Extension based Limited Lookahead Supervision of Discrete Event Systems [*]

Ratnesh Kumar, Hok M. Cheung
Department of Electrical Engineering
University of Kentucky, Lexington, KY 40506

Steven I. Marcus
Department of Electrical Engineering and ISR
University of Maryland, College Park, MD 20742

Abstract

Supervisory control of discrete event systems using limited lookahead has been studied by Chung-Lafortune-Lin, where control is computed by *truncating* the plant behavior up to the limited lookahead window. We present a modification of this approach in which the control is computed by *extending* the plant behavior by arbitrary traces beyond the limited lookahead window. The proposed supervisor avoids the notion of pending traces. Consequently the need for considering either a conservative or an optimistic attitude regarding pending traces (as in the work of Chung-Lafortune-Lin) does not arise. It was shown that an optimistic attitude may result in violation of the desired specifications. We demonstrate here that a conservative attitude may result in a restrictive control policy by showing that in general the proposed supervisor is less restrictive than the conservative attitude based supervisor. Moreover, the proposed approach uses the notion of relative closure to construct the supervisor so that it is non-blocking even when the desired behavior is not relative closed (Chung-Lafortune-Lin assume relative closure). Finally, the proposed supervisor possesses all the desirable properties that a conservative attitude based supervisor of Chung-Lafortune-Lin possesses. We illustrate our approach by applying it to concurrency control in database management systems.

Keywords: Discrete Events Systems, Supervisory Control, Limited Lookahead, Controllability, Relative Closure.

[*]This research was supported in part by the Center for Robotics and Manufacturing Systems, University of Kentucky, in part by the National Science Foundation under the Grants NSF-ECS-9409712, NSF-ECS-9709796 and NSF-EEC-9402384, and in part by the Office of Naval Research under the Grant ONR-N00014-96-1-5026.

1 Introduction

Discrete Event Systems (DESs) are those dynamical systems which evolve according to the asynchronous occurrence of certain discrete qualitative changes, called *events*. Many man made systems including manufacturing systems, communication protocols, database transaction systems, traffic systems, etc., are examples of DESs. The theory of supervisory control of DESs initiated by Ramadge and Wonham [25, 26] deals with control of the "orderly flow" of events in such systems. The behavior of a DES, also called a *plant*, is described using the set of all finite length sequences of events that it can execute, and a certain subset of the plant behavior represents a desired or *target* behavior. A supervisor, based on its observation of the sequence of events executed by the plant, dynamically disables some of the *controllable* events from occurring, so that the constraints imposed by the desired behavior specification are satisfied.

In the conventional approach for supervisor synthesis, the entire control policy for the given DES is computed *off-line*. This requires automata models which describe the entire behavior of the DES and the target behavior. However, in many situations it is difficult to perform the off-line computations:

- The DES is very complex and contains a large number of states. The off-line computation for the entire control policy is computationally too complex to be feasible.
- The DES is time varying and its complete description cannot be given as a fixed automaton.
- Even if the DES is time invariant, the entire description of the DES is not known initially but possibly becomes known during the execution time.
- Due to the complexity of the desired behavior constraints, it is difficult to construct an automaton consistent with desired specifications.
- The desired behavior itself may be time varying. The desired behavior is incompletely specified initially and must be specified using sensory information during the execution time.

In view of these observations, Chung-Lafortune-Lin [5] have proposed a control scheme using *Limited Lookahead Policy (LLP)*. This control scheme allows control actions to be calculated *on-line* instead of off-line. The next control action is computed on the basis of the DES behavior truncated up to the next N-steps. The control action can be computed based on two extreme attitudes regarding the *pending* sequences—*conservative* and *optimistic*. In [6] the authors have further studied how to use previous computations to help in the next computation of the control action. Effect of taking an undecided attitude to help improve the computational complexity has been investigated by the authors in [7], on-line computational technique in which no constraint is imposed on the depth of the lookahead window has been investigated in [9] assuming that the additional state-information is also available. The application to the case of partial observation has been considered in [11, 10].

Lookahead policies have also been employed earlier for instance for deadlock avoidance in flexible manufacturing systems using Petri net models [1, 27], for planning in artificial intelligence [24, 20, 23], in robotics [13], etc.

In an attempt to propose a modification of the limited lookahead supervision, we note the following limitations of the limited lookahead policy based supervisor studied in [5]:

- Due to the presence of the pending sequences, the limited lookahead policy based supervisor either takes an optimistic attitude which may result in violation of constraints imposed by the desired specifications, or it takes a conservative attitude which may result in a restrictive control policy. This is demonstrated by showing that the proposed supervisor is in general more permissive.

- The assumption of the relative closure (also known as the L_m-closure in the literature) of the desired behavior can only be verified if the entire plant and the desired behavior is known at the outset. So such an assumption is *not* practical for the limited lookahead setting, and should be relaxed.

- Similarly, the assumption that the plant is non-blocking, i.e., each generated trace is a prefix of some marked trace cannot be verified in the setting of limited lookahead and must be relaxed to be realistic.

- The LLP supervisor becomes *blocking* if the target behavior is not a relative closed language. As mentioned above an apriori assumption of relative closure is impractical in the limited lookahead setting as it cannot be verified. Moreover, such an assumption may not always be realistic. To see this consider the following simple example from Ramadge-Wonham [26]: In this example the plant consists of two tandemly operating machines, where the first machine receives an incoming part, and upon processing, puts it into a buffer from where the second machine fetches the part for the final step of processing. Each machine starts from its "idle" state, which is also its only "final" or "marked" state. It is desired that the machines operate in a manner that the buffer never overflows/underflows. Suppose it is further required that upon completion of the task all machines must be in their idle state and the buffer be in its empty state. Then it is easy to see that this desired behavior is not relative closed: The event sequence corresponding to arrival of a part into the first machine followed by departure from that machine has the property that (i) it is a prefix of the desired behavior (it can be extended by the trace arrival into the second machine and departure from the second machine to yield a trace that satisfies the given specification of the buffer overflow/underflow constraint and leaves the machines in their idle states and the buffer in its empty state); and (ii) it also leaves both the machines in their idle state, i.e., it is a trace belonging to the "marked behavior" of the plant. However, this trace does not belong to the desired behavior since the buffer state is not empty at this point.

- The optimistic LLP supervisor requires the knowledge of the desired behavior *beyond* the N-step lookahead window: Determination of whether a next N-step continuation would yield a *prefix* of the desired behavior actually requires the knowledge of the desired behavior beyond the N-step window. So the optimistic LLP supervisor is ill-defined. (Also refer to the footnote below in Section 2.1.)

These observations motivate us to consider a modification of the limited lookahead based supervisor, which avoids these limitations. We call it to be the *extension based Limited Lookahead (ELL)* supervisor. The proposed supervisor estimates the plant behavior as well as the desired behavior based on its next N-step knowledge of the plant behavior. The estimate of the plant behavior is obtained by appending the set of *all* finite length event sequences beyond the N-step projection of the plant behavior. For example when $N = 0$, then the estimate of the plant behavior equals the set of all finite length event sequences. The estimate of the desired behavior is obtained by considering the legal portion of the estimate of the plant behavior when the specification is prefix closed, and otherwise in the non-prefix closed case it is obtained as N-step truncation of the legal portion of the estimate of the plant behavior. The next control action is computed by computing the supremal relative closed and controllable sublanguage of the estimated desired behavior with respect to the estimated plant behavior.

The ELL supervisor is defined for each value of the number of steps of lookahead, and we show that it is non-blocking even when the desired behavior is not relative closed. The ELL supervisor is in general more permissive than the conservative LLP supervisor. In some cases such as when there are no uncontrollable events, or when an upper bound of the number of consecutive uncontrollable events that can occur in the plant is available, the ELL supervisor is strictly more permissive than the conservative LLP one. Moreover, the ELL supervisor possesses many of the desirable properties of the conservative LLP supervisor, and is computationally equally viable.

2 Notation and Preliminaries

Let Σ denote the set of events that occur in a given discrete event plant to be controlled. Σ^* is used to denote the set of all finite length sequences of events from Σ, including the zero length sequence, denoted ϵ. A member of Σ^* is called a *string* or a *trace*, and a subset of Σ^* is called a *language*. Given a string $s \in \Sigma^*$, $|s| \in \mathcal{N}$ is used to denote the length of s; if $t \in \Sigma^*$ is a *prefix* of s, then it is written as $t \leq s$. t is said to be a *proper prefix* of s, denoted $t < s$, if $t \leq s$ and $|t| < |s|$. The notation $pr(s) \subseteq \Sigma^*$ denotes the set of all prefixes of s, i.e., $pr(s) := \{t \in \Sigma^* \mid t \leq s\}$. Given a language $L \subseteq \Sigma^*$, the *prefix closure* of L denoted $pr(L) \subseteq \Sigma^*$, is defined as $pr(L) := \{s \in \Sigma^* \mid \exists t \in L \text{ s.t. } s \leq t\}$; L is said to be *prefix closed* if $L = pr(L)$.

Let P denote a plant. The language pair $(L(P), L_m(P))$ is used to denote the language model of P, which is also denoted as $P \equiv (L(P), L_m(P))$. $L(P) \subseteq \Sigma^*$ is called the *generated* language of P, and represents the set of all finite traces of events which P can execute. Clearly, $L(P)$ is prefix closed and nonempty. $L_m(P) \subseteq L(P)$ is called the *marked* language of P, and represents the set of all those traces whose executions represent completion of a certain task. We use $L_{complete} \subseteq \Sigma^*$ to denote the set of all traces corresponding to completion of a task. Note that this set may contain traces which are not physically possible in the plant. For example, in concurrency control for database transaction systems $L_{complete}$ equals the set of all *completed schedules* [21]. Thus $L_m(P) = L_{complete} \cap L(P)$. In this paper we will assume that the set of complete traces $L_{complete}$ is specified. Hence if the generated language of a plant is known, its marked language can be determined. Given two plants P_1 and P_2, we use $P_1 \leq P_2$ to denote that $L_m(P_1) \subseteq L_m(P_2)$ and $L(P_1) \subseteq L(P_2)$; and $P_1 = P_2$ to denote $P_1 \leq P_2$ and $P_2 \leq P_1$.

A nonempty language $K \subseteq L_m(P)$ is used to denote the *desired* or *target* marked language. We use $K_{legal} \subseteq \Sigma^*$ to denote the set of all *legal* traces. Note that this language may contain traces that are not physically possible in the plant. In database transaction systems, for instance, K_{legal} represents the set of all *serializable* and *strict schedules* [21]. Thus $K = K_{legal} \cap L_m(P)$. In this paper we will assume that the set of legal traces is specified so that the desired marked language can be computed if the marked language of the plant is known.

As in [26], the event set Σ is partitioned into the set of uncontrollable events, denoted $\Sigma_u \subseteq \Sigma$, and the set of controllable events $\Sigma_c = \Sigma - \Sigma_u$. It is assumed that all events are observable [19]. Given a language $L \subseteq \Sigma^*$, it is said to be *controllable* with respect to plant P if $pr(L)\Sigma_u \cap L(P) \subseteq pr(L)$; it is said to be *relative closed* with respect to P if $pr(L) \cap L_m(P) = L \cap L_m(P)$. Note that this is equivalent to $pr(L) \cap L_m(P) \subseteq L$ when $L \subseteq L_m(P)$. We use the following notations to denote the set of controllable sublanguages, the set of relative closed sublanguages, and the set of relative closed and controllable sublanguages of L with respect to P, respectively:

$$\begin{aligned}
C(L,P) &:= \{H \subseteq L(P) \mid H \subseteq L, pr(H)\Sigma_u \cap L(P) \subseteq pr(H)\}, \\
R(L,P) &:= \{H \subseteq L_m(P) \mid H \subseteq L, pr(H) \cap L_m(P) \subseteq H\}, \\
RC(L,P) &:= C(L,P) \cap R(L,P).
\end{aligned}$$

It is known that $supC(L,P), supR(L,P), supRC(L,P)$, namely, the supremal controllable sublanguage, the supremal relative closed sublanguage, the supremal relative closed and controllable sublanguage of L with respect to P, respectively, exist [25]. From definition, $C(L,P) = C(L \cap L(P), P)$, $R(L,P) = R(L \cap L_m(P), P)$, and $RC(L,P) = RC(L \cap L_m(P), P)$; hence $supC(L,P) = supC(L \cap L(P), P)$, $supR(L,P) = supR(L \cap L_m(P), P)$, and $supRC(L,P) = supRC(L \cap L_m(P), P)$. It is shown in [14] that $supR(L,P) = L - [(pr(L) \cap L_m(P)) - L]\Sigma^*$. A formula for $supC(L,P)$ when L is prefix closed is given in [2] and a computationally optimal algorithm for computing it is given in [17]. Computations of $supR(L,P)$ and $supC(L,P)$ can be used to compute $supRC(L,P)$, as $supRC(L,P) = supC(supR(L,P))$ [15].

The *after* operation on a language $L \subseteq \Sigma^*$ by a trace $s \in \Sigma^*$, denoted $L \backslash s \subseteq \Sigma^*$, is defined as $L \backslash s := \{t \in \Sigma^* \mid st \in L\}$; the *truncation* operation on L by a non-negative integer $N \in \mathcal{N}$, denoted $L|_N \subseteq \Sigma^*$, is defined as $L|_N := \{s \in L \mid |s| \leq N\}$. The following lemma appeared in [5], the proofs of which can be found in [4].

Lemma 1 [5] Let $K_1, K_2 \subseteq \Sigma^*$ and $s \in \Sigma^*$. Then

1. $(K_1 \cap K_2)\backslash s = K_1\backslash s \cap K_2\backslash s$,
   $(K_1 \cup K_2)\backslash s = K_1\backslash s \cup K_2\backslash s$.

2. $pr(K_1)\backslash s = pr(K_1\backslash s)$.

3. $(K_1\backslash s)K_2 \subseteq (K_1K_2)\backslash s$,
   $[s \in pr(K_1)] \Rightarrow [(K_1\backslash s)K_2 = (K_1K_2)\backslash s]$.

In order to obtain our main results of Section 3, we need the results of the following two lemmas, the proof of the first one is straightforward, whereas that of the second one is given in Appendix A.

Lemma 2 Let P_1, P_2 be two plants with $P_1 \leq P_2$. Then for $K \subseteq L_m(P_1)$, $supRC(K, P_2) \subseteq supRC(K, P_1)$.

The following lemma generalizes [5, Theorem A.1].

Lemma 3 Let $K \subseteq L_m(P)$, and $s \in \Sigma^*$. Then

1. $supRC(K, P)\backslash s \subseteq supRC(K\backslash s, P\backslash s)$.

2. $[s \in pr(supRC(K, P))] \Rightarrow [supRC(K, P)\backslash s = supRC(K\backslash s, P\backslash s)]$.

2.1 Review of LLP Supervision

For the LLP supervisor, it is assumed that $pr(L_m(P)) = L(P)$, K is relative closed with respect to P,[1] and that the supervisor knows the possible future behavior of the plant within the next N steps at any point during the execution. The LLP supervisor consists of five different blocks depicted in Figure 1. Each block performs a particular operation. The first block f_P^N computes the plant behavior N steps beyond the previously executed trace s, where one step corresponds to the execution of one event. This block generates the corresponding N-tree, which also contains the marking information, i.e.,

$$f_P^N(s) \quad := \quad P\backslash s|_N \equiv (L(P)\backslash s|_N, L_m(P)\backslash s|_N).$$

The block f_K^N determines which traces in the N-tree are legal.

$$f_K^N \circ f_P^N := (pr(K)\backslash s|_N, K\backslash s|_N).\ [2]$$

[1]Note that these two assumptions cannot be verified unless the entire plant and desired behavior is known at the outset which is unrealistic in the limited lookahead setting and must be relaxed.

[2]The computation of the next N-step continuations which belong to the prefix of the desired behavior, i.e., traces belonging to the set $pr(K)\backslash s|_N$, requires the knowledge of continuations *beyond* the next N-step. This is because to determine whether a certain N-step continuation is a *prefix* of the desired behavior, it requires the knowledge of the desired behavior *beyond* the N-step window. This is inappropriate for the limited lookahead setting.

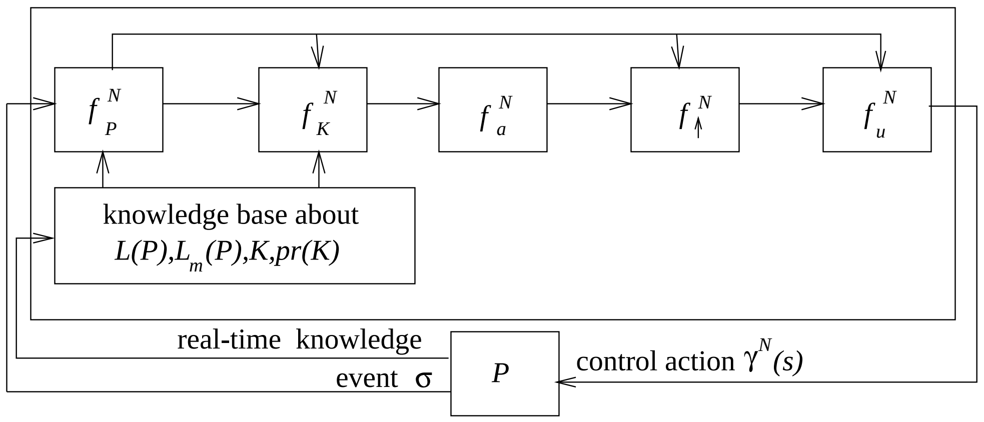

Figure 1: Block diagram of the limited lookahead supervisor

The legal traces of length N in the N-tree are called *pending* traces. The block f_a^N decides whether or not the pending traces are desired. Two attitudes are used, namely, *conservative* and *optimistic*. For conservative attitude, all pending traces are considered as undesired. This results in exclusion of all pending traces from the legal behavior. For optimistic attitude, all pending traces are considered as desired and marked. This results in inclusion of all pending traces into the desired behavior.

$$f_a^N \circ f_K^N \circ f_P^N(s) := \begin{cases} \text{conservative}: & K\backslash s|_{N-1} \\ \text{optimistic}: & K\backslash s|_N \cup (pr(K)\backslash s|_N - pr(K)\backslash s|_{N-1}). \end{cases}$$ [3]

If K is prefix closed, then

$$f_a^N \circ f_K^N \circ f_P^N(s) := \begin{cases} \text{conservative}: & K\backslash s|_{N-1} \\ \text{optimistic}: & K\backslash s|_N. \end{cases}$$

The block $f_\uparrow^N$ computes the supremal controllable sublanguage of the language marked by the modified tree output by f_a^N with respect to the language generated by the tree output by f_P^N.

$$\begin{aligned} f^N(s) &:= f_\uparrow^N \circ f_a^N \circ f_K^N \circ f_P^N(s) \\ &:= supC(f_a^N \circ f_K^N \circ f_P^N(s), P\backslash s|_N). \end{aligned}$$

The block f_u^N computes the control action $\gamma^N(s)$ by including the next allowable events in the tree output by $f_\uparrow^N$ and the set of all uncontrollable events that P can execute after s.

$$\begin{aligned} \gamma^N(s) &:= f_u^N \circ f^N(s) \\ &:= [pr(f^N(s)) \cap \Sigma] \cup [\Sigma_u \cap \Sigma_{L(P)}(s)], \end{aligned}$$

[3]Observe that the optimistic attitude requires the knowledge of the legal behavior beyond the N-step of the lookahead window for the same reason given in the previous footnote.

where $\Sigma_{L(P)}(s) := L(P)\backslash s \cap \Sigma$. Observe that the computation of the control action requires the knowledge of the events that are immediately executable in plant, i.e., at least one-step lookahead is needed.

The generated controlled plant language under the control policy $\gamma^N : L(P) \rightarrow 2^\Sigma$, denoted $L(P, \gamma^N)$, is defined recursively as follows:

- $\epsilon \in L(P, \gamma^N)$;
- $\forall s \in \Sigma^*, \sigma \in \Sigma : s \in L(P, \gamma^N), s\sigma \in L(P), \sigma \in \gamma^N(s) \Rightarrow s\sigma \in L(P, \gamma^N)$.

The marked controlled plant behavior, denoted $L_m(P, \gamma^N)$, is defined as:

$$L_m(P, \gamma^N) := L(P, \gamma^N) \cap L_m(P).$$

The notation γ^N_{cons} and f^N_{cons} (respectively, γ^N_{optm} and f^N_{optm}) is used to indicate that the conservative (respectively, optimistic) attitude is chosen in the module f^N_a.

3 Extension based Limited Lookahead Supervision

In this section, we propose a new limited lookahead based supervisor, which we call the *extension based limited lookahead (ELL)* supervisor. If the generated plant language $L(P)$ is known N steps beyond a previously executed trace, it is natural to assume that any sequence of events could happen after the known N steps. Therefore, we have the following definition.

Definition 1 Given $s \in L(P)$ and $N \geq 0$, the *estimates* of the generated plant language, marked plant language, and the desired marked language (after the trace s based on N-step lookahead), denoted $\overbrace{[L(P)\backslash s]}^N, \overbrace{[L_m(P)\backslash s]}^N, \overbrace{[K\backslash s]}^N$, respectively, are defined as:

$$\begin{aligned}
\overbrace{[L(P)\backslash s]}^N &:= \begin{cases} L(P)\backslash s|_{N-1} \cup (L(P)\backslash s|_N \cap \Sigma^N)\Sigma^* & \text{if } N \geq 1 \\ \Sigma^* & \text{otherwise,} \end{cases} \\
\overbrace{[L_m(P)\backslash s]}^N &:= [L_{complete}]\backslash s \cap \overbrace{[L(P)\backslash s]}^N \\
&(= \overbrace{[L(P)\backslash s]}^N, \text{ when spec. closed}) \\
\overbrace{[K\backslash s]}^N &:= \begin{cases} [[K_{legal}]\backslash s \cap \overbrace{[L_m(P)\backslash s]}^N]|_N \\ [K_{legal}]\backslash s \cap \overbrace{[L(P)\backslash s]}^N, \text{when spec. closed} \end{cases}
\end{aligned}$$

In other words, we append Σ^* to all the traces of $L(P)\backslash s$ of length N to obtain the estimate of the generated plant language. The estimate of the marked language equals that portion of the estimate of the generated language which corresponds to the completion of a task. In the special case when the specification is a safety specification so that it is prefix closed, we have $L_{complete} = \Sigma^*$, which implies $\overbrace{[L_m(P)\backslash s]}^N = \overbrace{[L(P)\backslash s]}^N$. The estimate of the desired marked language equals the legal portion of the estimate of the marked plant language when the specification is prefix closed, and otherwise (when the specification is not prefix closed) it is obtained as the N-step truncation of the legal portion of the estimate of the plant marked language. The reason for applying truncation in the non-prefix closed case is discussed in Remark 3 below.

Remark 1 It should be noted that the estimate of the plant behavior can be further refined by incorporating any additional knowledge regarding the plant behavior. For instance, if we know that the number of consecutive uncontrollable events that can occur in the plant cannot exceed a fixed value, say M, then our estimate of generated behavior can be refined as:

$$\overbrace{[L(P)\backslash s]}^{N} := [L(P)\backslash s|_{N-1} \cup (L(P)\backslash s|_N \cap \Sigma^N)\Sigma^*] \cap (\Sigma_c^*.\Sigma_u^{\leq M}.\Sigma_c^*)^*,$$

where $\Sigma_u^{\leq M} := \{s \in \Sigma_u^* : |s| \leq M\}$. For example, in the concurrency control of database systems at most one consecutive "crash" event can occur, i.e., $M = 1$.

The estimates of the marked plant language and the desired behavior are obtained using the languages $L_{complete}$ and K_{legal}. However, if these languages are not specified and instead $L_m(P)$ and K are given (as is the case in the setting of LLP [5]), then these estimates may be defined as follows (without resulting in loss of any of the results obtained in the paper):

$$\begin{aligned} \overbrace{[L_m(P)\backslash s]}^{N} &:= L_m(P)\backslash s|_{N-1} \cup (L_m(P)\backslash s|_N \cap \Sigma^N)\Sigma^* \\ \overbrace{[K\backslash s]}^{N} &:= K\backslash s|_N. \end{aligned}$$

$\diamondsuit$

Remark 2 The estimate of the generated plant language as given in Definition 1 is a regular language. However, (1) the estimate of the marked plant language when the specification language is non-prefix closed and $L_{complete}$ is non-regular, and (2) that of the desired language when the specification is prefix closed and K_{legal} is non-regular may turn out to be non-regular, and as a result the N-step ELL supervisor may not be computable in those cases.

The first case is not a real problem since when the specification language is non-prefix closed, the desired marked language estimate by definition contains traces of length at most N, and as a result in the computation of the N-step ELL supervisor the estimate of the marked plant language can be replaced by its truncation of length $N + 1$ (thus making it a regular language), without affecting the supervisor.

In the second case some compromise in the permissiveness of the supervisor needs to be made by approximating the desired language estimate by a regular language by truncating it beyond a certain length, say $N_{design} > N$, which is left as a design parameter decided for example by the amount of available computational resources:

$$\overbrace{[K\backslash s]}^{N} := \begin{cases} [K_{legal}\backslash s \cap \overbrace{[L_m(P)\backslash s]}^{N}]|_N \\ [K_{legal}\backslash s \cap \overbrace{[L(P)\backslash s]}^{N}]|_{N_{design}}, \text{ when spec. closed} \end{cases}$$

Using the fact that $N_{design} > N$, it is easy to show that all the results in the paper remain valid even with this revised definition of the estimate of the desired language. $\diamondsuit$

Similar to a LLP supervisor in [5], the ELL supervisor also consists of a series of blocks which perform different operations, and is depicted in Figure 2. The block g_P^N knows the

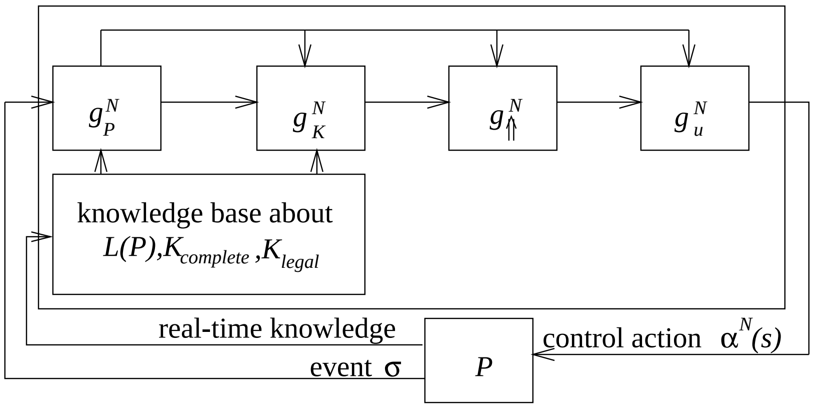

Figure 2: Block diagram of the extension based limited lookahead supervisor

plant behavior N steps beyond a previously executed trace s. It then generates the estimates of the generated and marked plant language.

$$g_P^N(s) \;:=\; \overbrace{[P\backslash s]}^N \equiv (\overbrace{[L(P)\backslash s]}^N, \overbrace{[L_m(P)\backslash s]}^N).$$

The block g_K^N determines which traces in the output of g_P^N are desired.

$$g_K^N \circ g_P^N(s) \;:=\; (pr(\overbrace{[K\backslash s]}^N), \overbrace{[K\backslash s]}^N).$$

The block $g_\Uparrow^N$ computes the supremal relative closed and controllable sublanguage of $\overbrace{[K\backslash s]}^N$ with respect to $\overbrace{[P\backslash s]}^N$.

$$\begin{aligned} g^N(s) \;&:=\; g_\Uparrow^N \circ g_K^N \circ g_P^N(s) \\ &:=\; supRC(\overbrace{[K\backslash s]}^N, \overbrace{[P\backslash s]}^N) \\ (&=\; supC(\overbrace{[K\backslash s]}^N, \overbrace{[P\backslash s]}^N) \text{ when spec. closed}) \end{aligned}$$

The control action $\alpha^N(s)$ is defined slightly differently from that in [5] since we do not require $N \geq 1$, which is required in [5]. This requirement implies that the next event set after the trace s, namely $\Sigma_{L(P)}(s)$, is known. We define $\alpha^N(s)$ as follows:

$$\alpha^N(s) := g_u^N \circ g^N(s) := (pr(g^N(s)) \cap \Sigma) \cup \Sigma_u.$$

Thus, since the next event set $\Sigma_{L(P)}(s)$ may not be known, all the uncontrollable events are enabled by the control action. However, only those uncontrollable events that are also physically possible will occur in the controlled plant.

The generated language of the controlled plant under the control policy $\alpha^N : L(P) \rightarrow 2^\Sigma$, denoted $L(P, \alpha^N) \subseteq L(P)$, is recursively defined as follows:

- $\epsilon \in L(P, \alpha^N)$;
- $\forall s \in \Sigma^*, \sigma \in \Sigma : s \in L(P, \alpha^N), s\sigma \in L(P), \sigma \in \alpha^N(s) \Rightarrow s\sigma \in L(P, \alpha^N)$.

The marked controlled plant behavior is defined as

$$L_m(P, \alpha^N) := L(P, \alpha^N) \cap L_{complete} = L(P, \alpha^N) \cap L_m(P).$$

The ELL supervisor $\alpha^N : L(P) \to 2^\Sigma$ is non-blocking if $pr(L_m(P, \alpha^N)) = L(P, \alpha^N)$. In this paper, we are interested in the synthesis of non-blocking ELL supervisors.

Example 1 Suppose $\Sigma = \{a, b\}$, $\Sigma_u = \{b\}$, $L_{complete} = \Sigma^* b$, and

$$L(P) = K_{legal} = \{s \in \Sigma^* \mid \forall t \le s : \#(a, t) - \#(b, t) \ge 0\},$$

where $\#(x, t)$ denotes number of x's in trace t.

First suppose $N = 1$. Then

$$\overbrace{[L(P)\backslash\epsilon]}{}^{N} = pr(a) \cup a\Sigma^*; \quad \overbrace{[(K)\backslash\epsilon]}{}^{N} = \emptyset.$$

Hence $supRC(\overbrace{[K\backslash\epsilon]}{}^{N}, \overbrace{[P\backslash\epsilon]}{}^{N}) = \emptyset$. So $\alpha^N(\epsilon) = \{b\}$ (all uncontrollable events are always enabled). However, since b is not physically possible, $L(P, \alpha^N) = \epsilon$.

Next suppose $N = 2$. Then

$$\overbrace{[L(P)\backslash\epsilon]}{}^{N} = pr(\{aa, ab\}) \cup \{aa, ab\}\Sigma^*; \quad \overbrace{[(K)\backslash\epsilon]}{}^{N} = \{ab\}.$$

Again, $supRC(\overbrace{[K\backslash\epsilon]}{}^{N}, \overbrace{[P\backslash\epsilon]}{}^{N}) = \emptyset$, and hence $L(P, \alpha^N) = \epsilon$.

Consider a third case when $N = 3$. Then

$$\overbrace{[L(P)\backslash\epsilon]}{}^{N} = pr(H) \cup H\Sigma^*; \quad \overbrace{[(K)\backslash\epsilon]}{}^{N} = \{ab, aab\},$$

where $H = \{aaa, aab, aba\}$. So $supRC(\overbrace{[K\backslash\epsilon]}{}^{N}, \overbrace{[P\backslash\epsilon]}{}^{N}) = \{ab\}$. Hence initially both a and b are enabled. Since a only is physically possible, we have $a \in L(P, \alpha^N)$. In order to compute the next control action we have

$$\overbrace{[L(P)\backslash a]}{}^{N} = pr(H') \cup H'\Sigma^*; \quad \overbrace{[(K)\backslash a]}{}^{N} = \{aab, ab, abb, b, bab\},$$

where $H' = \{aaa, aab, aba, abb, baa, bab\}$. It is easy to see that $supRC(\overbrace{[K\backslash a]}{}^{N}, \overbrace{[P\backslash a]}{}^{N}) = \{b\}$. Hence $ab \in L(P, \alpha^N)$. Continuing in this manner one can conclude that $L(P, \alpha^N) = pr((ab)^*)$.

Closed-loop behaviors for other values of N can be computed in a similar manner. $\diamondsuit$

Remark 3 Since no truncation is involved in computation of the estimate for the desired behavior in the prefix closed case, the supervisor in this case becomes more permissiveness compared to the non-prefix closed case. However, the reason for applying truncation for the non-prefix closed case is that otherwise the supervisor may violate the desired behavior specification as shown by the following example:

Suppose $\Sigma = \{a, b\}, \Sigma_u = \emptyset$, $L(P) = pr(aa), L_{complete} = \{aa, ab\}, K_{legal} = pr(ab)$. Then $L_m(P) = \{aa\}$ and $K = \emptyset$. With one step lookahead when no truncation is used in the computation of the estimate for desired behavior we get:

$$\overbrace{[L(P)\backslash\epsilon]}^{N} = pr(a) \cup a\Sigma^*; \quad \overbrace{[L_m(P)\backslash\epsilon]}^{N} = \{aa, ab\}; \quad \overbrace{[K\backslash\epsilon]}^{N} = ab.$$

So $supRC(\overbrace{[K\backslash\epsilon]}^{N}, \overbrace{[P\backslash\epsilon]}^{N}) = ab$, and the supervisor enables a initially. This, however, violates the desired behavior specification since $K = \emptyset$. $\diamondsuit$

It is clear from the definition of the estimates that the following properties hold for each $N \geq 0$, $s \in \Sigma^*$, and $\sigma \in \Sigma$:

P1 (Anti-Monotonicity): $\overbrace{[L(P)\backslash s]}^{N+1} \subseteq \overbrace{[L(P)\backslash s]}^{N}$;
P2 (Consistency): $\overbrace{[L(P)\backslash s\sigma]}^{N} \subseteq \overbrace{[L(P)\backslash s]}^{N+1}\backslash\sigma$.

It follows from P1 that

$$\overbrace{[L_m(P)\backslash s]}^{N+1} \subseteq \overbrace{[L_m(P)\backslash s]}^{N}.$$

Similarly, it follows from P2 that the following hold:

$$\overbrace{[L_m(P)\backslash s]}^{N+1}\backslash\sigma = \overbrace{[L_m(P)\backslash s\sigma]}^{N}; \quad \overbrace{[K\backslash s]}^{N+1}\backslash\sigma = \overbrace{[K\backslash s\sigma]}^{N}.$$

Consequently, from P1 and P2 we have

$$\overbrace{[P\backslash s]}^{N+1} \leq \overbrace{[P\backslash s]}^{N}; \quad \overbrace{[P\backslash s]}^{N+1}\backslash\sigma = \overbrace{[P\backslash s\sigma]}^{N}.$$

The following lemma presents a few properties of $g^N(s)$, which is defined as $g^N(s) := supRC(\overbrace{[K\backslash s]}^{N}, \overbrace{[P\backslash s]}^{N})$.

Lemma 4 Let $s \in \Sigma^*$ and $\sigma \in \Sigma$. Then for $N \geq 0$,

1. $supRC(\overbrace{[K\backslash s]}^{N}, \overbrace{[P\backslash s]}^{N}) \subseteq supRC(\overbrace{[K\backslash s]}^{N+1}, \overbrace{[P\backslash s]}^{N+1})$.

2. $supRC(\overbrace{[K\backslash s]}^{N}, \overbrace{[P\backslash s]}^{N}) \subseteq supRC(K\backslash s, P\backslash s)$.

3. $[supRC(\overbrace{[K\backslash s]}^{N}, \overbrace{[P\backslash s]}^{N})\backslash\sigma \subseteq supRC(\overbrace{[K\backslash s\sigma]}^{N}, \overbrace{[P\backslash s\sigma]}^{N})]$.

Proof: We begin by proving the first part for the non-prefix closed case first:

$$\begin{aligned}
&supRC(\overbrace{[K\backslash s]}{}^{N}, \overbrace{[P\backslash s]}{}^{N}) \\
&\subseteq supRC(\overbrace{[K\backslash s]}{}^{N}, \overbrace{[P\backslash s]}{}^{N+1}) && \text{(Lemma 2)} \\
&\subseteq supRC(\overbrace{[K\backslash s]}{}^{N+1}, \overbrace{[P\backslash s]}{}^{N+1}) && (\overbrace{[K\backslash s]}{}^{N} \subseteq \overbrace{[K\backslash s]}{}^{N+1})
\end{aligned}$$

In the prefix closed case we do not have $\overbrace{[K\backslash s]}{}^{N} \subseteq \overbrace{[K\backslash s]}{}^{N+1}$, but we have $\overbrace{[K\backslash s]}{}^{N} \cap \overbrace{[L(P)\backslash s]}{}^{N+1} = \overbrace{[K\backslash s]}{}^{N+1}$, which we use to prove the assertion as follows:

$$\begin{aligned}
&supRC(\overbrace{[K\backslash s]}{}^{N}, \overbrace{[P\backslash s]}{}^{N}) \\
&\subseteq supRC(\overbrace{[K\backslash s]}{}^{N}, \overbrace{[P\backslash s]}{}^{N+1}) && \text{(Lemma 2)} \\
&= supRC(\overbrace{[K\backslash s]}{}^{N} \cap \overbrace{[L(P)\backslash s]}{}^{N+1}, \overbrace{[P\backslash s]}{}^{N+1}) && \text{(by definition, see Section 2)} \\
&= supRC(\overbrace{[K\backslash s]}{}^{N+1}, \overbrace{[P\backslash s]}{}^{N+1}) && (\overbrace{[K\backslash s]}{}^{N} \cap \overbrace{[L(P)\backslash s]}{}^{N+1} = \overbrace{[K\backslash s]}{}^{N+1})
\end{aligned}$$

Next we prove the second part.

$$\begin{aligned}
&supRC(\overbrace{[K\backslash s]}{}^{N}, \overbrace{[P\backslash s]}{}^{N}) \\
&\subseteq supRC(\overbrace{[K\backslash s]}{}^{N}, P\backslash s) && \text{(Lemma 2, as } P\backslash s \leq \overbrace{[P\backslash s]}{}^{N} \text{ and } \overbrace{[K\backslash s]}{}^{N} \subseteq L_m(P)\backslash s) \\
&\subseteq supRC(K\backslash s, P\backslash s) && (\overbrace{[K\backslash s]}{}^{N} \subseteq K\backslash s)
\end{aligned}$$

Finally, we prove the last part.

$$\begin{aligned}
&supRC(\overbrace{[K\backslash s]}{}^{N}, \overbrace{[P\backslash s]}{}^{N})\backslash\sigma \\
&\subseteq supRC(\overbrace{[K\backslash s]}{}^{N+1}, \overbrace{[P\backslash s]}{}^{N+1})\backslash\sigma && \text{(part 1, Lemma 4)} \\
&\subseteq supRC(\overbrace{[K\backslash s]}{}^{N+1}\backslash\sigma, \overbrace{[P\backslash s]}{}^{N+1}\backslash\sigma) && \text{(part 1, Lemma 3)} \\
&= supRC(\overbrace{[K\backslash s\sigma]}{}^{N}, \overbrace{[P\backslash s\sigma]}{}^{N}) && \text{(consistency property of estimates)}
\end{aligned}$$

■

The third part of Lemma 4 can be generalized in a straightforward manner using induction to obtain the following corollary.

Corollary 1 Let $s, t \in \Sigma^*$. For $N \geq 0$,

$$[supRC(\overbrace{[K\backslash s]}{}^{N}, \overbrace{[P\backslash s]}{}^{N})\backslash t \subseteq supRC(\overbrace{[K\backslash st]}{}^{N}, \overbrace{[P\backslash st]}{}^{N})].$$

4 Comparing Conservative LLP and ELL Supervisors

In this section, we compare the performance of the ELL supervisor with the conservative LLP supervisor studied in [5]. Since the LLP supervisor assumes K is relative closed with respect to P, in order to make the comparison, we also assume for our ELL supervisor that K is relative closed with respect to P. This implies that $supRC(K, P) = supC(K, P)$.

Lemma 5 For $N \geq 1$, $supC(K\backslash s|_{N-1}, P\backslash s|_N) \subseteq supC({\overbrace{[K\backslash s]}}^{N}, {\overbrace{[P\backslash s]}}^{N})$.

Proof: Since $K\backslash s|_{N-1}$ contains traces with length no more than $N-1$, clearly

$$\begin{aligned} supC(K\backslash s|_{N-1}, {\overbrace{[P\backslash s]}}^{N}) &= supC(K\backslash s|_{N-1}, {\overbrace{[P\backslash s]}}^{N}|_N) \\ &= supC(K\backslash s|_{N-1}, P\backslash s|_N), \qquad (1) \end{aligned}$$

where the last equality follows from the fact that $P\backslash s|_N = {\overbrace{[P\backslash s]}}^{N}|_N$. Also, since $K\backslash s|_{N-1} \subseteq {\overbrace{[K\backslash s]}}^{N}$,

$$supC(K\backslash s|_{N-1}, {\overbrace{[P\backslash s]}}^{N}) \subseteq supC({\overbrace{[K\backslash s]}}^{N}, {\overbrace{[P\backslash s]}}^{N}). \qquad (2)$$

Hence it follows from (1) and (2) that

$$supC(K\backslash s|_{N-1}, P\backslash s|_N) \subseteq supC({\overbrace{[K\backslash s]}}^{N}, {\overbrace{[P\backslash s]}}^{N}),$$

which completes the proof. ■

The following theorem states that the ELL supervisor is in general less restrictive than the conservative LLP supervisor studied in [5].

Theorem 1 For $N \geq 1$, $L(P, \alpha^N) \supseteq L(P, \gamma^N_{cons})$.

Proof: We need to show that if $s \in L(P, \gamma^N_{cons})$, then $s \in L(P, \alpha^N)$. We use induction on the length of s to show this. If $|s| = 0$, then $s = \epsilon$. By definition $\epsilon \in L(P, \alpha^N)$, which establishes the base step.

In order to prove the induction step, let $s = \bar{s}\sigma$, where $\sigma \in \Sigma$. Since $s \in L(P, \gamma^N_{cons})$, which is prefix closed, we have $\bar{s} \in L(P, \gamma^N_{cons})$. Hence it follows from induction hypothesis that $\bar{s} \in L(P, \alpha^N)$. Since $s = \bar{s}\sigma \in L(P, \gamma^N_{cons})$, $\sigma \in \gamma^N_{cons}(\bar{s})$. On the other hand, since $\bar{s} \in L(P, \alpha^N)$, it follows from the definition of ELL supervisor that $s = \bar{s}\sigma \in L(P, \alpha^N)$ if and only if $\sigma \in \alpha^N(\bar{s}) \cap \Sigma_{L(P)}(\bar{s})$. Thus in order to show that $s = \bar{s}\sigma \in L(P, \alpha^N)$, it suffices to show $\gamma^N_{cons}(\bar{s}) \subseteq \alpha^N(\bar{s}) \cap \Sigma_{L(P)}(\bar{s})$. We have

$$\begin{aligned} \gamma^N_{cons}(\bar{s}) &= [f^N_{cons}(\bar{s}) \cap \Sigma] \cup [\Sigma_u \cap \Sigma_{L(P)}(\bar{s})] \\ &= [pr(supC(K\backslash\bar{s}|_{N-1}, P\backslash\bar{s}|_N)) \cap \Sigma] \cup [\Sigma_u \cap \Sigma_{L(P)}(\bar{s})] \\ &\subseteq [pr(supC({\overbrace{[K\backslash\bar{s}]}}^{N}, {\overbrace{[P\backslash\bar{s}]}}^{N})) \cap \Sigma] \cup [\Sigma_u \cap \Sigma_{L(P)}(\bar{s})] \\ &= [pr(g^N(\bar{s})) \cap \Sigma] \cup [\Sigma_u \cap \Sigma_{L(P)}(\bar{s})] \\ &= [[pr(g^N(\bar{s})) \cap \Sigma] \cup \Sigma_u] \cap [[pr(g^N(\bar{s})) \cap \Sigma] \cup \Sigma_{L(P)}(\bar{s})] \\ &= [\alpha^N(\bar{s})] \cap [\Sigma_{L(P)}(\bar{s})], \end{aligned}$$

where the containment follows from Lemma 5; and in the final equality we have used the fact that $N \geq 1$, which implies that $[pr(g^N(\bar{s})) \cap \Sigma] \subseteq \Sigma_{L(P)}(\bar{s})$ so that $[[pr(g^N(\bar{s})) \cap \Sigma] \cup \Sigma_{L(P)}(\bar{s})] = \Sigma_{L(P)}(\bar{s})$. This completes the proof. ■

The following example illustrates that in general the reverse containment of Theorem 1 does not hold, i.e., in general $L(P,\alpha^N) \not\subseteq L(P,\gamma^N_{cons})$, and thus ELL supervisor is strictly more permissive than conservative LLP supervisor in both non-prefix closed and prefix closed cases.

Example 2 Let $\Sigma = \{c\}$, $\Sigma_u = \emptyset$, and

$$K_{legal} = c^*, L_{complete} = (c^M)^*, L(P) = pr((c^M)^*),$$

where $M > 1$ is a fixed number. Then $L_m(P) = K = (c^M)^*$. Then for $N = M$, $L(P,\alpha^N) = pr((c^M)^*)$, but $L(P,\gamma^N_{cons}) = \epsilon$, showing that the conservative LLP supervisor is more restrictive than the ELL supervisor proposed here. Note that the same results would be obtained if $L_m(P)$ and K are specified instead of $L_{complete}$ and K_{legal}.

To see that $L(P,\alpha^N) \not\subseteq L(P,\gamma^N_{cons})$ in the prefix closed case, consider the first example in the proof of Theorem 2 below, which actually shows that $L(P,\alpha^N) \supset L(P,\gamma^{N+1}_{cons})$. $\diamondsuit$

Remark 4 The first part of the above example can be easily modified to illustrate that the ELL supervisor is less restrictive than the conservative LLP supervisor even when the set of uncontrollable events is non-empty and there is a known upper bound on the number of consecutive uncontrollable events that can occur in the plant so that the refined plant behavior estimate of Remark 1 can be used in the computation of the ELL supervisor. $\diamondsuit$

Theorem 1 shows that a N-step lookahead based ELL supervisor is generally more permissive than a N-step lookahead based conservative LLP supervisor. In the next theorem we show that it is generally less permissive than a $(N+1)$-step lookahead based conservative supervisor whenever the specification is non-prefix closed, and otherwise the two supervisors are non-comparable in their permissiveness.

Theorem 2 For $N \geq 1$, $L(P,\alpha^N) \subseteq L(P,\gamma^{N+1}_{cons})$ in the non-prefix closed case, whereas $L(P,\alpha^N)$ and $L(P,\gamma^{N+1}_{cons})$ are non-comparable in the prefix closed case.

Proof: For the non-prefix closed case, by definition the following holds for any trace $s \in \Sigma^*$:

$$K\backslash s|_N = \overbrace{[K\backslash s]}^{N}; \quad P\backslash s|_{N+1} \leq \overbrace{[P\backslash s]}^{N}.$$

So by simply replacing $\overbrace{[K\backslash s]}^{N+1}$ with $K\backslash s|_N$ and $\overbrace{[P\backslash s]}^{N+1}$ with $P\backslash s|_{N+1}$ in the proof steps of the first part of Lemma 4 we can conclude that

$$supC(\overbrace{[K\backslash s]}^{N}, \overbrace{[P\backslash s]}^{N}) \subseteq supC(K\backslash s|_N, P\backslash s|_{N+1}).$$

Next by applying a similar modification in the proof steps of Proposition 1 we obtain the desired containment that $L(P,\alpha^N) \subseteq L(P,\gamma^{N+1}_{cons})$ (in the non-prefix closed case).

To see the non-comparability of $L(P,\alpha^N)$ and $L(P,\gamma^{N+1}_{cons})$ in the prefix closed case consider the following two examples. First suppose $\Sigma = \{a,b\}, \Sigma_u = \{b\}, L_{complete} = \Sigma^*, L(P) = pr(ab^*), = K_{legal}$, and $N = 1$ so that $N+1 = 2$. Then $K\backslash\epsilon|_N = pr(a), L(P)\backslash s|_{N+1} =$

$pr(ab)$. So $supC(K\backslash\epsilon|_N, L(P)\backslash\epsilon|_{N+1}) = \epsilon$, and a is initially disabled by γ^{N+1}_{cons}. However, $\overbrace{[K\backslash\epsilon]}^{N} = \overbrace{[L(P)\backslash s]}^{N} = pr(ab^*)$, and a is initially enabled by α^N. This shows that in this example $L(P, \alpha^N) \supset L(P, \gamma^{N+1}_{cons})$. Next suppose instead $L(P) = pr(aab^*)$, and $K_{legal} = pr(a)$. Then $K\backslash\epsilon|_N = pr(a), L(P)\backslash s|_{N+1} = pr(aa)$. So $supC(K\backslash\epsilon|_N, L(P)\backslash\epsilon|_{N+1}) = pr(a)$, and a is initially enabled by γ^{N+1}_{cons}. However, $\overbrace{[K\backslash\epsilon]}^{N} = pr(a)$, and $\overbrace{[L(P)\backslash s]}^{N} = pr(ab^*)$. So $supC(\overbrace{[K\backslash\epsilon]}^{N}, \overbrace{[L(P)\backslash\epsilon]}^{N}) = \epsilon$, and a is initially disabled by α^N. This shows that in this example $L(P, \alpha^N) \subset L(P, \gamma^{N+1}_{cons})$. ■

The following example illustrates that in the non-prefix closed case the reverse containment of Theorem 2 does not generally hold.

Example 3 Let $\Sigma = \{a, b\}$, $\Sigma_u = \{b\}$, $K_{legal} = L_{complete} = L(P) = pr(aa)$, and $N = 1$ (so $N+1 = 2$). Then $K\backslash\epsilon|_N = pr(a)$ and $L(P)\backslash\epsilon|_{N+1} = pr(aa)$. So $supC(K\backslash\epsilon|_N, L(P)\backslash\epsilon|_{N+1}) = pr(a)$. Hence $a \in L(P, \gamma^{N+1}_{cons})$.

On the other hand, $\overbrace{[K\backslash\epsilon]}^{N} = K\backslash\epsilon|_N = pr(a)$, and $\overbrace{[L(P)\backslash\epsilon]}^{N} = pr(a) \cup a\Sigma^*$. It follows that $supC(\overbrace{[K\backslash\epsilon]}^{N}, \overbrace{[P\backslash\epsilon]}^{N}) = \epsilon$. Since the uncontrollable event cannot occur initially, $L(P, \alpha^N) = \epsilon$. ♢

5 Properties of ELL Supervisor

In this section, we present some useful properties of the ELL supervisor. These properties are quite similar to some of those of the LLP supervisor obtained in [5]. We first discuss the properties for the non-prefix closed case.

5.1 Non-prefix Closed Case:

The first proposition of this section establishes the expected result that a larger lookahead results in a less restrictive supervision. A similar result was first presented in [5, Theorem 4.5] in context of LLP supervision.

Proposition 1 (Monotonicity) For $N \geq 0$, $L(P, \alpha^N) \subseteq L(P, \alpha^{N+1})$.

Proof: It suffices to show that for each $s \in \Sigma^*$, $\alpha^N(s) \cap \Sigma_{L(P)}(s) \subseteq \alpha^{N+1}(s) \cap \Sigma_{L(P)}(s)$. From the first part of Lemma 4 we have

$$g^N(s) \subseteq g^{N+1}(s),$$

which implies

$$[pr(g^N(s)) \cap \Sigma] \cup \Sigma_u \subseteq [pr(g^{N+1}(s)) \cap \Sigma] \cup \Sigma_u.$$

Hence it follows from the definition of $\alpha^N(\cdot)$ that

$$\alpha^N(s) \subseteq \alpha^{N+1}(s),$$

which implies

$$\alpha^N(s) \cap \Sigma_{L(P)}(s) \subseteq \alpha^{N+1}(s) \cap \Sigma_{L(P)}(s).$$

This completes the proof. ■

Proposition 1 establishes a monotonicity property of the controlled plant generated language under increasing length of lookahead. However, it does not provide any clue to the range in which the controlled plant generated language under ELL supervision lies. By definition, the controlled plant generated language under ELL supervision is non-empty, i.e., it is bounded below by the empty set. The next proposition establishes another expected result that if a minimally restrictive supervisor exists (namely, if $supRC(K, P) \neq \emptyset$), then the controlled plant generated language under ELL supervision is bounded above by the controlled plant generated language under minimally restrictive supervision. In fact it proves that the existence of a minimally restrictive supervisor is equivalent to the satisfaction of such a bound. This result is similar to the one derived in [5, Theorem 4.4] in context of LLP supervision.

Proposition 2 For $N \geq 0$, $supRC(K, P) \neq \emptyset$ if and only if $L(P, \alpha^N) \subseteq pr(supRC(K, P))$.

Proof: In order to see sufficiency, suppose $L(P, \alpha^N) \subseteq pr(supRC(K, P))$. By definition, $\epsilon \in L(P, \alpha^N)$. Hence it follows from the assumption that $\epsilon \in pr(supRC(K, P))$. This implies $supRC(K, P) \neq \emptyset$.

Conversely, suppose $supRC(K, P) \neq \emptyset$. We need to show that if $s \in L(P, \alpha^N)$, then $s \in pr(supRC(K, P))$. We use induction on the length of s to prove this. If $|s| = 0$, then $s = \epsilon$. By definition, $\epsilon \in L(P, \alpha^N)$; and $supRC(K, P) \neq \emptyset$ implies $\epsilon \in pr(supRC(K, P))$. Hence we trivially have the base step.

In order to prove the induction step, let $s = \bar{s}\sigma$, where $\sigma \in \Sigma$. Since $L(P, \alpha^N)$ is prefix closed, $s \in L(P, \alpha^N)$ implies $\bar{s} \in L(P, \alpha^N)$. Hence by the induction hypothesis $\bar{s} \in pr(supRC(K, P))$. Also, since $s = \bar{s}\sigma \in L(P, \alpha^N)$, we have $\sigma \in \alpha^N(\bar{s}) \cap \Sigma_{L(P)}(\bar{s}) = [pr(g^N(\bar{s})) \cap \Sigma_{L(P)}(\bar{s})] \cup [\Sigma_u \cap \Sigma_{L(P)}(\bar{s})]$.

Let us suppose $\sigma \in \Sigma_u \cap \Sigma_{L(P)}(\bar{s})$. Then from the controllability of $supRC(K, P)$ and the fact that $\bar{s} \in pr(supRC(K, P))$, it follows that $s = \bar{s}\sigma \in pr(supRC(K, P))$, as desired. It remains to show that if $\sigma \in pr(g^N(\bar{s})) \cap \Sigma_{(L(P)}(\bar{s})$, then $\sigma \in pr(supRC(K, P))\backslash\bar{s}$, as this is equivalent to $s = \bar{s}\sigma \in pr(supRC(K, P))$. We have

$$\begin{aligned}
& pr(g^N(\bar{s})) \cap \Sigma_{L(P)}(\bar{s}) \\
& = pr(supRC(\overbrace{[K\backslash\bar{s}]}{}^{N}, \overbrace{[P\backslash\bar{s}]}{}^{N})) \cap \Sigma_{L(P)}(\bar{s}) && \text{(definition of } g^N(\bar{s})) \\
& \subseteq pr(supRC(K\backslash\bar{s}, P\backslash\bar{s})) \cap \Sigma_{L(P)}(\bar{s}) && \text{(part 2, Lemma 4)} \\
& = pr(supRC(K, P)\backslash\bar{s}) \cap \Sigma_{L(P)}(\bar{s}) && \text{(part 2, Lemma 3, as } \bar{s} \in pr(supRC(K, P))) \\
& = pr(supRC(K, P))\backslash\bar{s} \cap \Sigma_{L(P)}(\bar{s}) && \text{(part 2, Lemma 1)}
\end{aligned}$$

This completes the proof. ■

The result of Proposition 2 is of theoretical interest as it provides an upper bound for the controlled plant generated language when ELL supervisor is employed, under the condition for the existence of a minimally restrictive supervisor. However, due to limited lookahead it is not possible to compute $\mathit{supRC}(K, P)$; consequently, it is not possible to check for its non-emptiness. Thus the result of Proposition 2 does not bear any practical interest. It turns out that a stronger condition of the *absence of starting error* in $L(P, \alpha^N)$ can be easily verified, so that the upper bound result of Proposition 2 can be concluded whenever there is no starting error in $L(P, \alpha^N)$. The following definition similar to that given in [5] defines starting error as well as run time error.

Definition 2 We say that there is a *run time error (RTE)* in $L(P, \alpha^N)$ at trace $s \in L(P, \alpha^N)$ if $g^N(s) = \emptyset$; the RTE is said to be a *starting error (SE)* if $s = \epsilon$. If there is no RTE in $L(P, \alpha^N)$ at all traces in $L(P, \alpha^N)$, then we say that there is no RTE in $L(P, \alpha^N)$.

Clearly, the absence of SE in $L(P, \alpha^N)$ can be easily verified by testing the non-emptiness of $\mathit{supRC}(\overbrace{[K\backslash\epsilon]}^{N}, \overbrace{[P\backslash\epsilon]}^{N})$. The next proposition states that the absence of SE in $L(P, \alpha^N)$ also implies the absence of RTE in $L(P, \alpha^N)$. For these reasons, all of the results that we present below are obtained under the *single* assumption of the absence of SE in $L(P, \alpha^N)$. We first show that the absence of SE in $L(P, \alpha^N)$ is a stronger condition than the non-emptiness of $\mathit{supRC}(K, P)$. A similar result was first proved in the context of LLP supervision in [5, Lemma 3.5].

Lemma 6 For $N \geq 0$, if there is no SE in $L(P, \alpha^N)$, then $\mathit{supRC}(K, P) \neq \emptyset$.

Proof: If there is no SE in $L(P, \alpha^N)$, then by definition $g^N(\epsilon) \neq \emptyset$. This is equivalent to $\mathit{supRC}(\overbrace{[K\backslash\epsilon]}^{N}, \overbrace{[P\backslash\epsilon]}^{N}) \neq \emptyset$. By the second part of Lemma 4, this implies that $\mathit{supRC}(K\backslash\epsilon, P\backslash\epsilon) = \mathit{supRC}(K, P) \neq \emptyset$, which completes the proof. ■

The following theorem follows in a straightforward manner from Proposition 2 and Lemma 6, and provides an upper bound for the controlled plant generated language when ELL supervisor is employed, under the assumption of the absence of SE in $L(P, \alpha^N)$. It is similar to [5, Corollary 4.2] obtained in context of LLP supervision.

Theorem 3 For $N \geq 0$, if there is no SE in $L(P, \alpha^N)$, then $L(P, \alpha^N) \subseteq \mathit{pr}(\mathit{supRC}(K, P))$.

Proof: If there is no SE in $L(P, \alpha^N)$, it follows from Lemma 6 that $\mathit{supRC}(K, P) \neq \emptyset$. Hence from Proposition 2 we have that $L(P, \alpha^N) \subseteq \mathit{pr}(\mathit{supRC}(K, P))$. ■

In the next proposition we establish a useful consequence of the absence of SE in $L(P, \alpha^N)$, namely, the absence of RTE in $L(P, \alpha^N)$. A similar result was first proved in [5, Theorem 4.6] in context of LLP supervision.

Proposition 3 For $N \geq 0$, if there is no SE in $L(P, \alpha^N)$, then there is no RTE in $L(P, \alpha^N)$.

Proof: We need to show that if there is no SE in $L(P, \alpha^N)$, then there is no RTE for all s in $L(P, \alpha^N)$. We use induction on the length of trace s. If $|s| = 0$, then no SE in $L(P, \alpha^N)$ implies no RTE at s. This establishes the base step.

In order to prove the induction step, let $s = \bar{s}\sigma \in L(P, \alpha^N)$, where $\sigma \in \Sigma$. Since $L(P, \alpha^N)$ is prefix closed, this implies that $\bar{s} \in L(P, \alpha^N)$. Hence it follows from the induction hypothesis that there is no RTE at $\bar{s}$, i.e., $supRC(\overbrace{[K\backslash\bar{s}]}{}^{N}, \overbrace{[P\backslash\bar{s}]}{}^{N}) \neq \emptyset$, which implies that $\epsilon \in pr(supRC(\overbrace{[K\backslash\bar{s}]}{}^{N}, \overbrace{[P\backslash\bar{s}]}{}^{N}))$. Hence it follows from the controllability of $supRC(\overbrace{[K\backslash\bar{s}]}{}^{N}, \overbrace{[P\backslash\bar{s}]}{}^{N})$ that

$$\begin{aligned}\Sigma_u \cap \Sigma_{L(P)}(\bar{s}) &\subseteq pr(supRC(\overbrace{[K\backslash\bar{s}]}{}^{N}, \overbrace{[P\backslash\bar{s}]}{}^{N})) \cap \Sigma_{L(P)}(\bar{s}) \\ &= pr(g^N(\bar{s})) \cap \Sigma_{L(P)}(\bar{s}). \end{aligned} \tag{3}$$

Since $\sigma \in L(P, \alpha^N)\backslash\bar{s}$, we have that

$$\begin{aligned}\sigma \in \alpha^N(\bar{s}) \cap \Sigma_{L(P)}(\bar{s}) &= [[pr(g^N(\bar{s})) \cap \Sigma] \cup \Sigma_u] \cap \Sigma_{L(P)}(\bar{s})] \\ &= [pr(g^N(\bar{s})) \cap \Sigma_{L(P)}(\bar{s})] \cup [\Sigma_u \cap \Sigma_{L(P)}(\bar{s})] \\ &= [pr(g^N(\bar{s})) \cap \Sigma_{L(P)}(\bar{s})],\end{aligned}$$

where the last equality follows from (3).

Thus we have $\sigma \in pr(g^N(\bar{s})) \cap \Sigma_{L(P)}(\bar{s}) = pr(supRC(\overbrace{[K\backslash\bar{s}]}{}^{N}, \overbrace{[P\backslash\bar{s}]}{}^{N})) \cap \Sigma_{L(P)}(\bar{s})$, which implies that $[supRC(\overbrace{[K\backslash\bar{s}]}{}^{N}, \overbrace{[P\backslash\bar{s}]}{}^{N})]\backslash\sigma \neq \emptyset$. Hence it follows from the third part of Lemma 4 that $supRC(\overbrace{[K\backslash\bar{s}\sigma]}{}^{N}, \overbrace{[P\backslash\bar{s}\sigma]}{}^{N}) \neq \emptyset$, i.e., there is no RTE at $s = \bar{s}\sigma$. ■

The following result was proved in the course of the proof of Proposition 3.

Corollary 2 For $N \geq 0$, if there is no RTE at $\bar{s} \in L(P, \alpha^N)$, then $\alpha^N(\bar{s}) \cap \Sigma_{L(P)}(\bar{s}) = pr(g^N(\bar{s})) \cap \Sigma_{L(P)}(\bar{s})$.

The following corollary can be obtained in a straightforward manner by combining the results of Proposition 3 and Corollary 2:

Corollary 3 For $N \geq 0$, if there is no SE in $L(P, \alpha^N)$, then for each $s \in L(P, \alpha^N)$, $\alpha^N(s) \cap \Sigma_{L(P)}(s) = pr(g^N(s)) \cap \Sigma_{L(P)}(s)$.

Using the results of Proposition 3 and Corollary 3 we establish in the following theorem that the ELL supervisor is non-blocking.

Theorem 4 For $N \geq 0$, if there is no SE in $L(P, \alpha^N)$, then $L(P, \alpha^N) = pr(L_m(P, \alpha^N))$.

Proof: Since $pr(L_m(P, \alpha^N)) \subseteq L(P, \alpha^N)$, we only need to show that if $s \in L(P, \alpha^N)$, then $s \in pr(L_m(P, \alpha^N))$. We use induction on the length of s to prove this. If $|s| = 0$, then $s = \epsilon$. Since there is no SE in $L(P, \alpha^N)$, $supRC(\overbrace{[K\backslash\epsilon]}{}^{N}, \overbrace{[P\backslash\epsilon]}{}^{N}) \neq \emptyset$, i.e., there exists a trace $t \in supRC(\overbrace{[K\backslash\epsilon]}{}^{N}, \overbrace{[P\backslash\epsilon]}{}^{N})$. However,

$$\begin{aligned}
& supRC(\overbrace{[K\backslash\epsilon]}{}^{N}, \overbrace{[P\backslash\epsilon]}{}^{N}) & \\
& \subseteq supRC(K\backslash\epsilon, P\backslash\epsilon) & \text{(part 2, Lemma 4)} \\
& = supRC(K, P) & \text{(definition of } (\cdot)\backslash\epsilon\text{)} \\
& \subseteq K & \text{(definition of } supRC(K,P)\text{)} \\
& \subseteq L_m(P) & (K = K_{legal} \cap L_m(P))
\end{aligned}$$

Then, we have that $t \in L_m(P)$. Thus it suffices to show that $t \in L(P, \alpha^N)$, so that $t \in L(P, \alpha^N) \cap L_m(P) = L_m(P, \alpha^N)$, implying that $\epsilon \in pr(L_m(P, \alpha^N))$, and thus establishing the base step. If $t = \epsilon$, then clearly, $t \in L(P, \alpha^N)$. Suppose $t \neq \epsilon$; and suppose for contradiction that there exists $t' < t$ and $\sigma \in \Sigma$ such that $t'\sigma \leq t$, $t' \in L(P, \alpha^N)$ and $t'\sigma \notin L(P, \alpha^N)$, i.e.,

$$\sigma \notin pr(supRC(\overbrace{[K\backslash t']}{}^{N}, \overbrace{[P\backslash t']}{}^{N})). \tag{4}$$

On the other hand, since $t \in supRC(\overbrace{[K\backslash\epsilon]}{}^{N}, \overbrace{[P\backslash\epsilon]}{}^{N})$, we have that

$$\sigma \in pr(supRC(\overbrace{[K\backslash\epsilon]}{}^{N}, \overbrace{[P\backslash\epsilon]}{}^{N})\backslash t'). \tag{5}$$

From Corollary 1 we have

$$supRC(\overbrace{[K\backslash\epsilon]}{}^{N}, \overbrace{[P\backslash\epsilon]}{}^{N})\backslash t' \subseteq supRC(\overbrace{[K\backslash t']}{}^{N}, \overbrace{[P\backslash t']}{}^{N}).$$

So (5) implies

$$\sigma \in pr(supRC(\overbrace{[K\backslash t']}{}^{N}, \overbrace{[P\backslash t']}{}^{N})),$$

which is contradictory to (4).

In order to prove the induction step, let $s = \bar{s}\sigma$, where $\sigma \in \Sigma$. Since $s \in L(P, \alpha^N)$, and $L(P, \alpha^N)$ is prefix closed, it follows that $\bar{s} \in L(P, \alpha^N)$. Hence from induction hypothesis we obtain that $\bar{s} \in pr(L_m(P, \alpha^N))$. Thus it suffices to show that

$$\begin{aligned}
& \sigma \in pr(L_m(P, \alpha^N))\backslash\bar{s} & \\
& = pr(L_m(P, \alpha^N)\backslash\bar{s}) & \text{(part 2, Lemma 1)} \\
& = pr([L(P, \alpha^N) \cap L_m(P)]\backslash\bar{s}) & \text{(definition of } L_m(P, \alpha^N)\text{)} \\
& = pr(L(P, \alpha^N)\backslash\bar{s} \cap L_m(P)\backslash\bar{s}) & \text{(part 1, Lemma 1)}
\end{aligned}$$

In other words, it suffices to show that there exists a string $\bar{t} \in \Sigma^*$ such that $t := \sigma\bar{t} \in L(P, \alpha^N)\backslash\bar{s} \cap L_m(P)\backslash\bar{s}$. Since there is no SE in $L(P, \alpha^N)$, it follows from Proposition 3 that there is no RTE at $\bar{s}$ in $L(P, \alpha^N)$. Hence it follows from Corollary 3 that $\sigma \in pr(g^N(\bar{s})) \cap \Sigma_{L(P)}(\bar{s}) \subseteq pr(supRC(\overbrace{[K\backslash\bar{s}]}{}^{N}, \overbrace{[P\backslash\bar{s}]}{}^{N}))$. This implies that there exists a trace $\bar{t} \in \Sigma^*$ such that $t := \sigma\bar{t} \in supRC(\overbrace{[K\backslash\bar{s}]}{}^{N}, \overbrace{[P\backslash\bar{s}]}{}^{N}) \subseteq L_m(P)\backslash\bar{s}$. It can be proved in a manner analogous to the proof of the base step that $t \in L(P, \alpha^N)\backslash\bar{s}$, which establishes the induction step and completes the proof. ■

5.2 Prefix Closed Case:

The theorem below of this section describes the important properties of the ELL supervisor in the prefix-closed case. The first part of the theorem is analog of Proposition 2, and the second part is the analog of Proposition 1. Note that since in the prefix closed case each trace is marked, the non-blockingness of supervisor trivially holds and so there is no analog of Theorem 4 in this section.

Theorem 5 The following hold for $N \geq 0$:

1. $supC(K, P) \neq \emptyset$ if and only if $L(P, \alpha^N) \subseteq supC(K, P)$.

2. $L(P, \alpha^N) \subseteq L(P, \alpha^{N+1})$.

Proof: In the first assertion, the sufficiency is obvious. The necessity can be shown using induction on the length of traces as follows: Since $supC(K, P) \neq \emptyset$, and since it is prefix closed (since K is prefix closed), $\epsilon \in supC(K, P)$, which establishes the base step. For the induction step consider $s\sigma \in L(P, \alpha^N)$, where $\sigma \in \Sigma$. Then $s \in L(P, \alpha^N)$ and $\sigma \in supC(\overbrace{[K\backslash s]}{}^{N}, \overbrace{[P\backslash s]}{}^{N}) \cup [\Sigma_u \cap \Sigma_{L(P)}(s)]$. From induction hypothesis $s \in supC(K, P)$. Hence if $\sigma \in [\Sigma_u \cap \Sigma_{L(P)}(s)]$, the from the controllability of $supC(K, P)$, $s\sigma \in supC(K, P)$. Otherwise (when $\sigma \notin [\Sigma_u \cap \Sigma_{L(P)}(s)]$), suppose for contradiction that $\sigma \notin supC(K, P)\backslash s$. Then there exists a sequence of uncontrollable events $u \in \Sigma_u^*$ such that $\sigma u \in L(P)\backslash s - K\backslash s = L(P)\backslash s - K_{legal}\backslash s$. Since $L(P)\backslash s \subseteq \overbrace{[L(P)\backslash s]}{}^{N}$ and $\overbrace{[K\backslash s]}{}^{N} \subseteq K_{legal}\backslash s$, it follows that

$$L(P)\backslash s - K\backslash s \subseteq \overbrace{[L(P)\backslash s]}{}^{N} - \overbrace{[K\backslash s]}{}^{N}.$$

So we have that $\sigma u \in \overbrace{[L(P)\backslash s]}{}^{N} - \overbrace{[K\backslash s]}{}^{N}$, which is contradictory to the fact that $\sigma \in supC(\overbrace{[K\backslash s]}{}^{N}, \overbrace{[P\backslash s]}{}^{N})$.

We prove the second assertion also by induction on the length of traces. By definition $\epsilon \in L(P, \alpha^N) \cap L(P, \alpha^{N+1})$, which proves the base step. For induction step, consider $s\sigma \in L(P, \alpha^N)$ with $\sigma \in \Sigma$. Then $s \in L(P, \alpha^N)$, and so from induction hypothesis $s \in L(P, \alpha^{N+1})$. If $\sigma \in \Sigma_u \cap \Sigma_{L(P)}(s)$, then by definition $\sigma \in L(P, \alpha^{N+1})\backslash s$. Otherwise (if $\sigma \notin \Sigma_u$) $\sigma \in L(P, \alpha^N)\backslash s]$ if and only if

$$\sigma\Sigma_u^* \cap \overbrace{[L(P)\backslash s]}{}^{N} \subseteq \overbrace{[K\backslash s]}{}^{N}, \tag{6}$$

and suffices to show that $\sigma\Sigma_u^* \cap \overbrace{[L(P)\backslash s]}{}^{N+1} \subseteq \overbrace{[K\backslash s]}{}^{N+1}$. We have

$$\begin{aligned}
&\sigma\Sigma_u^* \cap \overbrace{[L(P)\backslash s]}{}^{N+1} \\
&\subseteq \sigma\Sigma_u^* \cap \overbrace{[P\backslash s]}{}^{N} && (\overbrace{[P\backslash s]}{}^{N+1} \subseteq \overbrace{[L(P)\backslash s]}{}^{N}) \\
&\subseteq \overbrace{[K\backslash s]}{}^{N} && \text{(from 6)} \\
&= \overbrace{[L(P)\backslash s]}{}^{N} \cap K_{legal}\backslash s.
\end{aligned}$$

By intersecting each term in the above series of containments with $\overbrace{[L(P)\backslash s]}^{N+1}$, we obtain

$$\sigma\Sigma_u^* \cap \overbrace{[L(P)\backslash s]}^{N+1} \subseteq \overbrace{[L(P)\backslash s]}^{N+1} \cap K_{legal}\backslash s = \overbrace{[K\backslash s]}^{N+1},$$

as desired. ■

6 Valid and Non-Blocking ELL Supervisor

It follows from Proposition 2 and first part of Theorem 5 that a ELL supervisor is in general not maximally permissive. It is useful to determine any condition under which such a property (introduced as validity in [5]) will hold. In this section we obtain a sufficient condition on the length of lookahead for the ELL supervisor to be valid and non-blocking, so that the controlled plant generated language under ELL supervision equals the controlled plant generated language under minimally restrictive supervision.

Definition 3 A ELL supervisor with control policy $\alpha^N : L(P) \rightarrow 2^\Sigma$ is called *valid* and *non-blocking* if $L(P, \alpha^N) = pr(supRC(K, P))$. (This reduces to $L(P, \alpha^N) = supC(K, P)$ in the prefix closed case.)

One should note that for a valid and non-blocking ELL supervisor, we have

$$\begin{aligned} L_m(P, \alpha^N) &:= L(P, \alpha^N) \cap L_m(P) \\ &= pr(supRC(K, P)) \cap L_m(P) \\ &= supRC(K, P), \end{aligned}$$

where the last equality follows from the fact that $supRC(K, P)$ is relative closed with respect to P. This justifies the name non-blocking in Definition 3. For a ELL supervisor to be valid and non-blocking it should have sufficient lookahead to determine the existence or non-existence of all infimal controllable and relative-closed sublanguage of the "post-behavior" (at every prefix of the desired behavior). This requires a lookahead window in which all the infimal controllable and relative-closed sublanguages of the post-behavior are present. So a lookahead window in which all the maximal length traces in the union of all the infimal controllable and relative-closed superlanguages of the post-behavior are present would suffice. We call such traces to be the "neighboring frontier traces" borrowing the terminology first introduced in [5]. These traces have the property that it is possible to terminate execution at these traces without getting blocked, and these are the shortest such traces. It is expected that if the number of steps of lookahead is larger than the longest neighboring frontier trace, then the ELL supervisor will be valid and non-blocking.

Definition 4 Given $s \in pr(K)$, the set of *frontier* traces in K after the trace s, denoted $(K\backslash s)_f \subseteq K\backslash s$, is defined as:

$$(K\backslash s)_f := \{t \in K\backslash s \mid \forall \sigma \in \Sigma : [t\sigma \in L(P)\backslash s] \Rightarrow [\sigma \notin \Sigma_u]\}.$$

K_f is used to denoted $(K\backslash\epsilon)_f$. The set of *neighboring frontier* traces in K after the trace s, denoted $(K\backslash s)_{nf} \subseteq (K\backslash s)_f$, is defined as:

$$(K\backslash s)_{nf} := \{t \in (K\backslash s)_f \mid |t| \geq 1, \text{ and } \forall t' < t : t' \notin (K\backslash s)_f\}.$$

The length of the longest neighboring frontier trace, denoted N_{nf}, as:

$$N_{nf} := \begin{cases} \max_{s\in pr(K)} \left\{\max_{t\in(K\backslash s)_{nf}} |t|\right\} & \text{if it exists} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Remark 5 It is clear that in the prefix closed case N_{nf} simply equals the maximum number of consecutive uncontrollable events possible in the specification language. $\diamondsuit$

Note that N_{nf} is undefined when $(K\backslash s)_{nf} = \emptyset$ for some $s \in pr(K)$. We first prove the following lemma.

Lemma 7 Suppose $s \in pr(supRC(K,P))$, $\sigma \in \Sigma$, and N_{nf} is defined. Then the following are equivalent:

1. $\sigma \in pr(supRC(K,P)\backslash s)$.
2. There exists a nonempty $H \in RC(K\backslash s\sigma, P\backslash s\sigma)$.
3. $(K\backslash s\sigma)_{nf} \neq \emptyset$.

Proof: The equivalence of the first two assertions generalizes [9, Theorem 1], and can be proved analogously. In order to show the equivalence of the last two assertions, we first show that the last assertion implies the second one. Define $H := pr[(K\backslash s\sigma)_{nf}] \cap K\backslash s\sigma$, which is the set of marked prefixes of $(K\backslash s\sigma)_{nf}$. Then it is easy to see that $H \in RC(K\backslash s\sigma, P\backslash s\sigma)$, and its nonemptiness follows from the last assertion. On the other hand, suppose there exists a nonempty $H \in RC(K\backslash s\sigma, P\backslash s\sigma)$. Pick a frontier trace $t \in H$ (which exists since H is nonempty and N_{nf} is defined). So there exists a prefix of $t' \leq t$ such that $t' \in (K\backslash s\sigma)_{nf}$, showing that it is nonempty. ■

In the following theorem we give a condition for validity and non-blockingness of a ELL supervisor. It generalizes a similar result first proved in [5, Theorem 5.5] in context of LLP supervision.

Theorem 6 For $N \geq 0$, if there is no SE in $L(P,\alpha^N)$ and $N \geq N_{nf}+2$, then $L(P,\alpha^N) = pr(supRC(K,P))$.

Proof: The forward containment follows from Theorem 3. We prove the reverse containment, i.e., $pr(supRC(K,P)) \subseteq L(P,\alpha^N)$, using induction on the length of traces. Since $\epsilon \in L(P,\alpha^N)$, the base step holds. For induction step, consider a trace $s\sigma \in pr(supRC(K,P))$, where $\sigma \in \Sigma$. Then $s \in pr(supRC(K,P))$; so from induction hypothesis $s \in L(P,\alpha^N)$. It suffices to show that $\sigma \in pr(supRC(\overbrace{[K\backslash s]}^{N}, \overbrace{[P\backslash s]}^{N}))$, which we know is nonempty since there is no SE, and as a result no RTE.

Since $\sigma \in pr(supRC(K,P)\backslash s)$, it follows from Lemma 7 that $(K\backslash s\sigma)_{nf} \neq \emptyset$. Also, since $N \geq N_{nf}+2$, $\sigma(K\backslash s\sigma)_{nf} \subseteq \overbrace{[K\backslash s]}^{N}$. So the controllability and relative-closure of the set of marked prefixes of $(K\backslash s\sigma)_{nf}$, absence of RTE, and the fact that $N \geq N_{nf}+2$ together imply $\sigma(K\backslash s\sigma)_{nf} \subseteq pr(supRC(\overbrace{[K\backslash s]}^{N}, \overbrace{[P\backslash s]}^{N})$. This implies $\sigma \in pr(supRC(\overbrace{[K\backslash s]}^{N}, \overbrace{[P\backslash s]}^{N}))$, as desired. ∎

Remark 6 The bound on the length of lookahead in Theorem 6 can be improved from $N \geq N_{nf}+2$ to $N \geq N_{nf}+1$ by noting the fact that no uncontrollable events are feasible at the frontier traces, which can be used to refine the estimate of the plant language by intersecting it with the set $(\Sigma_c^*.\Sigma_u^{\leq N_{nf}}.\Sigma_c^*)^*$ as in Remark 1.

Under the assumption of absence of starting error, Theorem 6 gives a sufficient condition for the validity and non-blockingness of the ELL supervision. It can be shown in an analogous manner that the same result can also be obtained under a weaker condition of existence of a minimally restrictive supervision, i.e., under the condition of $supRC(K,P) \neq \emptyset$. However, as discussed above, due to limited lookahead it is not possible to compute $supRC(K,P)$, hence it is not possible to verify its non-emptiness. $\diamondsuit$

7 Application to Concurrency Control

This section presents an application of the ELL supervisor in concurrency control of transactions in a simple database management system (DBMS). Modeling of transaction execution in database systems using discrete event framework has been studied in [18] for the untimed issues and in [8] for the real-time issues. Both these work assume *complete-information structure* for concurrency control. The use of limited lookahead for concurrency control in the untimed setting has been considered very informally in [12, Section 5]. We provide a formal treatment here via an example, and construct a *ELL supervisor* for controlling transaction execution. The example is worked out in detail to illustrate the application of the ELL supervisor. We have made some simplifying assumptions so that we are able to focus on the main issues.

In the following, we introduce some terminologies for concurrency control of transaction execution in DBMS; interested readers may refer to [21, 22] for details. A *transaction* is defined to be a sequence of *read* and *write* operations on the data items of the database. The additional operation, called *commit* operation, is used to signify successful termination of the transaction. A transaction that is not committed is called *active*. Let X denote the set of data items of the database. For each $i \in \mathcal{N}$ and $x \in X$, notations $r_i(x), w_i(x), o_i(x)$, and c_i are used to denote read operation on data item x, write operation on data item x, any operation (read or write) on data item x, and commit operation, respectively, of transaction T_i. We impose the natural requirement that each T_i can only read and write at most once per data item, i.e., no multiple reads and writes on the same data item for each T_i is allowed, and no operation of T_i follows its commit operation c_i.

Given a transaction T_i, $L(T_i)$ is the language consisting of all prefixes of the sequence of operations from T_i. For example, if $T_i = r_i(a)w_i(a)c_i$, then $L(T_i) = pr[r_i(a)w_i(a)c_i]$. Letting P denote the DBMS, its generated language, denoted $L(P)$, is defined to be:

$$L(P) = \|_{i=1}^{N_T} L(T_i),$$

where the non-negative integer $N_T \in \mathcal{N}$ represents the maximum number of active transactions allowed in the DBMS, and the $\|$ operator denotes the interleaving of languages [16]. The value of N_T may be dictated by limitations of the DBMS in terms of processing power or the amount of memory available. Thus the event set of DBMS P is given by:

$$\Sigma = \Sigma_c = \bigcup_{i \leq N_T, x \in X} \{r_i(x), w_i(x), c_i\}.$$

Each member of $L(P)$ is called a schedule; i.e., a schedule is a sequence of operations obtained by interleaving operations from one or more T_i. A schedule is called *complete* if there are no active transactions in it. The set of complete schedules denoted $L_{complete}$ is defined to be:

$$L_{complete} := \{s \in \Sigma^* \mid o_i \text{ in } s \text{ implies } c_i \text{ also in } s\}.$$

Consider for example

$$T_1 = w_1(a)c_1; \quad T_2 = r_2(a)w_2(a)c_2, \tag{7}$$

where a is a data item in the database. Also, consider the following three sequences of operations:

$$\begin{aligned} s_1 &= w_1(a)r_2(a)c_1w_2(a)c_2, \\ s_2 &= w_1(a)r_2(a)c_1, \\ s_3 &= w_1(a)w_2(a)c_1r_2(a)c_2. \end{aligned}$$

s_1 is a complete schedule; s_2 is only a schedule but not complete since T_2 is not committed; s_3, on the other hand, is complete but not a schedule, as the order of operations $r_2(a)$ and $w_2(a)$ in s_3 is not the same as that in the transaction T_2.

For each $i, j \leq N_T, i \neq j$ and $x \in X$, a pair of operations $(o_i(x), o_j(x))$ of a schedule $s \in L(P)$ such that at least one of the operations is a write operation, is said to be a *conflicting pair* of operations of schedule s. A schedule s is called *serializable* if all its conflicting pairs are *consistently ordered*, i.e., the graph representing the executional precedence of conflicting operations should be acyclic. The set of serializable schedules defines the legal behavior of the DBMS. Hence we define:

$$K_{legal} = \{s \in \Sigma^* \mid s \text{ is serializable}\}.$$

The control objective of a concurrency controller is to ensure that only serializable schedule occur in the DBMS. Note that $L_{complete}$ and K_{legal} may contain traces that are not physically possible schedules.

Since transactions start and terminate within the DBMS, it is not known *a priori* how many transactions are involved in the interleaving language nor which T_i's are involved. This is an example of a time-varying discrete event system. The conventional supervisory control approach is not applicable here because a plant P that models *all* possible future behavior cannot feasibly be constructed.

For the purpose of illustrating the application of ELL supervisor, we consider the situation where $N_T = 2$. We assume for simplicity that there is only one data item a. The set of events consists of all read $r_i(a)$ and write $w_i(a)$ operations on data item a and the commit operations c_i of the transaction T_i within the system. Thus, in our case we have

$$\Sigma = \Sigma_c = \{r_1(a), w_1(a), r_2(a), w_2(a), c_1, c_2\}.$$

Suppose the two transactions given in (7) simultaneously enter the database system. (These transactions are fixed but not known to the ELL supervisor.) Since $N_T = 2$, no new transaction is allowed to enter the system until at least one of the existing ones terminates. Suppose for simplicity that no new transaction enters the system until both the existing transactions terminate. For notational simplicity, we omit the data item a in all the operations (e.g., $r_1(a)$ is denoted as r_1). The generated language $L(P)$, which is the set of all possible schedules of T_1 and T_2, is depicted in Figure 3(a). The dark node is a *marked state* which corresponds to complete schedules.

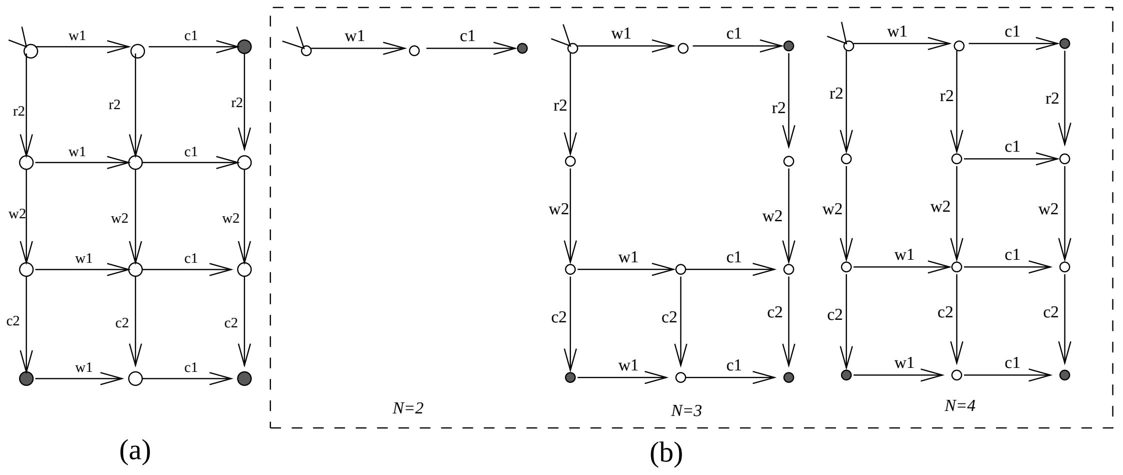

Figure 3: DBMS example: uncontrolled plant; controlled plant with increasing lookahead

The relevant portion of the language $L_{complete}$ is shown in Figure 4(a). Similarly, the graphical representation of the relevant portion of K_{legal} is depicted in Figure 4(b). In the graph of K_{legal}, all states are marked. For simplicity, we assume that the commit operation c_i performs self-loops around all nodes as it does not violate the criterion of serializability.

When the length of lookahead $N = 0, 1$, a starting error happens in $L(P, \alpha^N)$. Since all events are controllable, we have $L(P, \alpha^0) = L(P, \alpha^1) = \{\epsilon\}$. There is no starting error for $N \geq 2$. The generated languages of the closed-loop controlled system $L(P, \alpha^N)$ (for $N = 2, 3,$

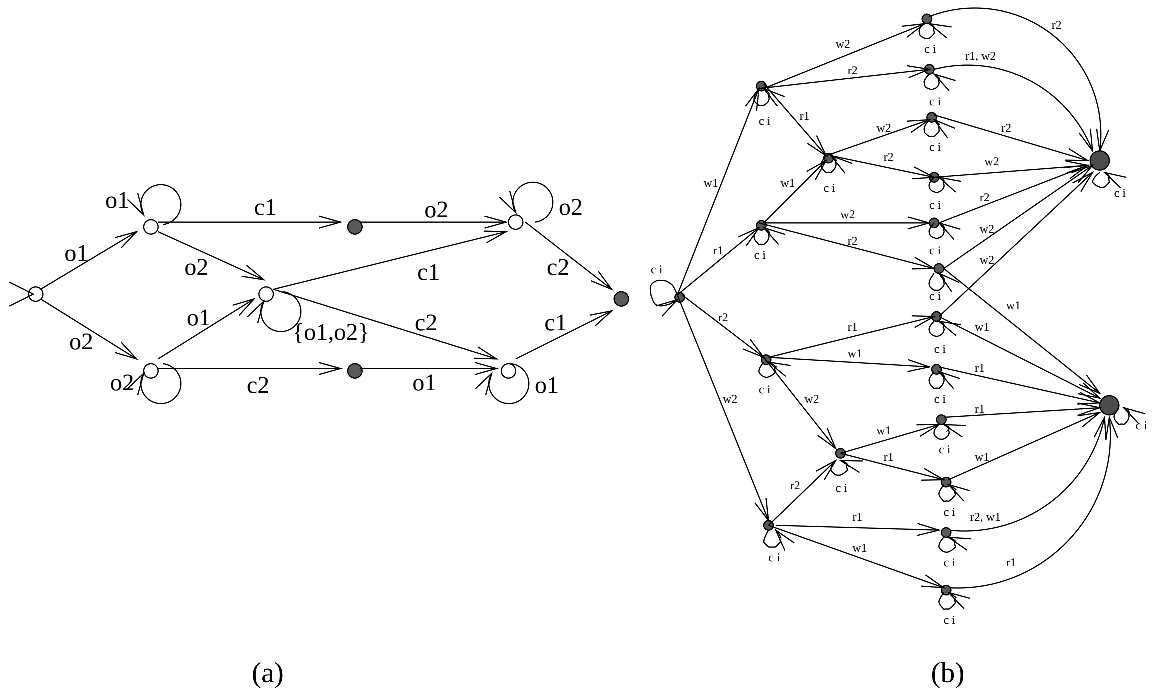

Figure 4: $L_{complete}$ and K_{legal} for the simple DBMS

and 4) are depicted in Figure 3(b). The closed-loop generated behavior $L(P, \alpha^4)$ is the set of all complete and serializable schedules, and thus equals the supremal relative closed and controllable sublanguage of the desired behavior. Therefore, the ELL supervisor with 4-step lookahead is valid and non-blocking in our example.

8 Conclusion and Discussion

In this paper, we have presented a new approach for extension based limited lookahead supervisor. The specific contributions of our work include the following:

1. The ELL supervisor avoids the need to choose an "attitude" regarding pending traces, which is required in LLP supervisor. This results in a unique choice for the supervisor.

2. The ELL supervisor is compared with the conservative LLP supervisor; it is shown that ELL supervisor is in general less restrictive than the conservative LLP supervisor.

3. The ELL supervisor is non-blocking even if the desired behavior is not a relative closed language.

4. A lower bound for N has been obtained which guarantees in the absence of starting error our on-line scheme performs as well as the traditional off-line scheme.

The complexity of computing control action at each point in the non-prefix closed case (which is that of computing the supremal relative closed and controllable sublanguage of the estimated desired marked language with respect to the estimated plant behavior) is of the *same order* as that of computing the control action at each point for the LLP supervisor. This is due to the facts that (i) the number of states in the estimate of the generated plant language is same as that in its N-step truncation, since appending Σ^* to the traces of length N is equivalent to adding self-loops on each event at each of the leaf nodes in the N-tree representing the N-step truncation; and (ii) the complexity of computing a supremal relative closed and controllable sublanguage is of the *same order* as that of computing a supremal controllable sublanguage. In the prefix-closed case, the computational complexity is also affected by the complexity of K_{legal}.

Finally, there are many possible ways that this work presented here could be extended:

1. As in [6, 7, 9], it is important to develop algorithm to perform recursive computation of $supRC(\cdot, \cdot)$ from one N-level tree to another as the limited lookahead windows roll through the execution of each event.

2. For some application areas, the representations of K_{legal} and $L_{complete}$ may be too complex. One possible solution to this implementation issue is that instead of taking set intersection to compute $\overbrace{[K\backslash s]}^{N}$, it is possible to perform and implement a legality test on $\overbrace{[P\backslash s]}^{N}$ to generate $\overbrace{[K\backslash s]}^{N}$. For instance, in the case of DBMS, *Serialization Graph Testing (SGT)* [22, 21] could be used to determine which schedules are serializable.

3. The non-blocking feature often results in restrictive control in the closed-loop behavior; thus, the generated controlled behavior, although non-blocking, may be restrictive. As noted by Chen and Lafortune [3], it is sometime better to allow some degrees of blocking if such blocking occurs very rarely, or the expense to correct such blocking situation is minimal, so that the controlled plant can achieve more of the desired behavior. For example in DBMS, rollback mechanism is used for recovery from blocking. (A rollback consists of undoing the effect of certain events until it is possible to resume the execution of the system.) Thus, it is possible to modify the ELL supervisor to allow blocking by redefining the function block $g^N(s)$ so that the ELL supervisor is even more permissive. However, appropriate mechanism for recovery from blocking must be incorporated.

4. In concurrency control of DBMS, locking and time-stamping [21, 22] are two popular concurrency control techniques used by many schedulers existing today. It is instructive to compare the performance between these two techniques with our ELL based on-line control scheme. The non-blocking requirement for ELL supervisor must be relaxed in order to make a meaningful comparison.

5. Abort operation, system failure, and rollback are also important events in DBMS. It will be useful to extend the application of ELL supervisor to include these events in the system.

A Proof of Lemma 3

Proof of Lemma 3: For notational simplicity, let $H := supRC(K, P)$, $H_1 := H\backslash s$, and $H_2 := supRC(K\backslash s, P\backslash s)$.

1. We need to show that $H_1 = H\backslash s \subseteq H_2$. Since H_2 is the supremal relative closed and controllable sublanguage of $K\backslash s$ with respect to $P\backslash s$, it suffices to show that H_1 is a relative closed and controllable sublanguage of $K\backslash s$ with respect to $P\backslash s$, i.e., (i) $H_1 \subseteq K\backslash s$, (ii) $pr(H_1) \cap L_m(P)\backslash s \subseteq H_1$, and (iii) $pr(H_1)\Sigma_u \cap L(P)\backslash s \subseteq pr(H_1)$. Since $H = supRC(K, P) \subseteq K$; clearly, $H_1 = H\backslash s \subseteq K\backslash s$. We first show that H_1 is relative closed with respect to $P\backslash s$.

$$\begin{aligned}
& pr(H_1) \cap L_m(P)\backslash s \\
&= pr(H\backslash s) \cap L_m(P)\backslash s && (H_1 = H\backslash s)\\
&= pr(H)\backslash s \cap L_m(P)\backslash s && \text{(part 2, Lemma 1)}\\
&= (pr(H) \cap L_m(P))\backslash s && \text{(part 1, Lemma 1)}\\
&\subseteq H\backslash s && (H \text{ is relative closed})\\
&= H_1 && (H_1 = H\backslash s)
\end{aligned}$$

Next we show that H_1 is controllable with respect to $P\backslash s$.

$$\begin{aligned}
& pr(H_1)\Sigma_u \cap L(P)\backslash s \\
&= pr(H\backslash s)\Sigma_u \cap L(P)\backslash s && (H_1 = H\backslash s)\\
&= (pr(H)\backslash s)\Sigma_u \cap L(P)\backslash s && \text{(part 2, Lemma 1)}\\
&\subseteq pr(H)\Sigma_u\backslash s \cap L(P)\backslash s && \text{(part 3, Lemma 1)}\\
&= (pr(H)\Sigma_u \cap L(P))\backslash s && \text{(part 1, Lemma 1)}\\
&\subseteq pr(H)\backslash s && (H \text{ is controllable})\\
&= pr(H\backslash s) && \text{(part 2, Lemma 1)}\\
&= pr(H_1) && (H_1 = H\backslash s)
\end{aligned}$$

2. The forward containment follows from part 1 above. Hence we only need to show the reverse containment, i.e., $H_1 = H\backslash s \supseteq H_2$, or equivalently, $H \supseteq \{s\}H_2$. Since H is the supremal relative closed and controllable sublanguage of K with respect to P, it suffices to show that $H \cup \{s\}H_2$ is relative closed and controllable sublanguage of K with respect to P, which implies that $H \cup \{s\}H_2 \subseteq H_1$; consequently, $\{s\}H_2 \subseteq H_1$. It is easy to see that $H \cup \{s\}H_2 \subseteq K$. We first prove that $H \cup \{s\}H_2$ is controllable with respect to P. Since $s \in pr(H)$ and H is controllable with respect to P, we have

$$pr(\{s\})\Sigma_u \cap L(P) \subseteq pr(H). \tag{8}$$

Also, since H_2 is controllable with respect to $P\backslash s$, we have

$$pr(H_2)\Sigma_u \cap L(P)\backslash s \subseteq pr(H_2),$$

or equivalently,

$$\{s\}pr(H_2)\Sigma_u \cap L(P) \subseteq \{s\}pr(H_2). \tag{9}$$

By considering the unions of left as well as right hand sides of (8) and (9) and using the fact that $pr(\{s\}) \cup \{s\}pr(H_2) = pr(\{s\}H_2)$, we obtain

$$pr(\{s\}H_2)\Sigma_u \cap L(P) \subseteq pr(H) \cup \{s\}pr(H_2). \tag{10}$$

Since H is controllable with respect to P, we have

$$pr(H)\Sigma_u \cap L(P) \subseteq pr(H). \tag{11}$$

By considering the union of left as well as right hand sides of (10) and (11) and using the fact that prefix operation distributes over the union operation, we obtain as desired:

$$pr(H \cup \{s\}H_2)\Sigma_u \cap L(P) \subseteq pr(H \cup \{s\}H_2).$$

Next we prove that $H \cup \{s\}H_2$ is relative closed with respect to P. Since $s \in pr(H)$ and H is relative closed, we have

$$pr(\{s\}) \cap L_m(P) \subseteq H. \tag{12}$$

Since H_2 is relative closed with respect to $L_m(P)\backslash s$, we have

$$pr(H_2) \cap L_m(P)\backslash s \subseteq H_2,$$

or equivalently,

$$\{s\}pr(H_2) \cap L_m(P) \subseteq \{s\}H_2. \tag{13}$$

By considering the unions of left as well as right hand sides of (12) and (13) and using the fact that $pr(\{s\}) \cup \{s\}pr(H_2) = pr(\{s\}H_2)$, we obtain

$$pr(\{s\}H_2) \cap L_m(P) \subseteq H \cup \{s\}H_2. \tag{14}$$

Since H is relative closed with respect to P, we have

$$pr(H) \cap L_m(P) \subseteq H. \tag{15}$$

By considering the union of left as well as right hand sides of (14) and (15) and using the fact that prefix operation distributes over the union operation, we obtain as desired:

$$pr(H \cup \{s\}H_2) \cap L_m(P) \subseteq H \cup \{s\}H_2.$$

This completes the proof. ■

References

[1] Z. Banaszak and B. H. Krogh. Deadlock avoidance in flexible manufacturing sytems with concurrently competing process flows. *IEEE Transactions on Robotics and Automation*, 6(6):724–734, December 1990.

[2] R. D. Brandt, V. K. Garg, R. Kumar, F. Lin, S. I. Marcus, and W. M. Wonham. Formulas for calculating supremal controllable and normal sublanguages. *Systems and Control Letters*, 15(8):111–117, 1990.

[3] E. Chen and S. Lafortune. Dealing with blocking in supervisory control of discrete event systems. *IEEE Transactions on Automatic Control*, 36(6):724–735, 1991.

[4] S. L. Chung, S. Lafortune, and F. Lin. Addendum to "Limited lookahead policies in supervisory control of discrete event systems": Proofs of technical results. Technical Report CGR-92-6, University of Michigan, Ann Arbor, Michigan, April 1992.

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

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

[7] S. L. Chung, S. Lafortune, and F. Lin. Supervisory control using variable lookahead policies. *Discrete Event Dynamic Systems: Theory and Applications*, 4(3):237–268, July 1994.

[8] A. Ghosh. Modeling and analysis of real-time database systems in the framework of discrete event systems. Technical Report MS 95-6, Institute of Systems Research, University of Maryland, 1995.

[9] N. B. Hadj-Alouane, S. Lafortune, and F. Lin. Variable lookahead supervisory control with state information. *IEEE Transactions on Automatic Control*, 39(12):2398–2410, December 1994.

[10] N. B. Hadj-Alouane, S. Lafortune, and F. Lin. Centralized and distributed algorithm for on-line synthesis of maximal control policies under partial observation. *Discrete Event Dynamical Systems: Theory and Applications*, 6(41):379–427, 1996.

[11] M. Heymann and F. Lin. On-line control of partially observed discrete event systems. *Discrete Event Dynamical Systems: Theory and Applications*, 4(3):221–236, July 1994.

[12] P. Kozak and W. M. Wonham. Synthesis of database management protocols. *IEEE Transactions on Automatic Control*, 41(9):1330–1335, September 1996.

[13] B. H. Krogh and D. Feng. Dynamic generation of subgoals for autonomous mobile robots using local feedback information. *IEEE Transactions on Automatic Control*, 34(5):483–493, May 1989.

[14] R. Kumar. Formulas for observability of discrete event dynamical systems. In *Proceedings of 1993 Conference on Information Sciences and Systems*, pages 581–586, Johns Hopkins University, Baltimore, MD, March 1993.

[15] R. Kumar and V. K. Garg. Extremal solutions of inequations over lattices with applications to supervisory control. *Theoretical Computer Science*, 148:67–92, November 1995.

[16] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems.* Kluwer Academic Publishers, Boston, MA, 1995.

[17] R. Kumar, V. K. Garg, and S. I. Marcus. On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17(3):157–168, 1991.

[18] S. Lafortune. Modeling and analysis of transaction execution in database systems. *IEEE Transactions on Automatic Control*, 33(5):439–447, 1988.

[19] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988.

[20] P. Liu. Goal-oriented behavior in autonomous systems. In *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pages 2–8, Herndon, VA, 1990.

[21] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[22] C. H. Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, Rockville, MD, 1986.

[23] K. M. Passino and P. J. Antsaklis. A system and control theoretic perspective on artificial intelligence planning systems. *International Journal of Applied Artificial Intelligence*, 3:1–32, 1989.

[24] J. C. Pemberton and R. E. Korf. Incremental path planning on graphs with cycles. In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*, pages 179–188, College Park, MD, 1992.

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

[26] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of IEEE: Special Issue on Discrete Event Systems*, 77:81–98, 1989.

[27] N. Viswanadham, Y. Narahari, and T. L. Johnson. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models. *IEEE Transactions on Robotics and Automation*, 6(6):713–723, December 1990.