Discrete Event Dynamic Systems, 4, 237–268 (1994)
© 1994 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Supervisory Control Using Variable Lookahead Policies

SHENG-LUEN CHUNG slchung@event.ee.ntit.edu.tw
Department of Electrical Engineering, National Taiwan Institute of Technology, Taipei, Taiwan 106

STÉPHANE LAFORTUNE stephane@eecs.umich.edu
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122

FENG LIN flin@ece.eng.wayne.edu
Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202

Received July 14, 1992; Revised March 4, 1994.

Abstract. This paper deals with the efficient on-line calculation of supervisory controls for discrete event systems (DES's) in the framework of limited lookahead control policies (or LLPs) that we introduced in previous papers. In the LLP scheme, the control action after a given trace of events has been executed is calculated on-line on the basis of an N-step ahead projection of the behavior of the DES. To compute these controls, one must calculate after the execution of each event the supremal controllable sublanguage of a finite language with respect to another finite larger language. In our previous work, we showed how the required supremal controllable sublanguage calculation can be performed by using a backward dynamic programming algorithm over the nodes of the tree representation of these two languages. In this paper, we pursue the same approach for the calculation of LLP controls, but instead we adopt a forward calculation procedure over the N-level tree of interest. This forward procedure improves upon previous work by avoiding the explicit consideration of all the nodes of the N-level tree, while still permitting tree-to-tree recursiveness as enabled events are executed by the system. The forward search ends whenever a control decision can be made unambiguously or whenever the boundary of the N-level tree is reached, whichever comes first. This motivates the name "Variable Lookahead Policy" (or VLP) for this implementation of the LLP supervisory control scheme. This paper presents a general VLP algorithm and studies the properties of several special cases of it. The paper also discusses the implementation of the VLP algorithms and presents computational results regarding the application of these algorithms to a "time-varying" DES.

Keywords: supervisory control, limited lookahead policies, dynamic programming

1. Introduction

This paper deals with the efficient on-line calculation of supervisory controls for discrete event systems (DES's) in the framework of limited lookahead control policies that we introduced in Chung, Lafortune, and Lin (1992). It builds upon the results presented in Chung, Lafortune, and Lin (1992, 1993a) concerning this framework.

Consider a DES that is being controlled by dynamically disabling/enabling events after the execution of each event by the controlled system. In supervisory control with limited lookahead policies (or LLPs), the control action after a given trace of events has been executed is calculated on-line on the basis of an N-step ahead projection of the behavior of

the DES under consideration; this procedure is repeated after the system executes any one of the enabled events. This is in contrast with the "conventional" supervisory control paradigm (cf. Lin and Wonham 1988, Ramadge and Wonham 1989) where the complete control policy is calculated off-line using automaton models of the DES and of the legal behavior. As discussed in Chung, Lafortune, and Lin (1992), LLPs allow to control certain classes of "time-varying" DES's and they also provide a means for dealing with the computational complexity of supervisor synthesis for DES's with large state spaces.

Chung, Lafortune, and Lin (1992) present a detailed study of the LLP scheme along with its optimality properties (in particular, in terms of N, the size of the "rolling window"). Chung, Lafortune, and Lin (1993a) are specifically concerned with the computation of the LLP controls. To compute these controls, one must calculate after the execution of each event by the system the supremal controllable sublanguage (cf. Wonham and Ramadge 1987) of a finite language with respect to another finite larger language. The latter language is the set of all traces of events that the system can generate in the next N steps, while the former language is the subset of these traces that are legal according to the specifications on event ordering. In Chung, Lafortune, and Lin (1993a), the required supremal controllable sublanguage calculation is approached as an optimal control problem with a $0/\infty$ cost structure. This optimal control problem is solved by using dynamic programming over a state space consisting of the nodes of a tree representation of the above-mentioned finite languages. Section 4.2 of Chung, Lafortune, and Lin (1993a) presents a backward dynamic programming algorithm over a given N-level tree for this calculation; this algorithm emphasizes the recursiveness between such calculations from N-level tree to N-level tree as the N-step window rolls with the execution of one (enabled) event by the DES, in addition to the inherent recursiveness from level to level of the tree at a given step.

In this paper, we pursue the dynamic programming approach of Chung, Lafortune, and Lin (1993a) for the calculation of LLP controls, but instead we adopt a forward calculation procedure over the state space (i.e., N-level tree) of interest. This forward procedure improves upon the algorithm in Chung, Lafortune, and Lin (1993a) by potentially avoiding the explicit consideration of all the nodes (or states) of the N-level tree, while still permitting step-to-step (i.e., tree-to-tree) recursiveness. The forward search ends whenever a control decision can he made unambiguously with respect to the future behavior or whenever the boundary of the N-level tree is reached, whichever comes first. This motivates the name "Variable Lookahead Policy" for this approach to LLP supervisory control.

We thus define a *Variable Lookahead Policy (or VLP)* to be a limited lookahead supervisory control policy whose on-line implementation at each step is by means of a forward search technique over the N-level tree of interest. Thus VLP's are a more efficient *implementation technique* of the LLP scheme. In particular, VLP's share all the properties of the LLP scheme that are presented in Chung, Lafortune, and Lin (1992). We will call a *VLP algorithm* the algorithm employed by a VLP for the required forward search over the N-level tree.

This paper presents a general VLP algorithm and studies several special cases of it in terms of maximum allowed value of N and "attitude" (conservative, optimistic, undecided, cf. Section 3) adopted when the Nth level of the tree is reached during a forward search. Our presentation is organized as follows. The problem at hand is precisely formulated in

Section 2. In Section 3, the general VLP algorithm is stated and its correctness proved. Section 4 presents some properties of this algorithm. The implementation of this algorithm is discussed in Section 5 while computational results regarding the application of four different variations of the general VLP algorithm to the train example of Chung, Lafortune, and Lin (1992) are reported in Section 6. Section 7 concludes the paper. (A preliminary and partial version of this work can be found in Chung, Lafortune, and Lin (1993b).)

2. Background and Problem Formulation

Consider a discrete event system G, generator of the closed language $L(G)$ and the marked language $L_m(G)$ over the set of events Σ. $\Sigma_u \subseteq \Sigma$ is the subset of uncontrollable events. As in Ramadge and Wonham (1987) and Chung, Lafortune, and Lin (1992), we assume that $\overline{L_m(G)} = L(G)$, where the overbar notation denotes prefix closure. G is to be controlled by means of limited lookahead policies (LLPs), according to the on-line scheme presented in Chung, Lafortune, and Lin (1992) and depicted in Fig. 1. In the following, we assume some familiarity on the part of the reader with the results in Chung, Lafortune, and Lin (1992) and Chung, Lafortune, and Lin (1993a). For convenience, we recall some notation from Chung, Lafortune, and Lin (1992) that we will be using.

The post-language 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 $N \in \mathbb{N}$ (the set of natural numbers) is the language

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

where $|t|$ is the length of trace t.

In general, control policies of LLP supervisors are defined as:

$$\gamma_{l.a}^N : L(G) \to 2^{\Sigma \cup \{\epsilon\}},$$

where l refers to LLP, a refers to the adopted attitude which can be either *cons* or *optm* representing "conservative" or "optimistic" (see below), and N is the size of the lookahead window. In order to determine the control action $\gamma_{l.a}^N(s)$ at the current trace $s \in L(G)$, the supervisor has to first generate the N-step lookahead 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 $\gamma_{l.a}^N(s)$. The resulting behavior when G is controlled by policy $\gamma_{l.a}^N$ is denoted by $L(G, \gamma_{l.a}^N)$ and is defined as follows:

i) $\epsilon \in L(G, \gamma_{l.a}^N)$, and

ii) $(\forall s \in L(G, \gamma_{l.a}^N))(\forall \sigma \in \Sigma \cup \{\epsilon\}) s\sigma \in L(G, \gamma_{l.a}^N) \Leftrightarrow \sigma \in \gamma_{l.a}^N(s)$.

The closed-loop marked behavior is

$$L_m(G, \gamma_{l.a}^N) = L(G, \gamma_{l.a}^N) \cap L_m(G).$$

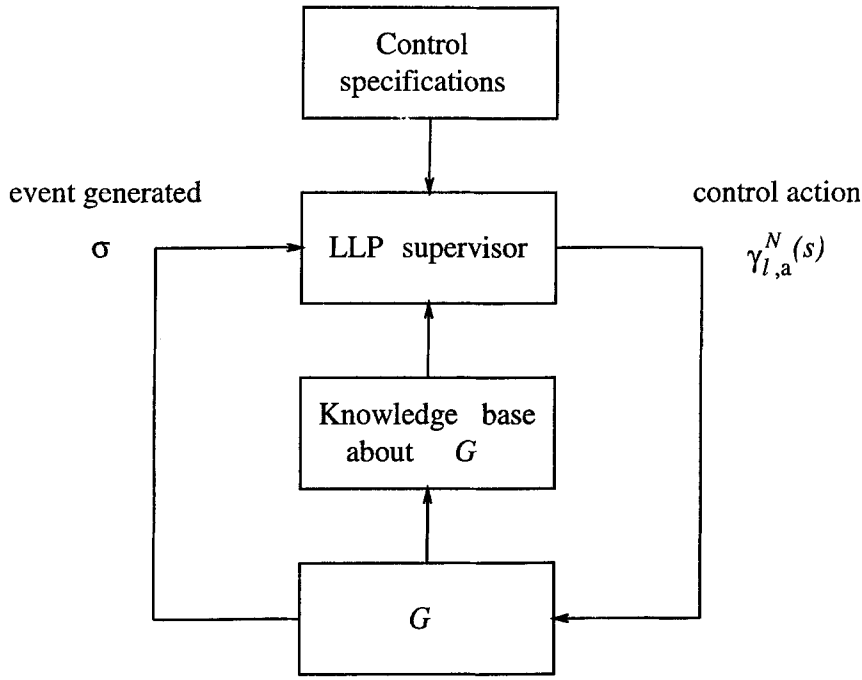


Figure 1. Limited lookahead supervisory control.

The control action $\gamma^N_{l.a}(s)$ 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 Chung, Lafortune, and Lin (1993a), we have considered the "standard" supervisory control problem (cf. Ramadge and Wonham 1987) with either a conservative or an optimistic attitude (cf. Chung, Lafortune, and Lin 1992). Given that the desired legal behavior for the closed-loop system $L(G, \gamma^N_{l.a})$ is the language $\overline{K}$ where $K \subseteq L_m(G)$, $K \neq \emptyset$ and $K = \overline{K} \cap L_m(G)$, the control action $\gamma^N_{l.a}(s)$ at trace $s \in L(G)$ is defined as in Chung, Lafortune, and Lin (1992), i.e.,

$$\gamma^N_{l.a}(s) = \overline{L_a^{\uparrow}|_1} \cup \overline{\Sigma_u \cap \Sigma_{L(G)}(s)}$$

where

$$L_a \equiv L_m(P) = \begin{cases} \textit{conservative}: & K/s|_N - (\overline{K}/s|_N - \overline{K}/s|_{N-1}) = K/s|_{N-1} \\ \textit{optimistic}: & K/s|_N \cup (\overline{K}/s|_N - \overline{K}/s|_{N-1}) \end{cases} \tag{2.1}$$

$L(P) \equiv L(G)/s|_N$

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

$\Sigma_{L(G)}(s) = \{\sigma \in \Sigma : s\sigma \in L(G)\}$.

The notation $L(P) \equiv L(G)/s|_N$ for the uncontrolled behavior and $L_a \equiv L_m(P)$ for the desired behavior is as in Chung, Lafortune, and Lin (1993a), except that now we explicitly identify the attitude as a subscript.

In this context, Chung, Lafortune, and Lin (1993a) reformulate the problem of finding the supremal controllable sublanguage in the limited lookahead window $L(P)$ as a finite-horizon optimal control problem. A tree generator (i.e., a generator whose digraph representation is a tree) is used to represent the finite language $L(P)$ and thus each trace in the lookahead window $L(P)$ is a state in the corresponding tree generator state space, denoted by X. Each state of X is classified, according to L_a, as a member of one of the following three sets:

$$\begin{aligned} X^a_{illegal} &= \{x \in L(P) : x \in L(P) - \overline{L_a}\} \\ X^a_m &= \{x \in L(P) : x \in L_a\} \\ X^a_{transient} &= \{x \in L(P) : x \in \overline{L_a} - L_a\}. \end{aligned}$$

The set of legal states is defined as

$$X^a_{legal} = \{x \in L(P) : x \in \overline{L_a}\}$$

which is the union of X^a_m and $X^a_{transient}$. The superscripts a in the above partition of X refer to the fact that the states of X are classified for legality and marking according to the particular attitude adopted, as can be seen in (2.1). In particular, states on the boundary

$$X_N = \{x \in X : |x| = N\}$$

will be either in $X^a_{illegal}$ or in X^a_{mc}, which is defined as

$$X^a_{mc} = \{x \in X^a_m : \Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset\}.$$

To reflect both the facts that a supervisor does not disable uncontrollable events and that we are interested in a nonblocking supervisor, we can appropriately assign the involved control costs and terminal costs of the optimal control problem under consideration (see Chung, Lafortune, and Lin 1993a). Furthermore, we have shown in Theorem 3.2 of Chung, Lafortune, and Lin (1993a) that the cost-to-go function of this optimal control problem, $V^N_{l.a} : X \longrightarrow \{0, \infty\}$, can be decided as follows:

$$V^N_{l.a}(x) = 0 \Leftrightarrow \begin{cases} x \in X^a_m \cup X^a_{transient} \\ \wedge\ (\forall \sigma \in \Sigma_u \cap \Sigma_{L(P)}(x)) V^N_{l.a}(x\sigma) = 0 \\ \wedge\ (x \in X^a_{transient} \Rightarrow (\exists \sigma \in \Sigma_{L(P)}(x)) V^N_{l.a}(x\sigma) = 0). \end{cases} \tag{2.2}$$

In relating the cost-to-go function $V^N_{l.a}(x)$ to the control action $\gamma^N_{l.a}(s)$ under consideration, we first recall Theorem 3.4 of Chung, Lafortune, and Lin (1993a), which states how to construct the least-restrictive optimal policy g^* from the cost-to-go function $V^N_{l.a}$ of the optimal control problem (see Definition 2.1 of Chung, Lafortune, and Lin (1993a)):

$$\text{if } V^N_{l.a}(x) = 0, \text{ then } g^*(x) = \{\sigma \in \Sigma_{L(P)}(x) : V^N_{l.a}(x\sigma) = 0\}.$$

The control action $\gamma_{l.a}^N(s)$ can then be derived from $g^*(x_0)$, where x_0 is the root of the tree at $L(P)$ and corresponds to the suffix ϵ of s (see Section 4.2 of Chung, Lafortune, and Lin (1993a)):

$$\gamma_{l.a}^N(s) = \begin{cases} g^*(x_0) \cup \{\epsilon\} & \text{if } V_{l.a}^N(x_0) = 0 \\ \overline{\Sigma_u \cap \Sigma_{L(P)}(x_0)} & \text{otherwise (i.e., if } V_{l.a}^N(x_0) = \infty). \end{cases}$$

In a nutshell, the control action $\gamma_{l.a}^N(s)$ equals the set of all next events $\sigma \in \Sigma_{L(P)}(s)$ with $V_{l.a}^N(x_0\sigma) = 0$.

Therefore, to facilitate the determination of the control actions $\gamma_{l.a}^N(s)$, an efficient scheme to calculate $V_{l.a}^N$ is required. In Chung, Lafortune, and Lin (1993a), $V_{l.a}^N$ is calculated by a backward dynamic programming technique, which computes the costs-to-go for all the states in the limited lookahead window. However, in general not all the states and their associated costs-to-go are relevant to specific control actions. Moreover, at any given window, we are only interested in the control action $\gamma_{l.a}^N(s)$ at the root s, and not in the complete control policy for the whole tree $L(P)$.

In light of the above considerations, we present in the next section a Variable Lookahead Policy algorithm which expands the lookahead window $L(P)$ in a forward manner and computes the associated costs-to-go only as necessary. In particular, only the part of the limited lookahead window directly relevant to the control action $\gamma_{l.a}^N(s)$ is expanded and evaluated for the cost-to-go, thus potentially reducing significantly the calculations required to obtain the control action.

3. The Variable Lookahead Policy Algorithm

3.1. *The Main Algorithm and its Special Cases*

When computing the supremal controllable sublanguage, or equivalently the costs-to-go, within a given lookahead window, one is faced with the problem of dealing with the uncertainty on the system behavior beyond N steps. As (2.1) shows, an LLP supervisor with the conservative (or the optimistic, respectively) attitude introduced in Chung, Lafortune, and Lin (1992) "fakes" the legal states on the boundary (Nth level of the tree) as illegal (or marked, respectively), thus assigning a faked ∞ (or 0, respectively) cost-to-go to these legal states for the dynamic programming algorithm. An VLP algorithm which adopts attitude a (where a stands for either *cons* or *optm*) in resolving the future uncertainty is denoted by its cost-to-go function $V_{v.a}^N$, which is defined as $V_{v.a}^N : X \longrightarrow \{0, \infty\}$; observe that although X is fixed, its partition is a function of the attitude a.

In addition to the conservative and optimistic attitudes, we now introduce a new attitude termed "undecided." A supervisor with an undecided attitude just assigns an "undecided" cost-to-go, denoted by U, to all the legal states on the boundary, while letting other *certain* information (explained below) decide the concerned control action, if possible. An VLP algorithm which adopts the "undecided" attitude in resolving the future uncertainty is denoted by its cost-to-go function V_{vu}^N. In this case the costs-to-go derived do not depend

on a faked state classification at the boundary. The associated cost-to-go function is defined as $V_{vu}^N : X \longrightarrow \{0, U, \infty\}$; this time the previous partitions of X do not apply since no attitude is adopted; the appropriate partition of X is implicit in the algorithm presented below for the calculation of V_{vu}^N. (The undecided attitude could also be used with other types of LLP algorithms, but it was not considered in Chung, Lafortune, and Lin (1992, 1993a).)

We will focus on V_{vu}^N throughout our presentation due to the following four reasons. First, besides the structural similarity between V_{vu}^N and $V_{v.a}^N$, costs-to-go derived by $V_{v.a}^N$ can be obtained directly from those derived by V_{vu}^N through a simple substitution. Second, by using V_{vu}^N, one can distinguish control decisions affected by the uncertainty on the behavior beyond the boundary from those that are not. Third, in the context of LLP supervisory control schemes, V_{vu}^N permits re-utilization of previous computational results. Finally, with simple modifications, V_{vu}^N reduces to $V_{v.a}^N$ as well as to V_v, the VLP algorithm without a boundary, which will be presented in Section 4.

We make three remarks on the notation before we state the main algorithm:

1. We define $\overline{L_u} = \overline{K}/s|_N$, while leaving L_u undefined.
2. $X_{mc} = \{x \in K/s|_{N-1} : \Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset\}$; therefore, the states in X_{mc} are marked states of the legal behavior with no uncontrollable continuation in $L(P)$.
3. Because U will be assigned to either 0 or ∞ when the uncertainty is resolved, we adopt the following convention in computing max and min : $0 \leq U \leq \infty$.

Main VLP Algorithm:
(define function cost-to-go (x); this function returns $V_{vu}^N(x)$.
case:

1. $|x| = N$:

$$V_{vu}^N(x) = \begin{cases} \infty & \text{if } x \in L(P) - \overline{L_u} \\ U & \text{otherwise} \end{cases} \tag{3.1}$$

2. $|x| < N \wedge x \in X_{mc} : V_{vu}^N(x) = 0$; return.
3. $|x| < N \wedge x \notin \overline{L_u} : V_{vu}^N(x) = \infty$; return.
4. $|x| < N \wedge x \notin X_{mc} \wedge x \in \overline{L_u}$:

case:

(a) $\Sigma_{L(P)}(x) \cap \Sigma_u \neq \emptyset$:
let $V_{vu}^N(x) = 0$
(do for all $(\sigma_u \in \Sigma_{L(P)}(x) \cap \Sigma_u)$ until $V_{vu}^N(x\sigma_u) = \infty$:
(cost-to-go $(x\sigma_u)$)
$V_{vu}^N(x) = \max(V_{vu}^n(x\sigma_u), V_{vu}^N(x)))$
return.

(b) $\Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset$:

let $V_{vu}^N(x) = \infty$

(do for all $(\sigma_c \in \Sigma_{L(P)}(x) \cap \Sigma_c)$ until $V_{vu}^N(x\sigma_c) = 0$:

(cost-to-go $(x\sigma_c)$)

$V_{vu}^N(x) = \min(V_{vu}^N(x\sigma_c), V_{vu}^N(x))$

return.)

Observe that 4(a) and 4(b) are implementations of the following two operations, respectively:

$$V_{vu}^N(x) = \max_{\sigma_u \in \Sigma_{L(P)}(x) \cap \Sigma_u} [V_{vu}^N(x\sigma_u)] \tag{3.2}$$

$$V_{vu}^N(x) = \min_{\sigma_c \in \Sigma_{L(P)}(x) \cap \Sigma_c} [V_{vu}^N(x\sigma_c)]. \tag{3.3}$$

Note that when $\Sigma_{L(P)}(x) = \emptyset$, x must be in X_{mc} due to the nonblocking assumption $\overline{L_m(G)} = L(G)$.

The algorithms $V_{v.cons}^N$ and $V_{v.optm}^N$ are exactly the same as V_{vu}^N, with the following provisos:

1. $\overline{L_u}$ is to be replaced by $\overline{L_{cons}}$ and $\overline{L_{optm}}$, respectively.

2. Instead of U, ∞ (0, respectively) is assigned to the legal states at the boundary in the second part of (3.1).

In the next section, we show that the V_{vu}^N algorithm and its variations $V_{v.cons}^N$ and $V_{v.optm}^N$ are correct implementations of the LLP scheme by showing that their costs-to-go for all relevant states are the same as those of $V_{l.a}^N$, given either the conservative or the optimistic attitude.

3.2. *Correctness of the Algorithms*

In order to show that the costs-to-go derived by V_{vu}^N are the same as the costs-to-go derived by $V_{l.u}^N$ up to a simple substitution, we show that the costs-to-go derived by $V_{v.a}^N$ are equal to those derived by $V_{l.a}^N$ with a fixed attitude (Theorems 3.1 and 3.2). Then we show how to obtain $V_{v.a}^N$ from V_{vu}^N (Theorem 3.3).

As an intermediate step in showing that the costs-to-go derived by $V_{v.a}^N$ are equal to those derived by $V_{l.a}^N$, we first define $W_a^N : X \longrightarrow \{0, \infty\}$ as follows (since W_a^N is attitude dependent, so is its associated partition of X):

1. If $x \in X_{illegal}^a$, then $W_a^N(x) = \infty$.

2. If $x \in X_{mc}^a$, then $W_a^N(x) = 0$.

3. If $x \in X^a_{legal} - X^a_{mc}$, then:

$$W^N_a(x) = \begin{cases} 0 \quad \text{if} \begin{cases} (\Sigma_{L(P)}(x) \cap \Sigma_u \neq \emptyset \wedge (\forall \sigma \in \Sigma_{L(P)}(x) \cap \Sigma_u) W^N_a(x\sigma) = 0) \vee \\ (\Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset \wedge (\exists \sigma \in \Sigma_{L(P)}(x)) W^N_a(x\sigma) = 0) \end{cases} \\ \infty \quad \text{otherwise.} \end{cases}$$

$W^N_a(x)$ is always defined for all $x \in X$. In particular, since states in X_N will be either in $X^a_{illegal}$ or in X^a_{mc}, W^N_a will return a well-defined value, either 0 or ∞, based on the first two case conditions.

THEOREM 3.1 *Under the same attitude,* $V^N_{v.a}(x) = W^N_a(x)$ *for all* $x \in L(P)$.

Proof: If x satisfies any of the first three case conditions in the algorithm for $V^N_{v.a}$, then x must also satisfy either of the first two case conditions in the definition of W^N_a. Therefore, for such x's,

$$V^N_{v.a}(x) = W^N_a(x).$$

On the other hand, the third case condition in the definition of $W^N_a(x)$ can be rewritten as:

1. $\Sigma_u \cap \Sigma_{L(P)}(x) \neq \emptyset$:

   If $(\exists \sigma \in \Sigma_u \cap \Sigma_{L(P)}(x)) V^N_{v.a}(x\sigma) = \infty$, then $W^N_a(x) = \infty$; else $W^N_a(x) = 0$.

2. $\Sigma_u \cap \Sigma_{L(P)}(x) = \emptyset$:

   If $(\exists \sigma \in \Sigma_c \cap \Sigma_{L(P)}(x)) V^N_{v.a}(x\sigma) = 0$, then $W^N_a(x) = 0$; else $W^N_a(x) = \infty$.

This is the fourth case condition in the algorithm for $V^N_{v.a}$. Therefore, for all $x \in L(P)$, $V^N_{v.a}(x) = W^N_a(x)$. ■

LEMMA 3.1 *If* $(\forall \sigma \in \Sigma_{L(P)}(x)) V^N_{l.a}(x\sigma) = W^N_a(x\sigma)$, *then* $V^N_{l.a}(x) = W^N_a(x)$.

Proof: We only need to show that $V^N_{l.a}(x) = 0 \Leftrightarrow W^N_a(x) = 0$.

From the definition of $W^N_a(x)$, we have

$$W^N_a(x) = 0$$
$$\Leftrightarrow \begin{cases} x \in X^a_{mc} \vee \\ \{x \in (X^a_{legal} - X^a_{mc}) \wedge [(\Sigma_{L(P)}(x) \cap \Sigma_u \neq \emptyset \wedge \\ (\forall \sigma \in \Sigma_{L(P)}(x) \cap \Sigma_u) W^N_a(x\sigma) = 0)) \vee \\ (\Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset \wedge (\exists \sigma \in \Sigma_{L(P)}(x)) W^N_a(x\sigma) = 0)]\}. \end{cases}$$

Assume $(\forall \sigma \in \Sigma_{L(P)}(x))V_{l.a}^N(x\sigma) = W_a^N(x\sigma)$. Then

$$
\begin{aligned}
& W_a^N(x) = 0 \\
\Leftrightarrow & \left\{ \begin{array}{l} x \in X_{mc}^a \vee \\ \{x \in (X_{legal}^a - X_{mc}^a) \wedge \\ [(\Sigma_{L(P)}(x) \cap \Sigma_u \neq \emptyset \wedge (\forall \sigma \in \Sigma_{L(P)}(x) \cap \Sigma_u) V_{l.a}^N(x\sigma) = 0)) \vee \\ (\Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset \wedge (\exists \sigma \in \Sigma_{L(P)}(x)) V_{l.a}^N(x\sigma) = 0)]\} \end{array} \right. \\
\Leftrightarrow & \left\{ \begin{array}{l} (x \in X_{mc}^a \vee x (\in X_{legal}^a - X_{mc}^a)) \wedge \\ \{(x \in X_{mc}^a \vee [(\Sigma_{L(P)}(x) \cap \Sigma_u \neq \emptyset \wedge (\forall \sigma \in \Sigma_{L(P)}(x) \cap \Sigma_u) V_{l.a}^N(x\sigma) = 0)) \vee \\ (\Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset \wedge (\exists \sigma \in \Sigma_{L(P)}(x)) V_{l.a}^N(x\sigma) = 0))]\}. \end{array} \right.
\end{aligned}
$$

To facilitate the presentation, we define

$$
\begin{aligned}
A &= x \in X_m^a \\
B &= x \in X_m^a \cup X_{transient}^a \\
C &= \Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset \\
D &= (\forall \sigma \in \Sigma_{L(P)}(x) \cap \Sigma_u) V_{l.a}^N(x\sigma) = 0 \\
E &= (\exists \sigma \in \Sigma_{L(P)}(x)) V_{l.a}^N(x\sigma) = 0.
\end{aligned}
$$

Since

$$x \in X_{mc}^a \vee x \in (X_{legal}^a - X_{mc}^a) \Leftrightarrow x \in X_m^a \cup X_{transient}^a$$

and

$$x \in X_{mc}^a \Leftrightarrow x \in X_m^a \wedge \Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset \Leftrightarrow A \wedge C$$

we have

$$
\begin{aligned}
W_a^N(x) = 0 &\Leftrightarrow B \wedge [(A \wedge C) \vee (\neg C \wedge D) \vee (C \wedge E)] \\
&\Leftrightarrow B \wedge (C \vee D) \wedge (A \vee D \vee E) \wedge (\neg C \vee A \vee E). \\
W_a^N(x) = 0 &\Leftrightarrow B \wedge D \wedge (\neg C \vee A \vee E) \\
&\Leftrightarrow B \wedge D \wedge ((C \wedge \neg A) \Rightarrow E) \\
&\Leftrightarrow \left\{ \begin{array}{l} x \in X_m^a \cup X_{transient}^a \wedge \\ (\forall \sigma \in \Sigma_{L(P)}(x) \cap \Sigma_u) V_{l.a}^N(x\sigma) = 0 \wedge \\ (\Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset \wedge x \in X_{transient}^a \Rightarrow \\ (\exists \sigma \in \Sigma_{L(P)}(x)) V_{l.a}^N(x\sigma) = 0) \end{array} \right. \\
&\Leftrightarrow V_{l.a}^N(x) = 0.
\end{aligned}
$$

■

The following result shows that the costs-to-go derived by $V_{l.a}^N$ are the same as those derived by W_a^N, and thus the same as those derived by $V_{v.a}^N$.

THEOREM 3.2 *Under the same attitude,* $V_{l.a}^N(x) = W_a^N(x)$ *for all* $x \in L(P)$.

Proof: According to Lemma 3.1, it suffices to show that for all states $x \in X_N$, $V^N_{l.a}(x) = W^N_a(x)$. Recall from Chung, Lafortune, and Lin (1993a) that

$$\text{if } x \in X_N, \text{ then } V^N_{l.a}(x) = \begin{cases} 0 & \text{if } x \in L_a \\ \infty & \text{otherwise.} \end{cases}$$

However, given a fixed attitude a, states in X_N will be either in $X^a_{illegal}$ or in X^a_{mc}. Thus

$$\text{if } x \in X_N, \text{ then } V^N_{l.a}(x) = \begin{cases} 0 & \text{if } x \in X^a_{mc} \\ \infty & x \in X^a_{illegal}. \end{cases}$$

Therefore, according to the first two case conditions in the definition of W^a_N,

$$(\forall x \in X_N) V^N_{l.a}(x) = W^N_a(x). \qquad \blacksquare$$

COROLLARY 3.1 $V^N_{v.a}(x) = V^N_{l.a}(x)$ *for all* $x \in L(P)$.

The costs-to-go derived by V^N_{vu} belong to the set $\{0, U, \infty\}$. If $V^N_{vu}(x)$ happens to be either 0 or ∞, then the cost-to-go is certain, irrespective of the system behavior beyond N steps. This is one of the advantages that V^N_{vu} has over $V^N_{v.cons}$ and $V^N_{v.optm}$; it enables us to tell whether or not the control action is affected by the uncertainty of the system behavior beyond N steps away. On the other hand, if $V^N_{vu}(x)$ turns out to be U, the supervisor still has the option of taking either the conservative or the optimistic attitude. In fact, $V^N_{v.cons}(x)$ and $V^N_{v.optm}(x)$ can be derived from $V^N_{vu}(x)$ by means of a simple substitution as we now show.

Let us define two new functions over X:

$$V^N_{v\infty}(x) = \begin{cases} V^N_{vu}(x) & \text{if } V^N_{vu}(x) \in \{0, \infty\} \\ \infty & \text{if } V^N_{vu}(x) = U \end{cases}$$

$$V^N_{v0}(x) = \begin{cases} V^N_{vu}(x) & \text{if } V^N_{vu}(x) \in \{0, \infty\} \\ 0 & \text{if } V^N_{vu}(x) = U. \end{cases}$$

LEMMA 3.2

(i) $(\forall \sigma \in \Sigma_{L(P)}(x)) V^N_{v\infty}(x\sigma) = V^N_{v.cons}(x\sigma) \Rightarrow V^N_{v\infty}(x) = V^N_{v.cons}(x)$.

(ii) $(\forall \sigma \in \Sigma_{L(P)}(x)) V^N_{v0}(x\sigma) = V^N_{v.optm}(x\sigma) \Rightarrow V^N_{v0}(x) = V^N_{v.optm}(x)$.

Proof: We only prove (i); the proof of (ii) is similar.

1. $\Sigma_{L(P)}(x) \cap \Sigma_u \neq \emptyset$: According to the algorithms,

$$V^N_{vu}(x) = \max_{\sigma_u \in \Sigma_{L(P)}(x) \cap \Sigma_u} [V^N_{vu}(x\sigma_u)]$$

$$V^N_{v.cons}(x) = \max_{\sigma_u \in \Sigma_{L(P)}(x) \cap \Sigma_u} [V^N_{v.cons}(x\sigma_u)].$$

But it is also the case that $V^N_{v\infty}(x) = \max_{\sigma_u \in \Sigma_{L(P)}(x) \cap \Sigma_u} [V^N_{v\infty}(x\sigma_u)]$ because:

$$\begin{aligned} RHS = \infty &\Rightarrow V^N_{vu}(x) \geq U \\ &\Rightarrow V^N_{v\infty}(x) = \infty \\ RHS = 0 &\Rightarrow V^N_{vu}(x) = 0 \\ &\Rightarrow V^N_{v\infty}(x) = 0. \end{aligned}$$

2. $\Sigma_{L(P)}(x) \cap \Sigma_u = \emptyset$: According to the algorithms,

$$\begin{aligned} V^N_{vu}(x) &= \min_{\sigma_c \in \Sigma_{L(P)}(x) \cap \Sigma_c} [V^N_{vu}(x\sigma_c)] \\ V^N_{v.cons}(x) &= \min_{\sigma_c \in \Sigma_{L(P)}(x) \cap \Sigma_c} [V^N_{v.cons}(x\sigma_c)]. \end{aligned}$$

■

Again, it is also the case that $V^N_{v\infty}(x) = \min_{\sigma_c \in \Sigma_{L(P)}(x) \cap \Sigma_c} [V^N_{v\infty}(x\sigma_c)]$ because:

$$\begin{aligned} RHS = \infty &\Rightarrow V^N_{vu}(x) \geq U \\ &\Rightarrow V^N_{v\infty}(x) = \infty \\ RHS = 0 &\Rightarrow V^N_{vu}(x) = 0 \\ &\Rightarrow V^N_{v\infty}(x) = 0. \end{aligned}$$

THEOREM 3.3

(i) $V^N_{v\infty}(x) = V^N_{v.cons}(x)$ *for all* $x \in L(P)$.

(ii) $V^N_{v0}(x) = V^N_{v.optm}(x)$ *for all* $x \in L(P)$.

Proof: We only prove (i); the proof of (ii) is similar.

According to Lemma 3.2(i), it suffices to show that for all states $x \in X_N$, $V^N_{v\infty}(x) = V^N_{v.cons}(x)$. Recall from (3.1) that

$$\text{if } x \in X_N, \text{ then } V^N_{vu}(x) = \begin{cases} \infty & \text{if } x \in L(P) - \overline{L_u} \\ U & \text{otherwise.} \end{cases}$$

Hence, $\forall x \in X_N$, $V^N_{v\infty}(x) = \infty$. On the other hand, the counterpart of (3.1) for the conservative case is

$$(\forall x \in X_N) V^N_{v.cons}(x) = \infty.$$

Therefore,

$$(\forall x \in X_N) V^N_{v\infty}(x) = V^N_{v,cons}(x). \qquad \blacksquare$$

4. Properties of the VLP Algorithms

We state in this section two computational properties of the VLP algorithms. The first property is related to the depth of the underlying forward search, and the second one is related to the re-utilization of previous computational results.

As suggested by (3.1), when a VLP algorithm is employed, if the forward search hits the boundary, an undecided cost-to-go can be rippled back to the upstream nodes. This implies that: (i) the final decision on the control action can be affected by the adopted attitude, and (ii) the derived path up to the bouncing point may make no decisive contribution to the control action: derivation on other paths (thus extra computation) is likely to follow. In general, a larger window size N makes it less likely to hit the boundary, while risking increased computational complexity. The following result shows that, conditional on N and on both the uncontrolled system behavior and the associated legal specification, the forward search of the VLP algorithm is guaranteed to terminate before hitting the boundary. Moreover, the resultant system behavior equals the optimal solution as would be obtained by the "conventional" off-line solution.

We first define

$$N' = \begin{cases} \max\{|t| : (\exists s \in \overline{K})(st \in K_{mc} \cup (L(G) - \overline{K}) \\ \wedge(\forall \epsilon < p < t) sp \notin K_{mc} \cup (L(G) - \overline{K}))\} & \text{if it exists} \\ \text{undefined} & \text{otherwise} \end{cases} \tag{4.1}$$

where $p < t$ denotes that p is a strict prefix of s and K_{mc} is defined as

$$K_{mc} = \{s \in K : (\forall \sigma \in \Sigma_u) s\sigma \notin L(G)\}.$$

Note that N' can be undefined either because it is infinite, or in degenerate situations when the set in the braces in the definition is empty.

Let $N_u(K)$ denote the longest finite subtrace of uncontrollable events in the language K, i.e.,

$$N_u(K) = \begin{cases} \max\{|t| : t \in \Sigma_u^* \wedge (\exists s, v \in \Sigma^*) stv \in K\} \text{ if it exists} \\ \text{undefined otherwise.} \end{cases}$$

LEMMA 4.1

(i) If $K = \overline{K}$ and $K \subset L(G)$, then either $N' = N_u(K)$ or $N' = N_u(K) + 1$.

(ii) If $K = \overline{K}$ and $K = L(G)$, then $N' = N_u(K)$.

Proof: The results follow from the definitions of N' and $N_u(K)$. $\blacksquare$

LEMMA 4.2 *If $N \geq N' + 1$ and the VLP algorithms are employed to calculate the costs-to-go, then the values of the costs-to-go of x_0 (the root) and $x_0\sigma$, for all $\sigma \in \Sigma_{L(P)}(x_0)$*

1. *can be calculated without calculating any of the costs-to-go of the states in X_N (the boundary); and*
2. *are independent of the particular algorithm employed (i.e., attitude adopted).*

Proof: This is immediate from the definition of N' and from the definitions of the VLP algorithms. ■

Effectively, the boundary condition (3.1) in the VLP algorithm can be removed, as long as the search depth (or the window size) N is larger than N'. Accordingly, we call the VLP algorithm without the boundary condition as the "unbounded" VLP algorithm; the cost-to-go function of this algorithm is denoted by V_v, and its associated control policy is denoted by γ_v. More specifically, we define $\gamma_v : L(G) \longrightarrow 2^{\Sigma}$ as follows:

$$\gamma_v(s) = \begin{cases} \{\sigma \in \Sigma_{L(P)}(x_0) : V_v(x_0\sigma) = 0\} & \text{if } V_v(x_0) = 0 \\ \Sigma_u \cap \Sigma_{L(P)}(x_0) & \text{otherwise.} \end{cases} \tag{4.2}$$

THEOREM 4.1 *Assume that $K^{\uparrow} \neq \emptyset$. If N' exists, then the closed-loop behavior under the control of γ_v is $L(G, \gamma_v) = \overline{K^{\uparrow}}$.*

Proof: To show that $L(G, \gamma_v) = \overline{K^{\uparrow}}$, we first reformulate the problem of finding $K^{\uparrow}$ as an optimal control problem, as we have proposed in Chung, Lafortune, and Lin (1993a).

Consider the cost function corresponding to the control policy $g : L(G) \to 2^{\Sigma}$

$$J(g) = \Sigma_{s \in L(G,g)} c(s, g(s))$$

where $c(s, g(s))$ is defined as:

$$c(s, g(s)) = \begin{cases} 0 & \text{if } s \notin L(G) - \overline{K} \wedge \Sigma_u \cap \Sigma_{L(G)}(s) \subseteq g(s) \\ & \quad \wedge (s \in (\overline{K} - K) \Rightarrow g(s) \neq \emptyset) \\ \infty & \text{otherwise.} \end{cases} \tag{4.3}$$

We now make the following claim which follows from the results in Section 2 of Chung, Lafortune, and Lin (1993a).

Claim. Under the present hypotheses that $K^{\uparrow} \neq \emptyset$ and that N' is finite, the least-restrictive policy g^* that achieves zero cost exists and $L(G, g^*) = \overline{K^{\uparrow}}$.

Proof of Claim: This result is proved in the same manner as Lemma 2.2 and Theorem 2.1 in Chung, Lafortune, and Lin (1993a) once the following two observations are made. First,

the finiteness of N' guarantees that the language K is "livelock-free" (see Chung, Lafortune, and Lin (1993a) for definition). Second, if Assumption 2.1(ii) in Chung, Lafortune, and Lin (1993a) is changed to require that K (L in the notation of that paper) be livelock-free (as opposed to requiring that $L_m(G)$ be livelock-free), then Lemma 2.2 and Theorem 2.1 in that paper remain true because, in the notation of Chung, Lafortune, and Lin (1993a), equation (3) in Lemma 2.2 plus the fact that $L(P, g) - L_m(P) \subseteq \overline{L} - L_m(P) = \overline{L} - \overline{L} \cap L_m(P) = \overline{L} - L$ (since L is $L_m(P)$-closed), contradicts the livelock-freeness of L. ■

In view of this result, in order to prove that $L(G, \gamma_v) = K^{\uparrow}$, it thus suffices to show that

$$\gamma_v(s) = g^*(s) \; \forall s \in L(G, g^*).$$

For this purpose, we define

$$V^{\infty}(s) = \sum_{t \in L(G/s, g^*/s)} c(st, g^*(st)) \tag{4.4}$$

where G/s and g^*/s denote the system G and the control policy g^* after s has occurred (see Chung, Lafortune, and Lin 1993a). The superscript ∞ reflects the fact that we are considering an infinite horizon problem.

Equation (4.4) can be rewritten as

$$V^{\infty}(s) = c(s, g^*(s)) + \sum_{\sigma \in g^*(s)} V^{\infty}(s\sigma). \tag{4.5}$$

Observe that the proofs of Theorems 3.2 and 3.4 of Chung, Lafortune, and Lin (1993a) depend only on the above recursive decomposition (4.5) for the value function V, on the definition of the cost function c (cf. (4.3)), and on the definition of the least-restrictive optimal control policy g^* (for Theorem 3.4 of Chung, Lafortune, and Lin (1993a)); these proofs hold for finite as well as infinite horizons. Therefore, similarly to Theorem 3.2 of Chung, Lafortune, and Lin (1993a), we can show that

$$V^{\infty}(s) = 0 \Leftrightarrow \begin{cases} s \in X_m^{\infty} \cup X_{transient}^{\infty} \wedge \\ (\nexists \sigma_u \in \Sigma_u \cap \Sigma_{L(G)}(s)) V^{\infty}(s\sigma_u) = \infty \wedge \\ (s \in X_{transient}^{\infty} \Rightarrow (\exists \sigma \in \Sigma_{L(G)}(s)) V^{\infty}(s\sigma) = 0) \end{cases} \tag{4.6}$$

where X_m^{∞} is the set of states associated with the traces in K, and $X_{transient}^{\infty}$ is the set of states associated with the traces in $\overline{K} - K$.

Also, similarly to Theorem 3.4 of Chung, Lafortune, and Lin (1993a):

$$g^*(s) = \{\sigma \in \Sigma_{L(G)}(s) : V^{\infty}(s\sigma) = 0\} \text{ if } V^{\infty}(s) = 0. \tag{4.7}$$

Observe that $V^{\infty}(s) = 0$ for all $s \in L(G, g^*)$ since $K^{\uparrow} \neq \emptyset$. Take $s \in L(G, g^*)$.

We argue that $V^{\infty}(s) = V_v(x_0)$ and $V^{\infty}(s\sigma) = V_v(x_0\sigma)$ $(\forall \sigma \in \Sigma_{L(G)}(s) = \Sigma_{L(P)}(x_0))$. Consider an "overlapping" of a finite tree over an infinite tree: the finite tree represents $L(P) = L(G)/s|_N$ with the legal behavior $L = K/s|_N$, and the infinite tree represents $L(G)$ with the legal behavior K. In the overlapping, x_0 of the finite tree of $L(P)$ is placed over $s \in L(G, g^*)$ of the infinite tree of $L(G)$ to highlight the mapping between $L(P)$ and $L(G)$. By Lemma 4.2, $V_v(x_0)$ and $V_v(x_0\sigma)$ $(\forall \sigma \in \Sigma_{L(P)}(x))$ are all defined. In addition, because the defining properties of both functions (cf. (2.2) and (4.6)) are identical (up to the mapping of the associated arguments),

$$V^{\infty}(s) = V_v(x_0) \; \forall s \in L(G, g^*)$$

$$V^{\infty}(s\sigma) = V_v(x_0\sigma) \; \forall \sigma \in \Sigma_{L(G)}(s) = \Sigma_{L(P)}(x_0).$$

Therefore, comparing (4.2) and (4.7), we conclude that

$$\gamma_v(s) = g^*(s) \;\; \forall s \in L(G, g^*).$$

This completes the proof. ■

Since the VLP algorithms are implementations of the LLP scheme, the results on sufficient conditions for the validity of limited lookahead policies in terms of $N_{mcf\bar{c}}$ and N_{mcmc} in Section 5 of Chung, Lafortune, and Lin (1992) hold for the VLP algorithms as well. Theorem 4.1 strengthens these results in two ways. First, Theorem 4.1 is attitude-independent, since when $N > N'$ the attitude is irrelevant as shown by Lemma 4.2. Second, N' is smaller than both $N_{mcf\bar{c}}$ and N_{mcmc}. Even if both N_{mcmc} and $N_{mcf\bar{c}}$ are undefined, N' can still be defined. Nonetheless, this sufficient bound N' may not always be finite. In such cases, the forward search of the unbounded VLP algorithm may continue forever. In order to return a control action in finite time, we need to impose a bound on the lookahead window and to adopt an attitude.

In particular, when the undecided attitude is adopted, costs-to-go at some states may be undefined (i.e., U). However, if the cost-to-go at a state is defined, then, as we show in the next theorem, it will not change as the system progresses. As this result holds for all N, we will drop the superscript N. Instead, in order to distinguish successive windows, we use a superscript $t = |s|$ to index all the relevant notation and the computational results of the current window rooted at s. We denote $s = s'\alpha$, where s' is the previous trace before the execution of $\alpha \in \gamma_{vu}^{t-1}(s')$. In particular, V_{vu}^t is now defined as $V_{vu}^t : X^t \to \{0, U, \infty\}$, where the states in X^t are labeled by the unique continuation of s that they correspond to. Therefore, $x \in X' \equiv \alpha x \in X^{t-1}$.

THEOREM 4.2

(1) $V_{vu}^{t-1}(\alpha x) = 0 \Rightarrow V_{vu}^t(x) = 0.$

(2) $V_{vu}^{t-1}(\alpha x) = \infty \Rightarrow V_{vu}^t(x) = \infty.$

Proof: We only prove (1). The proof of (2) is similar.

$$
\begin{aligned}
V_{vu}^{t-1}(\alpha x) = 0 \;&\Rightarrow\; V_{v.cons}^{t-1}(\alpha x) = 0 \text{ (by Theorem 3.3)} \\
&\Rightarrow\; V_{l.cons}^{t-1}(\alpha x) = 0 \text{ (by Corollary 3.1)} \\
&\Rightarrow\; V_{l.cons}^{t}(\alpha x) = 0 \text{ (by Corollary 4.1 of Chung, Lafortune, and Lin 1993a)} \\
&\Rightarrow\; V_{v.cons}^{t}(\alpha x) = 0 \text{ (by Corollary 3.1)} \\
&\Rightarrow\; V_{vu}^{t}(\alpha x) = 0 \text{ (by Theorem 3.3)}
\end{aligned}
$$

■

When the VLP algorithm V_v is applicable (i.e., when N' exists), the costs-to-go derived are all defined. As a result, all the costs-to-go derived by V_v are final. Similarly, the costs-to-go derived by $V_{v.cons}^N$ and $V_{v.optm}^N$ will be final whenever the sufficient conditions on N derived in Section 5 of Chung, Lafortune, and Lin (1992) for the validity of these attitudes hold. On the other hand, when the sufficient conditions are not satisfied, $V_{v.cons}^N$ and $V_{v.optm}^N$ still can take advantage of this final value property if the supervisor can distinguish costs-to-go with the "real" 0 and ∞ from those with the "faked" ones (i.e., those affected by the attitude at the boundary level).

5. Implementation of the VLP Algorithm

5.1. *VLP Supervisory Control*

When VLP algorithms are employed, only a variable lookahead window, instead of a whole limited lookahead window, is expanded in deriving a control action at any point of the undergoing system trajectory. A new variable lookahead window is expanded each time the system advances one step, i.e., executes one event. Between successive steps, some of the previous calculations can be re-utilized according to the properties stated in Theorem 4.2.

We first discuss how the underlying variable lookahead window can be represented by a generic tree structure, which is amenable for different system models (e.g., automata, Petri nets, etc.). Then we focus on how to implement the most general on-line control scheme, when the legal language is not closed and the V_{vu}^N algorithm is adopted. Finally, special cases will be examined.

5.2. *Data Structure*

During a VLP supervisory control operation, let s be the current event trajectory from the system behavior $L(G)$. A partial lookahead from s can be viewed as a finite tree, which is represented as a pair of lists:

$$tree(s) = (working\_plate(s)\ subtrees(s))$$

where

$$working\_plate(s) = (last\_event(s)\ seed(s)\ legality\_classification(s)\ cost\_to\_go(s))$$

$$subtrees(s) = \begin{cases} nil & \text{when the depth of } tree(s) \text{ equals 0} \\ (tree(s\sigma_i))_i & \text{otherwise} \end{cases}$$

and where $\sigma_i \in \Sigma_{L(G)}(s)$.

The meaning of each item is as follows.

- $working\_plate(s)$ carries the information to expand the tree from s and contains all the computational results relevant to deciding the control action at s.
- $last\_event(s)$ is the last event of s.
- $seed(s)$ contains the required information to expand the tree from s. $seed(s)$, for instance, can be a state in an automaton model, a token marking in a Petri net model (see, e.g., Peterson 1981, Murata 1989), or a post-process in a Finitely Recursive Process model or a Communicating Sequential Process model (see e.g., Inan and Varaiya (1989), Hoare (1985)), etc.
- $legality\_classification(s)$ records the legality status of s against the given legal constraint for the system behavior and can be any of the following: $\{illegal, marked, transient\}$.
- $cost\_to\_go(s)$ is equal to the cost-to-go at the root of the tree, which can be any of the following: $\{0, U, \infty\}$, depending on the particular variation of VLP employed. $cost\_to\_go(s)$ is used to decide the control action at s.
- $subtrees(s)$ consists of a list of subtrees, $(tree(s\sigma_i))_i$ when $tree(s)$ is not a singleton, or the associated length is not 0. Each of the subtrees itself is a tree with the root located at $s\sigma_i$. In general, not all events in the active set of $\Sigma_{L(G)}(s)$ are to be generated in the list by the particular VLP algorithm selected; only those relevant to the computation at hand will be generated.

5.3. *Flow Charts*

We first explain how the V_{vu}^N algorithm operates in an on-line supervisory control scheme. Then, operation by other variations of the VLP algorithm will be elaborated upon as special cases of V_{vu}^N.

The overall on-line operation of a VLP supervisor is divided into a startup phase and successive updating phases. During the startup phase, both the initial system configuration and the window size N are to be decided. All the required calculations must be performed from scratch. On the other hand, during the successive updating phases, whose flow chart is depicted in Fig. 2, many of the previous calculations can be re-utilized from one step to the next; the only necessary additional calculations are those that reflect the new information

gained with the one-step transition of the system. (The flow chart for the startup phase is similar to that in Fig. 2.)

In order to decide on the control action at the current trace s, the cost-to-go at the associated root state x_0 has to be decided first. The cost-to-go routine, depicted in Figures 3 and 4, decides the cost-to-go at the root of the current working tree. Events in the active set are generated one by one. Due to the properties of the VLP algorithms, not all of the events in the active set may have to be generated in deciding the cost-to-go. One reason is that some of them may have already been generated before. According to Theorem 4.2, all decided costs-to-go are final, which makes re-utilization of previous results possible, thus sparing repetitive tree expansion and the associated cost-to-go calculation.

Another reason why not all events may have to be generated is that not necessarily all of the events in the active set contribute to deciding the cost-to-go at the current node. When there is at least one uncontrollable event present in the active set, controllable events will have no say in deciding the current node's cost-to-go. In addition, during the process of generating the uncontrollable events one by one, if any of them has a cost-to-go equal to ∞, then this will suffice to conclude that the cost-to-go at the current node is also ∞. Similarly, if all the costs-to-go of the uncontrollable events in the active set are 0 with the exception of one or more with cost U, then the cost-to-go at the current node is also U. On the other hand, if no uncontrollable event is present in the active set, then the controllable events will be generated one by one until any one of them is found to have cost-to-go equal to 0. Then the cost-to-go at the current node will also be 0, unless all of them are U or ∞.

If an ∞ or an U cost-to-go results at the current trace s, the system will be in danger of being uncontrollably dragged into the illegal region or into blocking, occurrences that are termed Starting Errors or Run Time Errors in Chung, Lafortune, and Lin (1992). In particular, if an U cost-to-go results, the supervisor would have the option of taking either the conservative or the optimistic attitude in resolving the U cost-to-go. Otherwise, a 0 cost-to-go signifies the possibility of obtaining a valid (as defined in Chung, Lafortune, and Lin 1992) control action.

As mentioned earlier, in order to decide the cost-to-go at the root of the current tree, it need not be necessary to generate all the events in the active set. However, in order to decide the control action at s, it is required that the costs-to-go of all events in the active set of s be determined (cf. definition of $\gamma_{l,a}^{N}(s)$ in Section 2). This is done by the routine continue-at-controllable (see Fig. 2), which calls the cost-to-go routine for all controllable events in the current active set; this routine is necessary because the routine cost-to-go may have skipped some or all of them, for the reasons described above.

The control-map routine, depicted in Fig. 5, is then called to decide the control action at s, which corresponds to all the subsequent events leading to costs-to-go of 0. If some of the events in the active set result in a cost-to-go of U, the supervisor again has the option of taking a specific attitude in deciding if such an uncertain continuation should be included or not in the control action. In particular, uncontrollable events leading to a cost-to-go of U have to be included; otherwise, the supervisor would have to disable uncontrollable events, thus violating the premise of supervisory control.

After the control action is decided, the system will advance one step, i.e., execute one event from the control action. Subsequently, the subtree following the event executed can

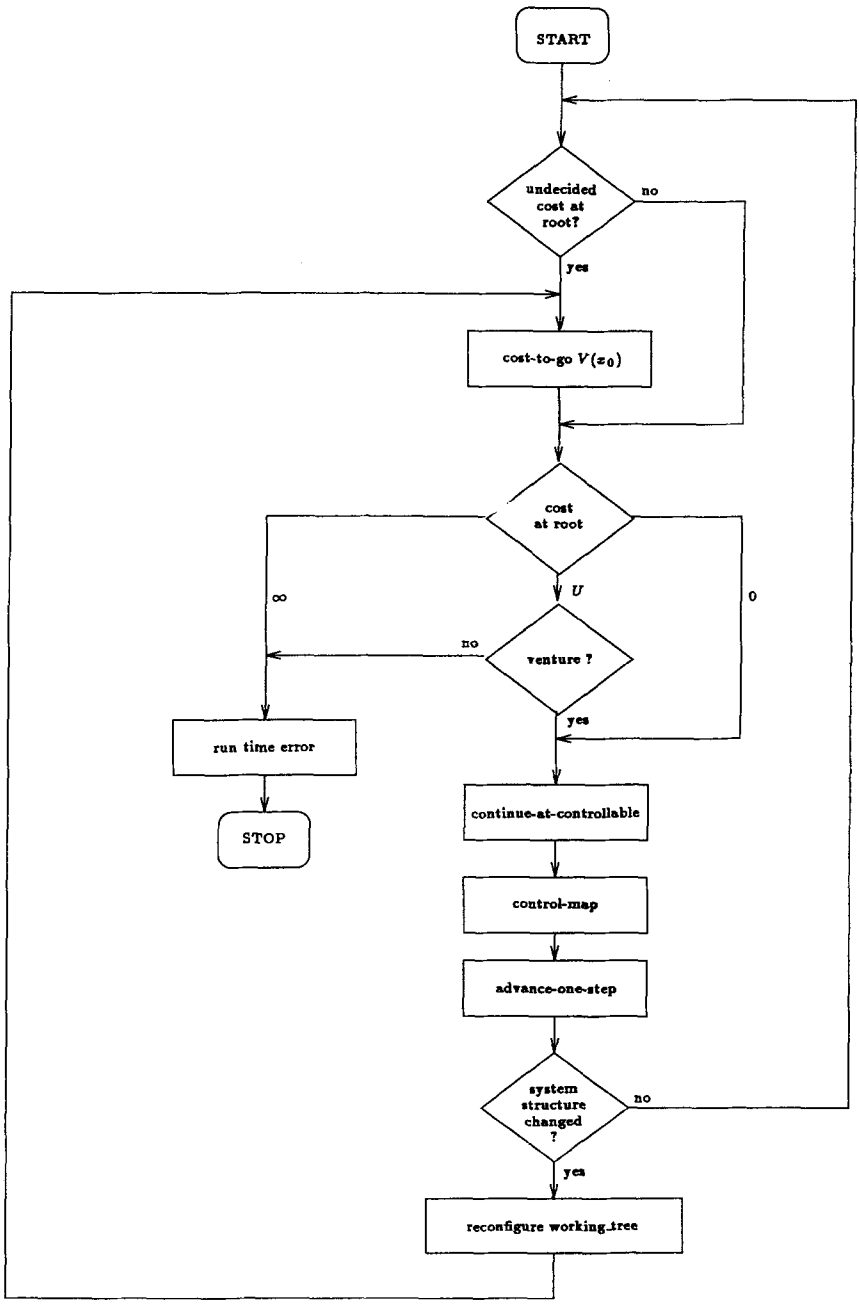
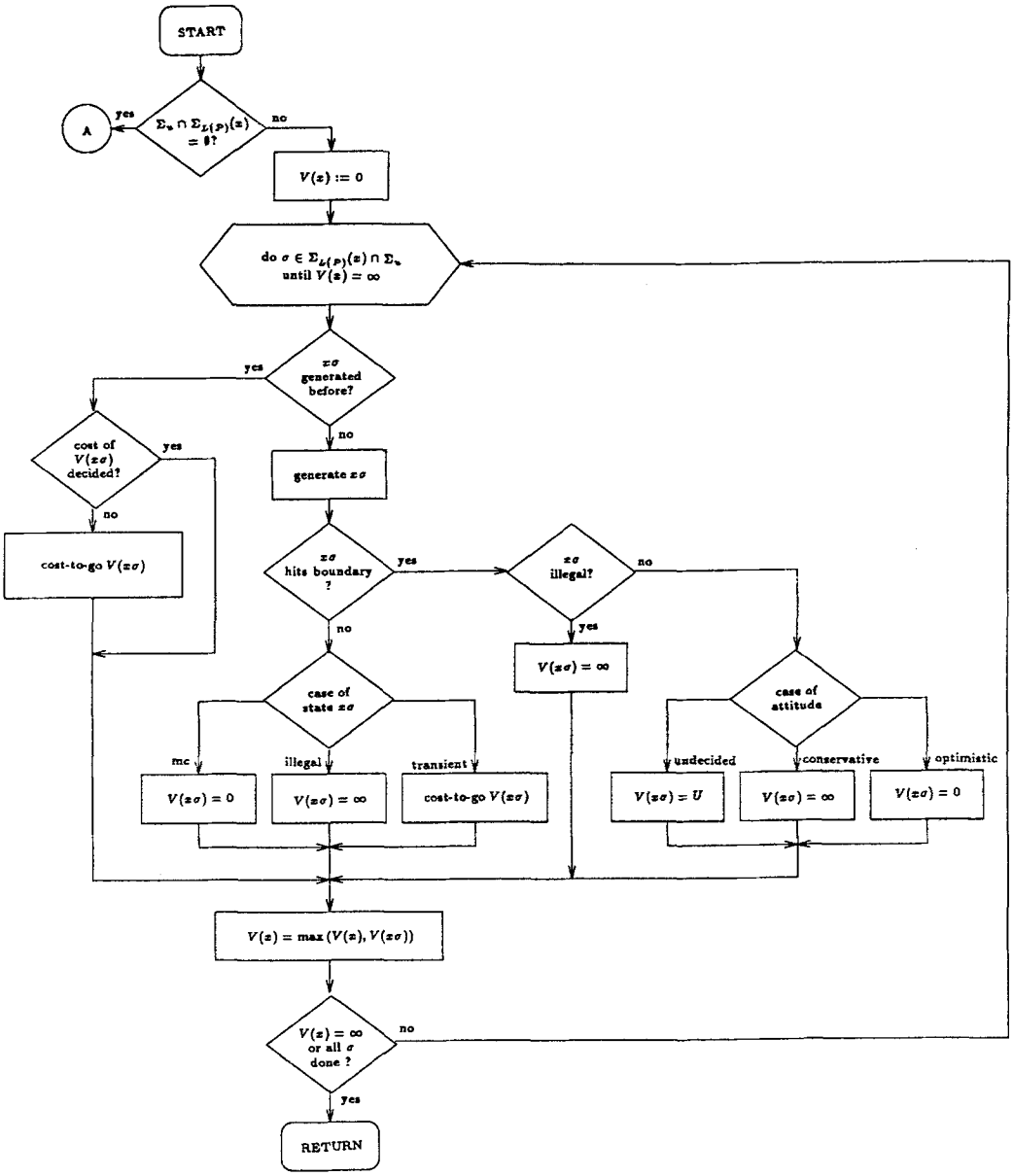


Figure 2. Successive updating phases.


START
A
yes
$\Sigma_s \cap \Sigma_{L(p)}(x) = \emptyset$?
no
$V(x) := 0$
do $\sigma \in \Sigma_{L(p)}(x) \cap \Sigma_s$ until $V(x) = \infty$
yes
$x\sigma$ generated before?
no
generate $x\sigma$
yes
cost of $V(x\sigma)$ decided?
no
cost-to-go $V(x\sigma)$
$x\sigma$ hits boundary ?
yes
$x\sigma$ illegal?
no
yes
$V(x\sigma) = \infty$
no
case of state $x\sigma$
mc
illegal
transient
$V(x\sigma) = 0$
$V(x\sigma) = \infty$
cost-to-go $V(x\sigma)$
case of attitude
undecided
conservative
optimistic
$V(x\sigma) = U$
$V(x\sigma) = \infty$
$V(x\sigma) = 0$
$V(x) = \max(V(x), V(x\sigma))$
$V(x) = \infty$ or all σ done ?
no
yes
RETURN

state x

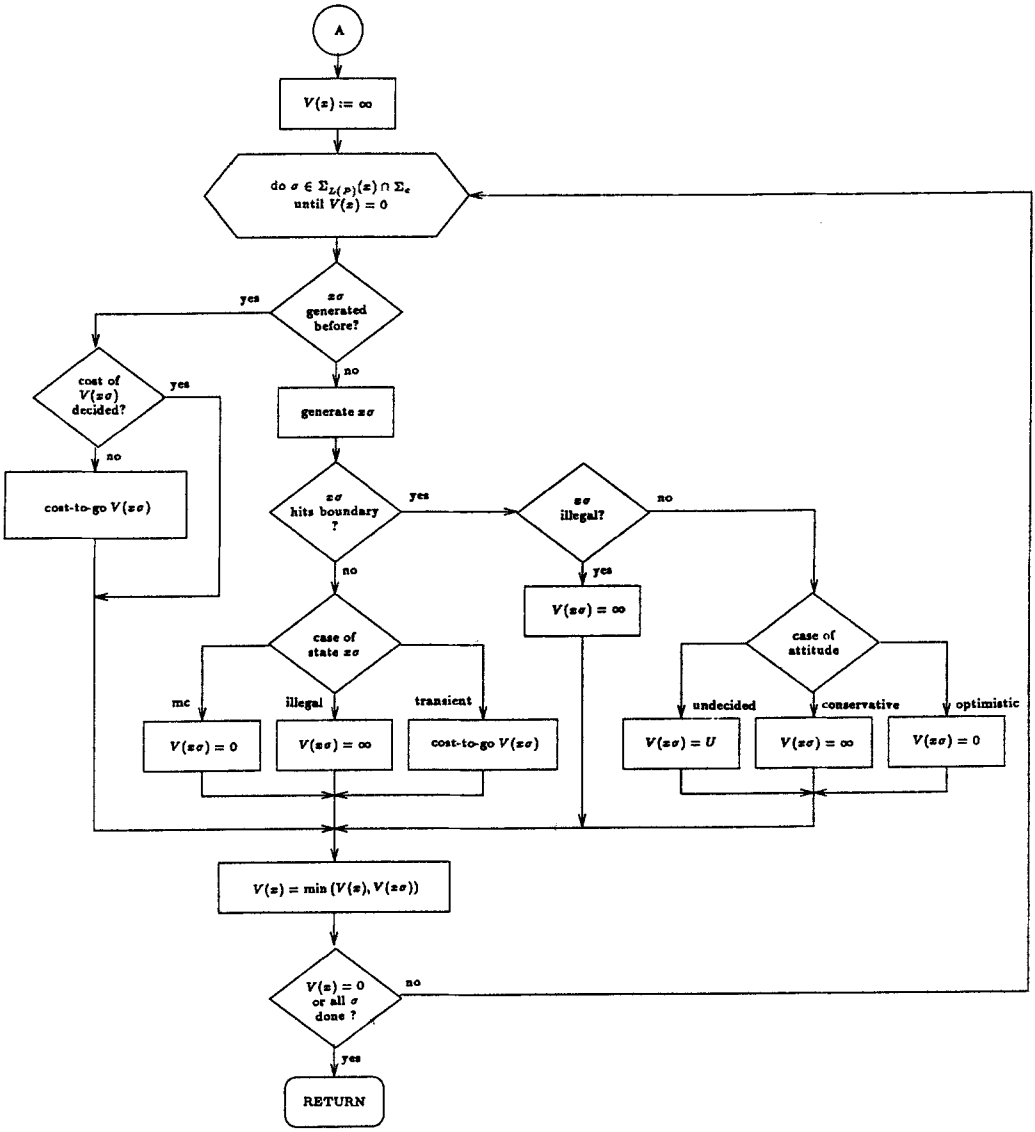


Figure 4. Cost-to-go at state x (continued).

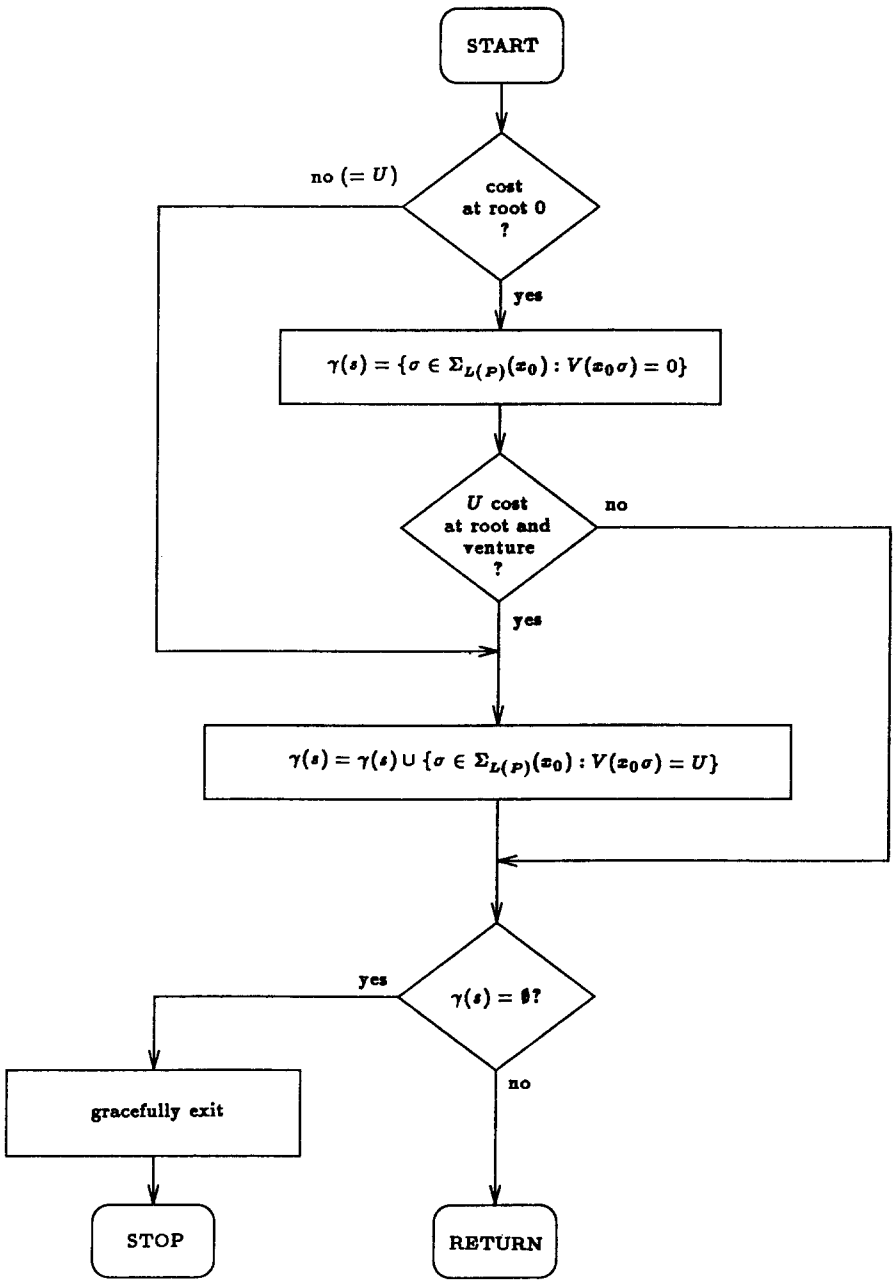


Figure 5. Control-map.

be re-utilized in deciding the control action at the next step. However, before doing so, the supervisor needs to check if the system to be controlled has changed in case it is "time-varying." If the system has changed (e.g., G consists of the synchronous composition of a time-varying set of subprocesses and some subprocesses have terminated or new ones have arrived; see the example in the next section), then a new initial state should be constructed accordingly and the previous variable lookahead window should be discarded. Then the whole aforementioned procedure is repeated.

The other VLP algorithms discussed in the preceding sections reduce to special cases of V_{vu}^N:

- If either the conservative or the optimistic attitude is adopted, then the costs-to-go for legal states at the boundary will be either ∞ or 0, instead of U.

- If the unbounded VLP algorithm V_v is adopted, all the test conditions "$x\sigma$ hits boundary?" present in the cost-to-go and the continue-at-controllable routines should not be included. Also, because costs-to-go for all the states generated in the working tree are final, the following test conditions are automatically satisfied and should be skipped: the test condition "undecided cost at root?" present in the successive updating phases and all the test conditions "cost of $V(x\sigma)$ decided?" in the cost-to-go routine.

In the foregoing discussion, the legal constraint is not assumed to be prefix-closed. When the legal constraint is closed, the left or "A" branch of the cost-to-go routine will never be executed. This is because if s is the first instance of this, then when the cost-to-go was called at the preceding step, the recursive call should have stopped there because the case condition "mc" in the case test "case of state $x\sigma$" should have been satisfied. Thus the cost-to-go routine at s should not have been called. Consequently, for systems with closed constraints, tree expansions only trail traces with uncontrollable continuations.

6. Example and Experiments

We use a train example to demonstrate how the proposed VLP supervisory control scheme can be applied to general "time-varying" open systems. In this context, four variations of the VLP algorithm have been implemented in LISP. The associated trade-off between performance and on-line processing time requirement will be discussed.

6.1. Train Example

We now revisit the example of a simple train system that was first presented in Chung, Lafortune, and Lin (1992) to illustrate the LLP scheme. We slightly modify the system to render it a "time-varying" open system. In addition to showing how the VLP algorithms perform for the original closed legal constraint, we will also show how the VLP algorithms perform when the original legal constraint is made non-closed.

The train system consists of three stations, two junctions and two tunnels, as depicted in Fig. 6. Trains enter the system from stations 1 and 2, while leaving from stations 2

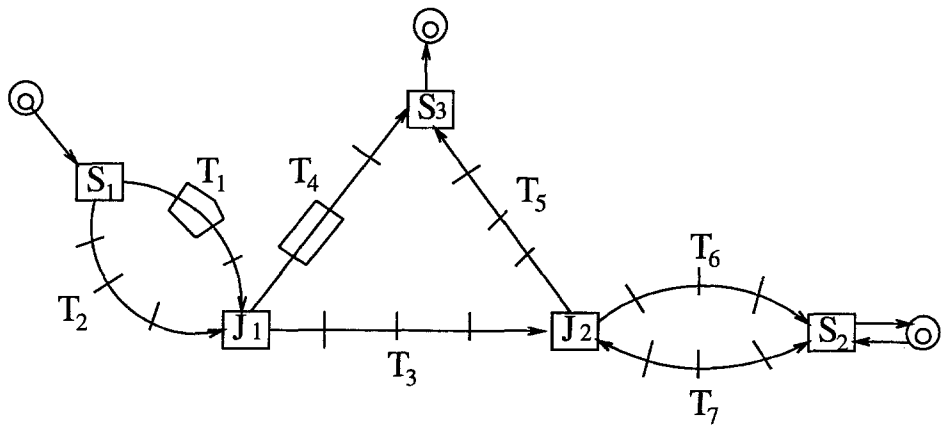


Figure 6. A train system.

and 3. Once in the system, a train can go anywhere as long as it does not change direction when traveling on a track. There are seven tracks connecting various stations and junctions, labeled $T_1, T_2, \ldots, T_7$. In particular, T_7 is a two-way track. Each track is divided into four sections.

To model each individual train, we consider the following events. For $i = 1, \ldots, 6$, $j = 1, \ldots, 4$, $k = 1, 2$, and $l = 2, 3$:

α_j^i: a train enters section j of track T_i;

α_5^i: a train leaves section 4 of track T_i;

β_j: a train traveling from junction 2 to station 2 enters section j of track T_7;

β_5: a train traveling from junction 2 to station 2 leaves section 4 of track T_7;

γ_j: a train traveling from station 2 to junction 2 enters section $5 - j$ of track T_7;

γ_5: a train traveling from station 2 to junction 2 leaves section 1 of track T_7.

γ_k: a train entering station k.

θ_l: a train leaving station l.

Some of these events are made controllable by installing lights at pertinent locations along a track to halt a train from proceeding.

$$\Sigma_c = \{\alpha_j^i, \beta_j, \gamma_j, \gamma_k, \theta_l : j = 1, 3, 5; i = 1, \ldots, 6; k = 1, 2; l = 2, 3\}.$$

Each train is modeled by a generator G_i. The states of G_i are naturally the set of locations spanned by the events defined above: the entry/exit points, the stations, the junctions, and

the four sections along a track. Trains starting at different entry points are to be modeled by different G_i's. Let $\mathcal{G}$ be the set of generators that model all the possible behaviors of a train in the system. At any time instant t, the system is modeled by

$$G = \|_{i=1}^{M(t)} G_i, G_i \in \mathcal{G}$$

where $M(t)$ is the number of trains in the system at time t. For simplicity, assume all the states in G_i are marked. Hence, $L_m(G) = L(G)$.

In an open system like this, it is not known *a priori* how many trains are involved in the synchronous composition nor which $G_i \in \mathcal{G}$ are involved. Thus, the conventional supervisory control approach is for all practical purposes not applicable here because a G that could generate *all* possible future behaviors cannot practically be constructed (unless $\mathcal{G}$ has small cardinality and there is a small upper bound to the total number of trains in the systems).

The legal constraint of the system, in terms of requirements on safety and flow control, can he specified as follows (a detailed description can be found in Chung, Lafortune, and Lin (1992)). Let $s \leq t$ denote that s is a prefix of t.

1. No more than one train is allowed to enter the same section:

$$K_1 = \{t \in L(G): \quad (\forall s \leq t)(\forall i = 1, \ldots, 6)(\forall j = 1, \ldots, 4)|\alpha_j^i|(s) - |\alpha_{j+1}^i|(s) \leq 1$$
$$\wedge\, |\beta_j|(s) - |\beta_{j+1}|(s) \leq 1 \wedge |\gamma_j|(s) - |\gamma_{j+1}|(s) \leq 1\}$$

   where $|\sigma|(s)$ denotes the number of occurrences of σ in s.

2. At least one section is empty between every two trains:

$$K_2 = \{t \in L(G): \quad (\forall s \leq t)(\forall i = 1, \ldots, 6)(\forall j = 1, 2, 3)$$
$$|\alpha_j^i|(s) - |\alpha_{j+1}^i|(s) = 1 \Rightarrow |\alpha_{j+1}^i|(s) - |\alpha_{j+2}^i|(s) = 0$$
$$\wedge\, |\beta_j|(s) - |\beta_{j+1}|(s) = 1 \Rightarrow |\beta_{j+1}|(s) - |\beta_{j+2}|(s) = 0$$
$$\wedge\, |\gamma_j|(s) - |\gamma_{j+1}|(s) = 1 \Rightarrow |\gamma_{j+1}|(s) - |\gamma_{j+2}|(s) = 0\}.$$

3. At most two trains are allowed to occupy junction 1 simultaneously:

$$K_3 = \{t \in L(G): (\forall s \leq t)0 \leq |\alpha_5^1|(s) + |\alpha_5^2|(s) - |\alpha_1^3|(s) - |\alpha_1^4|(s) \leq 2\}.$$

4. At most two trains are allowed to occupy junction 2 simultaneously:

$$K_4 = \{t \in L(G): (\forall s \leq t)0 \leq |\alpha_5^3|(s) + |\gamma_5|(s) - |\alpha_1^5|(s) - |\alpha_1^6|(s) - |\beta_1|(s) \leq 2\}.$$

5. At any given time, trains on T_7 must all travel in the same direction:

$$K_5 = \{t \in L(G): (\forall s \leq t)|\beta_1|(s) - |\beta_5|(s) = 0 \vee |\gamma_1|(s) - |\gamma_5|(s) = 0\}.$$

6. Flows in T_6 and T_7 are balanced (the maximum error=10):

$$K_6 = \{t \in L(G) : (\forall s \leq t) - 10 \leq |\beta_1|(s) + |\gamma_1|(s) - |\alpha_1^6|(s) \leq 10\}.$$

7. Approximately equal numbers of trains pass through tunnels 1 and 2 (the maximum error=10):

$$K_7 = \{t \in L(G) : (\forall s \leq t) - 10 \leq |\alpha_3^1|(s) - |\alpha_4^4|(s) \leq 10\}.$$

Therefore, the legal behavior is described by the prefix-closed language $K = K_1 \cap K_2 \cap \cdots \cap K_7$. Since the longest uncontrollable trace in any G_i has length one, the longest possible uncontrollable trace in $L(G)$ has length equal to the total number of trains in the system. As a result, according to Lemma 4.1 and Theorem 4.1, algorithm V_v of Section 4 guarantees valid control actions during the whole on-line control operation, provided the number of trains in the system is finite.

The depth of the lookahead window required for a supervisor to be valid in the case of a non-closed constraint system is in general longer than when the constraint is closed. To illustrate how all the VLP algorithm variations would perform in the case of a non-closed constraint, in the following experiments, we select the following states occurring in any G_i as marked states: the entry/exit points, the stations and the junctions. Let G_{nc} denote the result of the synchronous composition of these marked generators. Then the language of the new marked traces, denoted by $L_m(G_{nc})$, corresponds to the set of event trajectories where each train in the system is in any of the new marked states. In this context, we can define the non-closed constraint K_{nc} to be

$$K_{nc} = K \cap L_m(G_{nc}).$$

For the non-closed constraint K_{nc}, the sufficient bound stated in Theorem 4.1 does not exist in general. For instance, consider a loop, denoted by $(loop)$, from station 2 to station 2 through junction 2. Then for a trace of two trains $s = \alpha_1^1(loop)^* \in L(G)$, $s \notin L_m(G_{nc})$ while the length of s can be arbitrarily large.

6.2. *Results of the Experiments*

Before we present the results of our experiments, we make the disclaimer that while the proposed VLP algorithms are more efficient than those in Chung, Lafortune, and Lin (1993a) in finding on-line control actions in terms of the number of calculations required, their LISP implementation used for the experiments may be far from the fastest one in terms of execution time. We have chosen LISP because it is easy to implement. We believe that the execution time can be significantly reduced if the flow charts outlined in Section 5.3 are implemented in other languages, such as C, or if special techniques are employed to speed up the time-consuming legality checking procedure (detailed below) during the tree expansion.

Initially, two trains are present at both stations 1 and 2. During the on-line operation, a random mechanism is included to render the system time-varying. Each time after the system executes one event and before it moves on to the next window, a random number R is drawn from a uniform distribution between 0 to 1. If either of the following two conditions is satisfied, then a new train enters the system from station 1 or 2 with equal probability: (1) $R > 0.8$ and the number of trains in the current system is less than 10; or (2) $R < 0.8$ and the number of trains in the current system is less than 2.

The following simulations consider all the variations of the VLP algorithm and were run on a DEC 5000 workstation. The issues of primary concern in these simulations are:

(i) the number of trains in the system;

(ii) the decision time, in units of seconds, required to reach a control action at each step;

(iii) the newly generated states between successive steps;

(iv) the depth of the expanded tree (or the variable lookahead window);

(v) the "certainty ratio" of the control actions with regard to the ambiguity given by the adopted attitude; it is measured by the ratio of the number of events with 0 cost-to-go to that of events with either 0 or uncertain cost-to-go in the active set. For V_{vu}^N, the uncertain costs-to-go correspond to U; for $V_{v,cons}^N$ and $V_{v,optm}^N$, the uncertain costs-to-go correspond to the "faked" values of 0 or ∞ (in our implementation, costs-to-go derived without hitting the boundary are differentiated from those affected by the attitude at the boundary).

(vi) the number of arrivals of new trains during the simulation;

(vii) the percentage of decision time spent in legality checking; and

(viii) the percentage of re-utilization of computations between successive steps, which is measured by the average ratio of the survived portion of a previous window to the whole window at the following step.

Among these statistics, (i) is the result of the random mechanism involved in generating new trains and in selecting an event from a control action for execution. (ii), (iii), and (iv) are particular to the special variation of the VLP algorithm adopted. (v) is always certain for V_v. (vi) and (vii) are similar for all VLP algorithms. (viii) only makes sense for V_v and V_{vu}^N. We will address all of these statistics for V_v, and only (i)–(v) for the other variations.

For the case of the closed constraint K, only V_v is implemented. As shown in Table 1, during a one-hundred-step simulation, it takes about 6.5 seconds to calculate a control action for an average of 7 trains. Almost 90% of the decision time spent to calculate a control action during each cycle is spent in legality checking. This task consists of comparing relevant event counters as specified in constraints K_1 through K_7, one by one. The fact that legality checking requires a significant portion of the decision time shows that this issue is a worthwhile direction of investigation for improving the overall system response time.

In addition, as the system "rolls" from one step to another, the percentage of re-utilization is about 12.5%. On average, the re-utilization rate should be about the inverse of the average

Table 1. Simulation results of V_v over a 100-step simulation.

constraint	new arrivals during simulation	number of trains			decision time per step (seconds)			new states generated at each step			tree depth at each step			percentage of decision time used in legality checking	percentage of re-utilization of computations
		min	ave	max	min	ave	max	min	ave	max	min	ave	max		
closed	21	3	7.04	10	0.0	6.438	92.05	0	159.1	2283	2	4.2	7	89.99	12.45
non-closed	21	3	6.78	10	0.016	28.0	178.9	2	693.8	4334	5	22	51	89.24	8.84

Table 2. Simulation results for all variations of VLP algorithms.

constraint	length of simulation	algorithm	window size	number of trains			decision time per step (seconds)			new states generated at each step			tree depth at each step			certainty ratio of control actions
				min	ave	max	min	ave	max	min	ave	max	min	ave	max	
non-closed	40	$V^N_{v.optm}$	50	4	5.07	7	0.0	61.36	613.2	2	1701	17577	5	26	50	0.93
non-closed	40	$V^N_{v.optm}$	40	4	5	7	0.0	14.02	69.35	0	387.1	1886	5	19	40	0.95
non-closed	40	$V^N_{v.optm}$	30	4	4.18	5	0.0	2.921	11.32	0	83.5	322	5	14	30	0.9
non-closed	40	$V^N_{v.optm}$	20	4	4.18	5	0.0	2.81	11.28	0	80.4	322	5	13	20	0.8
non-closed	40	$V^N_{v.optm}$	10	3	4.1	6	0.1	1.902	7.367	4	53.6	210	5	9.3	10	0.5
non-closed	40	$V^N_{v.cons}$	10	3	4.1	6	0.1	49.48	583.8	4	1411	16297	5	9.3	10	0.5
non-closed	40	$V^N_{v.u}$	10	3	4.1	6	0.116	138.6	622.5	4	3963	18149	5	9.3	10	0.62
non-closed	100	V_v	n/a	3	6.78	10	0.016	28.0	178.9	2	693.8	4334	5	22	51	1.0
closed	100	V_v	n/a	3	7.04	10	0.0	6.438	92.05	0	159.1	2283	2	4.2	7	1.0

number of events in an active set. However, the 21 new arriving trains during the one-hundred-step simulation are partially responsible for such a degradation of re-utilization, because the resultant structural changes virtually make all previous calculations irrelevant.

In Table 2, we show for comparison purposes the simulation results when the original closed language K is replaced by the non-closed language K_{nc}. While during the one-hundred-step simulation V_v always terminates with finite computations, the readers are reminded that the sufficient bound defined in (4.1) does not exist. Table 2 also repeats some results of Table 1 for comparison purposes.

For the case of K_{nc}, four VLP algorithm variations have been implemented: $V^N_{v.optm}$, $V^N_{v.cons}$, V^N_{vu}, and V_v. Table 2 summarizes the simulation results. In our opinion, these results, although limited in scope, demonstrate that the VLP approach is feasible for on-line control applications.

Based on all the simulations that we have performed, we make the following observations. The first one is especially important as it validates, in the context of this example, the advantages of forward search techniques that were claimed throughout this paper.

(1) *The VLP algorithms only expand a small portion of the tree that would have been expanded by a basic limited lookahead policy.*

Over a one-hundred-step simulation using the V_v algorithm, we have observed the following. For the closed legal constraint K, the maximum number of states ever generated in a single step is 2283, which is less than 0.04% of the total number of states in the whole lookahead window with depth 8, the required bound stated in Theorem 4.1. For the non-closed legal constraint K_{nc}, the maximum number of states ever generated in a single step is 4334, which is less than 3×10^{-40} of the total number of states in the whole window with depth 51, the longest trace ever generated in the simulation.

(2) *Longer window sizes reduce the uncertainty of the control actions, while increasing the required calculations.*

During the process of resolving an undecided cost-to-go, the supervisor continues expanding the working tree until either a marked state without any uncontrollable continuation (i.e., a state in X^a_{mc} or "mc state") or an illegal state (i.e., a state in $X^a_{illegal}$) is encountered, or until the boundary (X^a_N) is reached, where the attitude is enforced. For a fixed attitude, an increase in the window size means a longer boundary, which implies that more states may be generated (and then the associated calculations increased). This results in a higher probability to reach an mc state or an illegal state, thus decreasing the influence of "faked" states at the boundary imposed by the adopted attitude, or equivalently reducing the uncertainty in the final control action.

(3) *As compared to $V^N_{v.cons}$ and $V^N_{v.optm}$, V^N_{vu} reduces the uncertainty in control actions at the cost of more calculations.*

Given a fixed window size N, more calculations are required to reach a control action for the undecided attitude than for either the conservative attitude or the optimistic one. An undecided cost assignment U at the boundary makes no decisive contribution to resolving the undecided cost-to-go. In contrast, for $V^N_{v.cons}$ and $V^N_{v.optm}$, cost assignments with either 0 or ∞, imposed at the boundary to reflect the attitude adopted, force the supervisor to make a decision earlier (or terminate the expansion earlier). However, in V^N_{vu}, there is supplementary information that is gained with the extra tree expansion (breadth-wise); this technique then enhances the probability of resolving the undecided cost-to-go with a certain cost. This is the trade-off gained by the extra time spent.

(4) *V_v is better on average than V^N_{vu}, $V^N_{v.cons}$ and $V^N_{v.optm}$.*

When the sufficient window bound stated in Theorem 4.1 exists, V_v is better than V^N_{vu}, $V^N_{v.cons}$ and $V^N_{v.optm}$. V_v is a depth-first search, while V^N_{vu}, $V^N_{v.cons}$ and $V^N_{v.optm}$ all have more characteristics of a breadth-first search. When the bound exists, V_v is preferable to the rest in general. However, if such sufficient bounds are difficult to check or if the user is willing to trade performance for decision time, other candidates than V_v should be used to force the tree expansion to terminate.

7. Conclusion

We have presented new algorithms that implement the Limited Lookahead Policy supervisory control scheme more efficiently than the algorithms proposed in previous work. These algorithms are termed Variable Lookahead Policy algorithms because they perform the calculation of LLP controls by means of a forward search technique over the behavior of the discrete event system under consideration. While the VLP algorithms have the same worst case complexity as the algorithm in Chung, Lafortune, and Lin (1993a), namely that of dynamic programming, the VLP algorithms are guaranteed to perform as well, and possibly much better, in any particular instance. In order to quantify this improvement, it would be necessary to undertake a probabilistic analysis that would depend on assumptions about (i) the average cardinality of the active set after each trace of events, (ii) the probability that an event is controllable, (iii) the probability that an event leads to an illegal state, (iv) the probability that an event leads to a marked state, and so forth. Such an analysis is beyond the scope of this paper. Rather, we have chosen to perform experiments to assess the performance of the VLP algorithms. A detailed example, that would be intractable using conventional supervisory control (cf. Chung, Lafortune, and Lin 1992), was considered for this purpose. For this example, the VLP approach proved feasible and resulted in significant computational savings over the algorithms in previous work. As a final remark, we mention that an illustrative application of the VLP algorithms to the familiar "Cat-and-Mouse" example of the DES literature is discussed in Chung, Lafortune, and Lin (1993c).

Acknowledgements

We wish to thank the reviewers for their useful comments and suggestions.

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.

References

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

S. L. Chung, S. Lafortune, and F. Lin. Recursive computation of limited lookahead supervisory controls for discrete event systems. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 3(1):71–100, March 1993.

S. L. Chung, S. Lafortune, and F. Lin. Supervisory control using variable lookahead policies. In *Proc. 1993 American Control Conf.*, pp. 1203–1208, San Francisco, CA, June 1993.

S. L. Chung, S. Lafortune, and F. Lin. Supervisory control with variable lookahead policies: Illustrative example. In S. Balemi, P. Kozák, and R. Smedinga, editors, *Discrete Event Systems: Modeling and Control—Proceedings of a Joint Workshop on Discrete Event Systems*, pp. 207–214. Birkhäuser Basel Verlag, 1993.

C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall, Englewood Cliffs, NJ, 1985.

K. M. Inan and P. P. Varaiya. Algebras of discrete event models. *Proc. IEEE*, 77(1):24–38, January 1989.

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

T. Murata. Petri nets: Properties, analysis, and applications. *Proc. IEEE*, 77(4):541–580, April 1989.

J. L. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice-Hall, 1981.

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

P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. IEEE*, 77(1):81–98, January 1989.

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