A LINGUISTIC FRAMEWORK FOR CONTROLLED HIERARCHICAL DES

by

Peyman Gohari-Moghadam

A thesis submitted in conformity with the requirements for the Degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright by Peyman Gohari-Moghadam, 1998
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-34133-X
A Linguistic Framework for Controlled Hierarchical DES

A thesis submitted in conformity with the requirements for the Degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright by Peyman Gohari-Moghadam, 1998

Abstract

A language-based framework is developed for expressing the behavior of complex discrete-event systems (DES). Alphabet symbols at different levels of the linguistic hierarchy represent events observed at different levels of abstraction. Correspondingly, the DES is modeled "top-down" as a hierarchy of subsystems, and its trajectories are represented by a structured language. Within this context, a question regarding the behavior of a system can be addressed recursively, resulting in a natural decomposition of the problem into subproblems. In particular, recursive definitions are provided for language containment and controllability, and an effective recursive procedure is presented for computing a supremal controllable sublanguage.
Acknowledgments

First and foremost I would like to thank my advisor Prof. W. M. Wonham for teaching me the alphabet of research and for making numerous helpful suggestions manifested all through my thesis. Much of this work has been inspired by his thoughtful observations on system architecture in connection with the role it could play in solving the mystery of complexity.

Special thanks go to people at Systems Control Group for their friendship and creating an atmosphere conducive to innovative research.

Financial support during my studies was provided in part by University of Toronto Open Fellowship and in part by research assistantship from Prof. Wonham, which I gratefully acknowledge.

At last but certainly not the least I would like to express my deepest appreciation to my family for their love and support.
To my parents
Contents

1 Introduction 1
  1.1 RW Supervisory Control of DES .......................... 1
  1.2 Motivations and Related Works .......................... 2
  1.3 Organization of the Thesis .............................. 4

2 Structured Languages 5
  2.1 Introduction ............................................. 5
  2.2 Syntax ...................................................... 9
  2.3 Semantics .................................................. 11
  2.4 Discussion ................................................ 16
    2.4.1 Syntax and Semantics Review ......................... 16
    2.4.2 Expanding I ........................................... 19

3 Behavior Containment and Controllability 24
  3.1 Introduction .............................................. 24
  3.2 Preorder $\preceq$ on $L_k$ ............................... 28
  3.3 Join and Meet on $L_k/\sim$ ............................... 36
  3.4 Controllability ........................................... 43
  3.5 Upper Semilattice Property of the Controllable Subclass .... 52

4 Computation of the Supremal Controllable Sublanguage 57
  4.1 Introduction ............................................... 57
  4.2 Top-Level Sublanguages ................................... 58
4.3 Defining Ω: Towards Fixed-Point Characterization of the Supremal Controllable Element .................................................. 62
4.4 The Iteration Scheme .......................................................... 71
4.5 Example ................................................................. 77

5 Effective Computation: Regularity ........................................... 82
5.1 Right Congruences on \((\Sigma_k \cup \mathcal{X}_k)^*\) ....................................... 82
5.2 Example ................................................................. 85
5.3 Regularity and Finite Time Convergence ................................. 89

6 Conclusions and Future Research ........................................... 104
List of Figures

2.1 The system $S$ .......................................... 6
2.2 Level-wise specification of $S$ ............................ 6
2.3 Revised—level-wise specification of $S$ .................. 8
2.4 The system $S'$ and its new components ............... 8
2.5 The system $S$ with states labeled with tokens ........ 15
2.6 Tree representation .................................. 17
2.7 A generic reachability tree in $L_k$ ..................... 18
2.8 Restricted vs. unrestricted substitution ............... 19
2.9 Example 2: system on the left side is not expressible in $L_2$ .... 22
2.10 Example 3: system on the left side is not expressible in $L_3$ .... 22
2.11 Graphical expansion of $I(L)$ .......................... 23

3.1 Representation of $E$ and $L$—level $k$ (current level) .... 25
3.2 Representation of $x_0$ and $y_0$—level $k + 1$ .......... 27
3.3 Illustration of $E \preceq L$ .............................. 28
3.4 Language $E \in L_0$ ($p = 2$) ............................. 32
3.5 Verifying $E \preceq L$ .................................. 33
3.6 Illustration of premeet and prejoin in $L_0$ ($p = 2$, $\Sigma_0 = \{\alpha, \beta, \gamma\}$, and $\Sigma_1 = \{a, b\}$) .......... 38
3.7 $I(L_1) \cup I(L_2) \nsubseteq I(L_1 \cap L_2)$ ............... 44
3.8 Illustration of controllability .......................... 46
3.9 Controllable: $u \in E$ & $\sigma \in \Sigma_{k,u}$ & $v\sigma \in L \Rightarrow u\sigma \in E$ .......... 47
3.10 Controllable, though $u \in E$, $\sigma \in \Sigma_{k,u}$, $v\sigma \in L$ but $u\sigma \not\in E$ .... 47
3.11 Uncontrollable, despite having $(\forall \sigma \in \Sigma_{k,u}, u \in E) u \sigma \in L \Rightarrow u \sigma \in E$  

3.12 $E$ is controllable with respect to $L$  

3.13 The system $L$ and its requirement $E$  

3.14 Controllable sublanguages of $E$  

3.15 Sub-semilattice $(\mathcal{C}_L(E), \preceq)$  

4.1 $E$ is a restriction of $L$ only at level $k$ when $L_x = L_y \& \theta_x = \theta_y - \{\gamma\}$  

4.2 The system $L$, together with the projection of its requirement  

4.3 Top level sublanguage $E$, constructed based on the information in Fig 4.2  

4.4 Unsatisfiable requirement for $y_4$ affects the requirement for $L$  

4.5 Weeding out those $u' \in L_x$ which make $E$ uncontrollable at $u$  

4.6 Computing $K := \Omega(E, L)$  

4.7 Consistency at the Boundaries  

4.8 $\Omega(E, L)$, where $E$ is consistent at the boundaries  

4.9 $\Omega(K, L)$—standard vs. structured approach  

4.10 $\Sigma_{\text{illegal}}$ for $j \in \mathbb{N}$  

4.11 Problem with inconsistent sublanguages  

4.12 The system $L$ together with its requirement  

4.13 The requirement for $L$, given as a structured language  

4.14 $K_1 := \Omega(K_0, L)$  

4.15 $K_2 := \Omega(K_1, L)$ and $\Omega(K_2, L) = K_2$  

5.1 Illustration of Lemmas 5.1 and 5.2  

5.2 The language $L$ together with its recognizer $R(L)$  

5.3 Alternative recognizer for $L_i$; the state set is the quotient set of $\equiv_L$  

5.4 Recognizers for $x_i$'s  

5.5 Recognizers for $x_{ij}$'s  

5.6 Expanding $R(L)$  

5.7 The canonical recognizer of $\mathcal{T}(L)$  

5.8 State-tree structure $S$  

5.9 State-tree structure of $R(L)$  

vi
# List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma_k$</td>
<td>Set of events at level $k$</td>
<td>9</td>
</tr>
<tr>
<td>$X_k$</td>
<td>Set of tokens at level $k$</td>
<td>9</td>
</tr>
<tr>
<td>$G_k$</td>
<td>Grammar</td>
<td>10</td>
</tr>
<tr>
<td>$P_k$</td>
<td>Projection map</td>
<td>10</td>
</tr>
<tr>
<td>$\text{length}$</td>
<td>Length function</td>
<td>10</td>
</tr>
<tr>
<td>$\text{first}$</td>
<td>First symbol function</td>
<td>10</td>
</tr>
<tr>
<td>$\text{last}$</td>
<td>Last symbol function</td>
<td>10</td>
</tr>
<tr>
<td>$\mathcal{L}_k$</td>
<td>Class of $p-k$-layered structured languages</td>
<td>10</td>
</tr>
<tr>
<td>$\Sigma_{k:p}$</td>
<td>Union of alphabets at levels $k, \ldots, p$</td>
<td>12</td>
</tr>
<tr>
<td>$\mathcal{F}(\Sigma_{k:p})$</td>
<td>Class of closed sublanguages over $\Sigma_{k:p}$</td>
<td>12</td>
</tr>
<tr>
<td>$I_k$</td>
<td>Semantics function</td>
<td>12</td>
</tr>
<tr>
<td>$L_x$</td>
<td>Language part of $I(x)$</td>
<td>12</td>
</tr>
<tr>
<td>$\theta_x$</td>
<td>Output map of $I(x)$</td>
<td>12</td>
</tr>
<tr>
<td>$prcl$</td>
<td>Prefix closure operator</td>
<td>13</td>
</tr>
<tr>
<td>$\equiv_k$</td>
<td>Level-$k$ equivalence relation</td>
<td>25</td>
</tr>
<tr>
<td>$\Sigma_u$</td>
<td>Set of eligible events at $u$</td>
<td>26</td>
</tr>
<tr>
<td>$\preceq$</td>
<td>Pre-order on $L_k$</td>
<td>28</td>
</tr>
<tr>
<td>$cell$</td>
<td>Cell of a partition (equivalence relation)</td>
<td>30</td>
</tr>
<tr>
<td>$\sim$</td>
<td>Behavior equivalence relation</td>
<td>35</td>
</tr>
<tr>
<td>$\mathcal{L}_k/\sim$</td>
<td>Quotient set of $L_k$ modulo $\sim$</td>
<td>35</td>
</tr>
<tr>
<td>$\preceq$</td>
<td>Partial order on $\mathcal{L}_k/\sim$</td>
<td>35</td>
</tr>
<tr>
<td>$\bowtie$</td>
<td>Pre-join on $L_k$</td>
<td>36</td>
</tr>
<tr>
<td>$\hat{\bowtie}$</td>
<td>Pre-meet on $L_k$</td>
<td>36</td>
</tr>
<tr>
<td>$\vee$</td>
<td>Join on $\mathcal{L}_k/\sim$</td>
<td>41</td>
</tr>
<tr>
<td>$\wedge$</td>
<td>Meet on $\mathcal{L}_k/\sim$</td>
<td>41</td>
</tr>
<tr>
<td>$\Sigma_{k,c}$</td>
<td>Set of controllable events at level $k$</td>
<td>44</td>
</tr>
<tr>
<td>$\Sigma_{k,u}$</td>
<td>Set of uncontrollable events at level $k$</td>
<td>44</td>
</tr>
</tbody>
</table>
\( \mathcal{C} \)  \hspace{1cm} \text{Controllable with respect to} \hspace{1cm} 49
\( \mathcal{C} \)  \hspace{1cm} \text{Class of controllable sublanguages} \hspace{1cm} 54
\( \text{Sup} \)  \hspace{1cm} \text{Supremal element} \hspace{1cm} 54
\( \preceq_T \)  \hspace{1cm} \text{Top-level sublanguage} \hspace{1cm} 58
\( S_{x,y} \)  \hspace{1cm} \text{Specification language} \hspace{1cm} 62
\( \Omega \)  \hspace{1cm} \text{Fixed-point operator} \hspace{1cm} 63
\( \Sigma_{\text{illegal}} \)  \hspace{1cm} \text{Set of illegal output events} \hspace{1cm} 69
\( \equiv_L \)  \hspace{1cm} \text{Nerode equivalence relation (N.e.r.)} \hspace{1cm} 82
\( \equiv_{\mathcal{C}(L)} \)  \hspace{1cm} \text{Structured N.e.r.} \hspace{1cm} 82
\( \equiv_\theta \)  \hspace{1cm} \text{Dynamic output kernel relation} \hspace{1cm} 83
\( \equiv_{\mathcal{C}(L,\theta)} \)  \hspace{1cm} \text{Structured N.e.r. with output map} \hspace{1cm} 83
\( ||L||_\mathcal{C} \)  \hspace{1cm} \text{State size of } L \hspace{1cm} 94
\( \Sigma_{\text{excl}} \)  \hspace{1cm} \text{Set of excluded output events} \hspace{1cm} 97
\( ||x||_\mathcal{X} \)  \hspace{1cm} \text{State size of } x \hspace{1cm} 99
Chapter 1

Introduction

1.1 RW Supervisory Control of DES

This work parallels Ramadge and Wonham supervisory control of Discrete-Event Systems (DES) in a hierarchical framework, and therefore it is appropriate to start with a brief tour of RW theory.

In [WR87] and [RW87] a DES is modeled as the generator of a formal language $L$. Thus, $L$ can be thought of as the set of trajectories—or the behavior—of the system. A string of $L$ is a sequence of events executable by the system, and therefore it represents a run of the system.

Let $\Sigma$ denote the set of event labels over which $L$ is defined. In order to tune the system behavior, a control structure is adjoined to $\Sigma$ by partitioning it into the set of controllable and uncontrollable events. A supervisor is an external agent in charge of controlling the system. It observes a sequence of events generated by the system as it unfolds, and it can restrict its possible extensions by disabling a subset of controllable events. Thus, the closed loop behavior of the system is a restriction—or a sublanguage—of $L$.

An important issue is to characterize those sublanguages of $L$ which are implementable by some supervisory control. A general notion of controllability is introduced and it is shown that a sublanguage $K$ of $L$ can be implemented by some
supervisory control\footnote{More specifically, marking nonblocking supervisory control.} if and only if it is controllable with respect to \( L \).

When the system behavior is expected to comply with a certain requirement, the desired closed-loop behavior of the system can be represented by some sublanguage \( E \) of \( L \). By the result just quoted, the requirement cannot be exactly met if \( E \) is not controllable with respect to \( L \). In such a case, one could settle for a best approximation, namely, the largest controllable (thus implementable) sublanguage of \( E \) (thus the requirement is met, though conservatively), if it ever exists. It is proven that the class of controllable sublanguages of \( E \) forms a complete upper subsemilattice of \((\Sigma^*, \subseteq)\) and therefore the supremal controllable sublanguage of \( E \) indeed exists. Such a supremal element is characterized as the largest fixed-point of a function \( \Omega : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*} \), which can be computed by successive approximation. The algorithm is proven to be effective when the languages \( E \) and \( L \) are both regular, i.e. their associated generators have finite numbers of states.

1.2 Motivations and Related Works

The idea of specifying a complex system as a hierarchy of subsystems has long been familiar. Nobel laureate Herbert Simon speculates on “the observed predominance of hierarchies among the complex systems nature presents to us” [Sim81]. Besides, because of its limited resources, mankind tends to deal with complexities through gradual refinement. Our psyche is incapable of processing a large volume of unorganized information, and organizing thus becomes an inseparable part of learning.

In the realm of system theory, Harel was among the first who realized how naturally and conveniently a complex system can be specified hierarchically. In his much-quoted article [Har87], he introduced statecharts as a visual specification formalism. The set of nodes in a statechart is enriched by a hierarchy relation captured by enclosure: At any level, the system is described by a State-Transition Diagram (STD), where the transitions might be equipped with special mechanisms for communication and concurrency. Every node in such a diagram, in turn, can have its
own STD, detailing the system specification at that node, and this pattern continues down to the bottom of the hierarchy, where additional information is not available or else is of no interest to the system analyst.

Previous work on hierarchical control of DES includes [ZW90], [BH93], [CW95] and [Wan95]. Zhong and Wonham [ZW90] study the issue of hierarchical consistency in the context of supervisory control. The aggregate high level model is refined (through refining the "reporter" map) until the behavior implemented by the low level operator matches what the high level manager expects of the aggregate model.

Heymann and Brave [BH93] specify a system as an Asynchronous Hierarchical State Machine, which is a simplified version of a statechart in which parallel components are not synchronized. Algorithms for testing reachability as well as for solving the forbidden configuration control problem are then developed. Because of the fact that the requirements are specified by predicates over the states and since parallel components are assumed to be asynchronous, one can ignore the order in the interleaving of events of parallel components, and therefore the complexity of the algorithms grows only polynomially in the number of parallel components.

Caines and Wei [CW95] study faithful aggregation of a finite-state machine based on a "dynamically consistent" partition of its state space. Such a partition has the property that either from none or all of the states in a cell $X_i$ the system can be taken to a cell $X_j$ by some sequence of transitions. Therefore, an aggregate transition between $X_i$ and $X_j$ would be a true representative of all system dynamics between $X_i$ and $X_j$. It is then shown that the resulting aggregate transition system is controllable—in the sense defined by the authors—if and only if the original system model is controllable.

A study similar to ours is carried out in [LPM96] on hierarchical Petri nets. The present research report is most closely related to [Wan95]. Taking our cue from Wang's state-tree structures (a control-theoretic version of statecharts), we provide a new linguistic basis and explicitly recursive approach. The linguistic basis provides a suitable framework for expressing the system behavior, while recursion emerges naturally from the hierarchical representation of the system. In a sense, this work
can be seen as a formal treatment of Wang's original ideas.

The crucial step is to define a class of languages—called *structured languages*—specially tailored to describe the behavior of hierarchical DES. Like the system model itself, a string representing a trajectory of the system is defined recursively: it is composed of the system events at the current level, as well as special symbols—called *tokens*—representing detailed behavior wherever a subsystem is specified in the model. Thus, a token \( x \) is interpreted by specifying 1) a language \( L_x \), defined over the subsystem event set, unioned with a new set of tokens representing the subsystem details, and 2) an output map \( \theta_x \), which specifies how the subsystem trajectories are to be connected to those of the system, thereby setting rules for defining the "global" behavior of the system. Within this context, a question regarding the system behavior can be posed and decided recursively, where each recursive call corresponds to a subsystem's share of that problem.

### 1.3 Organization of the Thesis

The rest of this report is organized as follows: In Chapter 2 we define the syntax and semantics of structured languages. Chapter 3 explains how language containment and controllability can be expressed in the new setup. It also establishes that the controllable sublanguages of a given language form a complete upper sub-semilattice. A fixed-point characterization of the supremal controllable element is carried out in Chapter 4, and an iteration scheme is presented for computation of such fixed-points. In Chapter 5 the notion of *regularity* is adapted to the structured case, and for regular languages the effectiveness of the iteration scheme is established. Chapter 6 concludes the thesis.
Chapter 2

Structured Languages

2.1 Introduction

Consider the system $S$ modeled by the hierarchical DES shown in the Figure 2.1. It is assumed that the system is specified at two levels of abstraction. At level $k = 0$, or the top level, we are given an automaton model $A$, defined over the alphabet $\Sigma_0 = \{\alpha, \beta, \gamma\}$. Such a model provides us with a high level overview of $S$, within which all the details at nodes are hidden. With this model in hand, one can quickly figure out general properties concerning the projection of the behavior of $S$ on $\Sigma_0$, e.g. whether or not $\alpha$ can follow $\beta$ in some trajectory of $S$, etc.

Yet in many other studies of the behavior of $S$, the wirings inside the nodes cannot be ignored. If for no other reason, an independent analysis of the behavior of subsystems of $S$ might be sought. More importantly, when a system like $S$ is specified hierarchically, some aspects of the system behavior may depend on certain properties of the subsystems. In the previous example, we made implicit the assumption that a transition out of a node of $A$ is "wired" internally to the incoming transition.

Thus, whenever applicable and conditional on the availability of such information, models for subsystems of $S$ at nodes of $A$ are specified. In case of $S$, the automata $A_0$ and $A_3$, defined over the alphabet $\Sigma_1 := \{a, b\}$, reveal the wiring inside the black-boxes marked by '0' and '3' in $A$, respectively (see Figure 2.2). In this case, we might say "$A_0$ and $A_3$ expand $A$ at nodes '0' and '3', respectively". $A_0$ and $A_3$ constitute
Figure 2.1: The system $S$

Figure 2.2: Level-wise specification of $S$
the system components at the level indexed by \( k = 1 \).

Obviously, the triple \((A, A_0, A_3)\)—which we would rather write in a list format as \( A[A_0, A_3] \)—falls short of specifying \( S \) completely, as we have not yet specified how the automata \( A_0 \) and \( A_3 \) should be plugged into \( A \).\(^1\) As a way of doing that, we represent the subsystems with labeled automata (or automata with output maps) models instead, where a node is labeled by \( \sigma \in \Sigma_0 \) if and only if there is a \( \sigma \) transition out of that node which returns the control back to the high level. We have denoted the labeled version of \( A_0 \) and \( A_3 \) by \( \hat{A}_0 \) and \( \hat{A}_3 \), respectively, and the revised and complete characterization of \( S : A[\hat{A}_0, \hat{A}_3] \) is shown in Figure 2.3.

Note that theoretically we can continue the process of adding details at nodes as long as we wish—of course, subject to the possibility of gathering such information and the need for doing that. For example, the system \( S' \) shown in Figure 2.4 is obtained by adding expansions to node ‘1’ of \( A \) and node ‘2’ of \( A_3 \), with the expanding structures given by \( \hat{A}_1 \) and \( \hat{A}_{32} \), respectively. Then, with other components defined as before, we get the representation \( S' : A[\hat{A}_0, \hat{A}_1, \hat{A}_3[\hat{A}_{32}]] \) for \( S' \), which shows how this representation is robust with respect to adding details at the leaf nodes. Finally, it is worthwhile to notice how conveniently a subsystem can be expressed using this notation: \( A_0, A_1, \) and \( A_3[\hat{A}_{32}] \) are all subsystems of \( S' \).\(^1\)

The objective of this chapter is to set up a suitable framework for expressing the \textit{behavior} of systems like \( S \). A trajectory of \( S \) is represented by a string \( s \), which is composed of events from the alphabet \( \Sigma_0 \), as well as special symbols from a disjoint set, which represent the information hidden at nodes of \( A \). Thus, if the symbol \( x_0 \) represents the subsystem \( \hat{A}_0 \), then the maximal trajectory generated by \( S \) when it takes the upper path of Figure 2.1 can be expressed by \( s = x_0 \beta \alpha \). Indeed, what we propose in the next section is slightly different than this, namely, we take

\(^{1}\text{In fact, at one end it is intuitively clear how the plugging should be carried out: the transitions into, say, node ‘0’ of \( A \) enter the initial state of \( A_0 \). It is connections at the other end that we have no clue about, e.g., which nodes of \( A_0 \) the transitions out of ‘0’ in \( A \) exit. In system theory language, we have to specify how a subsystem reports back to the system after it is done with its processing.}\)

\(^{1}\text{Alternatively, we could have denoted the subsystem hidden at node ‘3’ of \( A \) by \( \hat{A}_3[\hat{A}_{32}] \), etc. It all depends on whether we see the subsystem as an independent entity or in the context of the system in which it is embedded.}\)
Figure 2.3: Revised—level-wise specification of $S$

Figure 2.4: The system $S'$ and its new components
\[ s = x_0 \beta x_1 \alpha x_2, \] where \( x_1 \) and \( x_2 \) represent the internal behavior of nodes '1' and '2' of \( A \), respectively, which is of course empty. By doing so we achieve an alternating pattern of \( x \)'s and \( \sigma \)'s among the strings, which facilitates our future explorations of the system behavior. The framework is defined rigorously in the next two sections.

### 2.2 Syntax

In this section we formally define the class of structured languages as a framework for modeling the behavior of multi-level DES. We illustrate the ideas through the system \( S \) of Section 2.1.

Let \( S \) be a hierarchical DES with levels indexed by \( 0 \leq k < p \). An event at level \( k \) is labeled from an alphabet \( \Sigma_k \); a subsystem at level \( k \) is represented by a token \( x \) from a nonempty set \( X_k \). Thus, in the example of Section 2.1, for \( k = 0 \) we have \( \Sigma_0 = \{ \alpha, \beta, \gamma \} \) and \( X_0 = \{ x_0, \ldots, x_5 \} \), where each token \( x_i \) represents the subsystem hidden at node 'i' of \( A \). Subsets \( X_{k,\sigma} \subseteq X_k \) are given for all \( \sigma \in \Sigma_k \), with the interpretation that \( x \in X_{k,\sigma} \) iff the event \( \sigma \in \Sigma_k \) extends some trajectories of the subsystem that \( x \) represents. In the example, \( X_{0,\alpha} = \{ x_0, x_1 \} \), \( X_{0,\beta} = \{ x_0, x_3 \} \), and \( X_{0,\gamma} = \{ x_3 \} \).

Given \( \Sigma_k \) and \( X_k \) as specified, a typical trajectory \( s \) of a component at level \( k \) (the system itself if \( k = 0 \)) can be formed by a sequence of \( x \)'s and \( \sigma \)'s, where \( x \in X_k \) and \( \sigma \in \Sigma_k \). Without loss of generality, we assume that \( s \) begins with some \( x \in X_k \), and that events and tokens alternate, with event \( \sigma \) following token \( x \) if \( x \in X_{k,\sigma} \), and the string containing \( s \) terminates (if at all) with some \( x \) satisfying \( (\forall \sigma \in \Sigma_k) x \notin X_{k,\sigma} \). For this, we might be required to include special tokens with "empty" behavior, appearing in \( s \) wherever no subsystem is specified in the model of that component.

The string \( s = x_0 \beta x_1 \alpha x_2 \) is a trajectory of \( S \) of Section 2.1. Note that \( x_0 \in X_{0,\beta} \), \( x_1 \in X_{0,\alpha} \), and \( x_2 \notin \bigcup_{\sigma \in \Sigma_0} X_{0,\sigma} \). In this example, the tokens \( x_1, x_2, x_4 \) and \( x_5 \) all represent empty behavior. Indeed, with the exception of \( x_1 \), they are indistinguishable semantically, in the sense that they have the same behavior (empty), and that they are interfaced with the parent structure—i.e. \( A \)—in exactly the same way (in this
case, no transition labeled by $\Sigma_0$ events exits their respective nodes in $A$).

To formalize the idea, let $\{\Sigma_k\}_{k=0}^{p-1}$ and $\{\mathcal{X}_k\}_{k=0}^{p-1}$ be two sequences of pairwise disjoint alphabets\(^1\) and assume that subsets $\mathcal{X}_{k,\sigma} \subseteq \mathcal{X}_k$ are given for all $\sigma \in \Sigma_k$, $0 \leq k < p$. In order to generate all strings of the desired pattern that can possibly be made from the pair $(\Sigma_k, \mathcal{X}_k)$, we define a grammar $G_k = (V, T, P, S)$, where:

- $V := \{S\}$ is the set of variables;
- $T := \Sigma_k \cup \mathcal{X}_k$ is the set of terminals;
- $S$ is the start symbol;

and the set of production rules $P$ is given by:

1. $(\forall x \in \mathcal{X}_k, \sigma \in \Sigma_k) \ x \in \mathcal{X}_{k,\sigma} \Rightarrow S \rightarrow x\sigma S$,
2. $(\forall x \in \mathcal{X}_k)(\forall \sigma \in \Sigma_k) x \notin \mathcal{X}_{k,\sigma} \Rightarrow S \rightarrow x$.

A string $s \in T^*$ is generated by $G_k$ if it can be derived by repeated application of the production rules on $S$. For example, if $T = \{\alpha, \beta, x_0, x_1, x_2\}$ and the production rules are

- $P1$: $S \rightarrow x_0\beta S$
- $P2$: $S \rightarrow x_1\alpha S$
- $P3$: $S \rightarrow x_2$

then by applying $P1$, $P2$ and $P3$ in order we will get the string $x_0\beta x_1\alpha x_2 \in T^*$:

$S \xrightarrow{P1} x_0\beta S \xrightarrow{P2} x_0\beta x_1\alpha S \xrightarrow{P1} x_0\beta x_1\alpha x_2$

Let $L(G_k)$ denote the language generated by $G_k$, and define $P_k : (\Sigma_k \cup \mathcal{X}_k)^* \rightarrow \Sigma_k^*$ to be the natural projection of $(\Sigma_k \cup \mathcal{X}_k)^*$ onto $\Sigma_k^*$. Also, for a string $s = a_1 \ldots a_n$ over an alphabet $\Sigma$, define $\text{length}(s) := n$ and for $n \geq 1$ denote $\text{first}(s) := a_1$ and $\text{last}(s) := a_n$. We define the class $L_k$ of languages according to:

$L_k := \{M \mid M \subseteq L(G_k) \land M \text{ satisfies (1) and (2)}\}$, where

\(^1\)Strictly speaking, $\mathcal{X}_k$'s are not alphabets since we let them have countable cardinality.
1 \[(\forall s \in \overline{M}, \sigma \in \Sigma_k) \text{last}(s) \in \mathcal{X}_{k,\sigma} \Rightarrow s \sigma \in \overline{M},\]

2 \[(\forall s, t \in \overline{M}) P_k(s) = P_k(t) \& \text{length}(s) = \text{length}(t) \Rightarrow s = t.\]

If \(\overline{M} \in \mathcal{L}_k\) is to represent the behavior of a component of \(S^t\), (1) reminds us to include all possible \(\sigma\)-continuations of a token if it appears somewhere in a string of \(M\). In the example, if \(M \subseteq L(G_0)\) contains \(x_0 \beta x_1 \alpha x_2\) and \(\overline{M}\) is to represent the behavior of \(S\), then \(M\) must also contain a string with the prefix \(x_0 \alpha\), since \(s = x_0 \in \overline{M}\) and \(x_0 \in \mathcal{X}_{0,\alpha}\). On the other hand, (2) in effect imposes a structural condition on \(S\) and its components: by following a sequence of events in \(\Sigma_k\) a unique subsystem of \(S\) can be reached. Take the string \(\alpha \beta \in \Sigma^*_0\) for example. Any path with this projection in \(S\)—\(\alpha \beta\) and \(\alpha \alpha \beta\) are possible choices—will take \(S\) to the same node. This would let us denote the collection of all such paths by a single string—say \(x_0 \alpha x_3 \beta x_5\)—in \(\mathcal{L}_k\) (see Figure 2.5 of Page 15).

**Example:** Let \(\Sigma_0 = \{\alpha, \beta, \gamma\}\) and \(\mathcal{X}_0 = \{x_0, \ldots, x_3\}\), with \(\mathcal{X}_{0,\alpha} = \{x_0, x_1\}\), \(\mathcal{X}_{0,\beta} = \{x_0, x_3\}\) and \(\mathcal{X}_{0,\gamma} = \{x_3\}\). The production rules in this case would be:

\[S \rightarrow x_0 \alpha S | x_0 \beta S | x_1 \alpha S | x_3 \beta S | x_3 \gamma S | x_2 | x_4 | x_5\]

(abbreviated form for \(S \rightarrow x_0 \alpha S, S \rightarrow x_0 \beta S\), etc.).

\(M_0 = x_0(\beta x_1 \alpha x_2 + \alpha x_3(\gamma x_4 + \beta x_5)), M_1 = x_1 \alpha x_2\) and \(M_2 = x_2\) are all subsets of \(L(G_0)\) satisfying the conditions (1) and (2), and \(\overline{M}_0\) gives a high level description of the system \(S\) of Section 2.1.

### 2.3 Semantics

In this section we define the semantics of languages in \(\mathcal{L}_k\) by identifying each token in \(\mathcal{X}_k\) with a language in \(\mathcal{L}_{k+1}\) equipped with an output map. For convenience we define \(\Sigma_p := \emptyset\) and \(\mathcal{L}_p := \emptyset, \{\epsilon\}\), which is a class whose only members, \(\emptyset\) and \(\{\epsilon\}\), are already interpreted.

---

\(^1\)\(S\) itself if \(k = 0\).
Define $\Sigma_{k,p} := \bigcup_{i=k}^{p} \Sigma_i$, $0 \leq k < p$, and let $\mathcal{F}(\Sigma_{k,p}^*)$ denote the class of closed sub-languages over $\Sigma_{k,p}$. We define the semantics function $\mathcal{I} : \mathcal{L}_k \rightarrow \mathcal{F}(\Sigma_{k,p}^*)$ recursively as follows: ($0 \leq k < p$)

- for any $x \in \mathcal{X}_k$, $\mathcal{I}(x) := (\mathcal{I}(L_x), \theta_x)$, where:
  - $L_x \in \mathcal{L}_{k+1}$, and it is required that $L_x \neq \emptyset$.
  - $\theta_x : L_x \rightarrow 2^{\Sigma_k}$, satisfying:
    1. $(\forall s \in L_x) \text{ last}(s) \in \Sigma_{k+1} \Rightarrow \theta_x(s) = \emptyset$
    2. $\theta_x(L_x) = \{ \sigma \in \Sigma_k \mid x \in \mathcal{X}_{k,\sigma} \}$

- for any $L \in \mathcal{L}_k$ and $s \in L$, $\mathcal{I}(s)$ is defined inductively as follows:
  1. $\mathcal{I}(\epsilon) := \{ \epsilon \}$
  2. $(\forall s \in L, x \in \mathcal{X}_k) \mathcal{I}(sx) := \mathcal{I}(s) \mathcal{I}(L_x)$
  3. $(\forall s \in L, x \in \mathcal{X}_k, \sigma \in \Sigma_k) \mathcal{I}(sx\sigma) := \mathcal{I}(s) \bigcup_{s' \in L_x \land \sigma \in \theta_x(s')} \mathcal{I}(s') \cdot \{ \sigma \}$

- for any $L \in \mathcal{L}_k$,

$$\mathcal{I}(L) := \bigcup_{s \in L} \mathcal{I}(s)$$

Back to our example, $x_0 \in \mathcal{X}_0$ is now identified with two components: $L_0$ and $\theta_0$. $L_0$ is a language in $\mathcal{L}_1$, defined over the alphabet $\Sigma_1 = \{ a, b \}$ and a new set of tokens $\mathcal{X}_1 = \{ x_{00}, x_{01} \}$, according to $L_0 := \overline{x_{00}}ax_{01}$ (see Figure 2.5 of Page 15). The tokens $x_{00}$ and $x_{01}$ represent the subsystems hidden at nodes of $A_0$, although degenerate in this case. The output map $\theta_0$, on the other hand, maps the strings of $L_0$ into subsets of $\Sigma_0$, where a string terminated by a token in $\mathcal{X}_1$ is mapped into the labels of the corresponding node in the $A_0$ automaton. Thus, we have $\theta_0(x_{00}) = \{ \alpha \}$ and $\theta_0(x_{00}ax_{01}) = \{ \beta \}$ and therefore, $\theta_0(L_0) = \{ \alpha, \beta \} = \bigcup \{ \sigma \in \Sigma_0 \mid x_0 \in \mathcal{X}_{0,\sigma} \}$.

---

\footnote{For $L \in \mathcal{L}_p$ we have $\mathcal{I}(L) = L$ and $\mathcal{I}(\epsilon) = \{ \epsilon \}$.}

\footnote{To render the notation less cumbersome, we have dropped $x$ from the subscripts of $L$ and $\theta$ since the tokens of $L$ are labeled consistently with $x$'s.}
Notice how $L_k$ and $X_k$ are interpreted recursively: a token in $X_k$ is a member of the alphabet over which the languages in $L_k$ are defined, and it is in turn interpreted by a language in $L_{k+1}$ with an output map. The base of this recursion (and therefore of all the inductive proofs in this work) is at $k = p$, where we conveniently regarded $L = \{\epsilon\}$ as the only nonempty member of $L_p$. In our example, $p = 2$ (since $S$ is specified at two levels) and $I(x_00) = (I(L_00), \theta_{00})$, where $L_00 = \{\epsilon\}$ (the only nonempty language in $L_2$) and $\theta_{00}(\epsilon) = \{a\}$.

A language $L \in L_k$ defined as prescribed is called a $(p - k)$-layered structured language. Structured languages provide a suitable framework for expressing the behavior of multi-level DES. In the following example, we formulate the behavior of the system $S$ of Section 2.1 in the new setup.

**Example:** Consider the system $S : A[\hat{A}_0, \hat{A}_3]$ introduced in Section 2.1. We have $\Sigma_0 = \{\alpha, \beta, \gamma\}$ and $\Sigma_1 = \{a, b\}$, and therefore we take $p = 2$. As the first step, we augment the list $S : A[\hat{A}_0, \hat{A}_3]$ into $S : A[\hat{A}_0, \hat{A}_1, \hat{A}_2, \hat{A}_3, \hat{A}_4, \hat{A}_5]$, in other words, we consider internal structures—artificial though they might be—for all nodes of $A$. Correspondingly, we assign a token $x_i$ from a set $X_0$ to represent the item $\hat{A}_i$ in the list, $i = 0, \ldots, 5$. The behavior of $A$ is then described by the language $L \in L_0$, defined as:

\[
L = prcl(x_0(\beta x_1 \alpha x_2 + \alpha x_3(\gamma x_4 + \beta x_5)))
\]

where $prcl(\cdot)$ is the prefix closure operator. Next, every item in the list $x_0, \ldots, x_5$ is interpreted by an ordered pair consisting of a language in $L_1$ and an output map, given by:

\[
x_0 : \begin{cases} L_0 = x_{00} ax_{01}; \\ \theta_0(x_{00}) = \{\alpha\}, \ \theta_0(x_{00} ax_{01}) = \{\beta\} \end{cases}
\]

\[
x_1 : \begin{cases} L_1 = x_{10}; \\ \theta_1(x_{10}) = \{\alpha\} \end{cases}
\]

\[
x_2 : \begin{cases} L_2 = x_{20}; \\ \theta_2 = \emptyset \end{cases}
\]
Finally, we shall interpret the token $x_{ij} \in X_1$, which represents the subsystem of $A_i$ indexed by $j$, by specifying a pair $(L_{ij}, \theta_{ij})$, $ij \in \{00, 01, 10, 20, 30, 31, 32, 40, 50\}$. The language part in all cases is $\{\epsilon\}$, the only nonempty language in $L_2$, while the output functions are given below:

$$
\begin{align*}
\theta_{00}(\epsilon) &= \theta_{30}(\epsilon) = \{a\} \\
\theta_{31}(\epsilon) &= \{b\} \\
\theta_{01}(\epsilon) &= \theta_{10}(\epsilon) = \theta_{20}(\epsilon) = \theta_{32}(\epsilon) = \theta_{40}(\epsilon) = \theta_{50}(\epsilon) = \emptyset
\end{align*}
$$

The above formulation would have been quite straightforward had we labeled the states of the system with appropriate $x$'s beforehand. Such a graphical representation provides all the information needed in writing $L$, and therefore can replace the textual formulation altogether (Figure 2.5).

Notes:

1. The system $S$ may alternatively be referred to as "the system $L$". Note that the specification of $S$ is complete only when all the items $L, x_i; i \in \{0, \ldots, 5\}$, and $x_{ij}; ij \in \{00, 01, 10, 20, 30, 31, 32, 40, 50\}$ are specified. Sometimes we refer to these items collectively as the definition hierarchy of $L$ (or $S$).

2. As we mentioned before, in our representations it might be required to add layers at "simple nodes", where no details are known about the system behavior, or else are of interest to the system analyst. Though this might seem unnecessary and it makes formulation of the language more cumbersome, we believe it will simplify
Figure 2.5: The system $\mathcal{S}$ with states labeled with tokens

the study of the system by unifying the approach in dealing with different states of a certain component: there are always $m$ levels to go down in an $m$-layered structured language, even if information is not specified in $m$ layers at the point of interest. The added layers contain no additional information and have no effect on the way we interpret the language: in case of simple states, the incoming string is simply catenated by the output symbol, with nothing in between.

3. In the previous example, observe that

\[
\mathcal{I}(x_{00}) = \mathcal{I}(x_{30}) \\
\mathcal{I}(x_{01}) = \mathcal{I}(x_{10}) = \mathcal{I}(x_{20}) = \mathcal{I}(x_{32}) = \mathcal{I}(x_{40}) = \mathcal{I}(x_{50})
\]

and so for the purpose of system identification they are interchangeable, in the sense that they represent exactly the same entity.

From this observation it follows that $\mathcal{I}(L_2) = \mathcal{I}(L_4) = \mathcal{I}(L_5)$. We also have:

\[
\theta_2(\epsilon) = \theta_4(\epsilon) = \theta_5(\epsilon) = \emptyset \\
\theta_2(x_{20}) = \theta_4(x_{40}) = \theta_5(x_{50}) = \emptyset
\]
and therefore $x_2$, $x_4$ and $x_5$ are interchangeable too. Note, however, that the same cannot be stated about $x_1$ and $x_2$ despite the fact that $\mathcal{I}(L_1) = \mathcal{I}(L_2)$, since

$$\theta_1(x_{10}) = \{\alpha\} \neq \emptyset = \theta_2(x_{20})$$

Also, note how the interchangeability of tokens, and thereby of strings and languages, is established recursively in this setup: $x_2$ and $x_4$ are interchangeable because it is already known that $x_{20}$ and $x_{40}$ are interchangeable. A rigorous treatment of interchangeability is carried out in Chapter 3.

4. A structured language $L \in \mathcal{L}_k$, is after all a language—i.e. a set of words (or strings)—over $\Sigma_k \cup \mathcal{X}_k$. Therefore, it can be specified by its reachability tree. Similarly, a token $x$ in the definition hierarchy of the system can be interpreted graphically by a labeled tree, representing the pair $(L_x, \theta_x)$, where the node $s$ in the tree of $L_x$ is labeled by $\sigma$ if and only if $\sigma \in \theta_x(s)$. Thus, in the previous example, one can alternatively use the set of trees shown in Figure 2.6 to specify the system.

2.4 Discussion

2.4.1 Syntax and Semantics Review

We start this section by enumerating some of the syntactic properties of a structured language $L \in \mathcal{L}_k$ which are recurrent in this thesis. Most of these properties have been addressed during our exposition of the subject but nevertheless, the reader is encouraged to investigate them for the language $L$ of the example.

1. $L$ is a closed language over $\Sigma_k \cup \mathcal{X}_k$.

2. All nonempty strings of $L$ start with a unique $x \in \mathcal{X}_k$. 

16
Figure 2.6: Tree representation
3. Tokens and events alternate in the strings of $L$, and if a finite string $s$ of $L$ has a maximal length (i.e. $\neg(\exists t \in L)s < t$) then $s$ is terminated by some token $x \in \mathcal{X}_k$.

4. In the reachability tree of $L$, nodes entered by edges labeled with tokens do not have siblings.

Therefore, the reachability tree of $L$ should look like the tree of Figure 2.7, where $\sigma$ and $x$ represent a typical member of $\Sigma_k$ and $\mathcal{X}_k$, respectively.

As for the semantics, recall that the key step was to assign a pair $(L_x, \theta_x)$ to every token $x \in \mathcal{X}_k$. The output map $\theta_x$ is nonempty only on those strings of $L_x$ which are terminated by tokens. We call such strings well-formed strings and unless otherwise stated, we reserve the letters $u$, $v$ and $w$ to denote them. By convention, $\epsilon$ is considered to be the only wf-string of $\{\epsilon\} \in L_p$.

If $\cdots \sigma_i x \sigma_{i+1} \cdots$, with $\sigma_i, \sigma_{i+1} \in \Sigma_k$, is a segment of some string in $L$, then $L_x$ provides the detailed account of the system behavior after the occurrence of $\sigma_i$, and $\theta_x$ specifies how the subsystem trajectories (whose collection is $L_x$) are to be connected to those of the system, thereby setting rules for defining the “global” behavior of the
system. Strictly speaking, whenever \( s' \in L_x \) and \( \sigma_{i+1} \in \theta_x(s') \), \( \cdots \sigma_i \mathcal{I}(s') \sigma_{i+1} \cdots \) will be a segment of a collection of global trajectories of the system. Observe that the existence of such an \( s' \) is established by the fact that \( \cdots \sigma_i x \sigma_{i+1} \cdots \) is a segment of \( s \in L \) only if \( x \in X_k, \sigma_{i+1} \), and that \( \theta_x(L_x) = \{ \sigma \in \Sigma_k \mid x \in X_k, \sigma \} \).

2.4.2 Expanding \( \mathcal{I} \)

The definition of \( \mathcal{I} \) of Page 12 is highly recursive. We interpret languages by interpreting their strings, strings by interpreting their constitutive tokens, and tokens by interpreting languages at the next level downward. Our treatment of the subject is somewhat similar to the use of substitution in the formal language theory [HU79]. To illustrate the similarity, consider the segment of a language \( L \) shown in Figure 2.8.

\[
\alpha \rightarrow x \rightarrow \beta \rightarrow \gamma
\]

Figure 2.8: Restricted vs. unrestricted substitution

In an unrestricted substitution, \( \cdots \alpha L_x(\beta + \gamma) \cdots \) will be part of the reachability tree of the expanded language. But in our case, not all the strings of \( L_x \) should be catenated by \( \beta \) or \( \gamma \). Roughly speaking, if \( s' \in L_x \) and \( \beta \in \theta_x(s') \), only then will \( \alpha s' \beta \) be a segment of a string in \( \mathcal{I}(L) \). The above account is not quite accurate, because it is the interpretation of \( L_x \), i.e. \( \mathcal{I}(L_x) \), which should substitute for \( x \) in \( L \), not \( L_x \) itself (same for \( s' \)).

When \( L \in \mathcal{L}_k \), one can expand the recursive definition of \( \mathcal{I}(L) \) to make explicit the fact that \( \mathcal{I}(L) \) is a language over \( \Sigma_{k,p} \), as the following example illustrates.

**Example 1:** We will expand \( \mathcal{I}(L) \) for the language \( L \) defined in the example of Page 13, and write it explicitly as a language over \( \Sigma_0 \cup \Sigma_1 \).
We start at the bottom level:

\[ \mathcal{I}(L_{ij}) = \{ \epsilon \}, \text{ where } ij \in \{00, 01, 10, 20, 30, 31, 32, 40, 50\} \]

Now we move up to level (1), starting with \( L_0 \):

\[ \mathcal{I}(\epsilon) = \{ \epsilon \} \]

\[ \mathcal{I}(x_{00}) = \mathcal{I}(\epsilon) \mathcal{I}(L_{00}) = \{ \epsilon \} \]

\[ \mathcal{I}(x_{00}a) = \mathcal{I}(\epsilon) \mathcal{I}(\epsilon). \{ a \} = \{ a \} \ (\epsilon \in L_{00} \& a \in \theta_{00}(\epsilon)) \]

\[ \mathcal{I}(x_{00}ax_{01}) = \mathcal{I}(x_{00}a) \mathcal{I}(L_{01}) = \{ a \}. \{ \epsilon \} = \{ a \} \]

Therefore,

\[ \mathcal{I}(L_0) = \mathcal{I}(x_{00}) \cup \mathcal{I}(x_{00}ax_{01}) = \{ \epsilon, a \} \]

Note that \( \mathcal{I}(\epsilon) \subseteq \mathcal{I}(x_{00}) \) and \( \mathcal{I}(x_{00}a) \subseteq \mathcal{I}(x_{00}ax_{01}) \). Therefore, in \( \mathcal{I}(L) := \bigcup_{s \in L} \mathcal{I}(s) \), taking the union over \( w\)-\textit{strings} will suffice.

Similarly, we get

\[ \mathcal{I}(L_1) = \mathcal{I}(L_2) = \mathcal{I}(L_4) = \mathcal{I}(L_5) = \{ \epsilon \}, \text{ and} \]

\[ \mathcal{I}(L_3) = \{ \epsilon, a, ab \} \]

Finally, we are ready to compute \( \mathcal{I}(L) \):

\[ \mathcal{I}(\epsilon) = \{ \epsilon \} \]

\[ \mathcal{I}(x_0) = \mathcal{I}(\epsilon) \mathcal{I}(L_0) = \{ \epsilon, a \} \]

\[ \mathcal{I}(x_0\beta x_1) = \mathcal{I}(x_0\beta) = \mathcal{I}(\epsilon) \mathcal{I}(x_{0}\alpha x_{01}). \{ \beta \} = \{ a\beta \} \]

\[ \mathcal{I}(x_0\beta x_1\alpha x_2) = \mathcal{I}(x_0\beta x_1\alpha) = \mathcal{I}(x_0\beta) \mathcal{I}(x_{10}). \{ \alpha \} = \{ a\beta\alpha \} \]

\[ \mathcal{I}(x_0\alpha) = \mathcal{I}(\epsilon) \mathcal{I}(x_{00}). \{ \alpha \} = \{ \alpha \} \]

\[ \mathcal{I}(x_0\alpha x_3) = \mathcal{I}(x_0\alpha) \mathcal{I}(L_3) = \{ \alpha, \alpha a, \alpha ab \} \]
\[ I(x_0 \alpha x_3 \gamma x_4) = I(x_0 \alpha x_3 \gamma) = I(x_0 \alpha) . I(x_3 \alpha x_31 b x_32) . \{ \gamma \} = \{ \alpha b \gamma \} \]

And finally,

\[ I(x_0 \alpha x_3 \beta x_5) = I(x_0 \alpha x_3 \beta) = I(x_0 \alpha) \cdot \left( I(x_3) \cup I(x_3 \alpha x_31 b x_32) \right) . \{ \beta \} = \{ \alpha \beta, \alpha a b \beta \} \]

Therefore,

\[
I(L) = \{ \epsilon, a, a \beta, a \beta \alpha, \alpha, \alpha a, \alpha a b, \alpha a b \gamma, \alpha \beta, \alpha a b \beta \} \\
= \alpha b \alpha + \alpha (\beta + a b (\beta + \gamma))
\]

Clearly, \( I(L) \) is closed since \( L \) itself is closed. Also, with the alphabets involved fixed, the map \( I \) is 1-1 if the alphabets are mutually disjoint, but not onto. Having a 1-1 map between \( \mathcal{L}_k \) and \( \mathcal{F}(\Sigma^*_{k,p}) \) translates into the possibility of moving in the opposite direction, and structuring a flat (standard) language based on a given partition of its alphabet.

We do not intend to put our efforts into proving the injectivity of \( I \), but we will provide counterexamples illustrating why it is not onto, as that will shed some light on the type of structures which are not expressible in this setup.

**Example 2:** With \( \Sigma_0 = \{ \alpha, \beta, \gamma \} \) and \( \Sigma_1 = \{ a \} \), the language \( L'_1 \in \mathcal{F}(\Sigma^*_{0,2}) \) (super-script \( f \) captures the flat, unstructured characteristic of the language) of Figure 2.9 cannot be the image of any language in \( \mathcal{L}_0 \) with \( p = 2 \). As is seen in the lower part of the figure, our attempts at formulating such a language fail when we face non-determinism at the top level (\( \gamma \) exiting \( x_0 \)). In the terminology we have used before, the \( \Sigma^*_0 \)-projection \( \gamma \) (seen as a string) takes the system to two different subsystems—simple nodes in this case. Thus, the tokens \( x_1 \) and \( x_3 \) are not interchangeable, and including both \( x_0 \gamma x_1 \) and \( x_0 \gamma x_3 \) would violate the syntax:

\[
P_0(x_0 \gamma x_1) = P_0(x_0 \gamma x_3) \& \text{length}(x_0 \gamma x_1) = \text{length}(x_0 \gamma x_3) \& x_1 \neq x_3
\]
Note that since the alphabets are disjoint, there is a unique way of doing the backward procedure of adding layers. In contrast, the language $L_2^f$, obtained by relabeling the $\alpha$-transition in $L_1^f$ to $\beta$, is the image of $L_2 \in \mathcal{L}_2$.

$$L_1^f \quad L_2^f$$

$\quad L_1 \notin \mathcal{L}_0 \quad L_2 \in \mathcal{L}_0$

Figure 2.9: Example 2: system on the left side is not expressible in $\mathcal{L}_2$

**Example 3:** With $\Sigma_0 = \{\alpha\}$, $\Sigma_1 = \{a\}$, and $\Sigma_2 = \{\tau\}$, the structure in part (a) of Figure 2.10 cannot be expressed by a 3-layered structured language, while the structure in part (b) can. In the $\mathcal{I}$-image of any language in $\mathcal{L}_0$ ($p = 3$), the event $\alpha \in \Sigma_0$ should be catenated with all or none of the strings with the same $\Sigma_0 \cup \Sigma_1$-projection (in this case, $\varepsilon$ and $\tau$).  

$$\tau \quad a$$

(a)  

$$\tau \quad a$$

(b)

Figure 2.10: Example 3: system on the left side is not expressible in $\mathcal{L}_3$

We conclude this section by presenting a graphical approach for expanding $\mathcal{I}(L)$, which matches the tree representation of Figure 2.6. First the trees for $x_1$, $x_2$, $x_5$ substitute for the corresponding tokens in $L$, and then all the $x_{ij}$'s are removed from the resulting graph (since $L_{ij} = \{\varepsilon\}$). The procedure is illustrated in Figure 2.11.
Figure 2.11: Graphical expansion of $I(L)$
Chapter 3

Behavior Containment and Controllability

3.1 Introduction

Having set up a new framework for expressing the behavior of complex systems, we are now in a position to address two basic questions of concern to a system analyst:

I How can "behavior containment" be expressed in the new setup? In other words, we would like to state conditions under which $E \in \mathcal{L}_k$ can be regarded as a restriction of $L \in \mathcal{L}_k$. We denote such a relation by $E \preceq L$.

II With some control technology in place, is it possible to restrict the behavior of the system $L$ to a certain $E \preceq L$ under the influence of some supervisory control? If this is the case, we say $E$ is controllable with respect to $L$.

Consider the system $L \in \mathcal{L}_k$ which is represented by its reachability tree, shown partially in the right hand side of Figure 3.1. Also given in the figure is the language $E \in \mathcal{L}_k$, and we would like to investigate whether it can be regarded as a restriction of $L$, i.e. whether the system $L$ can track every step that the system $E$ takes.

\footnote{We said "partially" because the trees for $y_i$'s have not been specified yet.}
If \( E \) qualifies as a restriction of \( L \), observe that \( L \) must be able to follow every \( \Sigma_k \)-projection of the strings of \( E \), in other words, it must be the case that \( P_k(E) \subseteq P_k(L) \).

Since both languages are closed, this condition can be expressed in an alternative way. Consider the \( \text{wf-strings} \) \( x_0 \in E \) and \( y_0 \in L \). At this point, the set of eligible events in \( E \) is \( \{ \alpha, \beta \} \), which is a subset of eligible events in \( L \), comprising \( \{ \alpha, \beta, \gamma \} \). Therefore, \( L \) can successfully follow any of \( E \)'s next steps. Let us take the \( \beta \)-path in both. By syntax of \( L_k \), both strings should be catenated by some tokens, namely, for some \( x_1 \) and \( y_1 \) in \( \mathcal{X}_k \), \( x_0\beta x_1 \in E \) and \( y_0\beta y_1 \in L \). Once again we pose the same question: is the set of eligible events in \( E \) a subset of that in \( L \)? If this exhaustive inquiry in the reachability trees of \( E \) and \( L \) succeeds, then one can claim that \( P_k(E) \subseteq P_k(L) \).

Notice that in the above analysis we paired the strings of \( E \) and \( L \), as in \((x_0, y_0)\) and \((x_0\beta x_1, y_0\beta y_1)\), for the purpose of eligible event set comparison. The strings in a pair are both \textit{wf-strings}, and are of the same length and \( \Sigma_k \)-projection. In order to facilitate such pairings we introduce an equivalence relation on \( (\Sigma_k \cup \mathcal{X}_k)^* \) which relates a string of \( E \) to a unique string of \( L \) with the same length and \( \Sigma_k \)-projection, if it indeed exists.

\textbf{Definition 3.1} For all \( s, t \in (\Sigma_k \cup \mathcal{X}_k)^* \), \( s \equiv_k t \iff \)

\[
P_k(s) = P_k(t) \text{ and } \text{length}(s) = \text{length}(t)
\]

25
In other words,

\[ \equiv_k = \ker P_k \land \ker \text{length} \]

Thus in Figure 3.1 we have \( x_0 \equiv_k y_0 \) and \( x_0 \beta x_1 \equiv_k y_0 \beta y_1 \). Observe that by syntax of \( \mathcal{L}_k \) it follows that:

\[ (\forall L \in \mathcal{L}_k)(\forall s, t \in L) s \equiv_k t \Rightarrow s = t. \]

In other words, a string of \( L \) can be uniquely identified by specifying its length and projection onto \( \Sigma^*_k \).

Based on the new notation we formulate the result we have got thus far in the following lemma. Let \( \Sigma_u \) and \( \Sigma_v \) be the subsets of events in \( \Sigma_k \) which are eligible to extend \( u \) and \( v \) in \( E \) and \( L \), respectively, i.e.:

\[ \Sigma_u := \{ \sigma \in \Sigma_k \mid u \sigma \in E \} \text{ and } \Sigma_v := \{ \sigma \in \Sigma_k \mid v \sigma \in L \}. \]

**Lemma 3.1** Let \( E \) and \( L \) be languages in \( \mathcal{L}_k \) and \( L \) be nonempty. Then

\[ (\forall u \in E, v \in L) u \equiv_k v \Rightarrow \Sigma_u \subseteq \Sigma_v \]

if and only if \( P_k(E) \subseteq P_k(L) \). \( \bullet \)

So far we have investigated the implications of the containment of \( E \) in \( L \) only for the current level, namely, the containment of \( P_k(E) \) in \( P_k(L) \). If we capture the notion of containment by \( \preceq \), then roughly speaking, \( E \preceq L \) requires not only \( P_k(E) \subseteq P_k(L) \), but also a similar containment relation between those tokens in the two structures which correspond to each other. That means in the example of Figure 3.1, we must have \( x_0 \preceq y_0 \) and \( x_1 \preceq y_1 \), where the relation \( \preceq \) is appropriately extended to the domain of tokens. Thus, in order to decide whether \( E \preceq L \) we need to take a closer look at the structure of \( E \) and \( L \). Note that the problem is decided with a "no" answer if it is not the case that \( P_k(E) \subseteq P_k(L) \).

Sample tree representations for \( x_0 \) and \( y_0 \) are given in Figure 3.2, and we would like to state conditions under which \( x_0 \) can be regarded as a restriction of \( y_0 \), denoted by \( x_0 \preceq y_0 \), and check if this is the case in the example. As a sufficient condition
we require that $L_{x_0} \subseteq L_{y_0}$, where $L_{x_0}$ and $L_{y_0}$ are languages in $\mathcal{L}_{k+1}$ assigned to the tokens $x_0$ and $y_0$, respectively. But this is not enough, and naturally we expect the output maps assigned to $x$ and $y$ to play a role in deciding $x \subseteq y$. To see what goes wrong if we do just the language check, observe that with the expansions given in Figure 3.2 for $x_0$ and $y_0$, the system $E$ can take a global sequence—$a\alpha$—which $L$ is unable to follow. (For simplicity we have assumed that the specification of the languages $E$ and $L$ terminates at this level, i.e. $k + 2 = p$.)

Thus, instead of comparing the "aggregate" output of the subsystems—in this case, $\theta_{x_0}(L_{x_0}) = \{\alpha, \beta\} \subseteq \{\alpha, \beta, \gamma\} = \theta_{y_0}(L_{y_0})$—we perform a point-to-point comparison of the output sets at the subsystem level, i.e.:

$$(\forall u' \in L_{x_0}, v' \in L_{y_0})u' \equiv_{k+1} v' \Rightarrow \theta_{x_0}(u') \subseteq \theta_{y_0}(v')$$

in order to make sure that the restriction is respected at the interface of adjacent levels. In the above example, we have:

$$x_{00}ax_{01} \equiv_{k+1} y_{00}ay_{01} \text{ but } \theta_{x_0}(x_{00}ax_{01}) = \{\alpha, \beta\} \nsubseteq \{\beta, \gamma\} = \theta_{y_0}(y_{00}ay_{01})$$

and therefore $x_0 \nsubseteq y_0$ and consequently, $E \nsubseteq L$.

Finally, observe that deciding language containment in $\mathcal{L}_k$ is carried out recursively: the truth of $E \subseteq L$ is known only after deciding $L_x \subseteq L_y$ in $\mathcal{L}_{k+1}$, where $L_x$ and $L_y$ represent the relevant subsystems in $E$ and $L$, respectively, and some containment checks at the interfaces.
3.2 Preorder \( \preceq \) on \( \mathcal{L}_k \)

In this section we define a preorder\(^1\) on \( \mathcal{L}_k \) which captures the notion of behavior containment.

**Definition 3.2 (Preorder \( \preceq \))** Let \( E, L \in \mathcal{L}_k \), \( 0 \leq k \leq p \), and \( L \neq \emptyset \). \( E \preceq L \) iff

- \( k = p \): true
- \( 0 \leq k < p \): \((\forall u \in E, v \in L)u \equiv_k v \Rightarrow x \preceq y\)
  
  where \( x := \text{last}(u) \) and \( y := \text{last}(v) \).

For all \( x, y \in \mathcal{X}_k \), \( x \preceq y \) iff

\[ L_x \preceq L_y \text{ and } (\forall u' \in L_x, v' \in L_y)u' \equiv_{k+1} v' \Rightarrow \theta_x(u') \subseteq \theta_y(v') \]

The idea is illustrated in Figure 3.3.

![Figure 3.3: Illustration of \( E \preceq L \)](image)

Observe that \( \emptyset \preceq L \) for all nonempty \( L \) in \( \mathcal{L}_k \), and to cover cases where \( L = \emptyset \) we add the following axiom:

**Axiom 3.1**

\[(\forall E \in \mathcal{L}_k) E \preceq \emptyset \Rightarrow E = \emptyset.\]

\(^1\)For an overview of preorders the reader is referred to [Wec92].
It is noteworthy that the definition makes no explicit reference to any kind of restriction at the current level. In particular, there is no trace of the aggregate output requirement

\[(\forall u \in E, v \in L) u \equiv_k v \Rightarrow \Sigma_u \subseteq \Sigma_v\]

that we have stipulated in the previous section. The following lemma is the key to establishing this result.

**Lemma 3.2** Let \( E, L \in \mathcal{L}_k \) and \( E \preceq L \ (0 \leq k \leq p) \). Then for every string \( s \in E \) there exists a unique string \( t \in L \) such that \( s \equiv_k t \).

**Proof:** By syntax of \( \mathcal{L}_k \) it suffices to prove the result only for \( \text{wf-strings} \). First we do a backwards induction \(^1\) on the level index \( k \):

- \( k = p \): For the nontrivial case where \( E \neq \emptyset \) we have \( E = L = \{\epsilon\} \) and \( \epsilon \equiv_p \epsilon \) obviously holds true.

- \( 0 \leq k < p \): We bring in a second induction on the length of \( \text{wf-strings} \). At the base of induction we have the strings \( x_0 \in E \) and \( y_0 \in L \) which are guaranteed to exist when both \( E \) and \( L \) are nonempty. As for the inductive step, suppose \( u := x_0\sigma_0x_1\sigma_1\cdots\tau_{n-1}x_n \in E \), and that there exists some \( v \in L \) of the form \( v := y_0\sigma_0y_1\sigma_1\cdots\tau_{n-1}y_n \) and observe that \( u \equiv_k v \). If \( u\sigma_nx_{n+1} \in E \), by syntax we will have \( x_n \in \mathcal{X}_{k,\sigma_n} \) and so by semantics,

\[(\exists u' \in L_{x_n})\sigma_n \in \theta_{x_n}(u') \quad (3.1)\]

Now since \( E \preceq L \) and \( u \equiv_k v \), we have \( x_n \preceq y_n \). Thus \( L_{x_n} \preceq L_{y_n} \) and by \( k \)-induction assumption it follows that to any \( u' \in L_{x_n} \) satisfying (3.1) there corresponds a \( v' \in L_{y_n} \) such that \( u' \equiv_{k+1} v' \). Therefore, \( \sigma_n \in \theta_{y_n}(v') \) since \( x_n \preceq y_n \) and hence

\[(\exists u' \in L_{y_n})\sigma_n \in \theta_{y_n}(v')\]

\(^1\)For brevity henceforth referred to as “induction”.

29
i.e. $y_n \in \mathcal{X}_{k,\sigma_n}$. By syntax then it follows that $u\sigma_n y_{n+1} \in L$ for some $y_{n+1} \in \mathcal{X}_k$ and we have $u\sigma_n x_{n+1} \equiv_k v\sigma_n y_{n+1}$ since $u \equiv_k v$. This completes the proof.

Immediate from this lemma is the following result:

**Corollary 3.1**

$$(\forall E, L \in L_k) E \preceq L \Rightarrow P_k(E) \subseteq P_k(L)$$

So far, so good. Indeed, if $x$ and $y$ are two "corresponding" tokens in $E$ and $L$, respectively, then $E \preceq L$ implies $x \preceq y$ and therefore $L_x \preceq L_y$. By Corollary 3.1 then it follows that $P_{k+1}(L_x) \subseteq P_{k+1}(L_y)$, and a similar argument can be applied to all corresponding items in the definition hierarchies of $E$ and $L$. But as we have seen in Figure 3.2, this does not guarantee containment of the global behavior of $E$ in that of $L$, because the containment may fail to hold at the interface of adjacent levels. In Proposition 3.1 we will prove directly that $\mathcal{I}(E) \subseteq \mathcal{I}(L)$ when $E \preceq L$ and therefore we are entitled to call $E$ a sublanguage of $L$ whenever $E \preceq L$. First we state a couple of preliminary results, which are immediate from the definitions.

**Lemma 3.3** Let $L \in L_k$. We will have:

$$(\forall v_1, v_2 \in L) \mathcal{I}(v_1) \cap \mathcal{I}(v_2) \neq \emptyset \Rightarrow v_1 = v_2$$

**Corollary 3.2** wf-strings of $L \in L_k$ induce a partition on the language $\mathcal{I}(L)$, according to:

$$(\forall s \in \mathcal{I}(L), v \in L) s \in \text{cell}(v) \iff s \in \mathcal{I}(v) \iff P_k(s) = P_k(v)$$

In Example 3 of Section 2.4.2, we have, e.g.

\[
\begin{align*}
\text{cell}(x_0) &= \{\epsilon, a\} \\
\text{cell}(x_0\alpha x_3) &= \{\alpha, \alpha\alpha, \alpha\alpha\beta\} \\
\text{cell}(x_0\beta x_1\alpha x_2) &= \{\alpha\beta\alpha\}
\end{align*}
\]
Proposition 3.1 Let $E, L \subseteq L_k$, $0 \leq k \leq p$, and $E \subseteq L$. Then

$$I(E) \subseteq I(L)$$

Proof: Let $t \in I(E)$. By the corollary, we have $t \in I(u)$ for some $u \in E$. Since $E \subseteq L$, there exists by Lemma 3.2 a $v \in L$ such that $u \equiv_k v$. We will show that $I(u) \subseteq I(v)$ and therefore $t \in I(L)$. We prove this result by induction on the level index $k$:

- $k = p$: For the nontrivial case $E \neq \emptyset$ we have $u = v = \epsilon$ which implies $I(u) = \{\epsilon\} = I(v)$.

- $0 \leq k < p$: By induction on the length of strings we will show that
  
  $$(\forall r \leq u) I(r) \subseteq I(s),$$
  
  where $s$ is the unique string of $L$ satisfying $r \equiv_k s$.

  • Base: $\text{length}(r) = 0$, i.e. $r = s = \epsilon \Rightarrow I(r) = \{\epsilon\} = I(s)$
  
  • Inductive step: $\text{length}(r) = n \geq 1$. Either

    1. $r =: px$ and $s =: qy$

       Since $p$ is of length $n - 1$ and $p \equiv_k q$, it follows that $I(p) \subseteq I(q)$. Also, since $L_x \subseteq L_y$, by $k$-induction assumption we will have $I(L_x) \subseteq I(L_y)$.

       Therefore,
       $$I(r) = I(p)I(L_x) \subseteq I(q)I(L_y) = I(s)$$
       
       or,

    2. $r =: px\sigma$ and $s =: qy\sigma$

       Again by induction on length we have $I(p) \subseteq I(q)$. Since $x \subseteq y$, for arbitrary $u' \in L_x$ we have

       $$\sigma \in \theta_x(u') \Rightarrow \sigma \in \theta_y(v')$$

       where $v' \in L_y$, whose existence is established by Lemma 3.2, satisfies $u' \equiv_{k+1} v'$. Since $L_x \subseteq L_y$ by $k$-induction assumption it follows that

       $$I(u') \subseteq I(v')$$

       and therefore,

       $$\bigcup_{u' \in L_x \& \sigma \in \theta_x(u')} I(u') \subseteq \bigcup_{v' \in L_y \& \sigma \in \theta_y(v')} I(v')$$

6
Finally, since
\[ I(r) = \bigcup_{u' \in U_x \& \sigma \in \theta_x(u')} I(u') \cup \{ \sigma \} \quad \text{and} \]
\[ I(s) = \bigcup_{v' \in U_y \& \sigma \in \theta_y(v')} I(v') \cup \{ \sigma \} \]
we will get \( I(r) \subseteq I(s) \) and this completes the proof.

**Example:** With \( L \) as in the example of Page 13 and \( E \) shown in Figure 3.4, we would like to investigate systematically whether \( E \subseteq L \). In order to comply with the notation of Definition 3.2, we relabel the tokens of \( L \) by \( y_i \)'s.

![Diagram](image)

Figure 3.4: Language \( E \in L_0 \) (\( p = 2 \))

As the diagram of Figure 3.5 of Page 33 suggests, the answer is positive. Branches under language containment queries enumerate the \textit{wf-strings} of the language in the \( E \) family, and double arrows point to a corresponding \textit{wf-string} from the same equivalence class (\textit{modulo} \( \equiv_k \)) in the \( L \) family. Finally, branches under token containment queries list conditions under which that particular containment relation is satisfied. Observe that by Proposition 3.1 we must have:

\[ I(E) = a + \alpha(a + \beta) \leq \alpha \beta \alpha + \alpha(\beta + ab(\beta + \gamma)) = I(L). \]

To conclude this section we will prove that the relation \( \leq \) defines a preorder (quasi-order) on \( L_k \) as well as on \( X_k \).

**Proposition 3.2** The relation \( \leq \) is a preorder on \( L_k \) and on \( X_k \).
Figure 3.5: Verifying $E \preceq L$
Proof:

1. reflexive: \((\forall L \in \mathcal{L}_k)L \preceq L\)

By the axiom the result is valid when \(L = \emptyset\). When \(L \neq \emptyset\) we prove the result by induction on the level index \(k\).

- \(k = p\) : trivially true.

- \(0 \leq k < p\) : For every \(u, v \in L\) observe that \(u \equiv_k v\) iff \(u = v\). Let \(v \in L\) and \(y := \text{last}(v)\). We have \(y \preceq y\) since by the induction assumption \(L_y \preceq L_y\) and moreover,

\[
(\forall v' \in L_y)\theta_y(v') \subseteq \theta_y(v').
\]

2. transitive: \((\forall K, E, L \in \mathcal{L}_k)K \preceq E \& E \preceq L \Rightarrow K \preceq L\)

If, say, \(L = \emptyset\) then by Axiom 3.1 the result is vacuously true unless \(E = \emptyset\) and \(K = \emptyset\), in which case \(K \preceq L\).

Suppose \(E, L \neq \emptyset\).

- \(k = p\) : trivially true.

- \(0 \leq k < p\) : Let \(K \preceq E\) and \(E \preceq L\) and take \(w \in K\) and \(v \in L\) such that \(v \equiv_k w\). Since \(w \in K\) and \(K \preceq E\), by Lemma 3.2 the existence of \(u \in E\) with \(u \equiv_k w\) is established.

Let \(x := \text{last}(u)\), \(y := \text{last}(v)\) and \(z := \text{last}(w)\). Since \(u \equiv_k v \equiv_k w\) it follows that \(z \preceq x\) and \(x \preceq y\). Therefore,

\[
L_z \preceq L_x \quad \text{and} \quad L_x \preceq L_y
\]

which by the induction assumption imply \(L_z \preceq L_y\). Let \(w' \in L_z\) and \(v' \in L_y\) such that \(v' \equiv_{k+1} w'\). By a similar argument there can be found a unique \(u' \in L_x\) such that \(u' \equiv_{k+1} w'\) and therefore we will have

\[
\theta_z(w') \subseteq \theta_x(u') \quad \text{and} \quad \theta_x(u') \subseteq \theta_y(v')
\]

implying \(\theta_z(w') \subseteq \theta_y(v')\). This completes the proof.
The preorder $\preceq$ induces an equivalence relation on $L_k$ and $X_k$, according to:

**Definition 3.3**

- $(\forall E, L \in L_k) E \sim L \iff E \preceq L \& L \preceq E$

- $(\forall x, y \in X_k) x \sim y \iff x \preceq y \& y \preceq x$

Form Proposition 3.1 the following result is immediate:

**Corollary 3.3**

- $(\forall E, L \in L_k) E \sim L \Rightarrow I(E) = I(L)$

- $(\forall x, y \in X_k) x \sim y \Rightarrow \begin{cases} I(L_x) = I(L_y) \\ (\forall u' \in L_x, v' \in L_y) u' =_{L+1} v' \Rightarrow \theta_x(u') = \theta_y(v') \end{cases}$

Thus, $E \sim L$ (x ~ y) means that E and L (x and y) are behaviorally equivalent. If, say, x represents a system component and $x \sim y$, then y can replace some occurrences of x in a string without changing the meaning of that string, i.e.:

$(\forall L \in L_k, x, y \in X_k) x \sim y \Rightarrow L \sim L(x \leftrightarrow y)$

where $L(x \leftrightarrow y)$ is obtained from L by replacing some occurrences of x by y. Therefore, one may justifiably say “x and y are interchangeable.” In Example 1 of Page 13 we have, for example, $x_2 \sim x_4 \sim x_5$.

Thus, in our study of systems we do not distinguish between the languages (tokens) from the same equivalence class (modulo $\sim$), and therefore of special importance would be the quotient set $L_k/ \sim (X_k/ \sim)$. It turns out that the preorder $\preceq$ on $L_k (X_k)$ induces a partial order $\leq$ on the quotient set $L_k/ \sim (X_k/ \sim)$, defined as follows. Let $[E] ([x])$ denote the equivalence class (modulo $\sim$) containing $E$ (x).

**Definition 3.4**

- $(\forall E, L \in L_k) [E] \preceq [L] \iff E \preceq L$

- $(\forall x, y \in X_k) [x] \preceq [y] \iff x \preceq y$

It is straightforward to verify that $\preceq$ is indeed a partial order. In particular, $\preceq$ is antisymmetric since if, say, $[E] \preceq [L]$ and $[L] \preceq [E]$, we will have $E \preceq L$ and $L \preceq E$ and therefore, $E \sim L$, i.e. $[E] = [L]$. 

35
3.3 Join and Meet on $L_k/\sim$

In this section we will show that $(L_k/\sim, \preceq)$ and $(\mathcal{X}_k/\sim, \preceq)$ are complete lattices. To this end we define two binary operations on $L_k$ ($\mathcal{X}_k$) which induce join and meet operations on the quotient set.

**Definition 3.5 (Prejoin)** Let $L_1$, $L_2$ and $L$ be nonempty languages in $L_k$, $0 \leq k \leq p$. $L = L_1 \overset{\sim}{\vee} L_2$ iff

\begin{itemize}
  \item $k = p$ : $L = \{\varepsilon\}$
  \item $0 \leq k < p$ : For every $v \in L$, $C_1$ or $C_2$ holds, and

  \[ y = \begin{cases} 
  y_1 \overset{\sim}{\vee} y_2 & ; \text{if } C_1 \& C_2 \\
  y_1 & ; \text{if } C_1 \& \neg C_2 \\
  y_2 & ; \text{if } \neg C_1 \& C_2 
  \end{cases} \]

  where $C_i := (\exists v_i \in L_i) v_i \equiv_k v^i$, $y := \text{last}(v)$ and $y_i := \text{last}(v_i)$; $i = 1, 2$.
\end{itemize}

For any $y_1, y_2, y \in \mathcal{X}_k$, $y = y_1 \overset{\sim}{\vee} y_2$ iff $L_y = L_{y_1} \overset{\sim}{\vee} L_{y_2}$ and for any $v' \in L_y$,

\[ \theta_y(v') = \begin{cases} 
  \theta_{y_1}(v'_1) \cup \theta_{y_2}(v'_2) & ; \text{if } C'_1 \& C'_2 \\
  \theta_{y_1}(v'_1) & ; \text{if } C'_1 \& \neg C'_2 \\
  \theta_{y_2}(v'_2) & ; \text{if } \neg C'_1 \& C'_2 
  \end{cases} \]

where $C'_i := (\exists v'_i \in L_{y_i}) v'_i \equiv_{k+1} v^i$; $i = 1, 2$.

**Definition 3.6 (Premeet)** Let $L_1$, $L_2$ and $L$ be nonempty languages in $L_k$, $0 \leq k \leq p$. $L = L_1 \overset{\sim}{\wedge} L_2$ iff

\begin{itemize}
  \item $k = p$ : $L = \{\varepsilon\}$
  \item $0 \leq k < p$ : For every $v \in L$, $(\exists v_1 \in L_1, v_2 \in L_2) v_1 \equiv_k v_2 \equiv_k v$ & $y = y_1 \overset{\sim}{\wedge} y_2$

  where $y := \text{last}(v)$ and $y_i := \text{last}(v_i)$; $i = 1, 2$.
\end{itemize}

\[ \uparrow \text{Note that by syntax such a } v_i \text{ is unique since if } u_i \in L_i \text{ also satisfies } u_i \equiv_k v \text{ then we will have } u_i \equiv_k v_i, \text{i.e.} \]

\[ P_k(u_i) = P_k(v_i) \& \text{length}(u_i) = \text{length}(v_i) \]

implying $u_i = v_i$. 36
For any \( y_1, y_2, y \in \mathcal{X}_k \), \( y = y_1 \triangleleft y_2 \) iff

\[
L_y = L_{y_1} \triangleleft L_{y_2} \text{ and } \\
(\forall v' \in L_y, u'_1 \in L_{y_1}, u'_2 \in L_{y_2}) \ u'_1 \equiv_{k+1} u'_2 \equiv_{k+1} v' \Rightarrow \theta_y(v') = \theta_{y_1}(u'_1) \cap \theta_{y_2}(u'_2)
\]

Despite their complicated appearance, the definitions have simple intuitive interpretations. We convey the idea through the following example:

**Example:** In the upper half of Figure 3.6, \( L_1 \) and \( L_2 \) are specified in the graphical format, and then \( L_1 \triangleleft L_2 \) and \( L_1 \triangle L_2 \) are derived by inspection as shown. Let us take the case of premeet first. Observe that a \( \Sigma^*_L \)-projection can be taken in \( L_1 \triangleleft L_2 \) only when both \( L_1 \) and \( L_2 \) can follow the same projected sequence (for example, \( \alpha \beta \)). In this case, the behavior of the substructure reached in \( L_1 \triangleleft L_2 \) is the premeet of that of \( L_1 \) and \( L_2 \)—hence the definition is recursive—and connections at the interface of levels \( k = 0 \) and \( k = 1 \) are obtained literally by taking the intersection of the associated connections in \( L_1 \) and \( L_2 \). A dual argument explains the structure of \( L_1 \triangle L_2 \), and observe that in this case a \( \Sigma^*_L \)-projection can be taken if either of \( L_1 \) or \( L_2 \) can follow that sequence. In cases where exclusively one is able to track the sequence, the substructure reached in \( L_1 \triangle L_2 \) is the exact copy of the respective component in \( L_1 \) or \( L_2 \).

**Proposition 3.3** For all \( L_1, L_2 \in \mathcal{L}_k \), \( L_1 \triangle L_2 \) and \( L_1 \triangleleft L_2 \) satisfy the following properties:

- \( L_i \preceq L_1 \triangledown L_2 \ &: \ (\forall T \in \mathcal{L}_k) L_i \preceq T \Rightarrow L_1 \triangledown L_2 \preceq T \ (i = 1, 2) \)

- \( L_1 \triangle L_2 \preceq L_i \ &: \ (\forall T \in \mathcal{L}_k) T \preceq L_i \Rightarrow T \preceq L_1 \triangle L_2 \ (i = 1, 2) \)

**Proof for \( \triangleleft \):**

It is straightforward to verify that \( L_i \preceq L_1 \triangledown L_2 ; \ i = 1, 2 \). Thus we only need to show

\[
(\forall T \in \mathcal{L}_k) L_1 \preceq T \& L_2 \preceq T \Rightarrow L_1 \triangledown L_2 \preceq T
\]

As usual we prove the result by induction on the level index \( k \):
Figure 3.6: Illustration of premeet and prejoin in $\mathcal{L}_0$ ($p = 2$, $\Sigma_0 = \{\alpha, \beta, \gamma\}$, and $\Sigma_1 = \{a, b\}$)
- \( k = p \): The result is obvious since \( \{\epsilon\} \) is the only nonempty member of \( \mathcal{L}_p \).

- \( 0 \leq k < p \):

Let \( v \in L_1 \sim L_2 \) and \( w \in T \) such that \( v \equiv_k w \). We have to show \( y \preceq z \), where \( y := \text{last}(v) \) and \( z := \text{last}(w) \). Regarding the definition of prejoin, Page 36, it might be the case that:

1. \((C_1 \& C_2)\) i.e. there exist \( v_i \in L_i \), \( i = 1, 2 \), such that

   \[ v_1 \equiv_k v_2 \equiv_k v \& y = y_1 \sim y_2 \quad (3.2) \]

   where \( y_i := \text{last}(v_i) \), \( i = 1, 2 \). Since \( L_i \preceq T \) and \( v_i \equiv_k w \) we have:

   \[ y_i \preceq z; \ i = 1, 2 \quad (3.3) \]

(3.2) and (3.3) together imply

\[ L_{y_1} \preceq L_z \& L_{y_2} \preceq L_z \& L_y = L_{y_1} \sim L_{y_2} \]

by induction assumption, therefore, \( L_y \preceq L_z \).

Now take arbitrary \( v' \in L_y \) and \( w' \in L_z \) such that \( v' \equiv_{k+1} w' \). We verify the output map containment by conditioning on the type of \( v' \):

(a) if \((C_1' \& C_2')\), then there exist \( v'_i \in L_{y_i} \), \( i = 1, 2 \), such that

   \[ v'_1 \equiv_{k+1} v'_2 \equiv_{k+1} v' \] and

   \[ \theta_y(v') = \theta_{y_1}(v'_1) \cup \theta_{y_2}(v'_2) \]

   By (3.3) and since \( v'_i \equiv_{k+1} w' \) it follows that: \((i = 1, 2)\)

   \[ \theta_{y_i}(v'_i) \subseteq \theta_z(w') \]

   Taking the union over \( i = 1, 2 \), we will get:

   \[ \theta_y(v') \subseteq \theta_z(w') \]

(b) if, say, \((C_1' \& \neg C_2')\), then there exists \( v'_i \in L_{y_i} \) with \( v'_i \equiv_{k+1} v' \) such that

   \[ \theta_y(v') = \theta_{y_1}(v'_1) \].
Again since \( v_1' \equiv_{k+1} w' \) we will have
\[
\theta_y(v') = \theta_{y_1}(v'_1) \subseteq \theta_z(w')
\]

In both cases, therefore, \( y \preceq z \).

2. say, \((C_1 \land \neg C_2)\), i.e. there exists \( v_1 \in L_1 \) such that \( v_1 \equiv_k v \) and \( y = y_1 \),
where \( y_1 := \text{last}(v_1) \).

Since \( v_1 \equiv_k w \), it follows from \( L_1 \preceq T \) that \( y_1 \preceq z \) and therefore \( y \preceq z \).

**Proof for \( \wedge \):** (Skip unless you doubt the duality of \( \bar{v} \) and \( \bar{\wedge} \! \! \)!

It is straightforward to verify that \( L_1 \wedge L_2 \preceq L_i \); \( i = 1, 2 \). Thus we only need to show
\[
(\forall T \in L_k) T \preceq L_1 \land T \preceq L_2 \Rightarrow T \preceq L_1 \wedge L_2
\]

We prove the result by induction on the level index \( k \):

- \( k = p \): The result is obvious since \( \{e\} \) is the only nonempty member of \( L_p \).

- \( 0 \leq k < p \):

Let \( v \in L_1 \wedge L_2 \) and \( w \in T \) such that \( v \equiv_k w \). We have to show \( z \preceq y \), where \( y := \text{last}(v) \) and \( z := \text{last}(w) \). By definition there exist \( v_i \in L_i, i = 1, 2 \), such that

\[
v_1 \equiv_k v_2 \equiv_k v \land y = y_1 \wedge y_2
\]

where \( y_i := \text{last}(v_i), i = 1, 2 \). Since \( T \preceq L_i \) and \( v_i \equiv_k w \) we have \( (i = 1, 2) \):

\[
z \preceq y_i; \ i = 1, 2
\]

(3.4) and (3.5) together imply
\[
L_z \preceq L_{y_1} \land L_z \preceq L_{y_2} \land L_y = L_{y_1} \wedge L_{y_2}
\]

by induction assumption, therefore, \( L_z \preceq L_y \).
Now take arbitrary $v' \in L_y$ and $w' \in L_z$ such that $v' \equiv_{k+1} w'$. To verify the output map relation, observe that since $v' \in L_{y_1} \wedge L_{y_2}$, there exist $v'_i \in L_{y_i}$, $i = 1, 2$, such that $v'_i \equiv_{k+1} v'_2 \equiv_{k+1} v'$ and

$$\theta_y(v') = \theta_{y_1}(v'_1) \cap \theta_{y_2}(v'_2)$$

By (3.5) and since $v'_i \equiv_{k+1} w'$ it follows that: ($i = 1, 2$)

$$\theta_y(w') \subseteq \theta_{y_i}(v'_i)$$

Taking the intersection over $i = 1, 2$, we will get:

$$\theta_y(w') \subseteq \theta_y(v')$$

and this completes the proof.

Q.E.D.

**Corollary 3.4** Join and meet in $\langle L_k, \sim, \preceq \rangle$ and $\langle X_k, \sim, \preceq \rangle$ exist, and are defined according to:

- $(\forall L_1, L_2 \in L_k)[L_1] \vee [L_2] := [L_1 \lor L_2] \& [L_1] \wedge [L_2] := [L_1 \land L_2]$

- $(\forall x_1, x_2 \in X_k)[x_1] \vee [x_2] := [x_1 \lor x_2] \& [x_1] \wedge [x_2] := [x_1 \land x_2]$

**Proof:** First observe that, say, if $L_1 \sim L'_1$ and $L_2 \sim L'_2$, then $[L_1 \lor L_2] = [L'_1 \lor L'_2]$ because

$$L'_i \preceq L'_1 \lor L'_2 \& L_i \preceq L'_i \Rightarrow L_i \preceq L'_1 \lor L'_2; \quad i = 1, 2 \Rightarrow L_1 \lor L_2 \preceq L'_1 \lor L'_2.$$  

Similarly, $L'_1 \lor L'_2 \preceq L_1 \lor L_2$ and therefore, $L'_1 \lor L'_2 \sim L_1 \lor L_2$.

Now for $T \in L_k$, if $[L_i] \preceq [T]$ we will have:

$$L_i \preceq T; \quad i = 1, 2 \Rightarrow L_1 \lor L_2 \preceq T \Rightarrow [L_1] \lor [L_2] = [L_1 \lor L_2] \preceq [T]$$

This proves that $[L_1] \lor [L_2] = [L_1 \lor L_2]$. The other cases can be verified similarly.

Thus, as seen and probably expected, $\lor$ and $\land$ define join and meet on $L_k / \sim$ simply because in the domain of sets, union and intersection are join and meet with respect to subset inclusion, respectively.

**Generalization** We generalize join and meet to arbitrary collections $\{[L_i]\}_{i \in \ell}$ and $\{[y_i]\}_{i \in \ell}$.
Definition 3.7 (Join-generalized)

- Let $I$ be an arbitrary index set and $L_i$, $i \in I$, be nonempty languages in $\mathcal{L}_k$, $0 \leq k \leq p$. We define $\bigvee_{i \in I}[L_i] := [\bigvee_{i \in I}L_i]$, where $L \in \mathcal{L}_k$ satisfies $L = \bigvee_{i \in I}L_i$ if and only if
  
  - $k = p$ : $L = \{\varepsilon\}$
  
  - $0 \leq k < p$ : For every $v \in L$, $\bigcup_{i \in I}^\dagger C_i$ holds, and
    
    $y = \bigvee_{j \in J} y_j$
    
    where $C_i := (\exists v_i \in L_i)v_i \equiv_k v$, $J := \{i \in I \mid C_i\}$, $y := \text{last}(v)$, and
    
    $y_j := \text{last}(v_j); j \in J$.

- Let $I$ be an arbitrary index set and $y_i$, $i \in I$, be tokens in $\mathcal{X}_k$, $0 \leq k < p$. We define $\bigvee_{i \in I}[y_i] := [\bigvee_{i \in I}y_i]$, where $y \in \mathcal{X}_k$ satisfies $y = \bigvee_{i \in I}y_i$ if and only if
  
  1. $L_y = \bigvee_{i \in I}L_{y_i}$
  2. $(\forall v' \in L_y)\theta_y(v') = \bigcup_{j \in J} \theta_{y_j}(v'_j)$

  where $C_i' := (\exists v'_i \in L_{y_i})v'_i \equiv_k v'$ and $J := \{i \in I \mid C_i'\}$.

Definition 3.8 (Meet-generalized)

- Let $I$ be an arbitrary index set and $L_i$, $i \in I$, be nonempty languages in $\mathcal{L}_k$, $0 \leq k \leq p$. We define $\bigwedge_{i \in I}[L_i] := [\bigwedge_{i \in I}L_i]$, where $L \in \mathcal{L}_k$ satisfies $L = \bigwedge_{i \in I}L_i$ if and only if
  
  - $k = p$ : $L = \{\varepsilon\}$
  
  - $0 \leq k < p$ : $(\forall v \in L)(\forall i \in I)(\exists v_i \in L_i)v_i \equiv_k v$ & $y = \bigwedge_{j \in J} y_j$

  where $y := \text{last}(v)$ and $y_i := \text{last}(v_i); i \in I$.

- Let $I$ be an arbitrary index set and $y_i$, $i \in I$, be tokens in $\mathcal{X}_k$, $0 \leq k < p$. We define $\bigwedge_{i \in I}[y_i] := [\bigwedge_{i \in I}y_i]$, where $y \in \mathcal{X}_k$ satisfies $y = \bigwedge_{i \in I}y_i$ if and only if

\footnote{\text{Here $\bigcup$ denotes arbitrary "or".}}
1. \( L_y = \bigwedge_{i \in I} L_{y_i} \)

2. \( (\forall v' \in L_y)[(\forall i \in I, v'_i \in L_{y_i}) v'_i \equiv_{k+1} v'_i \Rightarrow \theta_y(v') = \bigcap_{j \in I} \theta_{y_j}(v'_j)] \)

Notes:

- By definition we have:

\[
(\forall L \in L_k) L \lor \emptyset = L \land L \land \emptyset = \emptyset.
\]

- In the example of Page 37 it is seen that

\[
\mathcal{I}(L_1 \triangledown L_2) = \mathcal{I}(L_1) \cap \mathcal{I}(L_2) \text{ and }
\]

\[
\mathcal{I}(L_1 \triangledown L_2) = \mathcal{I}(L_1) \cup \mathcal{I}(L_2)
\]

It should be noted, however, that the latter is not valid in general, as \( \mathcal{I}(L_1) \cup \mathcal{I}(L_2) \) may not be expressible by a structured language. For example, consider the languages \( L_1 \) and \( L_2 \) shown in Figure 3.7 together with their prejoin \( L_1 \triangledown L_2 \). Also shown in the figure is the flat language \( \mathcal{I}(L_1) \cup \mathcal{I}(L_2) \), which is a proper subset of \( \mathcal{I}(L_1 \triangledown L_2) \).

**Important Note:** As mentioned earlier, the equivalence relation \( \sim \) captures the equality of behaviors among languages and tokens. Thus, \( L_1 \) and \( L_2 \) are “equal up to relabeling of tokens” whenever \( L_1 \sim L_2 \). With a little abuse of notation, in the rest of this work we write \( L_1 = L_2 \) when what we really mean is \( L_1 \sim L_2 \) or equivalently, \([L_1] = [L_2] \ (\text{modulo } \sim)\). Within this loosened terminology, one can use \( \preceq \), \( \lor \) and \( \land \) in place of \( \leq \), \( \triangledown \) and \( \triangledown \), respectively. Therefore, It is understood that, e.g., \( L_1 \lor L_2 \) is a representative from the cell \([L_1 \triangledown L_2] = [L_1] \lor [L_2] \).

### 3.4 Controllability

We now introduce the notion of “control” into our setup as an extension of the Ramadage and Wonham supervisory control theory for discrete-event systems [RW87].
We assume that the alphabet $\Sigma_k$, $0 \leq k < p$, is partitioned into two disjoint subsets, controllable events and uncontrollable events:

$$\Sigma_k =: \Sigma_{k,c} \cup \Sigma_{k,u}$$

A controllable event $\sigma \in \Sigma_{k,c}$ represents a system action which can be disabled by some external agent—called supervisor—in order to restrict the system behavior, usually in compliance with some (safety) requirements. In other words, controllable events collectively are the knobs one is provided with for tuning the system.

The supervisor in charge observes the sequence of events generated by the system—seen as a generator of symbols—as it unfolds, and can influence its future evolution by disabling a subset of controllable events which are eligible to occur at that point. Keeping any of these events enabled does not have any implications for their occurrence. In general, our models do not specify which event is to be executed next.
when there is more than one candidate available. This is an aspect of the system behavior which is obscured from the system analyst and therefore our models are nondeterministic in this sense.

Uncontrollable events, in contrast, cannot be controlled (read "disabled") by the supervisor. As a well known example, the possibility of failure in a process is usually modeled by the uncontrollable event "fail", since it is totally unpredictable if it is going to happen before the process completes normally, yet the possibility is always present and cannot be avoided.

Thus, applying supervisory control to a system will always restrict the open loop behavior; in other words, if $E$ represents the closed loop behavior of the system $L$, then $E \preceq L$. But notice that not all sublanguages of $L$ can be implemented by a supervisory control. The following definition is fundamental in our study, and it characterizes the class of sublanguages of a given structured language which can be synthesized by some supervisory control.

**Definition 3.9 (Controllability)** Let $E \preceq L \in L_k$, $0 \leq k \leq p$. $E$ is said to be controllable with respect to $L$ iff:

- $k = p : true$

- $0 \leq k < p :$

  $(\forall u \in E, v \in L) u \equiv_k v \Rightarrow x$ is controllable with respect to $y$

  where $x := last(u)$ and $y := last(v)$.

For $x \preceq y \in X_k$, $x$ is controllable with respect to $y$ iff:

1. $L_x$ is controllable with respect to $L_y$, and

2. $(\forall u' \in L_x, v' \in L_y) u' \equiv_{k+1} v' \Rightarrow \theta_x(u') \cap \Sigma_{k,u} = \theta_y(v') \cap \Sigma_{k,u}$. 

The definition basically requires generation of the output event $\sigma \in \Sigma_{k,u}$ by a string $u'$ in substructure $x$ of $E$ if the "corresponding" string $v'$ in the "corresponding" substructure $y$ of $L$ generates that event (Figure 3.8).
Figure 3.8: Illustration of controllability

Notes:

1. When $x \leq y$, we have $L_x \leq L_y$ and

   $$(\forall u' \in L_x, v' \in L_y)u' \equiv_{k+1} v' \Rightarrow \theta_x(u') \cap \Sigma_{k,u} \subseteq \theta_y(v') \cap \Sigma_{k,u}$$

Therefore, the second condition in the definition of controllability of $x$ with respect to $y$ merely requires inclusion in the other direction:

   $$(\forall u' \in L_x, v' \in L_y)u' \equiv_{k+1} v' \Rightarrow \theta_x(u') \cap \Sigma_{k,u} \supseteq \theta_y(v') \cap \Sigma_{k,u}$$

which can be written as

   $$(\forall u' \in L_x, \sigma \in \Sigma_{k,u})\sigma \in \theta_y(u') \Rightarrow \sigma \in \theta_x(u')$$  \hspace{1cm} (3.6)

where for any $u' \in L_x$ we have denoted by $v'$ the unique string of $L_y$ which satisfies $u' \equiv_{k+1} v'$ (see Lemma 3.2, Page 29).

2. In RW theory, $E \subseteq L$ is controllable with respect to $L$ iff

   $$(\forall \sigma \in \Sigma_u, s \in E)s\sigma \in L \Rightarrow s\sigma \in E$$  \hspace{1cm} (3.7)

Now suppose $E, L \in \mathcal{L}_k$, and $E \leq L$. A not well-thought generalization of (3.7) could be:

   $$(\forall \sigma \in \Sigma_{k,u}, u \in E)u\sigma \in L \Rightarrow u\sigma \in E$$  \hspace{1cm} (3.8)
where for any \( u \in E \) we have denoted by \( v \) the unique string of \( L \) satisfying \( u \equiv_k v \).

Let us discuss how this condition is compared with the one we used in the definition.

Suppose \( E \) is controllable with respect to \( L \), and take arbitrary \( \sigma \in \Sigma_{k,u} \) and \( u \in E \). Let \( v \in L \) such that \( u \equiv_k v \). If \( v \sigma \in L \), by semantics of \( L_k \) we will have

\[
(\exists v' \in L_y) \sigma \in \theta_y(v')
\]  

(3.9)

Take any of these \( v' \) in \( L_y \). It might be the case that:

(a) \((\exists u' \in L_x) u' \equiv_{k+1} v'\). Then by (3.6) and (3.9) we get \( \sigma \in \theta_x(u') \) and therefore \( u \sigma \in E \) (Figure 3.9).

(b) \( \neg(\exists u' \in L_x) u' \equiv_{k+1} v' \), i.e. at some point during the evolution of \( v' \) in \( L_y \), the language \( L_x \) gave up tracking the sequence (Figure 3.10).

Figure 3.9: Controllable: \( u \in E \& \sigma \in \Sigma_{k,u} \& v \sigma \in L \Rightarrow u \sigma \in E \)

Figure 3.10: Controllable, though \( u \in E, \sigma \in \Sigma_{k,u}, v \sigma \in L \) but \( u \sigma \notin E \)
Returning to (3.9), if for all $v'$ with the specified property case (b) applies, we can say that the system event $\sigma$ is blocked by some "feasible" supervisory control at the subsystem level, and therefore having found

$$u \in E, \sigma \in \Sigma_{k,u}, v\sigma \in L \text{ and } u\sigma \notin E$$

does not violate the principles of supervisory control.

On the other hand, if for some of these $v'$ case (a) holds, condition (3.8) will be satisfied for this particular pair of $(u, \sigma)$.

3. The above analysis might leave the impression that (3.8) is too strong a criterion for deciding controllability, since it overlooks the fact that restricting a subsystem behavior may result in blocking events at the system level if strings generating those events can be excluded from the closed loop behavior of the subsystem. But it turns out that in some cases this condition is not strong enough to serve the purpose. For example, in Figure 3.11 condition (3.8) is satisfied, yet no supervisory control is capable of preventing $\alpha$ from occurring at state "00". (In our transition graphs, we mark a controllable transition by crossing its arrow by a tick. So in this example, $\alpha \in \Sigma_{0,u}$, etc). The bottom line is that (3.8) is too aggregated a criterion for deciding controllability in the structured setup.

Figure 3.11: Uncontrollable, despite having $(\forall \sigma \in \Sigma_{k,u}, u \in E) v\sigma \in L \Rightarrow u\sigma \in E$

\[\text{i.e. a transition labeled by a controllable event.}\]
4. The definition of controllability allows any combination of control actions at different levels and their interfaces. For example, the supervisor in charge of the system $L$ in Figure 3.12 can prevent the occurrence of the upper $\alpha$ by disabling the $a$ transition preceding it, leave the one right at the start up intact, and directly disable the lower one, yielding the closed loop structure modeled by $E$.

![Diagram of $L$ and $E$]

Figure 3.12: $E$ is controllable with respect to $L$

**Example:** Let us apply the steps of Definition 3.9 to see if $E$ of Figure 3.12 is controllable with respect to $L$. For brevity we denote "controllable with respect to" by $C$. We have $p = 2$ and:

$$\Sigma_{0,c} = \{\alpha\}; \; \Sigma_{0,u} = \emptyset$$

$$\Sigma_{1,c} = \{\alpha\}; \; \Sigma_{1,u} = \{b\}$$

It can be verified that $E \preceq L$. In the procedural checks presented in the next page, the recursive nature of the definition has been illustrated by indenting the lines appropriately.

---

1In this work we do not discuss the implementation aspects of the supervisory control, and will leave this issue to the reader's imagination. It can be thought of as a hierarchy of supervisors (one for $L$, one for $L_y$ for every $y$ appearing in the definition hierarchy of $L$, etc) communicating through the output functions.
ECL?

$x_0C y_0$?

$L_{x_0}C L_{y_0}$?

$x_{00}C y_{00}$?

$L_{x_{00}}C L_{y_{00}} \checkmark$

$\theta_{x_{00}}(\varepsilon) \cap \Sigma_{1,u} = \{b\} = \theta_{y_{00}}(\varepsilon) \cap \Sigma_{1,u} \checkmark$

$x_{02}C y_{02}$?

$L_{x_{02}}C L_{y_{02}} \checkmark$

$\theta_{x_{02}}(\varepsilon) \cap \Sigma_{1,u} = \emptyset = \theta_{y_{02}}(\varepsilon) \cap \Sigma_{1,u} \checkmark$

$\theta_{x_{0}}(x_{00}) \cap \Sigma_{0,u} = \emptyset = \theta_{y_{0}}(y_{00}) \cap \Sigma_{0,u} \checkmark$

$\theta_{x_{0}}(x_{00} b x_{02}) \cap \Sigma_{0,u} = \emptyset = \theta_{y_{0}}(y_{00} b y_{02}) \cap \Sigma_{0,u} \checkmark$

$x_1C y_1$?

$L_{x_1}C L_{y_1}$?

$x_{10}C y_{10}$?

$L_{x_{10}}C L_{y_{10}} \checkmark$

$\theta_{x_{10}}(\varepsilon) \cap \Sigma_{1,u} = \emptyset = \theta_{y_{10}}(\varepsilon) \cap \Sigma_{1,u} \checkmark$

$\theta_{x_{1}}(x_{10}) \cap \Sigma_{0,u} = \emptyset = \theta_{y_{1}}(y_{10}) \cap \Sigma_{0,u} \checkmark$
We conclude this section by stating the following result which claims that the notion of controllability in the hierarchical setup is consistent with that of [RW87]: if the language $E \in \mathcal{L}_k$ is controllable with respect to $L$, one can find an omniscient RW supervisor which restricts $\mathcal{I}(L)$ to $\mathcal{I}(E)$.

**Proposition 3.4** For $E \sqsubseteq L \in \mathcal{L}_k$, if $E$ is controllable with respect to $L$, then $\mathcal{I}(E)$ is controllable with respect to $\mathcal{I}(L)$ in the usual RW sense.

**Proof:** We have to show

$$\left( \forall s \in \mathcal{I}(E), \sigma \in \Sigma_{k,p,u} \right) s \sigma \in \mathcal{I}(L) \Rightarrow s \sigma \in \mathcal{I}(E)$$

We prove this result by induction on the level index $k$:

- $k = p$: trivially valid since $\emptyset$ and $\{e\}$ are the only choices for $E$ and $L$.

- $0 \leq k < p$: Let $s \in \mathcal{I}(E)$ and $\sigma \in \Sigma_{k,p,u}$. Note that $s \in \mathcal{I}(L)$ since $\mathcal{I}(E) \subseteq \mathcal{I}(L)$.

  By Corollary 3.2 of Page 30, there are unique $u \in E$ and $v \in L$ satisfying $u \equiv_k v$ such that $s \in \mathcal{I}(u) \cap \mathcal{I}(v)$. Let $u$ and $v$ have the form

$$u := x_0 \sigma_0 x_1 \cdots \sigma_{n-1} x_n; \quad v := y_0 \sigma_0 y_1 \cdots \sigma_{n-1} y_n$$

where $n$ is some natural number. Then $s$ can be factorized accordingly into:

$$s = s_0 \sigma_0 s_1 \cdots \sigma_{n-1} s_n$$

where $s_j \in \mathcal{I}(L_{x_j}) \cap \mathcal{I}(L_{y_j}); \quad 0 \leq j \leq n$.

If $s \sigma \in \mathcal{I}(L)$, we consider two cases:

1. $\sigma \not\in \Sigma_k$.

   Then it must be the case that $s_n \sigma \in \mathcal{I}(L_{y_n})$. We have:

   $$s_n \in \mathcal{I}(L_x), \quad \sigma \in \Sigma_{k+1,p,u}, \quad s_n \sigma \in \mathcal{I}(L_{y_n})$$

   by induction assumption, therefore, $s_n \sigma \in \mathcal{I}(L_x)$, which in turn implies $s \sigma \in \mathcal{I}(E)$.

\[\text{\footnotesize\[In this case we must have } k + 1 < p.\]

51
2. \( \sigma \in \Sigma_k \).

Since \( s_n \in \mathcal{I}(L_{x_n}) \cap \mathcal{I}(L_{y_n}) \) and \( L_{x_n} \preceq L_{y_n} \), by invoking the corollary once more we will get unique \( u' \in L_{x_n} \) and \( v' \in L_{y_n} \) such that

\[
u' \equiv_{k+1} v' \& s_n \in \mathcal{I}(u') \cap \mathcal{I}(v')\]

Since \( s\sigma \in \mathcal{I}(L) \), it must be the case that \( \sigma \in \theta_{y_n}(v') \). This, together with controllability of \( x_n \) with respect to \( y_n \) (see (3.6), Page 46) imply \( \sigma \in \theta_{x_n}(u') \) and therefore, \( s\sigma \in \mathcal{I}(E) \).

3.5 Upper Semilattice Property of the Controllable Subclass

Theoretically, we can regard a sublanguage \( E \) of \( L \) as the "desired" closed-loop behavior of \( L \) under supervisory control. When \( E \) is not controllable with respect to \( L \), therefore, the requirement \( E \) cannot be exactly met. Still, the requirement is conservatively met if we can come up with some \( K \preceq E \) which is controllable with respect to \( L \). Now one might naturally ask: Since there may be several such \( K \)'s available, is it possible to meet this requirement in an optimal (read "minimally restrictive") fashion? The answer is positive if the controllable (with respect to \( L \)) sublanguages of \( E \) form a complete upper sub-semilattice with respect to the partial order \( \preceq \).

The following result is fundamental in our theory. It states that the class of controllable sublanguages of a given structured language form a complete upper sub-semilattice with respect to the partial order \( \preceq \).

**Proposition 3.5** Controllability in \( \mathcal{L}_k \) is preserved under arbitrary join. In other words, if \( K_i \preceq L \in \mathcal{L}_k \) is controllable with respect to \( L \) for all \( i \) in some index set \( I \), then \( \bigvee_{i \in I} K_i \) is also controllable with respect to \( L \).

**Proof:** Firstly, since \( K_i \preceq L \) for \( i \in I \), it is immediate that \( \bigvee_{i \in I} K_i \preceq L^\dagger \).

\[\text{1} \text{That is because } (\mathcal{L}_k/ \sim, \preceq) \text{ is a complete lattice. Therefore, rigorously, } \bigvee_{i \in I} [K_i] \preceq [L] \text{ when } [K_i] \preceq [L].\]
We prove the result on controllability by induction on the level index \( k \):

- \( k = p \): Trivially valid since \( \emptyset \) and \( \{\epsilon\} \) are the only choices for the languages involved.

- \( 0 \leq k < p \): We have to show

\[
(\forall w \in \bigvee_{i \in I} K_i, v \in L) w \equiv_k v \Rightarrow \text{last}(w) \text{ is controllable wrt last}(v) \tag{3.10}
\]

Take \( w \in \bigvee_{i \in I} K_i \) and \( v \in L \) such that \( w \equiv_k v \) and let \( z := \text{last}(w) \) and \( y := \text{last}(v) \). Then the validity of (3.10) is tantamount to establishing that

\[
L_z \text{ is controllable with respect to } L_y, \tag{3.11}
\]

and

\[
(\forall w' \in L_z, v' \in L_y) w' \equiv_{k+1} v' \Rightarrow \theta_z(w') \cap \Sigma_{k,u} = \theta_y(v') \cap \Sigma_{k,u} \tag{3.12}
\]

Let \( C_i := (\exists w_i \in K_i) w_i \equiv_k w, \ i \in I \), be a boolean, and define \( J \subseteq I \) according to \( J := \{i \in I \mid C_i\} \). Then by Definition 3.7 we will have:

\[
z = \bigvee_{j \in J} z_j \tag{3.13}
\]

where \( z_j := \text{last}(w_j) \) for \( j \in J \). Since \( K_j \) is controllable with respect to \( L \) and \( w_j \equiv_k w \equiv_k v \) it follows that

\[
z_j = \text{last}(w_j) \text{ is controllable with respect to } y = \text{last}(v) \tag{3.14}
\]
The language part of (3.13) and (3.14) would read:

\[ L_z = \bigvee_{j \in J} L_{z_j} \] is controllable with respect to \( L_y; j \in J \)

by induction assumption, therefore, \( L_z \) is controllable with respect to \( L_y \), i.e. (3.11) holds true.

For the output map side, take \( w' \in L_z \) and \( v' \in L_y \) such that \( w' \equiv_{k+1} v' \) and for \( j \in J \) let \( C'_j := (\exists w'_j \in L_{z_j}) w'_j \equiv_{k+1} w' \). Define \( J' := \{ j \in J \mid C'_j \} \). Then we will have:

\[ \theta_z(w') = \bigcup_{j \in J'} \theta_{z_j}(w'_j) \]

For \( j \in J' \), we have \( w'_j \equiv_{k+1} w' \equiv_{k+1} v' \) and therefore (3.14) implies

\[ \theta_{z_j}(w'_j) \cap \Sigma_{k,u} = \theta_y(v') \cap \Sigma_{k,u}; j \in J' \] (3.15)

Taking union over \( J' \), we will get:

\[ \bigcup_{j \in J'} (\theta_{z_j}(w'_j) \cap \Sigma_{k,u}) = \theta_y(v') \cap \Sigma_{k,u} \]

and therefore

\[ (\bigcup_{j \in J'} \theta_{z_j}(w'_j)) \cap \Sigma_{k,u} = \theta_y(v') \cap \Sigma_{k,u} \]

and since \( \theta_z(w') = \bigcup_{j \in J'} \theta_{z_j}(w'_j) \), (3.12) immediately follows. Q.E.D. 

**Theorem 3.1** Let \( E, L \in L_k \), and \( E \preceq L \). Define

\[ C_L(E) := \{ K \in L_k \mid K \preceq E \text{ & } K \text{ is controllable with respect to } L \} \]

Then \( (C_L(E), \preceq) \) is a complete upper sub-semilattice of \( (L_k, \preceq) \), with the supremal element:

\[ \text{Sup} C_L(E) = \bigvee_{K \in C_L(E)} K. \]
Proof: $C_L(E)$ is nonempty since $\emptyset \in C_L(E)$, and so the result is immediate from the proposition.

Note: An exact statement of the theorem would be:

Let $E, L \in \mathcal{L}_k$, and $E \preceq L$. Define

$$C_L(E) := \{K \in \mathcal{L}_k \mid K \preceq E \& K \text{ is controllable with respect to } L\}$$

Then $(C_L(E)/ \sim, \preceq)$ is a complete upper sub-semilattice of $(\mathcal{L}_k/ \sim, \preceq)$, with the supremal element:

$$SupC_L(E) = \bigvee_{K \in C_L(E)} [K].$$

As in the standard theory, $SupC_L(E)$ represents the closed-loop behavior of the system, where under the influence of some supervisory control, the behavior meets the requirement $E$ in a minimally restrictive fashion.

Example: Let $E$ and $L$ be given as shown in Figure 3.13.

![Diagram](image)

Figure 3.13: The system $L$ and its requirement $E$

The controllable sublanguages of $E$ are given in Figure 3.14 and therefore, $C_L(E) = \{\emptyset, K_0, K_1, K_2, K_3\}$. The Hasse diagram of Figure 3.15 shows the sub-semilattice $(C_L(E), \preceq)$ graphically.
Figure 3.14: Controllable sublanguages of $E$

Figure 3.15: Sub-semilattice $(C_L(E), \preceq)$
Chapter 4

Computation of the Supremal Controllable Sublanguage

4.1 Introduction

In this chapter we will propose a fixed-point algorithm for computing the supremal controllable sublanguage of a given structured language. We devise an operator \( \Omega : \mathcal{L}_k \times \mathcal{L}_k \to \mathcal{L}_k \) with the property that the fixed-points of \( \Omega(\cdot, L) \) coincide with the controllable sublanguages of \( L \in \mathcal{L}_k \). Therefore, \( \text{SupC}_L(E) \) would be the largest of such fixed-points which is smaller than \( E \).

As we will see in Section 4.3, the operator \( \Omega \) is defined recursively, in the sense that \( \Omega(\cdot, L) \)'s operation on \( E \preceq L \in \mathcal{L}_k \) involves computing supremal controllable elements in \( \mathcal{L}_{k+1} \), which in turn can be thought of as the fixed-points of \( \Omega(\cdot, L_z) \) for appropriate \( z \in \mathcal{X}_k \) hanging under \( L \) in its definition hierarchy. (We opt to use the notation \( \Omega \) uniformly throughout, as opposed to labeling \( \Omega \) with the level index \( k \).)

Before setting out to define \( \Omega \), we spare a few pages to characterize a subclass of sublanguages of \( L \in \mathcal{L}_k \) in which the complexity of computing the supremal controllable sublanguage can be significantly reduced. Referring to Definition 3.9 of Page 45, observe that for deciding the controllability of \( E \) with respect to \( L \) one should in general investigate the output map inclusion among all corresponding subsystems at level \( k + 1 \) even when, informally speaking, things are looking fine at the system level \( k \),
i.e. for $u \in E$ and $v \in L$ with $u \equiv_k v$ and $\sigma \in \Sigma_{k,u}$, $v \sigma \in L$ implies $u \sigma \in E$. This is a serious drawback since our decision procedure does not enjoy separation of concerns at different levels, which is the cornerstone of hierarchical analysis.

What we propose in the next section is to specify requirements on the behavior of the system $L$ in the form of a hierarchy of structured languages, where each language in the requirement hierarchy targets the language part of a single item in the definition hierarchy of $L$, and is indifferent to the way the rest of the system behaves. This will amount to obviating the application of supervisory control at the interface of different levels. Thus, either no instance or all instances of the occurrence of an event $\sigma \in \Sigma_k$ is cut off at the output of a subsystem at level $k + 1$.

### 4.2 Top-Level Sublanguages

In this section we introduce a subclass of sublanguages of $L \in \mathcal{L}_k$ whose members satisfy a certain structural property. Informally speaking, we will focus our attention only on those $E \preceq L$ which, when seen as specifications of the desired behavior of the system, specify requirements only at the current level—indexed by $k$—and let the system behave freely elsewhere. Thus, if $x$ is a token in the definition hierarchy of $E$, $\mathcal{T}(x)$ is only as much restricted as needed for complying with the high level requirement $P_k(E) \subseteq P_k(L)$. For example, let $y \in L$ and $\Sigma_y$, the set of eligible events at $y$, be specified by $\Sigma_y = \{\alpha, \beta, \gamma\}$. If the requirement specification for $L$, given by $E \in \mathcal{L}_k$, is just meant for restricting $\Sigma_y$ to $\{\alpha, \beta\}$, then $E$ will contain a string of length one, say $x$ where $x \in \mathcal{X}_k$, such that:

$$L_x = L_y^\dagger \text{ and } \theta_x = \theta_y - \{\gamma\} \text{ (Figure 4.1)}.$$

**Definition 4.1** Let $E, L \in \mathcal{L}_k$, $0 \leq k \leq p$. $E$ is called a top level sublanguage of $L$, denoted by $E \preceq_T L$, iff

- $k = p : \text{true}$

\[\dagger\text{Reminder: i.e. } L_x \sim L_y.\]
Figure 4.1: \( E \) is a restriction of \( L \) only at level \( k \) when \( L_z = L_y \) & \( \theta_x = \theta_y - \{\gamma\} \)

- \( 0 \leq k < p \):
  1. \( E \subseteq P_k^{-1}P_k(L) \)
  2. \((\forall u \in E, v \in L) u \equiv_k v \Rightarrow L_x = L_y \) & \( \theta_x = \theta_y - (\Sigma_v - \Sigma_u) \)

where \( x := \text{last}(u) \), \( y := \text{last}(v) \), and \( \Sigma_u \) and \( \Sigma_v \) are the set of eligible events at \( u \in E \) and \( v \in L \), respectively:

\[
\Sigma_u := \{ \sigma \in \Sigma_k \mid u\sigma \in E \} \quad \text{and} \quad \Sigma_v := \{ \sigma \in \Sigma_k \mid v\sigma \in L \}.
\]

**Proposition 4.1**

\((\forall E, L \in \mathcal{L}_k) \ E \sim_T L \Rightarrow E \preceq L \)

**Proposition 4.2** Let \( E, L \in \mathcal{L}_k \) and \( E \sim_T L \). Then

\[
E = \text{Sup}\{S \preceq L \mid P_k(S) = P_k(E)\}
\]

In other words, the \( \Sigma^*_k \)-projection of a top level sublanguage \( E \in \mathcal{L}_k \) conveys enough information for reconstructing \( E \), using \( L \) as a template. This fact is not surprising at all, since top level sublanguages are "indifferent" to the details represented by structures at levels \( k + 1 \) downwards, and therefore no information will be lost by projecting the \( x \)'s out.

Proposition 4.2 provides an equivalent characterization of top level sublanguages, which has its own merits:

**Corollary 4.1** For all \( E, L \in \mathcal{L}_k \),

\[
E \preceq_T L \iff E = \text{Sup}\{S \preceq L \mid P_k(S) = E_k\} \text{ for some } E_k \subseteq P_k(L).
\]
Example: In Figures 4.2 and 4.3, \( \Sigma_0 = \{\alpha, \beta\} \) and \( \Sigma_1 = \{a, b\} \) \( (p = 2) \). The language \( E \preceq_T L \) is constructed based on \( E_0 \), the desired \( \Sigma_0 \)-projection of the system behavior, and the language \( L \) itself.

![Diagram of system L](image)

The Desired High-Level Behavior \( E_0 = P_0(E) \)

Figure 4.2: The system \( L \), together with the projection of its requirement

\[ \Rightarrow E : \]

![Diagram of top level sublanguage E](image)

Figure 4.3: Top level sublanguage \( E \), constructed based on the information in Fig 4.2

When \( E \preceq_T L \), the condition for controllability of \( E \) with respect to \( L \) will reduce to:

\[ (\forall u \in E, v \in L, \sigma \in \Sigma_{k,u}) u \equiv_k v \Rightarrow [v \sigma \in L \Rightarrow u \sigma \in E] \]

which is quite similar to that of the RW theory.

Remark: The idea of having top-level sublanguages as models for requirements is in harmony with the structured nature of complex systems, where one would expect the requirements to be specified in a distributed fashion, and the system parts to be dealt with “one at a time”.

60
Therefore, the problem of designing supervisory control for the whole system is broken down into a number of associated subproblems for each of the system components.

We argue that priority should be given to subproblems associated with components at lower levels. If the requirement corresponding to a component at level $k$—represented, say, by $L_y \in \mathcal{L}_{k+1}$—is not satisfiable (i.e. $SupC_{L_y}()$ applied to that requirement yields an empty result), it is the responsibility of the supervisor in charge of the father of $y$—say $L \in \mathcal{L}_k$, where $y$ appears in some string of $L$—to block access to $y$. Therefore, it would be sensible if one added all such newly imposed requirements to the list of the existing ones before aiming at designing a supervisory control for $L$ itself (see Figure 4.4).

Figure 4.4: Unsatisfiable requirement for $y_4$ affects the requirement for $L$
4.3 Defining $\Omega$: Towards Fixed-Point Characterization of the Supremal Controllable Element

Let $E \in L_k$ be an arbitrary sublanguage of $L$, and $E$ not be controllable with respect to $L$. Then, referring to Figure 4.5, observe that one can find $(u, v, u', v')$—related to each other as in the figure—such that:

$$\theta_x(u') \cap \Sigma_{k,u} \subseteq \theta_y(v') \cap \Sigma_{k,u}$$

Therefore, on its way to making $E$ controllable, $\Omega$ just removes those $u'$ from $L_x$ that violate the controllability condition, leaving us with:

$$S_{x,y} := \text{Sup}\{T' \leq L_x \mid (\forall u' \in L_x, v' \in L_y, w' \in T')$$

$$(u', v', w')_{k+1} \Rightarrow \theta_z(u') \cap \Sigma_{k,u} = \theta_y(v') \cap \Sigma_{k,u}\}$$

where for all $i$ and $x, \ldots, z \in (\Sigma_k \cup X_k)^*$, $(x, \ldots, z)_i$ denotes $x \equiv_i \cdots \equiv_i z$.

Figure 4.5: Weeding out those $u' \in L_x$ which make $E$ uncontrollable at $u$

Notice that the language $S_{x,y}$ is determined solely by the tokens $x$ and $y$. More precisely, $S_{x,y}$ is a maximal sublanguage of $L_x$ satisfying a predicate on the output functions $\theta_x$ and $\theta_y$.
In the meantime, we have to make sure that the removal of these illegal low level strings is carried out in a controllable fashion. So, assuming that we have the means to compute $SupC_{L_y}(S_{x,y})$ and the result is nonempty, in formulating $\Omega(E, L)$ we replace the language part of $x$ by $SupC_{L_y}(S_{x,y})$, and therefore

$$\Omega(E, L) := \text{Sup}\{T \preceq E \mid (\forall u \in E, v \in L, w \in T)(u, v, w)_k \Rightarrow L_z = SupC_{L_y}(S_{x,y})\}$$

where $x := \text{last}(u)$, $y := \text{last}(v)$, and $z := \text{last}(w)$. Notice that $L_z = SupC_{L_y}(S_{x,y})$ implicitly requires $SupC_{L_y}(S_{x,y}) \neq \emptyset$ by semantics of $L_k$. That means when $SupC_{L_y}(S_{x,y}) = \emptyset$, one cannot find a $w \in \Omega(E, L)$ such that $w \equiv_k u \equiv_k v$.

The following definition summarizes these steps:

**Definition 4.2** ($\Omega(\cdot, \cdot)$) Let $0 \leq k < p$. We define the partial function $\Omega : L_k \times L_k \rightarrow L_k$ according to:

For all $E \preceq L \in L_k$,

$$\Omega(E, L) := \text{Sup}\{T \preceq E \mid (\forall u \in E, v \in L, w \in T)(u, v, w)_k \Rightarrow L_z = SupC_{L_y}(S_{x,y})\}$$

where

$$x := \text{last}(u), \quad y := \text{last}(v), \quad z := \text{last}(w) \quad \text{and}$$

$$S_{x,y} := \text{Sup}\{T' \preceq L_z \mid (\forall u' \in L_z, v' \in L_y, w' \in T')$$

$$(u', v', w')_{k+1} \Rightarrow \theta_x(u') \cap \Sigma_{k,u} = \theta_y(v') \cap \Sigma_{k,u}\}$$

**Example:** *Illustrating the idea.* Here is one more example on how $\Omega$ makes a sublanguage look "more controllable". Take the strings $v = \cdots \alpha y \in L$ and $u = \cdots \alpha x \in E$ as shown in Figure 4.6. Observe that $u \equiv_k v$ and that with the given interpretation for $x$ and $y$, we have:

$$\theta_x(x_0ax_1bx_3) \cap \Sigma_{k,u} = \{\beta\} \subseteq \{\beta, \gamma\} = \theta_y(y_0ay_1by_3) \cap \Sigma_{k,u}$$

Therefore, $E$ is not controllable with respect to $L$. To find $K := \Omega(E, L)$, we first compute $S_{x,y}$, which is basically the largest sublanguage of $L_x$ that does not contain $\ldots$  

\footnote{In this sense the definition of $\Omega(\cdot, \cdot)$ is recursive: The result of $\Omega(E, L)$ depends on fixed-point computations on substructures.}
any string which has \(ab\) as the prefix of its \(\Sigma_{k+1}^*\)-projection. Such a language is given in the figure, and observe how the maximality of \(S_{x,y}\) has been manifested in the relations:

\[
\tilde{x}_0 = x_0 \text{ and } \begin{cases} L_{\tilde{x}_1} = L_{x_1} \\ \theta_{\tilde{x}_1} = \theta_{x_1} - \{b\} \end{cases}
\]

\[
L \quad \alpha \quad y \quad \gamma \quad \beta \quad \quad E \quad \alpha \quad x \quad \beta
\]

The same projections onto \(\Sigma_k^*\) \((\Sigma_k = \Sigma_{k,u} = \{\alpha, \beta, \gamma\})\)

\[
y \quad \bullet \quad a \quad y_1 \quad \beta \quad y_2 \quad \gamma \quad y_3 \quad \bullet \quad \beta
\]

\[
x \quad \bullet \quad a \quad x_1 \quad \beta \quad x_2 \quad \beta
\]

\[
\Rightarrow S_{x,y} : \quad \tilde{x}_0 \quad a \quad \tilde{x}_1 \quad S_{x,y}^L(S_{x,y}), \text{ say: } \quad \bullet \quad x_0 \quad \bullet
\]

Note: \(\tilde{x}_0 = x_0\) and \(\begin{cases} L_{\tilde{x}_1} = L_{x_1} \\ \theta_{\tilde{x}_1} = \theta_{x_1} - \{b\} \end{cases}\)

\[
\Rightarrow K := \Omega(E, L) \quad \alpha \quad z \quad \bullet \begin{cases} L_z = S_{x,y}^L(S_{x,y}) \\ \theta_z = \emptyset \end{cases}
\]

Figure 4.6: Computing \(K := \Omega(E, L)\)

Now we claim that \(S_{x,y}\) is not controllable with respect to \(L_y\). Take \(\tilde{x}_0 a \tilde{x}_1 \in S_{x,y}\) and \(y_0 a y_1 \in L_y\) and observe that \(\tilde{x}_0 a \tilde{x}_1 \equiv_{k+1} y_0 a y_1\). Since \(x_0 a x_1 b \in L_{x_1}\), we must have

\((\exists u'' \in L_{x_1}) b \in \theta_{x_1}(u'')\)
But $L_{z_2} \leq L_{y_1}$, therefore there exists a unique $v'' \in L_{y_1}$ such that $u'' \equiv_{k+2} v''$ and $b \in \theta_{y_1}(v'')$. Notice that $L_{z_2} = L_{z_1}$ and hence we have found $u' := \tilde{x}_0 a \tilde{x}_1 \in S_{x,y}$, $v' := y_0 a y_1 \in L_y$, $u'' \in L_{z_1}$, and $v'' \in L_{y_1}$ with $u' \equiv_{k+1} v'$ and $u'' \equiv_{k+2} v''$ such that

$$b \in \theta_{y_1}(v'') \cap \Sigma_{k+1,u}$$

and so the claim is proved. Thus, we need to compute its supremal controllable sublanguage, i.e. $\text{Sup} C_{L_y}(S_{x,y})$. Assume that the interpretation given for $y_1$ is such that there is no feasible way of disconnecting $b$ at the output of $y_1$. Thus, in any controllable sublanguage of $S_{x,y}$ the access to this subsystem must be blocked, namely, the controllable event $a$ has to be disabled. Again the maximality of $\text{Sup} C_{L_y}(S_{x,y})$ is reflected through the relations:

$$L_{z_0} = L_{z_0} \text{ and } \theta_{z_0} = \theta_{z_0} - \{a\}.$$ 

Finally, if all similar computations for the prefixes of $\cdots \alpha z$ have yielded nonempty results, the maximality of $\Omega(E, L)$ requires having $\cdots \alpha z$ among its strings, where $L_z = \text{Sup} C_{L_y}(S_{x,y})$ and $\theta_z = \emptyset$. 

Remarks:

1. The supremum operator in the definition of $\Omega$ returns the "largest" language with the given property, in the sense that:

(a) Its strings have maximal length. A particular thread terminates when $\text{Sup} C_{L_y}(S_{x,y}) = \emptyset$.

(b) Let $K := \Omega(E, L)$, and for any $w \in K$ let $z := \text{last}(w)$. Then by virtue of the supremum operator, $\theta_z$ will contain the maximal subset of events it is allowed to possess according to:

$$\forall w' \in L_z, u' \in L_z(u', w')_{k+1} \Rightarrow \theta_z(w') = \theta_z(u') - \{\sigma \in \Sigma_k \mid w \sigma \notin K\}$$

In the example, we got $\theta_z = \theta_z - \Sigma_k = \emptyset$.

[Otherwise, if for $w' \in L_z, u' \in L_z$ with $(u', w')_{k+1}$

$$(\exists \sigma \in \Sigma_k) \sigma \in \theta_z(u') \land \sigma \notin \theta_z(w') \land w \sigma \in K$$

65
then create a new language $\hat{K}$ with exactly the same structure as $K$—in particular, $L_z = L_z$—except for

$$\theta_z(w') = \theta_z(w') \cup \{\sigma\}$$

Such a construction is valid since $w\sigma \in K$ and therefore $\sigma$ can be included in the output set of any $wf$-string of $L_z$ without violating the semantics. Also clearly $\hat{K} \leq E$ and hence

$$\hat{K} \in \{T \leq E \mid (\forall u \in E, v \in L, w \in T)(u, v, w)_k \Rightarrow L_z = SupC_L_y(S_{x,y})\}$$

and so it must be the case that $\hat{K} \leq K$. But from our construction it is evident that $K \not\leq \hat{K}$, and this is a contradiction.]

The output map relation in (1b) is somewhat too cryptic to be used efficiently in investigating the properties of $\Omega$. In particular, it is not elaborated why $w\sigma \notin K = \Omega(E, L)$ when $w\sigma \in E$, and therefore all possible scenarios must be examined whenever the formula is invoked. Two causes may have contributed to $K$ failing to follow $E$ beyond $w$:

i. The subsystem behavior shrinks from $L_x$ to $L_z$, as some strings of $L_z$ are excluded first when $S_{x,y}$ is formed, and then when $SupC_L_y(\cdot)$ is taken on $S_{x,y}$. The output event $\sigma$ may be exclusively generated by the excluded strings, in which case $\sigma$ disappears altogether from the output map of $z$.

ii. Immediately following $\sigma$ we may have reached a subsystem which is unable to block some uncontrollable event at its output, as specified by the system-level requirement. Therefore, as a tentative solution, the access to that subsystem—i.e. the $\sigma$ transition—must be shut down. This imposes a new requirement on the current subsystem, namely, blocking $\sigma$ at its output.† Rigorously, if for $u \in E, v \in L$ and $w \in K$ with $(u, v, w)_k$ we have $u\sigma \hat{x} \in E$ and $u\sigma \hat{y} \in L$ such that

---

†In cases where $\sigma$ is uncontrollable, the next iteration of $\Omega(\cdot, L)$ (i.e. the one transforming $K$ into $\Omega(K, L)$) will take care of this new requirement.
\[ \text{SupC}_{L_y}(S_{\hat{z}, \hat{y}}) = \emptyset, \text{ then } w\sigma \notin K. \]

To summarize, we can write:

\[
(\forall u' \in L_x, w' \in L_z)(u', w')_{k+1} \Rightarrow \\
\theta_z(w') = \theta_x(u') - \{\sigma \in \Sigma_k \mid \text{SupC}_{L_y}(S_{\hat{z}, \hat{y}}) = \emptyset \}
\]

where \( w\sigma \hat{x} \) and \( v\sigma \hat{y} \) are the \( \sigma \)-continuations of \( u \) and \( v \) in \( E \) and \( L \), respectively.

2. Under a certain structural condition satisfied by \( E \), the term \( S_{x,y} \) appearing in the definition of \( \Omega \) can be seen as a specification passed down to the subsystem \( y \) in order to make it comply with some system-level requirement. Currently, the only obstacle towards such characterization, informally speaking, comes from the possibility of arbitrarily restricting the behavior at the boundaries. The condition recursively requires \( E \) to be consistent in dealing with the boundary cases with respect to \( L \): an event \( \sigma \) is cut out from the output of a subsystem of \( E \) altogether or not at all.

**Condition 4.1** \( E \preceq L \in \mathcal{L}_k, \ 0 \leq k \leq p, \) is said to be consistent at the boundaries with respect to \( L \)\(^1\) iff

- \( k = p : \text{true} \)
- \( 0 \leq k < p : \) For all \( u \in E \) and \( v \in L \) with \( u \equiv_k v \),

  (a) \( L_x \) is consistent at the boundaries with respect to \( L_y \), and
  (b) \( (\forall \sigma \in \Sigma_k) \) if \( w\sigma \in E \) then

\[
(\forall u' \in L_x, v' \in L_y)(u', v')_{k+1} \Rightarrow [\sigma \in \theta_y(v') \Rightarrow \sigma \in \theta_x(u')] 
\]

where \( x := \text{last}(u) \) and \( y := \text{last}(v) \).

**Example:** In Figure 4.7, \( E_i \preceq L; i = 1, 2, 3, \) and while \( E_1 \) and \( E_2 \) are consistent at the boundaries, \( E_3 \) is not.

\(^1\)We drop the reference to \( L \) when clear from the context.
As evident from the example, a sublanguage not consistent at the boundaries, when seen as a model of requirement for the system, represents a non-realistic, overly-constrained situation and therefore should be avoided. Specifically, it requires restricting part of the system behavior which does not pertain to any particular level, i.e. the interaction at the interface of adjacent levels. We believe this runs counter to the spirit of hierarchical modeling and based on intuition, it is too much for a subsystem to distinguish between different exits with the same label. $E_3$ in the example of Figure 4.7 has exactly the same $\Sigma_0$-projection as $L$, and their detailed behaviors are the same too, but still $E_3$ is a proper restriction of $L$. Apparently, prohibiting the occurrence of $\alpha$ at node "0" of $L$ is not a system-level requirement, as the $\alpha$ event following $a$ is left intact, but for some reason the $\alpha$-exit following $b$ is shut down. This is a manifestation of poor hierarchical design, where concerns at different levels are not strongly separated and therefore the manager's commands to the operator have to be very specific: "block the $\alpha$ event following $b"$, as opposed to "block the $\alpha$ exit".
**Proposition 4.3** If \( E \preceq L \) is consistent at the boundaries, then

\[
(\forall u \in E, v \in L, u' \in L_z, v' \in L_y)(u, v)_k \text{ and } (u', v')_{k+1} \text{ imply }
\theta_y(v') \cap \Sigma_{k,u} = \theta_x(u') \cap \Sigma_{k,u} \Leftrightarrow \theta_y(v') \cap \Sigma_{\text{illegal}}(u) = \emptyset
\]

where \( x := \text{last}(u), \ y := \text{last}(v), \) and

\[
\Sigma_{\text{illegal}}(u) := \{ \sigma \in \Sigma_{k,u} \mid v\sigma \in L \& u\sigma \notin E \}
\]

**Proof:** Straightforward. Note that \((\Leftrightarrow)\) is valid independent of the consistency condition.

In the light of Proposition 4.3 and with \( \Sigma_{\text{illegal}}(u) \) defined as above, when \( E \preceq L \) is consistent at the boundaries, \( S_{x,y} \) in the definition of \( \Omega \) can be rewritten as:

\[
S_{x,y} := \text{Sup}\{T' \preceq L_z \mid (\forall v' \in L_y, w' \in T')(v', w')_{k+1} \Rightarrow \theta_y(v') \cap \Sigma_{\text{illegal}}(u) = \emptyset\}
\]

Thus if we consider \( \Sigma_{\text{illegal}}(u) \) as the subset of system-level events making \( E \) uncontrollable at \( u \), \( S_{x,y} \) can then be visualized as the maximal subsystem behavior which does not generate any of these illegal outputs. Therefore, whenever \( \Omega \) spots an uncontrollable point in \( E \)—just by inspecting projections at the current level, thanks to consistency at the boundaries—where a subset of uncontrollable events has been cut out, it first looks for some feasible solution at the subsystem-level by specifying \( S_{x,y} \) as the new subsystem requirement\(^1\). If such a solution cannot be found (i.e. \( \text{Sup}\mathcal{C}_{L_y}(S_{x,y}) = \emptyset \)), only then and as the last resort it acts on the system-level and removes that particular node from \( E \) by blocking access to it (Figure 4.8).

The validity of the following results on consistency can be established from the definitions:

**Proposition 4.4** \((\forall E, L \in \mathcal{L}_k)E \preceq_L L \Rightarrow E \) is consistent at the boundaries.

---

\(^1\)It was formerly just \( L_z \).
Proposition 4.5 For all $E \leq L \in \mathcal{L}_k$, if $E$ is consistent at the boundaries, so are $\Omega(E, L)$ and $\text{SupC}_L(E)$.

After the discussion of Page 58, if the system requirement was originally specified by a hierarchy of top level sublanguages, then from the above results it follows that the supremal controllable sublanguage would be consistent at the boundaries.

We conclude this section by proving that $\Omega$, as defined earlier in this section, will indeed do the job of fixed-point characterization of $\text{SupC}$.

Proposition 4.6 For all $E \leq L \in \mathcal{L}_k$,

$$\{K \leq E \mid \Omega(K, L) = K\} = \mathcal{C}_L(E)$$

Proof: Let $K \leq E$ be controllable with respect to $L$. Then for any $u \in K$ and $v \in L$ with $(u, v)_k$ we will have:

$$x := \text{last}(u) \text{ is controllable with respect to } y := \text{last}(v)$$

which in turn implies

$$(\forall u' \in L_x, v' \in L_y)(u', v')_{k+1} \Rightarrow \theta_x(u') \cap \Sigma_{k, u} = \theta_y(v') \cap \Sigma_{k, u}$$
by definition, therefore,

\[ S_{x,y} = L_x \]

Since by (4.1) \( L_x \) is controllable with respect to \( L_y \), we get:

\[ \text{SupC}_{L_y}(S_{x,y}) = \text{SupC}_{L_y}(L_x) = L_x \]

Therefore,

\[ \Omega(K, L) = \text{Sup}\{T \leq K \mid (\forall u \in K, v \in L, w \in T)(u, v, w)_k \Rightarrow L_x = \text{SupC}_{L_y}(S_{x,y})\} = K \]

i.e. \( K \) is a fixed-point of \( \Omega(\cdot, L) \).

Conversely, if \( K \) is a fixed-point of \( \Omega(\cdot, L) \) and \( K \leq E \), then for any \( u \in K \) and \( v \in L \) with \( (u, v)_k \) we will have \( L_x = \text{SupC}_{L_y}(S_{x,y}) \), where \( x := \text{last}(u) \) and \( y := \text{last}(v) \). This implies that

\[ L_x \text{ is controllable with respect to } L_y, \quad (4.2) \]

and that \( L_x \preceq S_{x,y} \). On the other hand, by definition \( S_{x,y} \preceq L_x \). Therefore, \( S_{x,y} = L_x \) and so we can write

\[ (\forall u' \in L_x, v' \in L_y)(u', v')_{k+1} \Rightarrow \theta_x(u') \cap \Sigma_{k,u} = \theta_y(v') \cap \Sigma_{k,u} \quad (4.3) \]

(4.2) and (4.3) together imply \( x = \text{last}(u) \) is controllable with respect to \( y = \text{last}(v) \) and this completes the proof.

4.4 The Iteration Scheme

By Proposition 4.6 of Section 4.3, we have the important result that

\[ \text{SupC}_L(E) = \text{Sup}\{K \leq E \mid \Omega(K, L) = K\} \]

which gives us an alternative formulation for the problem "Find \( \text{SupC}_L(E) \)", namely,

"Find the largest fixed-point of \( \Omega(\cdot, L) \) which is smaller than \( E \)."
Since

**Fact 4.1** $\Omega(\cdot, L)$ is **contractive**, i.e.

$$(\forall K \leq L) \Omega(K, L) \leq K$$

a natural way to compute such a fixed-point is by iterating $\Omega(\cdot, L)$ on $E$, i.e. forming the sequence $\{K_j\}_{j=0}^{\infty}$ according to:

$$
\begin{align*}
K_0 &:= E \\
K_{j+1} &:= \Omega(K_j, L); \ j \in \mathbb{N}
\end{align*}
$$

If this iteration terminates in a finite number of steps, say, for some $j^* \in \mathbb{N}$,

$$\Omega(K_{j^*}, L) = K_{j^*}$$

then we have actually reached a fixed-point of $\Omega(\cdot, L)$ which is smaller than $E$. To verify that such a fixed-point is in fact the largest one with the desired property (i.e. that we have not skipped over the largest one), we have to probe $\Omega$ further for more structural properties. In particular,

**Proposition 4.7** $\Omega(\cdot, L)$ is **monotone**, i.e.

$$(\forall E_1, E_2 \leq L) E_1 \leq E_2 \Rightarrow \Omega(E_1, L) \leq \Omega(E_2, L)$$

To prove this proposition we need the following lemma:

**Lemma 4.1** $\text{Sup}C(\cdot)$ is **monotone**.

**Proof of Proposition:** Let $K_i := \Omega(E_i, L)$ for $i = 1, 2$.

If $K_1 = \emptyset$ the result is trivial. Thus, suppose $K_1 \neq \emptyset$ and therefore, say, $z_1 \in K_1$. Since $K_1 \leq E_1 \leq E_2 \leq L$, by Lemma 3.2 there exist strings $x_1 \in E_1$, $x_2 \in E_2$ and $y \in L$ and we have $L_{z_1} = \text{Sup}C_{L_y}(S_{z_1,y})$, where: $(i = 1, 2)$

$$S_{x_1,y} := \text{Sup}\{T_i \leq L_{x_1} \mid (\forall u'_i \in L_{x_1}, v' \in L_y, w'_i \in T_i)
(u'_i, v', w'_i)_{k+1} \Rightarrow \theta_{x_1}(u'_i) \cap \Sigma_{k,u} = \theta_y(v') \cap \Sigma_{k,u}\}.$$}

Since $\text{Sup}C(\cdot)$ is monotone and

**Fact 4.2** $x_1 \leq x_2 \Rightarrow S_{x_1,y} \leq S_{x_2,y}$
it follows that

\[
\text{Sup}_{CL_y}(S_{x_1,y}) \preceq \text{Sup}_{CL_y}(S_{x_2,y}).
\]

(4.4)

Therefore, \( \text{Sup}_{CL_y}(S_{x_2,y}) \neq \emptyset \) and so the existence of \( z_2 \in K_2 \) with \( L_{z_2} = \text{Sup}_{CL_y}(S_{x_2,y}) \) has been established and hence \( K_2 \neq \emptyset \).

We now verify that \( K_1 \preceq K_2 \). Take arbitrary \( w_1 \in K_1 \) and \( w_2 \in K_2 \) with \( (w_1, w_2)_k \).

We must show that \( z_1 \preceq z_2 \), where \( z_i := \text{last}(w_i); \ i = 1, 2 \).

Observe that since \( K_i \preceq E_i \preceq L, \ i = 1, 2 \), we have the strings \( u_i \in E_i \) and \( v \in L \) corresponding to \( w_i \in K_i \) such that \( (u_i, v, w_i)_k \). Denote \( x_i := \text{last}(u_i) \) and \( y := \text{last}(v) \). Then by definition of \( \Omega \) we have the relations:

\[
L_{z_i} = \text{Sup}_{CL_y}(S_{x_i,y}); \ i = 1, 2
\]

and therefore since \( x_1 \preceq x_2 \) we will have \( L_{z_1} \preceq L_{z_2} \). It only remains to show the output map inclusion. Take \( w'_1 \in L_{z_1} \) and \( w'_2 \in L_{z_2} \) with \( (w'_1, w'_2)_{k+1} \). We have to show \( \theta_{x_1}(w'_1) \subseteq \theta_{x_2}(w'_2) \). Once again, since \( L_{z_i} \preceq L_{x_i} \preceq L_y, \ i = 1, 2 \), the existence of \( u'_i \in L_{x_i} \) and \( v' \in L_y \) with \( (u'_i, v', w'_i)_{k+1} \) is established. Observe that by Remark (1) of Page 65 we have

\[
\theta_{x_i}(w'_i) = \theta_{x_i}(u'_i) - \{ \sigma \in \Sigma_k | \text{Sup}_{CL_y}(S_{x_i,y}) = \emptyset \}; \ i = 1, 2
\]

where whenever \( u_i \sigma \in E_i \), we denote the \( \sigma \)-continuations of \( u_i \sigma \) and \( v \sigma \) in \( E_i \) and \( L \) by \( u_i \sigma \hat{x}_i \) and \( v \sigma \hat{y} \), respectively.

Clearly, \( \theta_{x_1}(u'_i) \subseteq \theta_{x_2}(u'_2) \) since \( x_1 \preceq x_2 \). Let \( u_1 \sigma \in E_1 \) and define \( S_{\hat{x}_1,\hat{y}}; \ i = 1, 2 \), accordingly and let \( \text{Sup}_{CL_y}(S_{\hat{x}_2,\hat{y}}) = \emptyset \).

Again we have \( S_{\hat{x}_1,\hat{y}} \preceq S_{\hat{x}_2,\hat{y}} \) since \( \hat{x}_1 \preceq \hat{x}_2 \) and by monotonicity of \( \text{Sup}(\cdot) \) it follows that \( \text{Sup}_{CL_y}(S_{\hat{x}_1,\hat{y}}) = \emptyset \). Therefore,

\[
\theta_{x_1}(w'_i) \subseteq \theta_{x_2}(w'_2)
\]

and this completes the proof.
Proof of the Fact: \((x_1 \leq x_2 \Rightarrow S_{x_1,y} \leq S_{x_2,y})\)

If \(S_{x_1,y} \neq \emptyset\), we will have \(z'_1 \in S_{x_1,y}\) for some \(z'_1 \in \mathcal{X}_{k+1}\). Since

\[S_{x_1,y} \subseteq L_{x_1} \leq L_{x_2} \subseteq L_y\]

by Lemma 3.2 we will have the strings \(x'_1 \in L_{x_1}, x'_2 \in L_{x_2}\) and \(y' \in L_y\) corresponding to \(z'_1 \in S_{x_1,y}\). Since \(z'_1 \in S_{x_1,y}\) it follows that

\[\theta_{x_1}(x'_1) \cap \Sigma_{k,u} \supseteq \theta_y(y') \cap \Sigma_{k,u}\]

On the other hand, since \(x_1 \leq x_2\) we have \(\theta_{x_1}(x'_1) \subseteq \theta_{x_2}(x'_2)\) and therefore

\[\theta_{x_2}(x'_2) \cap \Sigma_{k,u} \supseteq \theta_y(y') \cap \Sigma_{k,u}\]

which establishes the existence of \(z'_2 \in S_{x_2,y}\) for some \(z'_2 \in \mathcal{X}_{k+1}\), and hence \(S_{x_2,y} \neq \emptyset\).

We now verify that \(S_{x_1,y} \leq S_{x_2,y}\). Take \(w'_1 \in S_{x_1,y}\) and \(w'_2 \in S_{x_2,y}\) with \((w'_1, w'_2)_{k+1}\). We have to show that \(z'_1 \leq z'_2\), where we have denoted \(z'_i := \text{last}(w'_i); i = 1, 2\).

Since \(S_{x_1,y} \leq L_{x_1} \leq L_y\), we have the strings \(u'_i \in L_{x_i}\) and \(v' \in L_y\) corresponding to \(w'_i \in S_{x_i,y}\) such that \((u'_i, v', w'_i)_{k+1}\), \(i = 1, 2\). Denote \(x'_i := \text{last}(u'_i)\) and \(y' := \text{last}(v')\).

Observe that by virtue of the supremum operator in the definition of \(S_{.}\.\) we have \((i = 1, 2)\):

\[L_{x'_1} = L_{x'_1}, \text{ and}\]

\[(\forall u''_i \in L_{x'_i}) \theta_{x'_i}(u''_i) = \theta_{x'_1}(u''_i) - \{\sigma \in \Sigma_{k+1} | \theta_{x'_1}(\hat{u}'_i) \cap \Sigma_{k,u} \not\subseteq \theta_y(y') \cap \Sigma_{k,u}\}\]

where whenever \(u'_i \sigma \in L_{x_i}\), we denote the one-step \(\sigma\)-continuations of \(u'_i\) and \(v'\) in \(L_{x_i}\) and \(L_y\) by \(\hat{u}'_i\) and \(\hat{v}'\), respectively. From \(x'_1 \leq x'_2\) it follows that

\[L_{x'_1} = L_{x'_1} \leq L_{x'_2} = L_{x'_2}\]

To verify the output map inclusion, take \(u''_1 \in L_{x'_1}\) and \(u''_2 \in L_{x'_2}\) with \((u''_1, u''_2)_{k+2}\). Since \(x'_1 \leq x'_2\) we have \(\theta_{x'_1}(u''_1) \subseteq \theta_{x'_2}(u''_2)\). Also, if \(u'_i \sigma \in L_{x_i}\) and

\[\theta_{x_2}(\hat{u}'_2) \cap \Sigma_{k,u} \not\subseteq \theta_y(y') \cap \Sigma_{k,u}\]

then since \(\theta_{x'_1}(u'_1) \subseteq \theta_{x'_2}(u'_2)\) we have

\[\theta_{x'_1}(\hat{u}'_1) \cap \Sigma_{k,u} \not\subseteq \theta_y(y') \cap \Sigma_{k,u}\]
and so $\theta_{x_1'}(u''_1) \subseteq \theta_{x_1'}(u''_2)$.

We are now ready to prove the main result of this section:

**Theorem 4.1** If the iteration

$$
\begin{align*}
K_0 &:= E \\
K_{j+1} &:= \Omega(K_j, L); \ j \in \mathbb{N}
\end{align*}
$$

terminates in a finite number of steps, i.e. for some $j^* \in \mathbb{N}$, $K_{j^*} = \Omega(K_{j^*}, L)$, then $K_{j^*}$ is indeed the largest fixed-point of $\Omega(\cdot, L)$ which is smaller than $E$.

**Proof:** Let $K^*$ be an arbitrary fixed-point of $\Omega(\cdot, L)$ which is smaller than $E$. Applying $\Omega(\cdot, L)$ $j^*$-times, by monotonicity it follows that:

$$K^* \preceq E = K_0 \Rightarrow K^* = \Omega^{j^*}(K^*, L) \preceq \Omega^{j^*}(K_0, L) = K_{j^*}.$$

Q.E.D.

In the next chapter, conditions will be sought under which termination of the iteration of Theorem 4.1 in a finite number of steps can be guaranteed.

**Final Thoughts:**

1. As mentioned before, recursion is implicit in the definition of $\Omega$. The result of $\Omega(\cdot, L)$'s application on $E \preceq L$ depends on the value of $\text{SupC}_L(S_{x,y})$ for all pairs $(x, y)$ of corresponding tokens in $(E, L)$, which in turn can be computed as the largest fixed-point of $\Omega(\cdot, L_y)$ which is smaller than $S_{x,y}$.

2. **An overview of the way $\Omega$ works:** We assume that we are given a top level sublanguage $E$ of $L$ as the requirement specification, implying that initially substructures of $E$ are carbon copies of those of $L$. Therefore, one should only be concerned with violation of controllability at the system level. This happens when for some $u \in E$ and $\sigma \in \Sigma_{k,u}$, we have $u\sigma \in L$ while $u\sigma \notin E$, where $u \equiv_k v$.

When such a case arises, $\Omega$ first looks for some feasible solution at the subsystem level, where it tries to block generation of strings having $\sigma$ in their output set.

If it succeeds, we will have a restricted version of the subsystem behavior in which strings generating the illegal outputs are excluded. Note that such a
restriction is implementable by some appropriate supervisory control at the subsystem level. Also note that in this case no change has to be made on the system's wirings around that particular subsystem, though as a side effect we may have lost other output events which were generated exclusively by the discarded strings. In other words, depending on the internal structure of the subsystem, there may be a price to pay when generation of the "system" event $\sigma$ is prohibited by some control actions at the "subsystem" level. Nevertheless, we will end up having a less restricted closed-loop behavior of the system compared to what we would compute had we applied the standard method, i.e. blocking access to that particular subsystem, thereby cutting out all the transitions at its output (see Figure 4.9).

If a feasible low level solution cannot be found, $\Omega$ acts the way it would in the standard theory, namely, it excludes $w$—and all other strings whose prefix is $w$—from the next language in the sequence.

Note that since top level sublanguages are consistent at the boundaries, and this property is preserved under $\Omega$, at all stages through the iteration concern is raised only when controllability is violated at the system level, i.e. for some $u \in K$,

$$\Sigma_{\text{illegal}}(u) := \{\sigma \in \Sigma_{k,u} \mid u\sigma \notin K \& v\sigma \in L\} \neq \emptyset$$

If this set has been unchanged since the last time it was visited, the interpretation is that the problem has been already dealt with at some previous step of the iteration. However, if this set has grown, the requirement for the subsystem must be posed anew in order to prevent generation of the newly declared illegal output events\footnote{There is an exception in this case, which will be explained in the example of Section 4.5.} (see Figure 4.10).

Finally, as noted before, by restricting our attention only to languages which are consistent at the boundaries we avoid cases where despite no apparent violation
of controllability at the system level, an uncontrollable event $\sigma$ is barred from the output set of "some" of the subsystem strings (see Figure 4.11).

### 4.5 Example

In this section we apply the methods to an example involving finite (structured) languages. As we will see in the next chapter, the class of finite structured languages is a proper subclass of the regular subclass of $\mathcal{L}_K$, within which convergence of the iteration of Theorem 4.1 in finite time is guaranteed.
No need to analyze the subsystem because $\Sigma_{illegal}$ has been unchanged since the last time.

$K_0 \quad \Sigma_{illegal} = \{\alpha\}$

$K_1, \ldots, K_{j-1} \Sigma_{illegal} = \{\alpha\}$

$K_j \Sigma_{illegal} = \{\alpha, \beta\} \quad K_{j+1}, \ldots \Sigma_{illegal} = \{\alpha, \beta\}$

Figure 4.10: $\Sigma_{illegal}$ for $j \in \mathbb{N}$

Figure 4.11: Problem with inconsistent sublanguages
Example: A system is modeled by the language $L \in \mathcal{L}_0$ ($p = 2$) shown in Figure 4.12. Also shown in the figure is the requirement specification, a flat language over $\Sigma_0$. Note that:

$$\Sigma_{0,c} = \{\beta\}, \quad \Sigma_{0,u} = \{\alpha, \gamma\} \text{ and } \Sigma_{1,c} = \{b\}, \quad \Sigma_{1,u} = \{a\}$$

Therefore, the starting point of the iteration would be the top level sublanguage $K_0$, shown in Figure 4.13.

Observe that for computing $K_1 := \Omega(K_0, L)$ only the tokens $x_1$ and $x_2$ need to be examined, and we have:

$$\Sigma_{\text{illegal}}(x_0\beta x_1) = \{\alpha\} \text{ and } \Sigma_{\text{illegal}}(x_0a x_2) = \{\gamma\}.$$
Therefore, graphically, we will have:

\[ S_{x_1,y_1} : \quad \xrightarrow{b} \quad 10 \quad \xrightarrow{a} \quad 11 \quad \Rightarrow \quad \text{Sup}C_{L_{y_1}}(S_{x_1,y_1}) = 10 \]

\[ S_{x_2,y_2} : \quad \xrightarrow{a} \quad 20 \quad \xrightarrow{b} \quad 21 \quad \Rightarrow \quad \text{Sup}C_{L_{y_2}}(S_{x_2,y_2}) = \emptyset \]

Note that:

\[ L_{y_1} : \quad \xrightarrow{b} \quad 10 \quad \xrightarrow{a} \quad 11 \quad \xrightarrow{a} \quad 12 \]

\[ L_{y_2} : \quad \xrightarrow{a} \quad 20 \quad \xrightarrow{b} \quad 21 \quad \xrightarrow{a} \quad 22 \]

Hence we get \( K_1 := \Omega(K_0, L) \) as shown in Figure 4.14.

With a little abuse of notation, we reuse the symbols \( x_0 \) and \( x_1 \) for naming the tokens of \( K_1 \). As it turns out, this time only \( x_0 \) should be inspected. Note that despite the fact that the event \( \gamma \) has been added to the set \( \Sigma_{\text{illegal}}(x_0 \beta x_1) = \{\alpha, \gamma\} \) since the last iteration step, we should not be concerned about excluding those strings of \( L_{x_1} \) which generate \( \gamma \) from the next language in the sequence. As a matter of fact, the action of disabling the \( b \)-transition of subsystem "1", meant originally for blocking the \( \alpha \)-exit at node "12", has necessarily over-restricted the subsystem behavior so that neither \( \alpha \) nor \( \gamma \) are present at its output.

Thus we would concentrate only on \( x_0 \). We have

\[ \Sigma_{\text{illegal}}(x_0) = \{\alpha\} \]

and hence, graphically,
Therefore we get \( K_2 := \Omega(K_1, L) \) as shown in Figure 4.15, which can be verified to be a fixed-point of \( \Omega(\cdot, L) \).

\[
\begin{array}{c}
S_{x_0, y_0} : & \xymatrix@R=1em{00 \ar[r] & 01 \ar[l]}
\end{array}
\Rightarrow
\begin{array}{c}
\sup C_{L_{y_0}}(S_{x_0, y_0}) = S_{x_0, y_0}
\end{array}
\]

Figure 4.15: \( K_2 := \Omega(K_1, L) \) and \( \Omega(K_2, L) = K_2 \)

Remark: As the class of one-layered structured languages over an alphabet \( \Sigma \) is no different from the class of standard (flat) closed languages over the same alphabet, we may opt to apply the standard RW methods when computing the supremal controllable sublanguage of a 1-layered structured language, as we did when faced with computations like \( \sup C_{L_{y_1}}(S_{x_1, y_1}) \) in the example. We should stress however that our general approach is still applicable and would naturally lead to the same result.

Let us examine the case of \( \sup C_{L_{y_1}}(S_{x_1, y_1}) \).

Let \( K'_0 := S_{x_1, y_1} \) and again reuse \( x \)'s to denote the tokens of \( K'_0 \). Observe that \( u := x_{10}bx_{11} \) is the only point where attention must be paid. We have:

\[
\Sigma_{illega}(x_{10}bx_{11}) = \{a\} \text{ and therefore } S_{x_1, y_{11}} = \emptyset.
\]

(Note that \( L_{x_1} = \epsilon \) and \( \theta_{y_{11}}(\epsilon) = \{a\} \), therefore removing the only string of \( L_{x_1} \) which generates \( a \)—will leave us with the empty set.)

So we get:

\[
K'_1 : \xymatrix@R=1em{ \ar[r] & 10 }
\]

which is a fixed-point of \( \Omega(\cdot, L_{y_1}) \) since \( \Sigma_{illega}(x_{10}) = \emptyset \).
Chapter 5

Effective Computation: Regularity

5.1 Right Congruences on \((\Sigma_k \cup \mathcal{X}_k)^\ast\)

The iteration scheme of Theorem 4.1 can be used for effective computation of the supremal controllable sublanguage of \(E\) whenever the iteration terminates in a finite number of steps. Unfortunately, there is no such guarantee in the general case where the sublanguage \(K_0\)—the starting point of the iteration—is infinite, i.e. contains strings of unbounded length.

However, there is a subclass of infinite languages within which the problem is still tractable. Informally speaking, these are the languages that can be generated (or recognized) by structures of “finite memory”.

Let \(L \in \mathcal{L}_k\) and \(\theta : L \to 2^T\) be an output function, mapping the strings of \(L\) into subsets of an alphabet \(T\). Based on the pair \((L, \theta)\) we define the following equivalence relations on \((\Sigma_k \cup \mathcal{X}_k)^\ast\):

- \(\equiv_L\): (“Nerode equivalence relation with respect to \(L\)”)

\[(\forall s, t \in (\Sigma_k \cup \mathcal{X}_k)^\ast)s \equiv_L t \iff (\forall u \in (\Sigma_k \cup \mathcal{X}_k)^\ast)su \in L \iff tu \in L\]

- \(\equiv_{\mathcal{L}(L)}\):

\[(\forall s, t \in (\Sigma_k \cup \mathcal{X}_k)^\ast)s \equiv_{\mathcal{L}(L)} t \iff s \equiv_L t \text{ and whenever } s \in \mathcal{L},

\text{last}(s) \in \mathcal{X}_k \Rightarrow \text{last}(s) = \text{last}(t)\]
- $\equiv_\theta$: ("The dynamic output kernel relation")

$$(\forall s, t \in (\Sigma_k \cup X_k)^*) s \equiv_\theta t \iff (\forall u \in (\Sigma_k \cup X_k)^*) \theta(su) = \theta(tu)$$

where the domain of $\theta$ is extended to $(\Sigma_k \cup X_k)^*$ by defining $\theta(s) := \emptyset$ for $s \notin L$.

- $\equiv_{L,\theta}$:

$$(\forall s, t \in (\Sigma_k \cup X_k)^*) s \equiv_{L,\theta} t \iff s \equiv_{L,\theta \cup 0} t$$

Note that for $s \in L$ and $s \equiv_L t$, we have $t \in L$ and

$$\text{last}(s) \in X_k \iff \text{last}(t) \in X_k$$

[Otherwise if, say, $\text{last}(s) \in X_k$ but $\text{last}(t) \notin X_k$, by syntax of $L_k$ it follows that $(\exists x \in X_k)tx \in L$ and therefore it must be the case that $sx \in L$, which violates the syntax.]

The equivalence relation $\equiv_{L,\theta}$ defined above is indeed a refined version of the Nerode equivalence relation which is customized for the structured case. Namely, all well-formed strings comprising an equivalence class of this relation must be terminated with the same token in $X_k$.

Also observe that when $s, t \in L$ and $s \equiv_{L,\theta} t$, then $s$ and $t$ will have exactly the same possible continuations in $L$ and moreover, their "observed behavior" (through the output function) looks the same as they evolve. We can rephrase the previous statement by saying that the dynamics of $s$ and $t$ are the same, and that they are indistinguishable from the vantage point of the high level system in which $(L, \theta)$ is embedded.

**Lemma 5.1** Let $\Sigma := \Sigma_k \cup X_k$, $X := (\Sigma_k \cup X_k)^*/ \equiv_{L,\theta}$ and $\pi : s \mapsto [s] (\text{mod } \equiv_{L,\theta})$. Then there exist unique maps $\xi$ and $x_0$ such that diagram (a) of Figure 5.1 commutes.

\[\bullet\]

In the diagram, $\text{cat} : (s, \sigma) \mapsto s\sigma$ is the catenation operator and $1$ is a singleton whose only member is mapped to $\epsilon$ by $\epsilon : 1 \to L$.
Lemma 5.2 Let $\Sigma := \Sigma_k \cup \mathcal{X}_k$, $Y := (\Sigma_k \cup \mathcal{X}_k)^*/\equiv_{\mathcal{L}(L, \theta)}$ and $\rho : s \mapsto [s] \ (\text{mod } \equiv_{\mathcal{L}(L, \theta)})$. Then there exist unique maps $\eta$, $y_0$ and $\lambda$ such that diagram (b) of Figure 5.1 commutes.

Based on the equivalence relations $\equiv_{\mathcal{L}(L)}$ and $\equiv_{\mathcal{L}(L, \theta)}$, one can construct abstract structures to model $L$ and $(L, \theta)$, respectively. Take the $\equiv_{\mathcal{L}(L, \theta)}$ case for example. Define:

- $Y := (\Sigma_k \cup \mathcal{X}_k)^*/\equiv_{\mathcal{L}(L, \theta)}$ the set of states;
- $Y_m := L/\equiv_{\mathcal{L}(L, \theta)}$ the set of marker or accepting states;
- $y_0 := [\epsilon]$ the initial state;
- $\eta : Y \times (\Sigma_k \cup \mathcal{X}_k) \to Y : ([s], \sigma) \mapsto [s\sigma]$ the transition function;
- $\lambda : Y \to 2^T : [s] \mapsto \theta(s)$ the output map.

Now extend $\eta$ to $Y \times (\Sigma_k \cup \mathcal{X}_k)^*$ by induction on the length of strings:

- $\xi([s], \epsilon) := [s]$
- $\xi([s], t\sigma) := \xi([st], \sigma)$

Then the original information $(L, \theta)$ can be recovered from the 7-tuple $R := (Y, Y_m, y_0, \Sigma_k \cup \mathcal{X}_k, T, \eta, \lambda)$ via the relations:

$L = \{s \in (\Sigma_k \cup \mathcal{X}_k)^* \mid \eta(y_0, s) \in Y_m \}$ and

$\theta(s) = \lambda(\eta(y_0, s))$. 

Figure 5.1: Illustration of Lemmas 5.1 and 5.2
Remark: It might be more convenient to take \( \eta \) as a partial map, defined on \([s, \sigma]\) only when \( s\sigma \in L \). Then the 6-tuple \((Y_m, y_0, \Sigma_k \cup X_k, T, \eta, \lambda)\) can be regarded as a recognizer for \((L, \theta)\). Note that

\[ Y = Y_m \cup \{(\Sigma_k \cup X_k)^* \setminus L\}. \]

5.2 Example

Let \( L \in \mathcal{L}_0 \) be defined as follows: \((p = 2)\)

\[ L := prcl(x_0ax_1(\alpha x_2\beta x_3)^*) \]

\[
x_0 \in X_0 : \begin{cases} L_0 = \bar{x_0} \\
\theta_0(x_0) = \{\alpha\} \end{cases} \quad x_2 \in X_0 : \begin{cases} L_2 = \bar{x_2} \\
\theta_2(x_2) = \{\beta\} \end{cases}
\]

\[
x_1 \in X_0 : \begin{cases} L_1 = prcl(x_{10}ax_{11}(bx_{10}ax_{11})^*) \\
\theta_1(x_{10}ax_{11}(bx_{10}ax_{11})^m) = \{\alpha\}; \ m \geq 0 \end{cases}
\]

\[
x_3 \in X_0 : \begin{cases} L_3 = prcl(x_{30}(bx_{31}ax_{30})^*) \\
\theta_3(x_{30}(bx_{31}ax_{30})^m) = \{\alpha\}; \ m \geq 0 \end{cases}
\]

\[
x_{31} = x_{10} \in X_1 : \begin{cases} L_{10} = \bar{\epsilon} \\
\theta_{10}(\epsilon) = \{a\} \end{cases} \quad x_{30} = x_{11} \in X_1 : \begin{cases} L_{11} = \bar{\epsilon} \\
\theta_{11}(\epsilon) = \{b\} \end{cases}
\]

\[
x_{20} = x_{00} \in X_1 : \begin{cases} L_{00} = \bar{\epsilon} \\
\theta_{00}(\epsilon) = \emptyset \end{cases}
\]

Note that \( x_0 \not= x_2 \) and \( x_1 \not= x_3 \) (i.e. \( x_0 \sim x_2 \) and \( x_1 \sim x_3 \)).

Based on the tree representation of \( L \), we build up automaton recognizers for all components in the definition hierarchy of \( L \). Naturally, we represent a token with a labeled automaton.
We start from the language $L$ itself, whose recognizer, $R(L)$, is represented by the automaton shown in Figure 5.2.

![Diagram of automaton](image)

\[ \Rightarrow R(L)^\dagger : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]

\[ \Rightarrow R^*(L) : \quad \]
The 4-tuple \( (S_m, s_0, \Sigma_0 \cup \mathcal{X}_0, \xi) \) can be written by inspecting the automaton \( R(L) \), according to:

\[
S_m := \{0, 1, \ldots, 7\}; \quad s_0 := 0
\]

We also have \( \Sigma_0 \supseteq \{\alpha, \beta\} \) and \( \mathcal{X}_0 \supseteq \{x_0, x_1, x_2, x_3\} \), and \( \xi \) is defined via the graph’s transition table, e.g. \( \xi(2, x_1) = 3 \), etc.

At the next level down, indexed by \( k = 1 \), we have tokens \( x_i, i = 0, 1, 2, 3 \), which we represent by labeled automata \( R(x_i) \), shown in Figure 5.4.

\[
R(x_0) : \quad \xrightarrow{x_{20}} \bullet \xrightarrow{a} \bullet \quad R(x_2) : \quad \xrightarrow{x_{20}} \bullet \xrightarrow{\beta} \bullet
\]

\[
x_1 : \quad \xrightarrow{x_{10}} \bullet \xrightarrow{a} \bullet \xrightarrow{x_{11}} \bullet \xrightarrow{b} \bullet \quad \text{Repeats}
\]

\[
\Rightarrow R(x_1) : \quad \xrightarrow{x_{10}} \bullet \xrightarrow{a} \bullet \xrightarrow{x_{11}} \bullet \quad \text{Repeats}
\]

\[
x_3 : \quad \xrightarrow{x_{30}} \bullet \xrightarrow{a} \bullet \xrightarrow{x_{31}} \bullet \xrightarrow{b} \bullet \quad \text{Repeats}
\]

\[
\Rightarrow R(x_3) : \quad \xrightarrow{x_{30}} \bullet \xrightarrow{a} \bullet \xrightarrow{x_{31}} \bullet \quad \text{Repeats}
\]

Figure 5.4: Recognizers for \( x_i \)'s

And finally, we have the trivial constructs shown in Figure 5.5 at the bottom level \( (k = 2) \).

**Off-the-topic note:** The plugged-in structure as a recognizer for \( I(L) \): We inductively expand the recognizers of all components in the definition hierarchy of \( L \),

\footnote{We have changed the notation a bit to avoid name conflict with tokens.}
$R(x_{31}) = R(x_{10}) : \quad \rightarrow \alpha \quad R(x_{30}) = R(x_{11}) : \quad \rightarrow \beta$

$R(x_{20}) = R(x_{00}) : \quad \rightarrow \bullet$

Figure 5.5: Recognizers for $x_{ij}$'s

Starting from those located at the level next to the bottom, and then proceeding all the way up to the top of the hierarchy where $L$ is located. The result is shown in Figure 5.6.

Figure 5.6: Expanding $R(L)$

Observe that as one might suspect, $R_{exp}(L)$ recognizes the language $\mathcal{I}(L)$, but it has more than what it takes to get the job done, in other words, $R_{exp}(L)$ is not the canonical recognizer of $\mathcal{I}(L)$, which is shown in Figure 5.7.

Figure 5.7: The canonical recognizer of $\mathcal{I}(L)$

To put this observation into a clearer perspective, suppose the system whose behavior is given by $L$ was originally represented within a framework which supports multi-level modeling—for example, Harel’s Statecharts [Har87] or State-Tree Struc-
tures of Wang [Wan95], as shown in Figure 5.8.\footnote{Of course, as pointed out in Chapter 2, not all systems modeled within these frameworks can be expressed in our setup. For example, the set of trajectories of the system $S$ shown below cannot be expressed by a structured language over $\Sigma_0 = \{\alpha, \beta\}$ and $\Sigma_1 = \{a\}$.}

$$L: \begin{array}{c}
\alpha \\
\downarrow \\
\alpha \\
\end{array} \quad \begin{array}{c}
\alpha \\
\downarrow \\
\alpha \\
\end{array}$$

"Superstate"

Figure 5.8: State-tree structure $S$

Then, a natural representation of $R(L)$ of Page 86 within the same framework is shown in Figure 5.9.

$$R(L): \begin{array}{c}
\alpha \\
\downarrow \\
\alpha \\
\end{array} \quad \begin{array}{c}
\alpha \\
\downarrow \\
\alpha \\
\end{array}$$

Figure 5.9: State-tree structure of $R(L)$

It looks as if the original "superstate", which has two entrance states, has now been broken into two separate superstates with duplicate structures, but each having a single entrance (initial) state. This observation is of no consequence to our convergence analysis, because the process of reproducing a superstate with $N$ entrance states $N - 1$ times will increase the state size finitely if the original model is finite.

5.3 Regularity and Finite Time Convergence

We set out to introduce a notion of regularity—similar to the one adopted in the standard theory of languages—as a sufficient condition for termination of the iteration
scheme of Theorem 4.1. The key step is to place an upper bound on the state size of languages $K_j$ appearing in the sequence $\{K_j\}_{j=0}^{\infty}$.

Let $K_0 \leq L \in \mathcal{L}_k$, and the sequence $\{K_j\}_{j=0}^{\infty}$ be generated through the relation $K_{j+1} := \Omega(K_j, L); \ j \geq 0$. Since $\Omega(\cdot, L)$ is contractive we have

$$L \supseteq K_0 \supseteq \cdots \supseteq K_j \supseteq K_{j+1} \supseteq \cdots.$$ 

Our first main result is the following:

**Lemma 5.3** When restricted to $K_j$, the following refinement relation holds for all $j \geq 0$:

$$\equiv_{\mathcal{L}(L)} \land \equiv_{\mathcal{L}(K_0)} \leq \equiv_{\mathcal{L}(K_j)}$$

It states that

$$(\forall s_j, t_j \in K_j)s \equiv_{\mathcal{L}(L)} t \land s_0 \equiv_{\mathcal{L}(K_0)} t_0 \Rightarrow s_j \equiv_{\mathcal{L}(K_j)} t_j$$

where $(s, t)$ and $(s_0, t_0)$ are the unique pairs of strings in $L^2$ and $K_0^2$, respectively, which correspond to $(s_j, t_j)$ in $K_j^2$. [i.e. we have $s_0 \equiv_k s_j \equiv_k s$ and $t_0 \equiv_k t_j \equiv_k t$.]

Before getting involved in the proof, we elaborate on the definition of $\equiv_{\mathcal{L}(L)}$ and show how the result of this lemma would have failed had it been formulated with the Nerode equivalence relations instead.

Let us for the moment strip $\equiv_{\mathcal{L}(L)}$ of the condition

whenever $s \in L$, $last(s) \in \mathcal{X}_k \Rightarrow last(s) = last(t)$

and simply take the quotients by $\equiv_L$ as recognizer's states (ditto for $K_0$ and $K_1$). Then one can come up with numerous examples which do not obey the result of Lemma 5.3. Consider the example of Figure 5.10.

Although the substructures (tokens) have not been specified, here we present a plausible verbal description which matches the high level realities and can be used as a guide to complete the picture:
Take $K_0$ to be a top-level restriction of $L$ in which all instances of $\sigma_4$ are removed (note that there is only one) and assume that $\sigma_4$ is uncontrollable. Thus we must have

$$x_0 = y_0; \quad \begin{cases} L_{x_i} = L_{y_i} \\ \theta_{x_i} = \theta_{y_i} - \{\sigma_4\} \end{cases}; \ i = 1, 2, \text{ and } x_3 = y_3.$$  

There are two "subsystems" generating $\sigma_4$ at their outputs: "1" and "2". Assume that in both there are feasible ways to shut down this output event, but in case of "2" there is a price to pay, namely, $\sigma_3$ also gets disconnected at the output. Thus, all we know about the tokens of $K_1$ is:

$$z_0 = y_0, \ z_i \neq y_i; \ i = 1, 2, \text{ and } z_3 = y_3$$

where the behavior of subsystem "2"—represented by $L_{z_2}$—is now so restricted that it no longer generates $\sigma_4$ at its output.

Now take $s_1 = z_0\sigma_1z_1$ and $t_1 = z_0\sigma_2z_2$ in $K_1$. Clearly, $s_1 \not\equiv_{K_1} t_1$, while $s_0 \equiv_{K_0} t_0$ and $s \equiv_{L} t$, where $s_0 = x_0\sigma_1x_1$, $t_0 = x_0\sigma_2x_2$, $s = y_0\sigma_1y_1$ and $t = y_0\sigma_2y_2$.

In contrast, refining the underlying equivalence relations to $\equiv_{L(L)}$ would yield the recognizers of Figure 5.11, in which case $s \not\equiv_{L(L)} t$ since $y_1 \neq y_2$. 

Figure 5.10: Nerode cells: Lemma 5.3 does not hold
Proof of the Lemma: By induction. For $j = 0$ the result is trivial. Suppose it also holds true for all $j \leq i$, and for arbitrary $s_{i+1}, t_{i+1} \in K_{i+1}$, let $s \equiv_{L(L)} t$ and $s_0 \equiv_{L(K_0)} t_0$, where $(s, t)$ and $(s_j, t_j)$ are unique pairs of strings in $L^2$ and $K_j$, respectively, satisfying $s \equiv_k s_j \equiv_k s_{i+1}$ and $t \equiv_k t_j \equiv_k t_{i+1}$; $j < i$. By the induction assumption it follows that:

$s_i \equiv_{L(K_i)} t_i$.

We break the rest of this proof into two parts, together verifying the validity of $s_{i+1} \equiv_{L(K_{i+1})} t_{i+1}$:

1. Let $last(s_{i+1}) \in \mathcal{X}_k$. Then we must have $last(s_i) \in \mathcal{X}_k$ & $last(s) \in \mathcal{X}_k$ and therefore,

$$last(s_i) = last(t) \quad \text{&} \quad last(s) = last(t).$$

Let $z_s := last(s_{i+1})$, $z_t := last(t_{i+1})$, $y := last(s)$, and $x := last(s_i)$. Since
\( K_{i+1} = \Omega(K_i, L) \), it must be the case that

\[
L_{z_k} = L_{z_t} = \text{SupC}_{L_z}(S_{x,y})
\]

\[
(\forall u' \in L_{x}, w' \in L_{z_k})(u', w')_{k+1} \Rightarrow \theta_{z_k}(w') = \{ \sigma \in \theta_{x}(u') \mid \text{SupC}_{L_{z_k}}(S_{z_k,y_k}) \neq \emptyset \}
\]

\[
(\forall u' \in L_{x}, w' \in L_{z_t})(u', w')_{k+1} \Rightarrow \theta_{z_t}(w') = \{ \sigma \in \theta_{x}(u') \mid \text{SupC}_{L_{z_t}}(S_{z_t,y_t}) \neq \emptyset \}
\]

where whenever \( s_i \sigma \in K_i \) we denote the tokens extending the \( \sigma \)-continuations of the strings involved according to:

\[
s_i \sigma \hat{x}_s, \ t_i \sigma \hat{x}_t \in K_i \quad \text{and} \quad s \sigma \hat{y}_s, \ t \sigma \hat{y}_t \in L.
\]

But, say, \( s \equiv_L t \) implies

\[
s \sigma \hat{y}_s \in L \Rightarrow t \sigma \hat{y}_t \in L
\]

and by syntax, therefore, \( \hat{y}_s = \hat{y}_t \). Similarly, \( \hat{x}_s = \hat{x}_t \) and so we will have

\[
(\forall w' \in L_{z_k} = L_{z_t})(w')_{k+1} = \theta_{z_k}(w')
\]

and therefore, \( z_k = z_t \) (Figure 5.12).

2. Note: In this part, \( u \), \( v \) and \( w \) denote arbitrary words over \( (\Sigma_k \cup \mathcal{X}_k) \).

Let \( s_{i+1}u_{i+1} \in K_{i+1} \). If \( t_{i+1}u_{i+1} \notin K_{i+1} \), let us denote by \( v_{i+1} \) the largest prefix of \( u_{i+1} \) which satisfies \( t_{i+1}v_{i+1} \in K_{i+1} \). Observe that \( u_{i+1} \neq \epsilon \), and that \( v_{i+1} \) is well-defined since \( t_{i+1} \epsilon \in K_{i+1} \) and \( K_{i+1} \) is closed.

We may have the following cases:

(a) \( \text{last}(v_{i+1}) \in \Sigma_k \). Then by the same argument as in part (1) of the proof it follows that the strings \( s_{i+1}v_{i+1} \) and \( t_{i+1}v_{i+1} \) are extended by the same token, say \( z \in \mathcal{X}_k \), in \( K_{i+1} \). Therefore, \( t_{i+1}v_{i+1}z \in K_{i+1} \) and this violates the assumption that \( v_{i+1} \) is the largest prefix of \( u_{i+1} \) satisfying \( t_{i+1}v_{i+1} \in K_{i+1} \).
Figure 5.12: Illustration of Case (1)

(b) $\text{last}(v_{i+1}) \notin \Sigma_k$. Without loss of generality assume that $\text{last}(s_{i+1}) \in \mathcal{X}_k$ and $\text{last}(t_{i+1}) \in \mathcal{X}_k$. Let $u_{i+1} := v_{i+1}w_{i+1}$ and $\sigma := \text{first}(w_{i+1})$ and define

$$z := \begin{cases} 
\text{last}(v_{i+1}) & v_{i+1} \neq \varepsilon \\
\text{last}(s_{i+1}) = \text{last}(t_{i+1}) & v_{i+1} = \varepsilon 
\end{cases}$$

Since $s_{i+1}v_{i+1}\sigma \in K_{i+1}$ and $z = \text{last}(s_{i+1}v_{i+1})$, it must be the case that

$$(\exists w' \in L_z)\sigma \in \theta_z(w')$$

and since $t_{i+1}v_{i+1} \in K_{i+1}$ and $z = \text{last}(t_{i+1}v_{i+1})$, by syntax it follows that $t_{i+1}v_{i+1}\sigma \in K_{i+1}$, which violates the maximality of $v_{i+1}$.

The proof is thus complete.

For $L \in \mathcal{L}_k$ let $||L||_\mathcal{L} := card\{(\Sigma_k \cup \mathcal{X}_k)^*/ \equiv_{\mathcal{L}(L)}\}$. The following result follows from the lemma.

**Corollary 5.1** For all $j \in \mathbb{N}$, $||K_j||_\mathcal{L} \leq ||K_0||_\mathcal{L} \cdot ||L||_\mathcal{L} + 1$. 

94
Thus, whenever $|K_0|_L < \infty$ and $|L|_L < \infty$, we will get $|K_j|_L < \infty$ or more precisely,

$$(\exists m \in \mathbb{N})(\forall j \in \mathbb{N}) |K_j|_L \leq m$$

A structured language $L$ is called $L$-regular iff $|L|_L < \infty$. Note that the concept of regularity only concerns the language $L$ itself and does not have any implications for the regularity of its substructures. See the example of Page 99.

**Lemma 5.4** If $\mathcal{X}_k$ has finite cardinality, then for $m \in \mathbb{N},$

$$\text{card}\{L \in \mathcal{L}_k \mid |L|_L \leq m\} < \infty.$$  

**Note:** The exact statement of this lemma would be:

*If $\mathcal{X}_k/\sim$ has finite cardinality, then for $m \in \mathbb{N},$

$$\text{card}\{L/\sim \mid L \in \mathcal{L}_k \& |L|_L \leq m\} < \infty.$$*

The change of notation introduced in Page 43 is heavily used in this chapter. For example, the exact definition of the Nerode equivalence relation with respect to $L$ is:

$$(\forall s, t \in (\Sigma_k \cup \mathcal{X}_k)^*)s \equiv_L t \iff (\forall u, v \in (\Sigma_k \cup \mathcal{X}_k)^*)u \sim v \Rightarrow su \in L \Leftrightarrow tu \in L.$$  

**Proof:** Let $|L|$ denote the cardinality of the Nerode quotient set. Obviously for $L \in \mathcal{L}_k$, we have $|L| \leq |L|_L$ and therefore $|L|_L \leq m$ implies $|L| \leq m$. Also, since $L$ is after all a subset of $(\Sigma_k \cup \mathcal{X}_k)^*$, we will have:

$$\{L \in \mathcal{L}_k \mid |L|_L \leq m\} \subseteq \{L \subseteq (\Sigma_k \cup \mathcal{X}_k)^* \mid |L| \leq m\} \text{ and therefore}$$

$$\text{card}\{L \in \mathcal{L}_k \mid |L|_L \leq m\} \leq \text{card}\{L \subseteq (\Sigma_k \cup \mathcal{X}_k)^* \mid |L| \leq m\} < \infty$$

where the boundedness of the latter can be verified by a simple combinatoric argument.

The following lemma states that if we begin with a finite number of tokens in writing $L$ and $K_0$, only a finite number of tokens will be needed for writing the entire sequence $\{K_j\}_{j=0}^\infty$. In other words, a $K_0$-token transforms finitely into its more restricted versions as $\Omega(\cdot, L)$ iterates on $K_0$.
Lemma 5.5 Let $K_0 \preceq L \in \mathcal{L}_k$ and both languages be $\mathcal{L}$-regular. If $K_0$ is consistent at the boundaries then only a finite number of distinct tokens will appear in the languages of the sequence $\{(K_j)_{j=0}^\infty\}$.

Proof: First we refine the recognizer of $K_0$ by taking the quotient set $\{K_0/ \equiv_{\mathcal{L}(K_0)} \land \equiv_{\mathcal{L}(L)}\}$ as the state set. Observe that since both recognizers are finite, one can enumerate their transition labels; in particular, we are interested in those transition labels that are taken from $\mathcal{X}_k$:

$$y_0, y_1, \ldots, y_{m-1} \text{ for } L \quad \text{and} \quad x_0, x_1, \ldots, x_{n-1} \text{ for } K_0.$$ 

Note that the lists may contain repeated items.

Take an arbitrary token $x$ from the $K_0$-list. We will show that only a finite number of variants of $x$—namely, $z \in \mathcal{X}_k$ such that $z \preceq x$—can appear in the sequence $\{(K_j)_{j=0}^\infty\}$. We shall discuss possible choices for $L_z$ and $\theta_z$ separately.

- $L_z$: Let $y : (L_y, \theta_y)$ be the subsystem of $L$ which corresponds to $x : (L_z, \theta_z)$ of $K_0$. Formally, if the (unique) transition labeled by $x$ leads to the state $[s_0]$ in the $K_0$-automaton and $s$ is the unique string of $L$ satisfying $s_0 \equiv_k s$, then $y$ is the token labeling the transition entering $[s]$ in the $L$-automaton. Now as $\Omega(\cdot, L)$ iterates on the sequence $\{(K_j)_{j=0}^\infty\}$, $L_z$ can evolve only to those $L_z$ satisfying:

$$L_z = \text{Sup C}_{L_y}(S(\Sigma_{\text{illegal}}))$$

where

$$S(\Sigma_{\text{illegal}}) := \text{Sup}\{T' \preceq L_z | (\forall v' \in L_y, w' \in T')(v', w')_{k+1} \Rightarrow \theta_y(v') \cap \Sigma_{\text{illegal}} = \emptyset\}$$

and $\Sigma_{\text{illegal}}$ is some subset of $\Sigma_{k,u}$: $\Sigma_{\text{illegal}} \subseteq \Sigma_{k,u}$.

The above relation states that $L_z$ cannot have arbitrary forms; rather it can only be the supremal controllable sublanguage of $L_z$ which does not generate any output in the illegal event set $\Sigma_{\text{illegal}}$ (Figure 5.13).

Since each $L_z$ is characterized by a subset of $\Sigma_{k,u}$, there will be at most $2^{\mid \Sigma_{k,u} \mid}$ choices for any fixed $x$. 

96
Figure 5.13: Candidates for $L_z$

- $L_z$: For any $L_z$ found in the first part, $\theta_z$ can be at best the restriction of $\theta_x$ on $L_z$. Also, it may very well happen that an arbitrary subset of $\Sigma_k$ gets excluded (Figure 5.14).

Figure 5.14: Example: $\theta_{z_0} = \theta_{x_0} - \Sigma_{excl}$, where $\Sigma_{excl} = \{\beta\}$ is subtracted from the output map of $x_0$ since it turns out that no subsystem-level solution for cutting $\gamma$ off the output of $x_1$ can be found.

Therefore, for a fixed $L_z$, one can at most come up with $2^{|\Sigma_k|}$ different choices for the output function $\theta_z$.

Thus, a conservative bound on the number of distinct tokens that may appear in
the formulation of \( K_j \)'s would be
\[
_n,2^{|\Sigma_k|+|\Sigma_\L|}
\]
which is finite since \(|\Sigma_k| < \infty\).

**Remark:** We should mention that the result of this lemma will still hold if we relax the consistency condition. Indeed, in this case the tokens labeling the transitions of the refined \( K_1 \)-recognizer can have quite arbitrary forms, listed as, say:
\[
x_0^{(1)}, x_1^{(1)}, \ldots, x_{n-1}^{(1)}
\]
where some items in the list might be absent as the product structure typically shrinks while \( \Omega(\cdot, L) \) is iterating on \( K_0 \) (cf. Lemma 5.3).

But as soon as the first iteration step (i.e. the one taking \( K_0 \) to \( K_1 \)) completes, all the supremal controllable elements at levels \( k+1 \) downwards have been computed with respect to the multi-level requirement specification \( K_0 \). From this point on (i.e. for \( j > 1 \) in the sequence \( \{K_j\}_{j=0}^\infty \)), the low-level constructs \( x_k^{(j)} \) are modified only when changes are made to the transition structure of \( K_j \) around \( x_k^{(j)} \), and therefore its successor \( x_k^{(j+1)} \) can be determined solely by specifying the subsets \( \Sigma_{\text{illegat}} \) and \( \Sigma_{\text{excl}} \) introduced earlier.

From the previous lemmas the result below will follow:

**Theorem 5.1** Let \( K_0 \preceq L \in \mathcal{L}_k \) be structured languages and \( ||K_0||_{\mathcal{L}}, ||L||_{\mathcal{L}} < \infty \). Then only a finite number of distinct languages will appear in the sequence \( \{K_j\}_{j=0}^\infty \), defined by \( K_{j+1} := \Omega(K_j, L) \); \( j \in \mathbb{N} \). Moreover, we have \( ||K_j||_{\mathcal{L}} < \infty \) for all \( j \in \mathbb{N} \).

**Corollary 5.2** Let \( K_0 \preceq L \in \mathcal{L}_k \) be structured languages and \( ||K_0||_{\mathcal{L}}, ||L||_{\mathcal{L}} < \infty \). The iteration
\[
K_{j+1} := \Omega(K_j, L); \ j \in \mathbb{N}
\]
terminates in a finite number of steps. Moreover, if \( K_j^* \) is the resulting fixed point of \( \Omega(\cdot, L) \), we will have:
\[
||K_j^*||_{\mathcal{L}} \leq ||K_0||_{\mathcal{L}} ||L||_{\mathcal{L}} + 1
\]
The result just mentioned stipulates a regularity-like condition on the convergence of the iteration in a finite number of steps, and henceforth validates it as an effective computational algorithm for finding the supremal controllable sublanguage of a given language.

Since the result of $\Omega(\cdot, L)$'s application on $K \preceq L$ depends on $SupC(\cdot)$ computations on the underlying subsystems, the next task ahead is to verify whether the iteration scheme is applicable globally. In other words, we would like to investigate if subject to some structural conditions termination can be achieved at all levels. First we introduce the notion of "global regularity".

**Definition 5.1** $L \in \mathcal{L}_k$ is globally regular iff $L$ itself is $\mathcal{L}$-regular and every token $x : (L_x, \theta_x)$ appearing in the definition hierarchy of $L$ satisfies

$$||x||_\mathcal{X} := \text{card}\{(\Sigma_k \cup \mathcal{X}_k)^* / \equiv_{L_x, \theta_x}\} < \infty$$

in which case we say $x$ is $\mathcal{X}$-regular.

**Example:** The language $L$ in the example of Section 5.2 is globally regular, while $L \in \mathcal{L}_0$, partially defined in Figure 5.15, is not since $\text{card}\{(\Sigma_1 \cup \mathcal{X}_1)^* / \equiv_{L_{x_0}, \theta_{x_0}}\} = \infty$.

As mentioned before, in passing from $K_0$ to $K_1 = \Omega(K_0, L)$, we have to undertake computations of the form $L_z = SupC_{L_y}(S_{z,y})$, where $z$ is some token in the structure of $K_1$, and $x$ and $y$ are its counterparts in $K_0$ and $L$, respectively. To see if the iteration scheme is effective in this case, we have to make sure that $||S_{z,y}||_\mathcal{L} < \infty$. 

99
Fact 5.1 When restricted to $S_{x,y}$, the following refinement relation holds true:

$$\equiv_{L(x,y)} \land \equiv_{L(y,y)} \leq \equiv_{L(x,y)}$$

Proof: Take arbitrary $t_1, t_2 \in S_{x,y}$. Since $S_{x,y} \leq L_x \leq L_y$, we have the corresponding $r_1, r_2 \in L_x$ and $s_1, s_2 \in L_y$ such that $r_i \equiv_{k+1} s_i \equiv_{k+1} t_i$, $i = 1, 2$. Let $r_1 \equiv_{L(x,y)} r_2$ and $s_1 \equiv_{L(y,y)} s_2$.

1. Let $\text{last}(t_1) \in X_{k+1}$. Then we will have $\text{last}(r_1) \in X_{k+1}$ & $\text{last}(s_1) \in X_{k+1}$ and therefore,

$$\text{last}(r_1) = \text{last}(r_2) \land \text{last}(s_1) = \text{last}(s_2).$$

Let $x' := \text{last}(r_1)$, $y' := \text{last}(s_1)$, and $z_i' := \text{last}(t_i)$, $i = 1, 2$. By definition of $S_{x,y}$ it follows that: $(i = 1, 2)$

$$L_{x'} = L_{x'}$$

$$(\forall u' \in L_{x'})\theta_{x'}(u') = \{\sigma \in \theta_{x'}(u') | \theta_{x'}(r_i \sigma x'_i) \cap \Sigma_{k,u} = \theta_{y'}(s_i \sigma y'_i) \cap \Sigma_{k,u}\}$$

where whenever $r_1 \sigma \in L_x$ we denote by $\hat{x}_i'$ and $\hat{y}_i'$ the tokens which extend the $\sigma$-continuations of $r_i \sigma$ and $s_i \sigma$ in $L_x$ and $L_y$, respectively:

$$r_i \sigma \hat{x}_i' \in L_x \land s_i \sigma \hat{y}_i' \in L_y, i = 1, 2.$$ But, say, $s_1 \equiv_{L_y} s_2$ implies

$$s_1 \sigma \hat{y}_1' \in L_y \Rightarrow s_2 \sigma \hat{y}_1' \in L_y$$

and by syntax it follows that $\hat{y}_1' = \hat{y}_2'$. Similarly, $\hat{x}_1' = \hat{x}_2'$, and since $r_1 \equiv_{L_x} r_2$ and $s_1 \equiv_{L_y} s_2$ we will have:

$$\theta_x(r_1 \sigma \hat{x}_1') = \theta_x(r_2 \sigma \hat{x}_2')$$

and

$$\theta_y(s_1 \sigma \hat{y}_1') = \theta_y(s_2 \sigma \hat{y}_2').$$
Therefore,

\[(\forall u' \in L_{x'})\theta_{z'_1}(u') = \theta_{z'_2}(u')\]

and hence \(z'_1 = z'_2\).

2. Let \(t_1u \in S_{x,y}\). If \(t_2u \notin S_{x,y}\), let us denote by \(v\) the largest prefix of \(u\) which satisfies \(t_2v \in S_{x,y}\).

We may have the following cases:

(a) \(\text{last}(v) \in \Sigma_{k+1}\). By a similar argument as in part (1) of the proof it follows that the strings \(t_1v\) and \(t_2v\) are extended by the same token, say \(z' \in \Sigma_{k+1}\), in \(S_{x,y}\). Therefore, \(t_2vz' \in S_{x,y}\), and this violates the assumption that \(v\) is the largest prefix of \(u\) satisfying \(t_2v \in S_{x,y}\).

(b) \(\text{last}(v) \notin \Sigma_{k+1}\). Without loss of generality assume that \(\text{last}(t_i) \in \Sigma_{k+1}\), \(i = 1, 2\). Let \(u := vw\) and \(\sigma := \text{first}(w)\) and define

\[z' := \begin{cases} 
\text{last}(v) & v \neq \varepsilon \\
\text{last}(t_1) = \text{last}(t_2) & v = \varepsilon
\end{cases}\]

Since \(t_1v\sigma \in S_{x,y}\) and \(z' = \text{last}(t_1v)\) it must be the case that

\[(\exists w' \in L_{x'})\sigma \in \theta_{z'}(w')\]

And since \(t_2v \in S_{x,y}\) and \(z' = \text{last}(t_2v)\), by syntax it follows that \(t_2v\sigma \in S_{x,y}\), again violating the maximality of \(v\).

Therefore, \(t_1 \equiv_{\Sigma(S_{x,y})} t_2\) and the proof is complete.

Thus, if the languages \(L\) and \(K_0\) are globally regular, we have just proved that \(L_z = \text{Sup}C_{L_y}(S_{x,y})\) can be computed by forming the iteration:

\[
\begin{align*}
L' & := L_y \\
K'_0 & := S_{x,y} \\
K'_{j+1} & := \Omega(K'_j, L'); \ j \in \mathbb{N}
\end{align*}
\] (5.1)
To recap, the iteration step from $K_0$ to $K_1 := \Omega(K_0, L)$ completes only after all the supremal controllable elements at level $k + 1$ are computed using the scheme in (5.1).

Note that since with the above scheme $L_z = \text{SupC}_{L_y}(S_{z,y}) = K'_j$, for some $j^* \in \mathbb{N}$, by Lemma 5.3 we will have:

restricted to $L_z$, $\equiv \mathcal{C}(L_y) \land \equiv \mathcal{C}(S_{z,y}) \preceq \equiv \mathcal{C}(L_z)$

and therefore by Fact 5.1 (Page 100) we can write:

restricted to $L_z$, $\equiv \mathcal{C}(L_z, \theta_z) \land \equiv \mathcal{C}(L_y, \theta_y) \preceq \equiv \mathcal{C}(L_y) \land \equiv \mathcal{C}(S_{z,y}) \preceq \equiv \mathcal{C}(L_z)$ (5.2)

Finally, observe that the output function of $z$ is determined in the context of the high level language $K_1$ in which it is embedded. Namely, we have the relation $\theta_z = \theta_x - \Sigma_{excl}$, and it is not difficult to see that

restricted to $L_z$, $\equiv \theta_z \preceq \equiv \theta_x$

and so from (5.2) it follows that

restricted to $L_z$, $\equiv \mathcal{C}(L_y, \theta_y) \land \equiv \mathcal{C}(L_z, \theta_z) \preceq \equiv \mathcal{C}(L_z, \theta_z)$ (5.3)

This is a very important result. First of all, it completes the tedious inductive proof of the fact that if the languages $K_0$ and $L$ are globally regular, then for all $j \in \mathbb{N}$ and while taking the step from $K_j$ to $K_{j+1}$, the iteration scheme of (5.1) is always applicable for effective computation of $L_{x,j+1} = \text{SupC}_{L_y}(S_{x,j,y})$, since\(^1\)

restricted to $S_{x,j,y}$, $\equiv \mathcal{C}(L_y, \theta_y) \land \equiv \mathcal{C}(L_y, \theta_y) \preceq \equiv \mathcal{C}(S_{x,j,y})$

Since the results we have got thus far are independent of the level index $k$, one can slide the level frame from the top down to the bottom of the hierarchy and therefore the iteration scheme for finding the supremal controllable sublanguage is universally applicable.

\(^1\)In the analysis just performed, we used $x$ and $z$ for $x^0$ and $x^1$, respectively.
The relation (5.3) has also an interesting byproduct, which can best be expressed by the following corollary.

**Corollary 5.3** Let $K_0 \leq L \in \mathcal{L}_k$ be globally regular. Consider automata—with output maps whenever applicable—models for both languages and all tokens in their definition hierarchies, where we have refined the state set of every automaton in the $K_0$ family by forming the product structure $K_0 \times L$. Then the substructures of $K_0$ never grow as $\Omega(\cdot, L)$ iterates on $K_0$. In other words, no extra memory is needed at lower levels for implementing a high level command.
Chapter 6

Conclusions and Future Research

In this work we formalized common sense heuristics applied to the supervisory control of hierarchical discrete-event systems. The overall heuristic is divide and conquer, which suggests decomposing a problem of considerable size into more manageable subproblems. Decomposition of the problem is carried out recursively, bringing in a new dimension to the system exploration. Thus, the traditional exploration of the state space of a flat model is now carried out more efficiently along two dimensions, where a state lower down in the hierarchy is explored only when the information at the higher level is not conclusive for establishing a certain result. For example, consider the toy system of Figure 6.1 and suppose it is required that the system be shutdown after two cycles.

![Figure 6.1: Toy example](image)

Then the obvious control action is to disable 'a' after two occurrences of β. In this example, what happens inside the black box is immaterial and our horizontal
exploration of the state space is coupled with a downward move only at state '0'. Figure 6.2 illustrates how the state-tree of Toy has been traversed.

![State Tree Diagram]

**Figure 6.2: Traversal of the state-tree of Toy**

Observe that how the problem is broken down into subproblems is a question of modeling and is not discussed in this thesis. Generally, we take our cue from the system functionality, with special attention paid to spatial and temporal relations among the system components.

As an example, a counter that counts from 0 up to 999 can have a flat model with 1,000 states and an indiscriminating \( \text{incr} \) event. In contrast, it would be more sensible to structure the flat model as follows:

For fixed \( 0 \leq i, j \leq 9 \), group all states of the form \( ijk \) with \( 0 \leq k \leq 9 \) and label the resulting aggregate state with \( ij \). This amounts to existential quantification of the state space with respect to the variable \( k \). Now quantify the already aggregated state space with respect to \( j \) and label the resulting aggregate states with \( i \). Finally, the model is perfected by distinguishing between the increments of digits according to their significance. The resulting structured model is shown partially in Figure 6.3.

It is noteworthy that the choice of decimal representation for numbers is implicitly reflected in the structure of Figure 6.3. Thus, '203' is reached if three occurrences of \( \text{incr}_3 \) follow two occurrences of \( \text{incr}_1 \), etc.

In some applications, the requirement specification hints at a certain structure on the system model which may result in reducing the complexity of verification.
Figure 6.3: Structured model for a decimal counter

(or implementation) problem. For example, suppose in a large industrial unit a flag is set to announce the completion of a subprocess. The state space of the flat model enumerates all possible evaluations of the variables, including flag, and the internal states of the subprocess. Then the states of the subprocess can be quantified (aggregated) if a requirement is just concerned with the functionality of the subprocess as a whole.

Another related issue worthy of mentioning is that of the input-output abstraction of subsystems. The question can be posed in the following way: given a subsystem connected to its surroundings through input and output transitions, find a more compact, canonical structure that can replace the subsystem while maintaining the same input-output functionality. This process involves encoding the reachability as well as controllability information of subsystems. As far as reachability is concerned, all one needs to know is whether any of the output transitions is connected to any of the input transitions, which can be checked independently for any pair of input and output transitions.

But for controllability the coupling between the output transitions can be exceedingly complicated. For example, in Figure 6.4, $o_1$, $o_2$ and $o_3$ are all virtually controllable [Wan95] with respect to $i$, in the sense that they can be cut off by disabling a subset of controllable events inside the box. Moreover, for any of the three this can be achieved independently and without affecting the other two, e.g. by disabling '1' and '2' in the figure, $o_1$ is cut off but not $o_2$ or $o_3$. However, the other transition will be victim if one tries to exclude $\{o_1, o_3\}$ or $\{o_2, o_3\}$ simultaneously.
from the output.

![Diagram](image)

Figure 6.4: Coupling between the output events

Indeed, using TCT procedures Hiconsis and Higen one can show that the structure of Figure 6.4 cannot be simplified any further while preserving the same input-output reachability and controllability characteristics.

From this example one can envisage how complicated and perhaps infeasible derivation of a general case solution could be. As we have mentioned earlier, the techniques presented in this work, in contrast, explore the subsystems on-the-fly and only when it is needed. A thorough exploration of the system and abstraction of the internal structure of all subsystems is justified only when the system is to operate in different configurations—each associated with a certain requirement specification. Only then such an a priori analysis would help by eliminating the need for repeating a reachability/controllability test over and over again.

Before wrapping up the thesis by proposing a number of possible extensions to this work, we point out yet another case where our methods can have remarkable efficiency.

Naturally, since our algorithms are recursive, they exhibit their maximal power on inductive structures. As an example, consider the language $L$ of Figure 6.5.

Bring in the sublanguage $E$ of $L$, differing from $L$ only in lacking the $\gamma_0$-transition from '34' to '4', and suppose that it is desired to compute $\text{Sup}_{C_\gamma}(E)$. By the procedures of Chapter 4 it can be verified that $\text{Sup}_{C_\gamma}(E)$ has the same structure as $E$, except for substructure '3' being replaced by $\text{Sup}_{C_{L_3}}(E_3)$, where $E_3$ differs from $L_3$ only in lacking the $\gamma_1$-transition from '333' to '34'. This pattern continues down to
level $p - 1$, where $Sup C_{L_{33}}(E_{33})$ evaluates to $\{\epsilon\}$ ($E_{33}$ is defined similarly). It is notable that all the intermediate computational steps are equivalent up to a relabeling of events, and therefore the number of levels in this extreme case does not affect the complexity of the problem.

We close this chapter by an unordered list of some other issues untouched in this work but nevertheless important in their own right:

- The issue of deadlock freedom (or liveness) could be studied in our setup by extending $L_k$ to include non-closed languages. All definitions must be adapted accordingly.

- There are two different ways of composing a complex system from the simpler ones: sequential composition, which was what we have studied in this work, and parallel composition. Unfortunately, the latter may not be expressible directly in $L_k$. As an example, consider the languages $L_1$ and $L_2$ of Figure 6.6.

As explained in Example 1 of Page 21, the flat language $\mathcal{I}(L_1) || \mathcal{I}(L_2)$ cannot be expressed in $L_0$. Thus, one may be able to derive structural conditions under which parallel composition is well defined in $L_k$. For example, one way around this problem could be to force all parallel subsystems to synchronize on their output events (See [Law97]). I could not stress more the need for such a study, as the complexity introduced by parallel composition is exponential in the number of components (as opposed to linear in the sequential case),
and therefore the leading edge of hierarchical methodology is better manifested when parallel composition is incorporated into the theory.

- The techniques presented in this thesis yet have to prove their efficiency on real world examples. Using TCT, I have tested the flattened version of Toy of Figure 6.1\footnote{The black box was simulated by 20 two-state, single-transition processes running in parallel, resulting in a structure with $2^{20}$ states, which basically enumerates all possible interleavings of 20 transitions. Also for convenience I rearranged Toy a bit and made the black box to appear first in the composition.} on a Pentium system with 256 MB of RAM and 880 MB of swap space and not surprisingly, the system ran out of memory while attempting to compute $\mathcal{I}(E)$. This illustrates the potential of the hierarchical approach, but for greater credibility one should measure the performance on benchmarks and real world problems.

- Development of a software tool that implements the techniques of this work would be of particular relevance.
Bibliography


