed legal languages and the special properties that result from the assumption of finite languages (cf. Section 3.2).

The results in Section 4 show that limited lookahead control policies are amenable to a large amount of recursiveness as the lookahead window moves (or "rolls") from step to step. This recursiveness is not only present in the calculation of the control policy (14) which involves a supremal controllable sublanguage operation, but also in the calculation

of the control policy (17) which involves an infimal controllable superlanguage operation together with an intersection.

Further results relative to the efficient implementation of the LLP supervisory control scheme can be found in Chung et al. [1992b].

Acknowledgment

The authors wish to thank the reviewers for their comments and suggestions.

References

Brave, Y., and Heymann, M. 1990. On optimal attraction of discrete-event processes, Center for Intelligent Systems Report 9010, Technion, Haifa, Israel.

Chung, S.L., Lafortune, S., and Lin, F. 1992a. Limited lookahead policies in supervisory control of discrete event systems. *IEEE Trans. Automat. Control*, 37(12):1921–1935.

Chung, S.L., Lafortune, S., and Lin, F. 1992b. Supervisory control using variable lookahead policies, Technical Report CGR-92-9, College of Engineering Control Group Reports, University of Michigan.

Kumar, P.R., and Varaiya, P. 1986. *Stochastic Systems. Estimation, Identification, and Adaptive Control*. Prentice-Hall: Englewood Cliffs, NJ.

Kumar, R., and Garg, V. 1991. Optimal control of discrete event dynamical systems using network flow techniques, Preprint, Department of Electrical and Computer Engineering, University of Texas, Austin.

Lafortune, S., and Chen, E. 1990. The infimal closed controllable superlanguage and its application in supervisory control. *IEEE Trans. Automat. Control* 35(4): 398–405.

Lafortune, S., and Lin, F. 1991. On tolerable and desirable behaviors in supervisory control of discrete event systems. *Discrete Event Dynamic Systems* 1(1): 61–92.

Lin, F., and Wonham, W.M. 1988. On observability of discrete-event systems. *Information Sci.* 44: 173–198.

Passino, K.M., and Antsaklis, P.J. 1989. On the optimal control of discrete event systems. *Proc. 28th IEEE Conf. Decision and Control*, pp. 2713–2718, Tampa, FL.

Ramadge, P.J., and Wonham, W.M. 1987. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* 25(1): 206–230.

Sengupta, R., and Lafortune, S. 1992. A graph-theoretic optimal control problem for terminating discrete event processes. *Discrete Event Dynamic Systems* 2(2): 139–172.

Tsitsiklis, J.N. 1989. On the control of discrete-event dynamical systems. *Math. Control Signals Systems* 2: 95–107.

Wonham, W.M., and Ramadge, P.J. 1987. On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim.* 25(3): 637–659.

Wonham, W.M., and Ramadge, P.J. 1988. Modular supervisory control of discrete-event systems. *Math. Control Signals Systems* 1(1): 13–30.