Finite Transition Systems
C.A.R. Hoare, Series Editor

BACKHOUSE, R.C., Program Construction and Verification
BACKHOUSE, R.C., Syntax of Programming Languages
DEBAKKER, J.W., Mathematical Theory of Program Correctness
BARR, M. and WELLS, C., Category Theory for Computing Science
BEN-ARI, M., Principles of Concurrent and Distributed Programming
BEN-ARI, M., Mathematical Logic for Computer Science
BIRD, R. and WADLER, P., Introduction to Functional Programming
BORNAT, R., Programming from First Principles
BOVET, D.P. and CRESCENZI, P., Introduction to the Theory of Complexity
BUSTARD, D., ELDER, J. and WELSH, J., Concurrent Program Structures
CLARK, K. and MCCABE, F.G., Micro-Prolog: Programming in Logic
DAHL, O.-J., Verifiable Programming
DROMEY, R.G., How to Solve It by Computer
DUNCAN, E., Microprocessor Programming and Software Development
ELDER, J., Construction of Data Processing Software
ELLIOTT, R.J. and HOARE, C.A.R. (eds), Scientific Applications of Multiprocessors
FREEMAN, T.L. and PHILLIPS, R.C., Parallel Numerical Algorithms
GOLDSCHLAGER, L. and LISTER, A., Computer Science: A modern introduction (2nd edn)
GORDON, M.J.C., Programming Language Theory and Its Implementation
GRAY, P.M.D., KULKARNI, K.G. and PATON, N.W., Object-oriented Databases
HEHNER, E.C.R., The Logic of Programming
HENDERSON, P., Functional Programming: Application and implementation
HOARE, C.A.R., Communicating Sequential Processes
HOARE, C.A.R. and GORDON, M.J.C. (eds), Mechanized Reasoning and Hardware Design
HOARE, C.A.R. and SHEPHERDSOHN, J.C. (eds), Mechanical Logic and Programming Languages
HUGHES, J.G., Database Technology: A software engineering approach
HUGHES, J.G., Object-oriented Databases
INMOS LTD, Occam 2 Reference Manual
JACKSON, M.A., System Development
JOHNSTON, H., Learning to Program
JONES, C.B., Systematic Software Development Using VDM (2nd edn)
JONES, C.B. and SHAH, R.C.F. (eds), Case Studies in Systematic Software Development
JONES, G., Programming in Occam
JONES, G. and GOLDSMITH, M., Programming in Occam 2
JONES, N.D., GOMARD, C.K. and SESTOFF, P., Partial Evaluation and Automatic Program Generation
JOSEPH, M., PRASAD, V.R. and NATARAJAN, N., A Multiprocessor Operating System
KALDEWAIJ, A., Programming: The derivation of algorithms
KING, P.J.B., Computer and Communications Systems Performance Modelling
LALEMONT, R., Computation as Logic
LEW, A., Computer Science: A mathematical introduction
McCABE, F.G., Logic and Objects
McCABE, F.G., High-level Programmer's Guide to the 68000
MEYER, B., Introduction to the Theory of Programming Languages
MEYER, B., Object-oriented Software Construction
MILNER, R., Communication and Concurrency
MITCHELL, R., Abstract Data Types and Modula 2
MORGAN, C., Programming from Specifications
PEYTON JONES, S.L., The Implementation of Functional Programming Languages
PEYTON JONES, S. and LESTER, D., Implementing Functional Languages
POMBERGER, G., Software Engineering and Modula-2
POTTER, B., SINCLAIR, J. and TILL, D., An Introduction to Formal Specification and Z
REYNOLDS, J.C., The Craft of Programming
Finite Transition Systems
Semantics of Communicating Systems

André Arnold
Université Bordeaux-I

Translated by John Plaice
Université Laval, Québec

Masson

Prentice Hall
## Contents

Preface                                      vii  

1 Introduction                               1  

2 Transition systems                         5  
   2.1 Definitions and notations             5  
   2.2 A few examples                       10  
   2.3 Petri nets and transition systems    17  
   2.4 Process algebras and transition systems  20  
   2.5 Synchronous and asynchronous systems  24  

3 The synchronous product of transition systems  26  
   3.1 The free product of transition systems  26  
   3.2 The synchronous product of transition systems  27  
   3.3 An example: Peterson’s algorithm      29  
   3.4 The synchronous product and the semantics of systems  33  
   3.5 The synchronous product of parameterized transition systems  35  

4 Transition system logics                  46  
   4.1 Propositional logic                   47  
   4.2 Linear temporal logic                48  
   4.3 Wolper logic                         51  
   4.4 Hennessy–Milner logic                52  
   4.5 Dicky logic                          54  
   4.6 CTL                                  56  
   4.7 CTL*                                 57  
   4.8 Fairness conditions                  59
5 Verification of properties of transition systems
  5.1 Transition system logics as logics
  5.2 Transition system logics as algebras
  5.3 Verification

6 Fixpoints in transition systems
  6.1 Examples
  6.2 Fixpoints of monotone functions
  6.3 The μ-calculus
  6.4 Dicky calculus
  6.5 Strongly connected components and fairness

7 Indistinguishability in transition systems
  7.1 Indistinguishable states and transitions
  7.2 Indistinguishability classes
  7.3 Saturating homomorphisms

8 Equivalences of transition systems
  8.1 Equivalences induced by saturating homomorphisms
  8.2 Bisimulation
  8.3 Trace equivalences

9 The observation of transition systems
  9.1 Observational equivalence
  9.2 Observation criteria
  9.3 Other equivalences

10 Software tools

Bibliography

Index
Finite transition systems form one of the formalisms used to describe systems of processes. This formalism, although mathematically simple, can model most of the properties of such systems, and so plays an important rôle in the study of their semantics. Several theoretical tools, based on this formalism, have been developed, including equivalence relations between transition systems and languages stating their properties. The usefulness of these theoretical tools is supported by the existence of much analysis and validation software that is based on transition systems.

This book presents transition systems and the derived theoretical tools dealing with the semantics of systems of processes. It originates in a course for the French ‘DEA’, roughly equivalent to a master’s course, that attempted to synthesize the material on this subject, as well as in a research article [6]. This may explain why this work sometimes appears to be a tutorial and sometimes appears to be a research monograph.

The author would like to thank C. Kaiser and O. Rafiq for encouraging him to write such a book, M. Nivat, D. Bégay, P. Crubillé and A. Dicky for their contributions to its content, and J. Plaice for his help in improving the English version.

André Arnold
Chapter 1

Introduction

Most work on the semantics of ‘parallel’, ‘communicating’, ‘concurrent’ or ‘interacting’ processes is based on the concept of automaton. Not only can automata model the behavior of a system of processes at different levels of granularity, but verification tools, based on automata, can be developed: by representing a system by an automaton, it is possible to observe and verify certain properties, such as potential deadlock, of the system.

Although finite automaton theory was mainly developed in the framework of formal language theory and its applications to lexical and syntactic analysis, it has also been used in other domains, such as the study of the semantics of iterative sequential programs [58, 80]. For example, a flowchart is an automaton that describes the flow of control in a program.

More generally, a finite state automaton, formed of states and labeled transitions between those states, can describe a system whose state evolves over time. In fact, it was precisely for this purpose that automata were first introduced (see Minsky [68, first part]): they were used to model the behavior of neurons [65]. Kleene’s famous theorem classifying the languages recognized by a finite state automaton refers to the behavior of a neuron network [61].

In the case of a flowchart, a system’s state can be represented by the program counter. To model a sequential program, the state must be augmented to include not just the program counter’s value, but also the value of some of its variables (where their domain is finite, otherwise the associated automaton would no longer be finite). Each action of the program can only be executed from certain states, and executing an action can provoke a change of state. The actions are therefore represented by transitions in the automaton. Since an automaton can represent a system whose state changes with the execution of certain actions or with the occurrence of certain events, it can naturally represent systems of processes. Examples of the use of transition systems to describe systems of processes can be found in J. Beauquier and B. Bérard’s book on operating systems [12].

For systems of processes communicating through a shared memory, Karp and
Introduction

Miller [59] extended Ianov's model of sequential processes [58] by considering that certain actions made by the processes of a system can modify disjoint parts of the common memory. A system's control remains centralized, so it can still be represented by a finite state automaton. However, the actions which test or modify the contents of the common memory only test or modify part of the memory. By adding the states of the memory's states to the control states, the system can be described by a finite state automaton, assuming, of course, that the memory can only take a finite number of values. If the control itself is distributed, i.e., if the program counter is, as is the memory, composed of a vector and if each action tests and modifies only certain components of this vector, the resulting automaton belongs to the class of asynchronous automata, defined and studied by Zielonka [90].

Other models of systems of processes have been proposed: for example, Petri nets [17] and process algebras such as CCS (Calculus of Communicating Systems), SCCS (Synchronous Calculus of Communicating Systems) [53, 66, 67] and Meije [16]. All these models can be understood as specifying sets of states and transitions between these states. The states of a Petri net are its markings; the transitions between states are made by the firing, simultaneous or not (depending on the net's semantics), of transitions of the net. In process algebras a system's states are terms and the transitions are defined by the operational semantics of the computation, which indicates how and under what conditions a term transforms itself into another term.

Automata can therefore be considered to be the fundamental concept for the formal description of the behavior of a system of processes. The term transition system is used to better show that these automata are formal systems containing states and transitions, rather than machines to recognize certain languages. Unlike Keller's transition systems [60], which also model systems of processes, the transitions of the systems used here are labeled by action or event names, even though this is not absolutely necessary: as will be seen later, it is most important to be able to distinguish particular sets of transitions, e.g., the set of all transitions labeled by the same action name. Furthermore, although the definition of an automaton singles out one or more initial states and a set of terminal ('accepting') states, the definition of a transition system does not require the definition of these two particular sets. Rather, depending on the needs of the model, an arbitrary number of states can be distinguished. For example, to represent a CSP (Communicating Sequential Processes) [56] program, the set of states corresponding to a normal termination of the program and the set of states corresponding to a failure must be defined [43].

The set of states of a transition system representing a system of processes can be very large, even infinite. Although there is a clear theoretical line between finite and infinite transition systems, from a practical point of view, sets that are too large are infinite. In most of this work, a transition system's size is ignored. It is assumed to be finite where certain results require that assumption, and to be sufficiently small when certain algorithms are implemented. However, a notable
characteristic of these systems of processes is that they are represented by gigantic automata, generated by the combinatorial explosion of the number of states, and that the goal of much research in this area is to reduce the number of states of a system without losing essential information about its behavior.

Because of this problem of size, a system of processes is rarely described as a transition system. Formal description tools such as Petri nets or process algebras are not transition systems, but it is possible to translate a system's description in one of these formalisms into the transition system representing its behavior. Note that it is possible to give simple descriptions that generate infinite transition systems and that it is in general undecidable whether a description in a particular formalism generates a finite transition system. However, certain description formalisms do use the concept of transition system, as do certain languages for describing communication protocols [15, 28]. In fact, transition systems are a useful way to describe the behavior of such protocols (see Tanenbaum [83]). The purpose of this work is therefore to present transition systems as models of systems of interacting processes, to point out the major uses of these transition systems to define the semantics of those systems they model and to verify some of their properties.

Chapter 2 introduces transition systems and explains how they can represent the behavior of systems of processes. Chapter 3 defines a fundamental operation for transition systems: the synchronous product. This operation, introduced by Arnold and Nivat [3, 9, 70], allows the construction of a transition system representing a system composed of several interacting processes from the transition systems representing the component processes and from communication and synchronization constraints which those processes must satisfy. These constraints are the vectors of actions and events which can occur simultaneously in the system of processes. The synchronous product is important since it can express almost all communication and synchronization constraints.

The properties of transition systems are stated in formal languages called logics (this is perhaps not the best terminology, since we are not interested in proofs). Chapter 4 presents many logics, those most commonly used and those of special interest. We are not particularly interested in the different expressivities of these logics, so we do not go into the details about which properties they can or they cannot express. Instead, we emphasize what they have in common: how sets of objects (states, transitions and paths) are associated with formulas.

Chapter 5 shows that it is possible to verify properties defined in these logics. At this point, an algebraic point of view, used later and simpler to use, is introduced for these logics. This point of view amounts to interpreting the logic's operators as operators acting on the set of states, transitions or paths of a given transition system.

Many logical operators can be defined from simpler ones, using a fixpoint operator. Chapter 6 presents the general results about least fixpoints and shows how they can be used to define a great variety of properties using a very small number of logical operators.

Given a language to express properties, one can examine to what extent the
properties definable in this language can distinguish two states in a transition system. Chapter 7 introduces the concept of *indistinguishability*: two objects are indistinguishable under a set of properties if they satisfy exactly the same properties in that set. Some general results related to indistinguishability, applicable to any logic, are presented.

Having a formal definition of the semantics of a system of processes implies that it can be used to define precisely the concept of *equivalence* of two systems. Two systems can be compared or an implementation of a system can be compared with a specification. Chapter 8 examines how equivalences can be defined between transition systems. In fact, several different equivalence relations can be defined; depending on the case, the comparisons that one might wish to make can be different. Some of these equivalences, such as bisimulation equivalence, are based on the concept of indistinguishability. Chapter 9 presents other equivalences that take into account that certain transitions are not *observable* and that, intuitively, two transition systems whose executions differ only in the occurrences of unobservable actions must be equivalent.

To conclude, Chapter 10 briefly describes software tools implementing some of the theoretical concepts raised in this work.
Chapter 2

Transition systems

2.1 Definitions and notations

This chapter introduces transition systems and states the basic definitions. The intuition is that a transition system consists of a set of possible states for the system and a set of transitions—or state changes—which the system can effect. When a state change is the result of an external event or of an action made by the system, then that transition is labeled with that event or action. Of course, particular states or transitions in a transition system can be distinguished.

There are several kinds of transition systems (simple, labeled, parameterized), depending on the kind of information that is associated with states and transitions. For example, it is shown how transition systems can be used to model the components of a system of processes. Of particular interest is the modeling of asynchronous processes, where the sequence of actions and events can be defined without stating precisely when they occur, and synchronous processes, where each action can only take place with the tick of a clock.

2.1.1 Transition systems

A transition system is a quadruple $A = \langle S, T, \alpha, \beta \rangle$ where

- $S$ is a finite or infinite set of states,
- $T$ is a finite or infinite set of transitions,
- $\alpha$ and $\beta$ are two mappings from $T$ to $S$ which take each transition $t$ in $T$ to the two states $\alpha(t)$ and $\beta(t)$, respectively the source and the target of the transition $t$.

A transition $t$ with source $s$ and target $s'$ is written $t : s \rightarrow s'$. Several transitions can have the same source and target, i.e. the product mapping $\langle \alpha, \beta \rangle : T \rightarrow S \times S$ is not necessarily injective.
A transition system is finite if $S$ and $T$ are finite. Unless explicitly stated otherwise, it is assumed that only finite transition systems are considered.

**Paths**

A path of length $n$, $n > 0$, in a transition system $A$ is a sequence of transitions $t_1, \ldots, t_n$ such that $\forall i: 1 \leq i < n, \beta(t_i) = \alpha(t_{i+1})$. Similarly, an infinite path is an infinite sequence of transitions $t_1, \ldots, t_n, \ldots$ such that $\forall i \geq 1, \beta(t_i) = \alpha(t_{i+1})$.

Write $T^+$ for the set of finite paths and $T^\omega$ for the set of infinite paths. The mappings $\alpha$ and $\beta$ can be extended to $T^+$ by defining

$$\alpha(t_1 \cdots t_n) = \alpha(t_1),$$
$$\beta(t_1 \cdots t_n) = \beta(t_n).$$

A finite path $c$ therefore represents the evolution of a transition system from state $\alpha(c)$ to state $\beta(c)$.

Similarly the mapping $\alpha$ is extended to $T^\omega$ by defining

$$\alpha(t_1 \cdots \cdot) = \alpha(t_1).$$

An infinite path $c$ represents an infinite evolution of a transition system from state $\alpha(c)$.

A partial product over $T^+$ is defined as follows: if $c = t_1 \cdots t_n$ is a path of length $n$, if $c' = t'_1 \cdots t'_m$ is a path of length $m$, and if $\beta(c) = \alpha(c')$ then

$$c \cdot c' = t_1 \cdots t_n t'_1 \cdots t'_m$$

is a finite path of length $n + m$ and

$$\alpha(c \cdot c') = \alpha(c),$$
$$\beta(c \cdot c') = \beta(c').$$

This product, defined over $T^+ \times T^+$, can also be defined over $T^+ \times T^\omega$: if $c$ is a finite path and $c'$ an infinite path such that $\beta(c) = \alpha(c')$ then $c \cdot c'$ is an infinite path and $\alpha(c \cdot c') = \alpha(c)$.

Finally, it can be interesting to consider the paths of length zero, which are equated with the states of $S$: for each state $s$ of $S$, define the empty path $\varepsilon_s$ of length zero, and extend $\alpha$ and $\beta$ by

$$\alpha(\varepsilon_s) = \beta(\varepsilon_s) = s.$$ 

Write $T^*$ for the set of finite paths, empty or non-empty. The previously defined partial product is extended: if $c$ is a finite path and if $s = \alpha(c)$ and $s' = \beta(c)$, then $\varepsilon_s \cdot c = c = c \cdot \varepsilon_{s'}$; if $c$ is an infinite path and if $s = \alpha(c)$, then $\varepsilon_s \cdot c = c$. 
Labeled transition systems

A transition system labeled by an alphabet $A$ is a 5-tuple $A = (S, T, \alpha, \beta, \lambda)$ where

- $(S, T, \alpha, \beta)$ is a transition system,
- $\lambda$ is a mapping from $T$ to $A$ taking each transition $t$ to its label $\lambda(t)$.

Intuitively, the label of a transition indicates the action or the event which triggers the transition. It is therefore logical to assume that two different transitions cannot have the same source, target and label, i.e. it is not necessary to distinguish two transitions that are triggered by the same action and that both make the transition system pass from the same state $s$ to the same state $s'$. More formally, this property states that the product mapping $(\alpha, \lambda, \beta) : T \rightarrow S \times A \times S$ is injective. In this case, a transition $t$ can be designated by the triple $(\alpha(t), \lambda(t), \beta(t))$ and the set of transitions is a subset of the Cartesian product $S \times A \times S$. However, a labeled transition system can well be nondeterministic. In a given state, the same action can provoke two different transitions leading to different states: $\alpha(t) = \alpha(t')$ and $\lambda(t) = \lambda(t')$ do not necessarily imply $t = t'$. Write

$$t : s \rightarrow a \rightarrow s'$$

for a transition $t$ of source $s$, target $s'$ and label $a$.

If $c = t_1t_2\ldots$ is a path in a labeled transition system, the sequence of actions

$$\text{trace}(c) = \lambda(t_1)\lambda(t_2)\ldots$$

is called the trace of the path.

Regular expressions, extended to denote sets of infinite words, will be used to compactly denote certain sets of traces.

- If $a$ is a letter in alphabet $A$, $a$ is also a regular expression denoting the set formed of the single word $a$.
- If $e_1$ and $e_2$ are regular expressions denoting sets $E_1$ and $E_2$,
  - $e_1e_2$ is a regular expression denoting $E_1E_2 = \{u_1u_2 \mid u_i \in E_i\}$,
  - $e_1 + e_2$ is a regular expression denoting $E_1 \cup E_2$.
- If $e$ is a regular expression denoting $E$,
  - $e^*$ is a regular expression denoting the set
    $$\varepsilon \cup \{u_1\cdots u_n \mid u_i \in E, n \geq 1\}$$
    containing the empty word $\varepsilon$ and all the products of words in $E$,
  - $e^\omega$ is a regular expression denoting the set of infinite words
    $$\{u_1\cdots u_n \cdots \mid u_i \in E, u_i \neq \varepsilon\}.$$
Parameterized transition systems

Labeled transition systems can also be constructed by defining, for all $a \in A$, the sets $T_a = \lambda^{-1}(a) = \{ t \mid \lambda(t) = a \}$ of all the transitions labeled $a$. More generally, a transition system parameterized by $(\mathcal{X}, \mathcal{Y})$, where

- $\mathcal{X} = \{ X_1, \ldots, X_n \}$ is a finite set of state parameter names and
- $\mathcal{Y} = \{ Y_1, \ldots, Y_m \}$ is a finite set of transition parameter names,

is a transition system $\langle S, T, \alpha, \beta \rangle$ where the subsets $S_X$ of $S$ (state parameters) and $T_Y$ of $T$ (transition parameters) have been defined for each $X$ in $\mathcal{X}$ and each $Y$ in $\mathcal{Y}$.

A property can be associated with any subset of states or transitions of a transition system. For example, label $a$ is a property of all transitions labeled $a$; the parameter $T_a$ defined above regroups the transitions having that property. Transition properties that are not dependent on labels can also be defined using parameters. For example, transitions can be classified according to which 'critical section' they belong to. Should a transition system contain initial or final states, or other special states, appropriate parameters can be given. Other examples of parameters are given in subsection 2.2.5 (page 14) and in section 3.5 (page 35).

Let $\mathcal{A} = \langle S, T, \alpha, \beta, S_{X_1}, \ldots, S_{X_n}, T_{Y_1}, \ldots, T_{Y_m} \rangle$ be a transition system parameterized by $(\mathcal{X}, \mathcal{Y})$, $\mathcal{X} = \{ X_1, \ldots, X_n \}$ and $\mathcal{Y} = \{ Y_1, \ldots, Y_m \}$. Consider the two power sets $A_\sigma = \wp(\mathcal{X})$ and $A_\tau = \wp(\mathcal{Y})$, and the two mappings $\mu_\sigma : S \rightarrow A_\sigma$ and $\mu_\tau : T \rightarrow A_\tau$, called markings of the states and transitions, defined, respectively, by:

$$\mu_\sigma(s) = \{ X \mid s \in S_X \},$$
$$\mu_\tau(t) = \{ Y \mid t \in T_Y \}.$$

Then $\langle S, T, \alpha, \beta, \mu_\tau \rangle$ is a labeled transition system if the triple $\langle \alpha, \beta, \mu_\tau \rangle$ is injective, i.e. if for every pair of transitions $t$ and $t'$ such that $\alpha(t) = \alpha(t')$ and $\beta(t) = \beta(t')$, there exists $Y$ in $\mathcal{Y}$ such that $(t \in T_Y$ and $t' \not\in T_Y)$ or $(t \not\in T_Y$ and $t' \in T_Y)$.

Normally, the transition systems used will be labeled and parameterized, even though labeling the transitions can be done with transition parameters. In practice, the label that indicates an action or an event having triggered a state change plays a role quite different from that of other parameters.

A transition system which is neither labeled nor parameterized is a simple transition system.

### 2.1.2 Transition system homomorphisms

**Definition**

Let $\mathcal{A} = \langle S, T, \alpha, \beta \rangle$ and $\mathcal{A}' = \langle S', T', \alpha', \beta' \rangle$ be two transition systems. A homomorphism $h$ from $\mathcal{A}$ to $\mathcal{A}'$ is a pair $(h_\sigma, h_\tau)$ of mappings

$$h_\sigma : S \rightarrow S',$$
$$h_\tau : T \rightarrow T'.$$
which satisfies, for every transition \( t \) of \( T \):
\[
\alpha'(h_\tau(t)) = h_\sigma(\alpha(t)),
\]
\[
\beta'(h_\tau(t)) = h_\sigma(\beta(t)).
\]

A homomorphism \( h \) is surjective if the two mappings \( h_\sigma \) and \( h_\tau \) are surjective. If \( h \) is a surjective homomorphism from \( A \) to \( A' \), the transition system \( A' \) is the quotient of \( A \) under \( h \).

**Labeled transition system homomorphisms**

Let \( A = \langle S, T, \alpha, \beta, \lambda \rangle \) and \( A' = \langle S', T', \alpha', \beta', \lambda' \rangle \) be two transition systems labeled by the same alphabet. A labeled transition system homomorphism from \( A \) to \( A' \) is a homomorphism \( h \) which also satisfies the condition
\[
\lambda'(h_\tau(t)) = \lambda(t).
\]

**Parameterized transition system homomorphisms**

Let
\[
A = \langle S, T, \alpha, \beta, S_{X_1}, \ldots, S_{X_n}, T_{Y_1}, \ldots, T_{Y_m} \rangle
\]
and
\[
A' = \langle S', T', \alpha', \beta', S'_{X_1}, \ldots, S'_{X_n}, T'_{Y_1}, \ldots, T'_{Y_m} \rangle
\]
be two transition systems parameterized by \( (X, Y) \).

A parameterized transition system homomorphism must also satisfy the two additional properties:
\[
\forall s \in S, \forall X \in X, \quad \left( s \in S_X \Leftrightarrow h_\sigma(s) \in S'_X \right),
\]
\[
\forall t \in T, \forall Y \in Y, \quad \left( t \in T_Y \Leftrightarrow h_\tau(t) \in T'_Y \right),
\]
or else, using the previously defined markings \( \mu_\sigma \) and \( \mu_\tau \),
\[
\forall s \in S, \quad \mu_\sigma(s) = \mu'_\sigma(h_\sigma(s)),
\]
\[
\forall t \in T, \quad \mu_\tau(t) = \mu'_\tau(h_\tau(t)).
\]

**Transition system isomorphisms**

A simple (resp. labeled or parameterized) transition system isomorphism is a simple (resp. labeled or parameterized) transition system homomorphism where the two mappings \( h_\sigma \) and \( h_\tau \) are bijective. In this case, the inverse mappings \( g_\sigma \) and \( g_\tau \) can be defined, and \( g = \langle g_\sigma, g_\tau \rangle \) is itself a simple (resp. labeled or parameterized) transition system isomorphism. If two transition systems are isomorphic, the only difference between them can be the way in which the states and transitions are 'named'. But these names have no particular meaning and can be arbitrarily chosen, since it is through the labels of transitions or the parameters that particular properties of states or transitions are designated. In practice, however, the choice of names, particularly of states, can contribute significantly to the 'readability' of a transition system, as does the choice of identifiers in a program.
2.2 A few examples

It is shown here how the normal components of a system of processes can be represented by transition systems. There is nothing 'automatic' in the construction of such representations. This should come as no surprise, since whenever a model of an object is constructed, one must determine the salient aspects of that object that are to appear in the model and under what form they will appear.

2.2.1 A boolean variable

A boolean variable \(b\) can be viewed as a system that can change state, and can therefore be associated with a transition system \(B\). The states of this transition system correspond to the values of the boolean variable, called true and false; it is supposed that the variable will not take a third undefined value. The value of the variable can be modified by an assignment \(b := \text{true}\) or \(b := \text{false}\). These two assignment events imply four labeled transitions, shown using the notation presented on page 7:

\[
\begin{align*}
t_1 & : \text{true} \rightarrow b := \text{true} \rightarrow \text{true}, \\
t_2 & : \text{true} \rightarrow b := \text{false} \rightarrow \text{false}, \\
t_3 & : \text{false} \rightarrow b := \text{true} \rightarrow \text{true}, \\
t_4 & : \text{false} \rightarrow b := \text{false} \rightarrow \text{false}.
\end{align*}
\]

Assume now that one might wish to read or test the value of the variable. This action is an event that does not change the system’s state. The transitions of this event have the same source and target, and so there are two such transitions. But in fact, there are two different reading events: reading the value true and reading the value false. These events are written true! and false!, and to transition system \(B\) are added the two transitions

\[
\begin{align*}
t'_1 & : \text{true} \rightarrow \text{true!} \rightarrow \text{true}, \\
t'_2 & : \text{false} \rightarrow \text{false!} \rightarrow \text{false}.
\end{align*}
\]

Finally, for reasons which appear in the definition of the synchronous product, it is sometimes necessary to introduce the null event, written \(e\), which does nothing and does not modify the variable's state. These two transitions are therefore added:

\[
\begin{align*}
t''_1 & : \text{true} \rightarrow e \rightarrow \text{true}, \\
t''_2 & : \text{false} \rightarrow e \rightarrow \text{false}.
\end{align*}
\]

This transition system is represented in its normal graphical form in Figure 2.1 (page 15).

Only the initial state remains to be defined. This is done by introducing an initial parameter containing the states corresponding to the possible initial values
of the variable. It is arbitrarily assumed that \( b \) is always initialized with the value false, i.e. initial = \{false\}.

Suppose, and this is sometimes the case, that the reading is ‘destructive’: reading a variable returns the contents, but always leaves the variable in state false. The variable can still be modeled, by replacing transition

\[
t'_1 : \text{true} \rightarrow \text{true}! \rightarrow \text{true}
\]

by

\[
t'_1 : \text{true} \rightarrow \text{true}! \rightarrow \text{false}.
\]

### 2.2.2 A counter

Consider a counter that can take values 0, 1, 2 and 3, and that these values are used as the labels of the states of the representing transition system. The value of the counter can be incremented by action \text{inc} and decremented by action \text{dec}. The following transitions are obvious:

\[
\begin{align*}
0 & \rightarrow \text{inc} \rightarrow 1, \\
1 & \rightarrow \text{inc} \rightarrow 2, \\
2 & \rightarrow \text{inc} \rightarrow 3, \\
1 & \rightarrow \text{dec} \rightarrow 0, \\
2 & \rightarrow \text{dec} \rightarrow 1, \\
3 & \rightarrow \text{dec} \rightarrow 2.
\end{align*}
\]

What happens if the counter is incremented when its value is 3 or if it is decremented when its value is 0? Several choices are possible, such as:

1. ‘disallow’ incrementing 3 or decrementing 0; this is the simplest, but not necessarily the most realistic, choice;
2. define a counter modulo 4;
3. add a supplementary error (or overflow) value to the values taken by the counter.

In the first case, it suffices to disallow any transition with source 3 (resp. 0) and label \text{inc} (resp. \text{dec}) in the transition system.

In the second case, add the transitions

\[
3 \rightarrow \text{inc} \rightarrow 0
\]

and

\[
0 \rightarrow \text{dec} \rightarrow 3.
\]
Transition systems

In the third case, add the transitions

\[ 3 \rightarrow \text{inc} \rightarrow \text{error} \]

and

\[ 0 \rightarrow \text{dec} \rightarrow \text{error}. \]

One can then consider that there are no transitions with source error (once in this state, the counter does nothing) or that incrementing or decrementing error does not modify the value, yielding the two transitions:

\[
\begin{align*}
\text{error} & \rightarrow \text{inc} \rightarrow \text{error} \\
\text{error} & \rightarrow \text{dec} \rightarrow \text{error}.
\end{align*}
\]

As for the boolean variable case, it is possible to test the value of the counter by using the actions \((=n!\) and \(\neq n!\) for each value \(n\) that the counter can take, and by adding the transitions

\[ n \rightarrow (=n!) \rightarrow n \]

for each value \(n\) of the counter, and the transitions

\[ m \rightarrow (\neq n!) \rightarrow m \]

for each pair \((n, m)\) of distinct possible values of the counter.

‘Waiting’ transitions \(n \rightarrow e \rightarrow n\) can also be added and the initial state can also be defined.

2.2.3 A bounded buffer

Consider a two-slot buffer used as a queue and a two-letter alphabet with letters \(a\) and \(b\).

The states that the buffer can be in are labeled by the possible contents:

\[\text{empty}, a, b, aa, ab, ba, bb.\]

The events, or actions, are:

- enter a letter in the buffer, if the buffer is not full, or
- remove a letter from the buffer, if the buffer is not empty,

which yields the following transitions:

\[
\begin{align*}
\text{empty} & \rightarrow \text{enter}(a) \rightarrow a, \\
\text{empty} & \rightarrow \text{enter}(b) \rightarrow b, \\
a & \rightarrow \text{enter}(a) \rightarrow aa,
\end{align*}
\]
As previously, wait transitions can be added and state empty can be designated as the sole initial state.

A buffer of size $n$ with an alphabet of $k$ letters can be modeled similarly. The number of states in the corresponding system is

$$\frac{k^{n+1} - 1}{k - 1}.$$

One can also model an unbounded buffer by an infinite transition system.

### 2.2.4 A sequential program

Consider the program fragment:

```pseudocode
while true do
    1: if not b then
        begin
            2: b := true;
            3: proc;
            4: b := false
        end
```

It can be represented by a transition system $P$ with four states: 1, 2, 3 and 4 (the values of the 'program counter'). The possible actions for $P$ are $b := \text{true}$, $\text{proc}$, $b := \text{false}$, and the two tests, written $\text{b=true?}$ and $\text{b=false}$?, which correspond to the cases where $b$ was tested and found, respectively, to be equal to true or false. The result is the following set of transitions:

$t_1 : 1 \rightarrow \text{b = true?} \rightarrow 1,$
$t_2 : 1 \rightarrow \text{b = false?} \rightarrow 2,$
$t_3 : 2 \rightarrow \text{b := true} \rightarrow 3,$
$t_4 : 3 \rightarrow \text{proc} \rightarrow 4,$
$t_5 : 4 \rightarrow \text{b := false} \rightarrow 1.$
The initial state of this transition system is obviously 1. Its graphical representation is given in Figure 2.2 (page 15).

2.2.5 Peterson’s algorithm

The different transition systems constituting Peterson’s mutual exclusion algorithm for two processes, as presented in Raynal’s book [79], are given here. The purpose of such an algorithm is to ensure that the two processes will never be simultaneously in a particular part of their code, called a critical section. This algorithm uses three global variables: two boolean variables d0 and d1 initialized to false and an integer variable turn, initialized to 0, that can only take values 0 and 1. Process P0 executes the following code:

```
while true do
begin
1: {non-critical section}
2: d0 := true;
3: turn := 0;
4: wait(d1=false or turn=1);
5: {critical section}
6: d0 := false
end
```

while P1 executes the symmetric code obtained by permuting 0 and 1:

```
while true do
begin
1: {non-critical section}
2: d1 := true;
3: turn := 1;
4: wait(d0=false or turn=0);
5: {critical section}
6: d1 := false
end
```

The three variables d0, d1 and turn are represented by the three transition systems $\mathcal{D}_0$, $\mathcal{D}_1$ and $T$. The systems $\mathcal{D}_0$ and $\mathcal{D}_1$ represent boolean variables, identical to the one presented above. Since turn is an integer variable that can only take two values, 0 and 1, it is represented by a transition system $T$ similar to the one represented by a boolean variable. It is obtained by respectively replacing in $\mathcal{B}$:

- false, true, $b := false$, $b := true$, false!, true!

by

- 0, 1, $t := 0$, $t := 1$, 0!, 1!
Figure 2.1 Transition system $B$, representing a boolean variable.

Figure 2.2 Transition system $P$, representing the sequential program.
Process $P_i$ is represented by transition system $\mathcal{P}_i$. Action ncs represents access to the non-critical section and action cs represents access to the critical section; actions $t:=1$, $t:=0$, $t=1?$ and $t=0?$ represent the assignments to and the tests of variable turn; actions $d0:=\text{true}$, $d0:=\text{false}$, $d0=\text{true}?$ and $d0=\text{false}?$ represent the assignments to and the test of variable $d0$; finally, actions $d1:=\text{true}$, $d1:=\text{false}$, $d1=\text{true}?$ and $d1=\text{false}?$ represent the assignments to and the tests of variable $d1$.

The transitions of $\mathcal{P}_0$ are therefore

$$
\begin{align*}
&t_1 : \quad 1 \quad \rightarrow \quad \text{ncs} \quad \rightarrow \quad 2, \\
&t_2 : \quad 2 \quad \rightarrow \quad d0 := \text{true} \quad \rightarrow \quad 3, \\
&t_3 : \quad 3 \quad \rightarrow \quad t := 0 \quad \rightarrow \quad 4, \\
&t_4 : \quad 4 \quad \rightarrow \quad d1 = \text{false}? \quad \rightarrow \quad 5, \\
&t_5 : \quad 4 \quad \rightarrow \quad t = 1? \quad \rightarrow \quad 5, \\
&t_6 : \quad 5 \quad \rightarrow \quad cs \quad \rightarrow \quad 6, \\
&t_7 : \quad 6 \quad \rightarrow \quad d0 := \text{false} \quad \rightarrow \quad 1,
\end{align*}
$$

and the graphical representation of the transition system is given in Figure 2.3 (page 19). Similarly, transition system $\mathcal{P}_1$ is obtained by exchanging 0 and 1, which gives:

$$
\begin{align*}
&t'_1 : \quad 1 \quad \rightarrow \quad \text{ncs} \quad \rightarrow \quad 2, \\
&t'_2 : \quad 2 \quad \rightarrow \quad d1 := \text{true} \quad \rightarrow \quad 3, \\
&t'_3 : \quad 3 \quad \rightarrow \quad t := 1 \quad \rightarrow \quad 4, \\
&t'_4 : \quad 4 \quad \rightarrow \quad d0 = \text{false}? \quad \rightarrow \quad 5, \\
&t'_5 : \quad 4 \quad \rightarrow \quad t = 0? \quad \rightarrow \quad 5, \\
&t'_6 : \quad 5 \quad \rightarrow \quad cs \quad \rightarrow \quad 6, \\
&t'_7 : \quad 6 \quad \rightarrow \quad d1 := \text{false} \quad \rightarrow \quad 1.
\end{align*}
$$

In the two cases the initial state is 1.

Note that in $\mathcal{P}_0$, to pass from state 4 to state 5, either transition $t_4$ or transition $t_5$ must be used, which implies that the test of $d1$ returns false or the test of turn returns 1. So $\text{wait} (\text{dj=false or turn=j})$ is properly translated: if neither of the two conditions is verified the system cannot leave state 5 and execute its critical section.

To each system, add the transitions labeled by the null action $e$, which do not change the state. A process can remain inactive for an interval, as is the case on a single processor, in which case the execution of a process can be suspended so that another may be executed.

It is useful to add to each of the two transition systems the transition parameter $MB = \{t_2, t_3, t_4, t_5\}$ (resp. $MB' = \{t'_2, t'_3, t'_4, t'_5\}$) formed by the transitions that a process executes when it attempts to enter a critical section. It will then be possible to state properties about each process's critical section.
2.2.6 A variant of Peterson’s algorithm

In the above model of Peterson’s algorithm, \( \text{wait}(d1=\text{false or turn}=1) \) was translated into the two transitions

\[
\begin{align*}
t_4 : & \quad 4 \rightarrow d1 = \text{false}\? \rightarrow 5, \\
t_5 : & \quad 4 \quad \rightarrow t = 1\? \rightarrow 5.
\end{align*}
\]

If neither of the two transitions can be made the transition system remains in state 4, otherwise it goes into state 5. This model does not correctly model an active wait, where the value of this condition is repeatedly tested: if it is true the transition system passes into state 5, and if it is false the transition system remains in state 4 and the condition is again tested. The two transitions can therefore be replaced by:

\[
\begin{align*}
4 & \quad \rightarrow (d1 = \text{false} \text{? or } t = 1\?) \equiv \text{true} \rightarrow 5, \\
4 & \quad \rightarrow (d1 = \text{false} \text{? or } t = 1\?) \equiv \text{false} \rightarrow 4.
\end{align*}
\]

There are therefore two different loops on state 4, the loop \( 4 \rightarrow e \rightarrow 4 \), which states that the process voluntarily remained in the same state and the loop \( 4 \rightarrow (d1 = \text{false} \text{? or } t = 1\?) \equiv \text{false} \rightarrow 4 \), which states that the process tried to enter the critical section and could not do so because the condition was not satisfied. These two kinds of waiting, ‘voluntary’ and ‘forced’, are now differentiated, while they were not in the previous model. This differentiating is important since these models will be used to determine the possibilities of deadlock.

Since the condition to be tested to enter the critical section is the disjunction of two elementary conditions, these two elementary conditions could be successively tested, first \( d1=\text{false} \) then \( \text{turn}=1 \). A new intermediary state \( 4' \) could be introduced, along with the four transitions:

\[
\begin{align*}
4 & \quad \rightarrow d1 = \text{false}\? \rightarrow 5, \\
4 & \quad \rightarrow d1 = \text{true}\? \rightarrow 4', \\
4' & \quad \rightarrow t = 1\? \rightarrow 5, \\
4' & \quad \rightarrow t = 0\? \rightarrow 4.
\end{align*}
\]

The forced wait loop reappears as

\[
4 \quad \rightarrow d1 = \text{true}\? \rightarrow 4' \quad \rightarrow t = 0\? \rightarrow 4,
\]

but here the test to enter the critical section is no longer atomic (indivisible), which could have implications on the proper behavior of the algorithm.

2.3 Petri nets and transition systems

Petri nets are a well-known and widespread formalism to model concurrent systems. Since many references can be found in the literature (see [69, 74] for instance), the
definition of Petri nets is only recalled sufficiently to show that they can be seen as transition systems.

A Petri net is a tuple \( (P, T, Pre, Post) \) where

- \( P \) is a finite set of places,
- \( T \) is a finite set of transitions, and
- \( Pre \) and \( Post \) are two mappings from \( T \) into \( \wp(P) \).

The standard graphical representation of a Petri net consists of representing places by circles and transitions by thick short lines; the mappings \( Pre \) and \( Post \) are represented by arrows: there is an arrow from place \( p \) to transition \( t \) if \( p \in Pre(t) \), and an arrow from transition \( t \) to place \( p \) if \( p \in Post(t) \) (see Figure 2.4, page 19).

A marking of a Petri net is a mapping \( m \) from \( P \) into the set of natural numbers; it indicates how many tokens, represented by dots, the place \( p \) contains. In a marking \( m \), a transition \( t \) is fireable if and only if

\[
\forall p \in Pre(t), \quad m(p) > 0.
\]

If \( t \) is fireable, the firing of \( t \) produces a new marking \( m' \) defined by

\[
\forall p \in P, \quad m'(p) = m(p) - pre(p, t) + post(t, p),
\]

where

\[
pre(p, t) = \begin{cases} 
1 & \text{if } p \in Pre(t) \\ 
0 & \text{otherwise}, 
\end{cases}
\]

\[
post(t, p) = \begin{cases} 
1 & \text{if } p \in Post(t) \\ 
0 & \text{otherwise}. 
\end{cases}
\]

A marking \( m \) is reachable from an initial marking \( m_0 \) if there is a sequence of markings \( m_0, m_1, \ldots, m_n = m \) such that \( m_{i+1} \) is obtained by firing a transition \( t_i \) in the marking \( m_i \).

Thus, with a Petri net and an initial marking \( m_0 \), one can associate a transition system, labeled \( T \), whose states are the markings reachable from \( m_0 \), and whose labeled transitions are the triples \( m \rightarrow t \rightarrow m' \) such that \( t \) is fireable in \( m \) and produces \( m' \). Of course this transition system is only finite if the set of reachable markings is finite, i.e. is bounded above.

In some cases, independent transitions may be simultaneously fired. Two transitions \( t' \) and \( t'' \) are said to be independent if

\[
Pre(t') \cap \left( Pre(t'') \cup Post(t'') \right) = \emptyset
\]

and

\[
Pre(t'') \cap \left( Pre(t') \cup Post(t') \right) = \emptyset.
\]

In this case, if both transitions are fireable in \( m \), then \( m \rightarrow t' \rightarrow m' \) and \( m \rightarrow t'' \rightarrow m'' \), with \( t' \) fireable in \( m'' \) and \( t'' \) fireable in \( m' \), and there is a marking \( m_1 \).
Figure 2.3 Transition system $P_0$ of Peterson's algorithm.

Figure 2.4 A Petri net.
such that \( m'' \rightarrow t' \rightarrow m_1 \) and \( m' \rightarrow t'' \rightarrow m_1 \). Thus, the two transitions can be simultaneously fired to produce \( m_1 \), represented by \( m \rightarrow t', t'' \rightarrow m_1 \). In this case, the transition system associated with the Petri net is labeled by \( g(T) \) instead of \( T \).

As an example, consider the Petri net in Figure 2.4 (page 19), with three transitions \( t_1, t_2, t_3 \) and four places \( p_1, p_2, p_3, p_4 \), such that:

\[
\begin{align*}
Pre(t_1) &= \{p_1, p_2\}, & Post(t_1) &= \{p_3, p_4\} \\
Pre(t_2) &= \{p_3\}, & Post(t_2) &= \{p_1\} \\
Pre(t_3) &= \{p_4\}, & Post(t_3) &= \{p_2\}.
\end{align*}
\]

Starting from the initial marking \( m_0 \) defined by

\[
m_0(p_1) = 1, \ m_0(p_2) = 1, \ m_0(p_3) = 0, \ m_0(p_4) = 0,
\]

the following successive markings are accessible:

\[
\begin{align*}
m_1 &: m_1(p_1) = 0, \ m_1(p_2) = 0, \ m_1(p_3) = 1, \ m_1(p_4) = 1; \\
m_2 &: m_2(p_1) = 1, \ m_2(p_2) = 0, \ m_2(p_3) = 0, \ m_2(p_4) = 1; \\
m_3 &: m_3(p_1) = 0, \ m_3(p_2) = 1, \ m_3(p_3) = 1, \ m_3(p_4) = 0.
\end{align*}
\]

The system passes from one of these markings to another by firing one of the three transitions or by simultaneously firing the transitions \( t_2 \) and \( t_3 \), thereby yielding the transition system:

\[
\begin{align*}
m_0 & \rightarrow t_1 \rightarrow m_1, \\
m_1 & \rightarrow t_2 \rightarrow m_2, \\
m_1 & \rightarrow t_3 \rightarrow m_3, \\
m_1 & \rightarrow t_2, t_3 \rightarrow m_0, \\
m_2 & \rightarrow t_3 \rightarrow m_0, \\
m_3 & \rightarrow t_2 \rightarrow m_0.
\end{align*}
\]

If Petri nets are given a semantics whereby it is impossible to fire several transitions simultaneously, the transition \( m_1 \rightarrow t_2, t_3 \rightarrow m_0 \) can be removed from the system.

### 2.4 Process algebras and transition systems

Process algebras developed from Milner’s CCS [66] are descriptions of certain sets of processes. A process \( p \) is something that can execute some action \( a \), or react to an event \( a \), and transform itself into a new process \( p' \); this is indeed a labeled transition \( p \rightarrow a \rightarrow p' \). Therefore, it is natural to associate a labeled transition system with a process \( p \), similarly to that done with Petri nets. The states of such a transition system are all the processes in which \( p \) can be transformed by a sequence of actions or events. Its transitions are the triples \( p \rightarrow a \rightarrow p' \).
A process algebra is a formalism to describe processes. As the term ‘algebra’ suggests, a process is a term built up from elementary processes and operators on processes. Together with the ‘syntactic’ rules governing the construction of such terms, there are ‘semantic’ rules explaining which actions a process (represented by a term) can execute and how it is transformed. These rules are defined by induction on the structure of a term. Usually, they have the following form

\[
p_1 \mapsto a_1 \mapsto p'_1, \ldots, p_n \mapsto a_n \mapsto p'_n, f(p_1, \ldots, p_n) \mapsto b \mapsto f'(p_1, \ldots, p_n, p'_1, \ldots, p'_n), C
\]

where \( f \) and \( f' \) are operators of the algebra and \( C \) is a set of conditions on the rule’s application. Its meaning is that if the transitions above the bar exist (the assumption), then the transition below the bar exists, provided \( C \) is satisfied.

An example of such a process algebra is given; it is shown how a term of this algebra can be associated with a transition system.

It is supposed that this algebra contains a special process, \( NIL \), which does nothing. It is also supposed that there exists a set \( A \) of elementary events or actions such that

- \( A \) contains a special element, written \( \tau \), called the invisible event or the invisible action.
- If \( a \) is an event or action in \( A \) different from \( \tau \), then \( A \) also contains the complementary event or action, written \( \bar{a} \). If \( a \) is the complement of \( \bar{a} \) then \( \bar{a} \) is the complement of \( a \).

Finally, process variables are assumed to exist.

The following operators are used:

- **prefixing** if \( p \) is a process and \( a \) is an element of \( A \), then process \( ap \) is a process that executes \( a \) and then behaves as \( p \) (or that awaits event \( a \), and, upon its receipt, behaves like \( p \)). This is represented by

\[
\frac{ap \mapsto a \mapsto p'}{}
\]

which means that the transition below the line exists, without any condition.

- **choice** if \( p \) and \( q \) are two processes, and if \( p \) can execute \( a \) and then behaves as \( p' \) or if \( q \) can execute \( b \) and then behaves as \( q' \), then \( ap + bq \) is a process that executes \( a \) and then behaves as \( p \) or that executes \( b \) and then behaves as \( q \). This is represented by

\[
p \mapsto a \mapsto p' \quad q \mapsto b \mapsto q' \quad p + q \mapsto a \mapsto p' \quad p + q \mapsto b \mapsto q'.
\]

A generalized (associative and commutative) choice operator \( \Sigma \), with any number of arguments, can also be used. Let \( I \) be a set of indices and, for each \( i \in I \), let \( p_i \) be a process. Then \( \Sigma_{i \in I} p_i \) is a process. If for some \( i \in I \),
Transition systems

$p_i$ can execute $a$ and then behave as $q$, $\sum_{i \in I} p_i$ can also execute $a$ and then behave as $q$. This is represented by

\[ p_i \rightarrow a \rightarrow q, \quad i \in I. \]

**parallelism** if $p$ and $q$ are two processes, $p \parallel q$ is a process which can have three types of behavior:

- If $p$ can execute $a$ and then behaves as $p'$, then $p \parallel q$ can execute $a$ and behave as $p' \parallel q$.
- If $q$ can execute $b$ and then behaves as $q'$, then $p \parallel q$ can execute $b$ and behave as $p \parallel q'$.
- If $p$ can execute $a$ and then behaves as $p'$, if $q$ can execute $b$ and then behaves as $q'$, and if $a$ and $b$ are complementary, then $p \parallel q$ can execute $\tau$ and behave as $p' \parallel q'$.

This is represented by

\[ p \rightarrow a \rightarrow p' \quad q \rightarrow b \rightarrow q' \quad \frac{p \parallel q \rightarrow a \rightarrow p' \parallel q'}{p \parallel q \rightarrow b \rightarrow p \parallel q'}, \quad \text{if } a \text{ and } b \text{ are complementary.} \]

**restriction** if $p$ is a process and $a$ is an element of $A$, then $p \setminus a$ is a process; if $p$ can execute $b$ and then behave as $p'$ and if $b$ is different from $a$ and its complement, then $p \setminus a$ can execute $b$ and then behave as $p' \setminus a$.

This is represented by

\[ p \rightarrow b \rightarrow p' \quad p \setminus a \rightarrow b \rightarrow p' \setminus a, \quad \text{if } b \neq a, \bar{a}. \]

**recursion** if $x_1, x_2, \ldots, x_n$ are variables representing processes and if $p, p_1, p_2, \ldots, p_n$ are processes in which appear the variables $x_1, x_2, \ldots, x_n$, then $p$ where $(x_1 = p_1; x_2 = p_2; \ldots; x_n = p_n)$ is a process behaving as follows: let $p'$ be the process obtained by replacing in $p$ all the occurrences of the variables $x_i$ by the corresponding processes $p_i$, denoted by $p(x_1/p_1, \ldots, x_n/p_n)$. If $p'$ can execute $a$ and then behave as $p''$, then $p$ where $(x_1 = p_1; x_2 = p_2; \ldots; x_n = p_n)$ can execute $a$ and then behave as $p''$ where $(x_1 = p_1; x_2 = p_2; \ldots; x_n = p_n)$, which reads as

\[ p(x_1/p_1, \ldots, x_n/p_n) \rightarrow a \rightarrow p'' \quad q \rightarrow a \rightarrow q'', \]

where

\[ q = p \text{ where } (x_1 = p_1; x_2 = p_2; \ldots; x_n = p_n), \]

\[ q'' = p'' \text{ where } (x_1 = p_1; x_2 = p_2; \ldots; x_n = p_n). \]
As an example, consider the following process:

\[ p_0 = (x \parallel y) \downarrow a \text{ where } (x = ax + b:NIL ; y = \bar{a}y + c:NIL). \]

Let \( q_0 = (x \parallel y) \). Replacing the variables by their definitions gives

\[ q_0 = (ax + b:NIL \parallel \bar{a}y + c:NIL). \]

According to the rules for choice and parallelism, process \( q_0 \) can

- execute the invisible action \( \tau \) (the simultaneous execution of \( a \) by \( x \) and \( \bar{a} \) by \( y \)), transforming itself into \( q_0 = (x \parallel y) \);
- execute \( a \) in its first component and transform itself into
  \( (x \parallel \bar{a}y + c:NIL) \);
- execute \( b \) in its first component and transform itself into
  \( q_1 = (NIL \parallel \bar{a}y + c:NIL) \);
- execute \( \bar{a} \) in its second component and transform itself into
  \( (ax + b:NIL \parallel y) \);
- execute \( c \) in its second component and transform itself into
  \( q_2 = (ax + b:NIL \parallel NIL) \).

The rule about restriction ensures that \( q_0 \downarrow a \) can only be transformed into \( q_0 \downarrow a \) by \( \tau \), into \( q_1 \downarrow a \) by \( b \), and into \( q_2 \downarrow a \) by \( c \). Since

\[ p_0 = q_0 \downarrow a \text{ where } (x = ax + b:NIL ; y = \bar{a}y + c:NIL), \]

the recursion rule gives

\[
\begin{align*}
  p_0 & \rightarrow \tau \rightarrow p_0, \\
  p_0 & \rightarrow b \rightarrow p_1, \\
  p_0 & \rightarrow c \rightarrow p_2,
\end{align*}
\]

where

\[
\begin{align*}
p_1 &= q_1 \downarrow a \text{ where } (x = ax + b:NIL ; y = \bar{a}y + c:NIL), \\
p_2 &= q_2 \downarrow a \text{ where } (x = ax + b:NIL ; y = \bar{a}y + c:NIL).
\end{align*}
\]

Now, what can \( p_1 \) do? Replacing the variables by their definitions in \( q_1 \downarrow a \) gives

\( (NIL \parallel \bar{a}(\bar{a}y + c:NIL) + c:NIL) \downarrow a \), which can only execute \( c \), transforming itself into \( (NIL \parallel NIL) \downarrow a \), thus

\[ p_1 \rightarrow c \rightarrow p_3, \]

where

\[ p_3 = (NIL \parallel NIL) \downarrow a \text{ where } (x = ax + b:NIL ; y = \bar{a}y + c:NIL). \]

Similarly, \( p_2 \rightarrow a \rightarrow p_3 \).
The evolution of process $p_0$ is thus represented by the transition system:

\[
\begin{align*}
    p_0 & \rightarrow \tau \rightarrow p, \\
    p_0 & \rightarrow b \rightarrow p_1, \\
    p_0 & \rightarrow c \rightarrow p_2, \\
    p_1 & \rightarrow c \rightarrow p_3, \\
    p_2 & \rightarrow b \rightarrow p_3.
\end{align*}
\]

Thus, a process defines a transition system, as does a Petri net with an initial marking.

Conversely, let $A = \langle S, T, \alpha, \beta, \lambda \rangle$ be a finite transition system labeled by an alphabet $A$. Associate a term $q_s$ of this process algebra with each state $s$ in $S = \{s_1, s_2, \ldots, s_n\}$, as follows:

With each state $s$, associate a variable $x_s$ and a process $p_s$ such that

\[
p_s = \sum_{t \in T : \alpha(t) = s} \lambda(t)x_{\beta(t)}.
\]

It is then easy to see that the transition system associated with the term

\[
q_s = x_s \text{ where } (x_{s_1} = p_{s_1}, x_{s_2} = p_{s_2}, \ldots, x_{s_n} = p_{s_n})
\]

is the sub-transition system of $A$ consisting of states and transitions reachable from the state $s$.

### 2.5 Synchronous and asynchronous systems

Consider a transition system $A = \langle S, T, \alpha, \beta \rangle$. A finite path $c$ of source $s$ and target $s'$ is a sequence of transitions making the transition system pass from state $s$ to state $s'$. However, even if each of the transitions takes place in a well-defined interval, one cannot deduce the time at which each of the transitions takes place. One can only state that transition $t_{i+1}$ takes place in a finite but unbounded amount of time after transition $t_i$ (which preceded it on path $c$) takes place. This manner of interpreting the sequence of transitions that make up a path, the *asynchronous interpretation*, is standard. It is the one used, for example, for Petri nets.

One can also use a *synchronous interpretation*. Each transition then takes place during a tick provided by a clock, in such a way that two transitions follow each other in a path if they take place during two consecutive ticks of the clock. This implies, of course, that if the interval between two ticks is constant, then it is sufficiently large for each transition to finish during this interval; or, that a clock does not tick until the transition started with the previous tick has been completed. This is what is done when a program is executed on a processor: at each cycle, the instruction pointed to by the program counter is executed. Asynchronous systems can also be modeled with the synchronous interpretation. It suffices to state, using
a null transition, the fact that a transition system can remain in the same state for an arbitrary period of time; in the preceding examples, the null transition was written $e$. The name given to this action is of no matter: these wait transitions are ordinary transitions, and only in the study of the properties of a particular transition system might one decide to handle the transitions bearing a particular label differently.

Consider again Peterson's mutual exclusion algorithm. The two processes $P_0$ and $P_1$ are simultaneously executed on the same processor. This processor executes, during some cycles, instructions from $P_0$, leaving $P_1$ inactive. At other cycles, instructions from $P_1$ are executed and $P_0$ is inactive. A loop labeled $e$ is added to each state of $P_0$ and of $P_1$, allowing each process $P_i$ to be inactive at any moment.

In what follows, only the synchronous interpretation is considered, since it can represent asynchronous systems, using wait-loops, as well as systems which are a mix, since wait-loops need not be inserted on all states. Other advantages of the synchronous interpretation appear in the next chapter, when the interactions between the different components of a system of processes are studied.
Chapter 3

The synchronous product of transition systems

The components of a system of interacting processes, be they the processes themselves or other components, such as shared variables, can all be represented using transition systems. To obtain the transition system associated with the system itself, the synchronous product combines the component transition systems. This operation requires a formal description, given below, of the allowed interactions between the different components of a system of processes. To begin, the free product of transition systems, where there is no interaction between components, is examined.

3.1 The free product of transition systems

Consider $n$ transition systems $A_i = \langle S_i, T_i, \alpha_i, \beta_i \rangle$, $i = 1, \ldots, n$. The free product $A_1 \times \cdots \times A_n$ of those $n$ transition systems is the transition system $A = \langle S, T, \alpha, \beta \rangle$ defined by

\[
S = S_1 \times \cdots \times S_n, \\
T = T_1 \times \cdots \times T_n, \\
\alpha(t_1, \ldots, t_n) = \langle \alpha_1(t_1), \ldots, \alpha_n(t_n) \rangle, \\
\beta(t_1, \ldots, t_n) = \langle \beta_1(t_1), \ldots, \beta_n(t_n) \rangle.
\]

If, in addition, each $A_i$ is labeled by an alphabet $A_i$, the free product is a transition system labeled by the alphabet $A_1 \times \cdots \times A_n$; transitions are labeled as follows:

\[
\lambda(t_1, \ldots, t_n) = \langle \lambda_1(t_1), \ldots, \lambda_n(t_n) \rangle.
\]

If the transition system $A$ is in global state $s = \langle s_1, \ldots, s_n \rangle$, each component transition system $A_i$ is in state $s_i$. Each $A_i$ can independently effect transition $t_i$, changing to state $s'_i$. After having effected the global transition $t = \langle t_1, \ldots, t_n \rangle$, the transition system $A$ changes to global state $s' = \langle s'_1, \ldots, s'_n \rangle$. In the case of
labeled transition systems, the vector $\lambda(t)$ is the global action that triggered the global transition $t$.

The free product assumes that in a global system, all the component systems execute their transitions simultaneously, i.e. it is possible to divide time into intervals in such a way that during each of those intervals each component executes exactly one transition. In other words, the same 'clock' drives the different transition systems forming the product. This hypothesis is natural for synchronous systems, under the condition that the transitions of each component correspond to elementary actions or events. If this were not the case, one could replace each action or each event by a sequence of elementary actions or events.

### 3.2 The synchronous product of transition systems

When processes interact, not all possible global actions are useful, since the interaction is subject to communication and synchronization constraints. The transition system associated with the system of processes must therefore be a subsystem of the free product of the component transition systems. The communication and synchronization constraints that define the subsystem can always be simply expressed by the synchronous product, formally defined below. Some examples from the preceding chapter are re-examined before the formal presentation.

#### 3.2.1 Synchronization vectors

Consider the transition system $\mathcal{P}$ in Figure 2.2 (page 15) that represents the program in subsection 2.2.4:

\[
\begin{align*}
    t_1 & : 1 \quad \rightarrow b = \text{true?} \rightarrow 1, \\
    t_2 & : 1 \quad \rightarrow b = \text{false?} \rightarrow 2, \\
    t_3 & : 2 \quad \rightarrow b := \text{true} \rightarrow 3, \\
    t_4 & : 3 \quad \rightarrow \text{proc} \rightarrow 4, \\
    t_5 & : 4 \quad \rightarrow b := \text{false} \rightarrow 1,
\end{align*}
\]

and the transition system $\mathcal{B}$ that represents the boolean variable used by $\mathcal{P}$:

\[
\begin{align*}
    t'_1 & : \text{true} \quad \rightarrow b := \text{true} \rightarrow \text{true}, \\
    t'_2 & : \text{true} \quad \rightarrow b := \text{false} \rightarrow \text{false}, \\
    t'_3 & : \text{false} \quad \rightarrow b := \text{true} \rightarrow \text{true}, \\
    t'_4 & : \text{false} \quad \rightarrow b := \text{false} \rightarrow \text{false}, \\
    t'_5 & : \text{true} \quad \rightarrow \text{true!} \rightarrow \text{true}, \\
    t'_6 & : \text{false} \quad \rightarrow \text{false!} \rightarrow \text{false}, \\
    t'_7 & : \text{true} \quad \rightarrow e \rightarrow \text{true}, \\
    t'_8 & : \text{false} \quad \rightarrow e \rightarrow \text{false}.
\end{align*}
\]
Since $P$ has four states and five transitions and $B$ two states and eight transitions, their free product has eight states and forty transitions. For example, the global transition $(t_5, t'_1)$ makes the system pass from global state $(4, \text{true})$ to global state $(1, \text{true})$ by executing the global action $(b := \text{false}, b := \text{true})$.

Intuitively, $P$'s action $b := \text{false}$ sets the boolean variable to false. Process $B$, which represents this variable, must simultaneously execute a transition taking it to state false, i.e. a transition labeled $b := \text{false}$. Conversely, if the boolean variable executes a transition labeled $b := \text{false}$, then this variable must be set to false by $P$, i.e. $P$ must simultaneously execute action $b := \text{false}$. In other words, there is only one global action allowed by the first component when the first component is the action $b := \text{false}$: $(b := \text{false}, b := \text{false})$. It is also the only legal global action when the second component is action $b := \text{false}$.

Similarly, if $P$ executes action $b := \text{false?!}$, which tests the value of the boolean variable and finds it to be false, then the variable must simultaneously execute a transition labeled false!, which returns the value false upon reading. The only legal global action when the first component is $b := \text{false?!}$ or the second is false! is therefore $(b = \text{false?!}, \text{false!})$.

Finally, when proc is executed, process $P$ does not affect the variable. In this case, the boolean variable does nothing, represented by the null action e.

The authorized global actions are therefore:

$$
\begin{align*}
&\langle b := \text{true}, b := \text{true} \rangle, \\
&\langle b := \text{false}, b := \text{false} \rangle, \\
&\langle \text{true?!}, \text{true!} \rangle, \\
&\langle \text{false?!}, \text{false!} \rangle, \\
&\langle \text{proc}, \text{e} \rangle.
\end{align*}
$$

A synchronization constraint can now be defined. If $A_1, \ldots, A_n$ are alphabets representing actions or events, a synchronization constraint is a subset of the Cartesian product $A_1 \times \cdots \times A_n$. Each element of the subset is a synchronization vector representing a global action of the system of processes. The list of vectors given in the preceding example is a synchronization constraint.

This synchronization constraint is constant. It does not vary during the evolution of the system. In particular it does not depend on the state of the system or on one of its components.

3.2.2 Synchronous product

If $A_i, i = 1, \ldots, n$, are $n$ transition systems labeled by alphabets $A_i$, and if $I \subseteq A_1 \times \cdots \times A_n$ is a synchronization constraint, the synchronous product of the $A_i$ under $I$, written $(A_1, \ldots, A_n; I)$, is the transition subsystem of the free product of the $A_i$ containing only the global transitions $(t_1, \ldots, t_n)$ whose vectors of labels $(\lambda_1(t_1), \ldots, \lambda_n(t_n))$ are elements of $I$. In other words, the synchronous product
allows only those global transitions corresponding to action vectors included in the synchronization constraint.

Consider the synchronization constraint for the above example. The synchronous product of the transition systems \( P \) and \( B \) under this constraint gives the following transition system:

\[
\begin{align*}
\langle 1, \text{true} \rangle &\Rightarrow \langle b = \text{true}\?, \text{true}\! \rangle \Rightarrow \langle 1, \text{true} \rangle, \\
\langle 2, \text{true} \rangle &\Rightarrow \langle b := \text{true} , b := \text{true} \rangle \Rightarrow \langle 3, \text{true} \rangle, \\
\langle 3, \text{true} \rangle &\Rightarrow \langle \text{proc} , e \rangle \Rightarrow \langle 4, \text{true} \rangle, \\
\langle 4, \text{true} \rangle &\Rightarrow \langle b := \text{false} , b := \text{false} \rangle \Rightarrow \langle 1, \text{false} \rangle, \\
\langle 1, \text{false} \rangle &\Rightarrow \langle b = \text{false}\?, \text{false}\! \rangle \Rightarrow \langle 2, \text{false} \rangle, \\
\langle 2, \text{false} \rangle &\Rightarrow \langle b := \text{true} , b := \text{true} \rangle \Rightarrow \langle 3, \text{true} \rangle, \\
\langle 3, \text{false} \rangle &\Rightarrow \langle \text{proc} , e \rangle \Rightarrow \langle 4, \text{false} \rangle, \\
\langle 4, \text{false} \rangle &\Rightarrow \langle b := \text{false} , b := \text{false} \rangle \Rightarrow \langle 1, \text{false} \rangle.
\end{align*}
\]

Eliminating those states and transitions that not accessible from the initial global state \( \langle 1, \text{false} \rangle \), formed from the (in this case, unique) initial state of each component, gives the simpler system:

\[
\begin{align*}
\langle 1, \text{false} \rangle &\Rightarrow \langle b = \text{false}\?, \text{false}\! \rangle \Rightarrow \langle 2, \text{false} \rangle, \\
\langle 2, \text{false} \rangle &\Rightarrow \langle b := \text{true} , b := \text{true} \rangle \Rightarrow \langle 3, \text{true} \rangle, \\
\langle 3, \text{true} \rangle &\Rightarrow \langle \text{proc} , e \rangle \Rightarrow \langle 4, \text{true} \rangle, \\
\langle 4, \text{true} \rangle &\Rightarrow \langle b := \text{false} , b := \text{false} \rangle \Rightarrow \langle 1, \text{false} \rangle.
\end{align*}
\]

### 3.3 An example: Peterson’s algorithm

Consider the Peterson algorithm defined on page 14, composed of transition systems \( P_0, P_1, D_0, D_1 \) and \( T \). The synchronization constraint for this system is defined here, to construct the synchronous product representing this system of processes.

The intuitive meaning of the action \( t:=0 \) executed by process \( P_0 \) is to set the variable turn to 0. This variable must therefore simultaneously execute a transition labeled \( t:=0 \), which changes to state 0. Conversely, if variable turn executes a transition labeled \( t:=0 \), this variable is set to 0 by one of the processes which has access to it, and which simultaneously executes action \( t:=0 \) (in this particular example, only \( P_0 \) can execute this action). In the free product of transition systems \( P_0, P_1, D_0, D_1 \) and \( T \), global transitions \( t_1, t_2, t_3, t_4, t_5 \) of the following forms must be disallowed:

- transition \( t_1 \) is labeled \( t:=0 \) and transition \( t_5 \) is not labeled \( t:=0 \), or
• transition $t_1$ is not labeled $t := 0$ and transition $t_5$ is labeled $t := 0$.

Similarly, the global transitions where the second process does not set the variable turn to 1 at the same time that variable turn changes itself to 1 must also be disallowed.

As for the tests $t = 0?$ and $t = 1?$ executed by processes $P_0$ and $P_1$, considered as readings of a particular value of the variable turn, they must occur at the same time as the actions 0! and 1! of this variable. These latter actions can be interpreted as the generation of a particular value while reading.

Unlike during writing, the two processes might simultaneously test the value of the variable. However, it is supposed that their access to this variable is mutually exclusive and that, if the value of turn is read, exactly one of the two processes executes the action of reading the variable’s value. The same reasoning applies to the readings and writings of boolean variables $D_0$ and $D_1$.

Finally, and this is the case if Peterson’s algorithm is executed on a single processor, in each interval exactly one of the two processes can execute an action, and the other will wait, i.e., will execute the null action $e$.

Here is the complete list of all the global actions which can be executed by the system under the above assumptions (other assumptions, such as the possibility of two processes simultaneously executing independent actions, would give other lists):

\[
\begin{align*}
(d_0 := \text{false}, & \ e, & d_0 := \text{false}, & \ e, & \ e, & \ e) ; \\
(d_0 := \text{true}, & \ e, & d_0 := \text{true}, & \ e, & \ e, & \ e) ; \\
(e, & d_1 := \text{false}, & e, & d_1 := \text{false}, & \ e, & \ e) ; \\
(e, & d_1 := \text{true}, & e, & d_1 := \text{true}, & \ e, & \ e) ; \\
(t := 0, & \ e, & e, & e, & t := 0) ; \\
(e, & t := 1, & e, & e, & t := 1) ; \\
d_1 = \text{false}? , & \ e, & e, & \text{false!}, & \ e, & \ e) ; \\
d_1 = \text{true}? , & \ e, & e, & \text{true!}, & \ e, & \ e) ; \\
e, & d_0 = \text{false}? , & \text{false!}, & e, & e, & e) ; \\
e, & d_0 = \text{true}? , & \text{true!}, & e, & e, & e) ; \\
t = 1? , & \ e, & e, & e, & 1! ) ; \\
t = 0? , & \ e, & e, & e, & 0! ) ; \\
e, & t = 0? , & e, & e, & 0! ) ; \\
e, & t = 1? , & e, & e, & 1! ) ; \\
e, & cs , & e, & e, & e) ; \\
e, & ncs , & e, & e, & e) ; \\
cs , & e, & e, & e, & e) ; \\
ncs , & e, & e, & e, & e) ;
\end{align*}
\]

The initial state is $(1, 1, \text{false}, \text{false}, 0)$. If only the states and transitions reachable from that state are conserved, the resulting transition system (given in Figures 3.1, page 31, and 3.2, page 32, by the list of its transitions), contains no global state for which the two first components are both 5. State 5 is the state corresponding to the entry into the critical section. It is therefore impossible for both processes to enter their critical section at the same time, which is precisely the objective in a mutual exclusion algorithm.
Figure 3.1 The synchronous product of Peterson’s algorithm (beginning).
Figure 3.2 The synchronous product of Peterson’s algorithm (cont. and end).
3.4 The synchronous product and the semantics of systems

The transition system synchronous product is Arnold and Nivat’s [9, 70] fundamental operation for the definition of the semantics of a system of interacting processes. A few reasons are given for this thesis. From a purely theoretical vantage, if it is accepted that the transition system \( A = \langle S, T, \alpha, \beta \rangle \) associated with a set \( \{A_i = \langle S_i, T_i, \alpha_i, \beta_i \rangle | i = 1, \ldots, n \} \) of interacting transition systems is always a subsystem of the free product of these transition systems, it is always possible to label the transitions of each \( A_i \) so that \( A \) is a synchronous product: it suffices to label each transition of each \( A_i \) by its own name, i.e. to take \( A_i = T_i \) and \( \lambda_i(t) = t \) and to choose as synchronization constraint the set of all those global transitions that are to appear in the result. Clearly, this is not the way to proceed. In practice, given a process system and specifications of the interactions between the processes, one can a priori and quite naturally define a synchronization constraint which exactly describes these interactions, so that the synchronous product of the components represents the behavior of the system.

Examples

1. For processes interacting by shared variables, one can use the method illustrated above for Peterson’s algorithm: each process and each shared object is represented by a transition system. The synchronization vectors ensure that every access to an object must be executed at the same time as the corresponding action that the object takes. Depending on the situation, the simultaneous access to an object by several processes can be allowed, as can an action simultaneously accessing several objects. The synchronization vectors simply state the global actions allowed by the specification of the interactions.

2. Consider a system containing two boolean variables and a process that tests the equality of the two variables. The transition system representing this process contains the transitions labeled \( b=b' \) and \( b\neq b' \) and the transition systems representing the two boolean variables contain transitions labeled true! and false!, respectively looping on the states true and false. The synchronization vectors are:

\[
\begin{align*}
\{ & b = b', \quad \text{true!} \quad \text{true!} \}, \\
\{ & b = b', \quad \text{false!} \quad \text{false!} \}, \\
\{ & b \neq b', \quad \text{false!} \quad \text{true!} \}, \\
\{ & b \neq b', \quad \text{true!} \quad \text{false!} \}.
\end{align*}
\]

3. When processes communicate by message passing using communication channels, the channels can be considered to be shared objects, bringing the problem to that of example 1 above: sending a message over a channel is synchronized with the entry of the message into the channel; similarly, the reception of a message coming over the channel is synchronized with the exit of the message from the channel. As for the channel itself, the bounded buffer example (page 12) suffices if
the channel is bounded. Furthermore, if the channel can lose, duplicate, modify or reorder messages, this can easily be formalized in the transition system associated with it. In a study of Stenning's protocol, Vergamini [88] gave several transition systems corresponding to different kinds of channels.

4. Synchronization vectors can easily represent the rendezvous of CSP [42, 43]. For every variable $x$ and for every possible value $v$ of this variable, for every pair $(P, Q)$ of processes, the rendezvous ($P!v, Q?x$) is represented by the vector

$\langle \ldots, P!v, \ldots, Q?x, \ldots, \ldots, x := v, \ldots \rangle$,

whose components are equal to the empty action $e$, except for those associated with $P$, $Q$ and the variable $x$: emission by $P$ of a value $v$ to $Q$, reception by $Q$ in variable $x$ of a value coming from $P$, and assignment of $v$ to $x$ are the three simultaneous component elements of a CSP rendezvous. The interactions between processes using path expressions, as they are formalized in the COSY language [64], are also easily described using synchronization vectors [2].

5. Finally, the restriction operator in process algebras can force the simultaneous execution of two complementary actions and, in the case where a process can be represented by a set of transition systems, such as those proposed by Vergamini [88] and Guessarian and Niar-Dinedane [49], the limitations imposed on the behavior of the process by this operator are also easily transcribed under the form of a synchronization constraint.

As the previous examples show, writing synchronization vectors is a form of modeling, with all its arbitrariness. In particular, when one wishes to represent a system, it is often possible to make certain aspects appear in the component transition systems or in the synchronization vectors.

Modeling is still a difficult art that requires a certain know-how. This remark has also been made by Halpern and Fagin [52], who propose a model for communicating systems close to the synchronous product, where the synchronization vectors are called 'joint actions':

Systems can often be characterized by the types of actions that are allowed. Typical actions in a system might include reading and writing a shared variable, sending a message, receiving a message, and local computations. How these actions change the global state of the system will depend to some extent on the details of how we model the processes' local states and the environment. At this point, the choice of how to model a system, including choosing the state space for the processes and the environment and deciding on the set of runs that make up the system, is more of an art than a science.
3.5 The synchronous product of parameterized transition systems

Consider $k$ transition systems $A_i = \langle S_i, T_i, \alpha_i, \beta_i, \lambda_i \rangle$ labeled by an alphabet $A_i$. Let $I \subseteq A_1 \times \cdots \times A_k$ be a synchronization constraint and let $B = \langle S, T, \alpha, \beta, \lambda \rangle$ be the synchronous product, which is a transition system labeled by alphabet $I$.

Suppose that each transition system $A_i$ is also a transition system parameterized by $(X_i, Y_i)$, these types not necessarily being the same. Then $B$ is also a transition system parameterized by $(X, Y)$, where $X$ (resp. $Y$) is the disjoint union of the $X_i$ (resp. $Y_i$), i.e.

$$X = \{ \langle i, X \rangle | 1 \leq i \leq k, X \in X_i \},$$
$$Y = \{ \langle i, Y \rangle | 1 \leq i \leq k, Y \in Y_i \},$$

by defining

$$S_{(i,X)} = \{ \langle s_1, \ldots, s_i, \ldots, s_k \rangle \in S | s_i \in (S_i)_X \},$$
$$T_{(i,X)} = \{ \langle t_1, \ldots, t_i, \ldots, t_k \rangle \in T | t_i \in (T_i)_Y \}.$$ 

The alternating bit protocol

A standard example of a system easily modeled by transition systems—it was defined in this manner—is the alternating bit protocol [11]. It has often been used to illustrate the use of formal methods in the specification and validation of systems, in particular by Clarke et al. [24]

To simplify the handling of this example, Crubillé’s version [27] is used, since it allows synchronous products whose sizes are much smaller than those obtained with the original version. This simplification deals with three aspects:

- Instead of modeling a bidirectional (full-duplex) connection, only a unidirectional connection is modeled. One of the connected entities is the sender, which sends messages and receives acknowledgments labeled by a control bit. The other is the receiver, which receives the messages and sends the acknowledgments. The transition systems representing the sender and the receiver are the same as for the original algorithm, the only difference being for the initial states of the receiver, which here are states awaiting the arrival of a message, while in [11], they are, as for the sender, emission states.

- Another difference consists of making the assumption that the communications between the two entities are instantaneous: the messages or acknowledgments do not pass through a buffer.

- Finally, while in [11] and [27] two kinds of error are distinguished,
  - inversion of the control bit and
  - any other detected transmission error.
Only the first of those types is considered here. Since the two types of error provoke exactly the same transitions, it can be assumed that each of the two entities interprets a message that suffered from a transmission error as a message whose control bit is different from the expected one.

The actions made by the sender are:

- **em1** emission of a message labeled by a 1 bit,
- **em0** emission of a message labeled by a 0 bit,
- **ra1** reception of an acknowledgement labeled by a 1 bit,
- **ra0** reception of an acknowledgement labeled by a 0 bit.

The actions made by the receiver are:

- **rm1** reception of a message labeled by a 1 bit,
- **rm0** reception of a message labeled by a 0 bit,
- **ea1** emission of an acknowledgement labeled by a 1 bit,
- **ea0** emission of an acknowledgement labeled by a 0 bit.

Note that this list does not contain the empty action. It is supposed that no entity is ever inactive: it is the sequence of exchanges of messages and acknowledgements that is of interest, not the time separating those exchanges. The transition systems can therefore be interpreted synchronously, as below, and it can be assumed that at each instant each entity emits or receives information.

The transition system modeling the sender is:

\[
\begin{align*}
t_1 & : \quad \text{send}_1 \rightarrow \text{em1} \rightarrow \text{wait}_1, \\
t_2 & : \quad \text{send}_0 \rightarrow \text{em0} \rightarrow \text{wait}_0, \\
t_3 & : \quad \text{resend}_1 \rightarrow \text{em1} \rightarrow \text{wait}_1, \\
t_4 & : \quad \text{resend}_0 \rightarrow \text{em0} \rightarrow \text{wait}_0, \\
t_5 & : \quad \text{wait}_0 \rightarrow \text{ra0} \rightarrow \text{send}_1, \\
t_6 & : \quad \text{wait}_0 \rightarrow \text{ra1} \rightarrow \text{resend}_0, \\
t_7 & : \quad \text{wait}_1 \rightarrow \text{ra1} \rightarrow \text{send}_0, \\
t_8 & : \quad \text{wait}_1 \rightarrow \text{ra0} \rightarrow \text{resend}_1.
\end{align*}
\]

Its initial states are defined by a parameter, **initial**, equal to \{send\(1\), send\(0\}\).

Two transition parameters are also defined:

- \text{emission} = \{t_1, t_2\} contains the transitions where a message is emitted for the first time.
- \text{re-emission} = \{t_3, t_4\} contains the transitions where a message is re-emitted because it was not properly acknowledged.
The transition system modeling the receiver is:

\[ t'_1 : \text{wait}_1 \rightarrow \text{rm}_1 \rightarrow \text{send}_1, \]
\[ t'_2 : \text{wait}_1 \rightarrow \text{rm}_0 \rightarrow \text{resend}_0, \]
\[ t'_3 : \text{wait}_0 \rightarrow \text{rm}_0 \rightarrow \text{send}_0, \]
\[ t'_4 : \text{wait}_0 \rightarrow \text{rm}_1 \rightarrow \text{resend}_1, \]
\[ t'_5 : \text{send}_1 \rightarrow \text{ea}_1 \rightarrow \text{wait}_0, \]
\[ t'_6 : \text{send}_0 \rightarrow \text{ea}_0 \rightarrow \text{wait}_1, \]
\[ t'_7 : \text{resend}_1 \rightarrow \text{ea}_1 \rightarrow \text{wait}_0, \]
\[ t'_8 : \text{resend}_0 \rightarrow \text{ea}_0 \rightarrow \text{wait}_1. \]

Its initial states are defined by a parameter, initial, equal to \( \{\text{wait}_1, \text{wait}_0\} \). Two transition parameters are also defined here:

\[ \text{wellreceived} = \{t'_1, t'_3\} \] contains the transitions where a message is received and where the value of the control bit of that message is the expected value.

\[ \text{illreceived} = \{t'_2, t'_4\} \] contains the transitions where a message is received and where the value of the control bit of the message is not the expected value.

Several synchronization constraints are given to express various kinds of transmission between the two entities.

If the connection is perfect, every message sent by the sender is simultaneously received by the receiver without change of the control bit. The same applies to the acknowledgements sent by the receiver and received by the sender. The set \( I \) of synchronization vectors corresponding to this hypothesis is

\[ \{ \text{em}_0, \text{rm}_0 \}, \]
\[ \{ \text{em}_1, \text{rm}_1 \}, \]
\[ \{ \text{ra}_0, \text{ea}_0 \}, \]
\[ \{ \text{ra}_1, \text{ea}_1 \}. \]

If transmission errors may occur between the sender and the receiver, then a message sent with a particular control bit may also be received by the receiver with the control bit inverted, thereby yielding two supplementary vectors, to form \( I_\varepsilon \):

\[ \{ \text{em}_0, \text{rm}_0 \}, \]
\[ \{ \text{em}_1, \text{rm}_1 \}, \]
\[ \{ \text{ra}_0, \text{ea}_0 \}, \]
\[ \{ \text{ra}_1, \text{ea}_1 \}, \]
\[ \{ \text{em}_0, \text{rm}_1 \}, \]
\[ \{ \text{em}_1, \text{rm}_0 \}. \]

If the transmission is bad in the opposite direction, it is the control bit of the
acknowledgements that can be modified, which gives the set of vectors $I_r$:

\[
\{\begin{array}{l}
em0, \ rm0 \\
em1, \ rm1 \\
ra0, \ ea0 \\
ra1, \ ea0 \\
ra1, \ ea1 \\
ra0, \ ea1 \\
ra1, \ ea0
\end{array}\}
\]

Finally, the control bit can be inverted during any transmission, which is represented by the set $I_{cr} = I_c \cup I_r$:

\[
\{\begin{array}{l}
em0, \ rm0 \\
em1, \ rm1 \\
ra0, \ ea0 \\
ra1, \ ea1 \\
em0, \ rm1 \\
em1, \ rm0 \\
ra0, \ ea1 \\
ra1, \ ea0
\end{array}\}
\]

The synchronous products of the transition systems representing the sender and receiver can be built for each of the four synchronization constraints. The resulting transition systems are presented in the following pages. For each of the transition systems, the following transition parameters are shown:

**EM**, or, using the notation from earlier in this section, \((1, \text{emission})\), contains the transitions whose first component is in parameter \text{emission},

**RE**, or \((1, \text{re-emission})\), contains the transitions whose first component is in the parameter \text{re-emission},

**WR**, or \((2, \text{well-received})\), contains the transitions whose second component is in the parameter \text{well-received},

**IR**, or \((2, \text{ill-received})\), will contain the transitions whose second component is in the parameter \text{ill-received},

by indicating after each transition of each product transition system the parameters containing the transition.

The first of these products, obtained from the synchronization constraint $I$, is the transition system $\mathcal{A}$ shown in Figure 3.3 (page 39). By using $I_c$, transition system $\mathcal{A}_c$ is obtained, shown in Figure 3.5 (page 40). By using $I_r$, $\mathcal{A}_r$ is obtained, shown in Figure 3.7 (page 42). Finally, using $I_{cr}$ yields $\mathcal{A}_{cr}$, described in Figure 3.9 (page 44). These transition systems are graphically represented in Figures 3.4, 3.6, 3.8 and 3.10 (pages 39, 41, 43 and 45, respectively). One can follow the sequence of emissions and receptions of messages, according to the manner in which they were received, on these graphical representations, where the state names and the parameters are abbreviated and where the initial states are doubly circled.
Figure 3.3 Transition system $A$.

Figure 3.4 Graphical representation of transition system $A$. 

\[
\begin{align*}
\langle \text{send}_0, \text{wait}_0 \rangle & \rightarrow \langle \text{em}_0, \text{rm}_0 \rangle \rightarrow \langle \text{wait}_0, \text{send}_0 \rangle & \text{EM WR} \\
& \rightarrow \langle \text{em}_0, \text{rm}_1 \rangle \rightarrow \langle \text{wait}_0, \text{resend}_1 \rangle & \text{EM IR} \\
\langle \text{send}_1, \text{wait}_0 \rangle & \rightarrow \langle \text{em}_1, \text{rm}_0 \rangle \rightarrow \langle \text{wait}_1, \text{send}_0 \rangle & \text{EM WR} \\
& \rightarrow \langle \text{em}_1, \text{rm}_1 \rangle \rightarrow \langle \text{wait}_1, \text{resend}_1 \rangle & \text{EM IR} \\
\langle \text{resend}_0, \text{wait}_0 \rangle & \rightarrow \langle \text{em}_0, \text{rm}_0 \rangle \rightarrow \langle \text{wait}_0, \text{send}_0 \rangle & \text{RE WR} \\
& \rightarrow \langle \text{em}_0, \text{rm}_1 \rangle \rightarrow \langle \text{wait}_0, \text{resend}_1 \rangle & \text{RE IR} \\
\langle \text{send}_0, \text{wait}_1 \rangle & \rightarrow \langle \text{em}_0, \text{rm}_0 \rangle \rightarrow \langle \text{wait}_0, \text{resend}_0 \rangle & \text{EM IR} \\
& \rightarrow \langle \text{em}_0, \text{rm}_1 \rangle \rightarrow \langle \text{wait}_0, \text{send}_1 \rangle & \text{EM WR} \\
\langle \text{send}_1, \text{wait}_1 \rangle & \rightarrow \langle \text{em}_1, \text{rm}_0 \rangle \rightarrow \langle \text{wait}_1, \text{send}_0 \rangle & \text{EM IR} \\
& \rightarrow \langle \text{em}_1, \text{rm}_1 \rangle \rightarrow \langle \text{wait}_1, \text{send}_1 \rangle & \text{EM WR} \\
\langle \text{resend}_1, \text{wait}_1 \rangle & \rightarrow \langle \text{em}_1, \text{rm}_0 \rangle \rightarrow \langle \text{wait}_1, \text{resend}_0 \rangle & \text{RE IR} \\
& \rightarrow \langle \text{em}_1, \text{rm}_1 \rangle \rightarrow \langle \text{wait}_1, \text{resend}_1 \rangle & \text{RE WR} \\
\langle \text{wait}_0, \text{send}_0 \rangle & \rightarrow \langle \text{ra}_0, \text{ea}_0 \rangle \rightarrow \langle \text{send}_1, \text{wait}_1 \rangle \\
\langle \text{wait}_1, \text{send}_0 \rangle & \rightarrow \langle \text{ra}_0, \text{ea}_0 \rangle \rightarrow \langle \text{resend}_1, \text{wait}_2 \rangle \\
\langle \text{wait}_0, \text{send}_1 \rangle & \rightarrow \langle \text{ra}_1, \text{ea}_1 \rangle \rightarrow \langle \text{resend}_0, \text{wait}_0 \rangle \\
\langle \text{wait}_1, \text{send}_1 \rangle & \rightarrow \langle \text{ra}_1, \text{ea}_1 \rangle \rightarrow \langle \text{send}_0, \text{wait}_0 \rangle \\
\langle \text{wait}_0, \text{resend}_0 \rangle & \rightarrow \langle \text{ra}_0, \text{ea}_0 \rangle \rightarrow \langle \text{send}_1, \text{wait}_1 \rangle \\
\langle \text{wait}_1, \text{resend}_0 \rangle & \rightarrow \langle \text{ra}_0, \text{ea}_0 \rangle \rightarrow \langle \text{resend}_1, \text{wait}_1 \rangle \\
\langle \text{wait}_0, \text{resend}_1 \rangle & \rightarrow \langle \text{ra}_1, \text{ea}_1 \rangle \rightarrow \langle \text{resend}_0, \text{wait}_0 \rangle \\
\langle \text{wait}_1, \text{resend}_1 \rangle & \rightarrow \langle \text{ra}_1, \text{ea}_1 \rangle \rightarrow \langle \text{send}_0, \text{wait}_0 \rangle \\
\end{align*}
\]

Figure 3.5 Transition system \( A_c \).
Figure 3.6 Graphical representation of the transition system $A_e$. 
The synchronous product of transition systems

\[
\begin{align*}
(send_0, wait_0) & \rightarrow (em0, rm0) \rightarrow (wait_0, send_0) & EM & WR \\
(send_1, wait_0) & \rightarrow (em1, rm1) \rightarrow (wait_1, resend_1) & EM & IR \\
(resend_1, wait_0) & \rightarrow (em1, rm1) \rightarrow (wait_1, resend_1) & RE & IR \\
(send_0, wait_1) & \rightarrow (em0, rm0) \rightarrow (wait_0, resend_0) & EM & IR \\
(send_1, wait_1) & \rightarrow (em1, rm1) \rightarrow (wait_1, resend_1) & EM & WR \\
(resend_0, wait_1) & \rightarrow (em0, rm0) \rightarrow (wait_0, resend_0) & RE & IR \\
(wait_0, send_0) & \rightarrow (ra0, ea0) \rightarrow (send_1, wait_1) \\
(wait_1, send_1) & \rightarrow (ra0, ea0) \rightarrow (resend_0, wait_1) \\
(wait_0, resend_0) & \rightarrow (ra0, ea0) \rightarrow (send_1, wait_1) \\
(wait_1, resend_0) & \rightarrow (ra0, ea0) \rightarrow (resend_0, wait_1) \\
(wait_1, resend_1) & \rightarrow (ra0, ea0) \rightarrow (resend_1, wait_0) \\
(wait_1, send_1) & \rightarrow (ra1, ea0) \rightarrow (send_0, wait_0) \\
(wait_0, resend_0) & \rightarrow (ra1, ea0) \rightarrow (resend_0, wait_1) \\
(wait_1, resend_1) & \rightarrow (ra1, ea0) \rightarrow (resend_1, wait_0) \\
\end{align*}
\]

Figure 3.7 Transition system $A_r$. 

Figure 3.8 Graphical representation of the transition system $A_r$. 

The synchronous product of parameterized transition systems 43
Figure 3.9 Transition system $A_{cr}$. 
Figure 3.10 Graphical representation of the transition system \( A_{cr} \).
Chapter 4

Transition system logics

Transforming a process or a system of interacting processes into a transition system, possibly using the synchronous product, yields a formal description of that process or system. The system’s properties, written as properties of the transition system representing it, i.e. as properties of its states, transitions and paths, can be studied.

For example, in the transition system representing Peterson’s algorithm, given in Figure 3.1 (page 31), one might wish to determine

- if, in a given state, the two processes are both in their critical sections;
- if a given global state is not the source of any transition (deadlock in the strictest sense);
- if a given global state is not the source of an infinite path (deadlock in the broader sense);
- if in a global transition the two processes are trying to enter their critical sections;
- if from the target state of a global transition where a process tries to enter its critical section, the system necessarily reaches a global state where that process is in its critical section;
- if there exists an infinite path composed only of transitions where the two processes try to enter their critical sections and never do (livelock).

These properties are stated in languages called logics. A great variety of such logics exists; they apply to different kinds of transition systems (simple, labeled or parameterized) and depend on the kind of properties that are to be stated.

A number of logics will be presented here, more or less according to increasing expressivity. Propositional logics are studied first; these can only express purely local properties on states and transitions, and do not depend on the structure of the transition system being studied. Linear temporal logic is presented next; it can express properties about paths of a transition system. Finally, several branching-time temporal logics are presented; these can express state properties that can take into account the branching structure of a transition system, i.e. that a state can have several distinct successors.
4.1 Propositional logic

Let $A = \langle S, T, \alpha, \beta, S_{X_1}, \ldots, S_{X_n}, T_{Y_1}, \ldots, T_{Y_m} \rangle$ be a transition system parameterized by $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X} = \{X_1, \ldots, X_n\}$ and $\mathcal{Y} = \{Y_1, \ldots, Y_m\}$.

Consider a language formed of

- the constants 1 (true) and 0 (false),
- the elementary propositions $P_X$, for every $X$ in $\mathcal{X}$,
- the binary operators $\lor$ (disjunction) and $\land$ (conjunction),
- and the unary operator $\neg$ (negation).

The formulas of this logic are given by the following rules:

- Constants are formulas.
- Elementary propositions are formulas.
- If $F$ and $F'$ are formulas, then $F \lor F'$ and $F \land F'$ are formulas.
- If $F$ is a formula, then $\neg F$ is a formula.

For every formula $F$ formed this way, the satisfiability relation, written

$$A, s \models F,$$

is defined. It states that state $s$ of transition system $A$ satisfies the property defined by $F$. This relation is defined by induction over $F$'s construction:

- If $F = \mathbf{0}$, then $A, s \not\models F$.
- If $F = \mathbf{1}$, then $A, s \models F$.
- If $F = P_X$, then $A, s \models F$ if and only if $s \in S_X$.
- If $F = F_1 \lor F_2$, then $A, s \models F$ if and only if $A, s \models F_1$ or $A, s \models F_2$.
- If $F = F_1 \land F_2$, then $A, s \models F$ if and only if $A, s \models F_1$ and $A, s \models F_2$.
- If $F = \neg F'$, then $A, s \models F$ if and only if $A, s \not\models F'$.

Example

Consider the transition system of Figure 3.1, representing Peterson's algorithm. Let $CS_0$ (resp. $CS_1$) be the state parameter name associated with parameter $S_{CS_0}$ (resp. $S_{CS_1}$), composed of global states where process $P_0$ (resp. $P_1$) is in its critical section (in state 5) and associated with elementary proposition $P_{CS_0}$ (resp. $P_{CS_1}$).

The formula $P_{CS_0} \land P_{CS_1}$ is satisfied by the global states where the two processes are both in their critical sections: $A, s \models P_{CS_0} \land P_{CS_1}$ if and only if $s \in S_{CS_0} \cap S_{CS_1}$.

The formulas of this logic express state properties. If transition parameter propositions had been used instead of state parameter propositions, then the result would have been formulas expressing transition properties.
Example
Process $P_0$ (resp. $P_1$) was defined with the transition parameter name $MB_0$ (resp. $MB_1$). One can define, in the global transition system, the transition parameter $T_{MB_0}$ (resp. $T_{MB_1}$), formed of the global transitions whose first (resp. second) component is in $MB_0$ (resp. $MB_1$). If $Q_{MB_0}$ and $Q_{MB_1}$ are the elementary propositions associated with these two parameters, the global transitions where one of these two processes tries to enter its critical section are characterized by the formula $Q_{MB_0} \lor Q_{MB_1}$. □

4.2 Linear temporal logic

Linear temporal logic, which states path properties, can be used to study a system’s evolution (through time) by examining the state sequence in a system’s path; hence the term temporal. The logic is called linear because a system in a given state is only considered to have a single successor state in the next instant: the one given by the path currently being studied.

4.2.1 Definitions

Let $\mathcal{A} = (S, T, \alpha, \beta, S_{X_1}, \ldots, S_{X_n}, T_{Y_1}, \ldots, T_{Y_m})$ be a transition system parameterized by $(\mathcal{X}, \mathcal{Y})$. If the transition system $(S, T, \alpha, \beta, \lambda)$ is labeled by an alphabet $A$, it is assumed to be parameterized by $(\emptyset, A)$, taking each letter $a$ in $A$ to the parameter $T_a = \lambda^{-1}(a) = \{t \mid \lambda(t) = a\}$.

The logic presented here is the propositional logic built up from the elementary propositions associated with transition parameters, along with two new operators:

- the unary operator $\nu$ ('next'), and
- the binary operator $\mu$ ('until').

The satisfiability relation is defined by

$A, c \models F$,

where $c$ is a path of $\mathcal{A}$, empty or non-empty, finite or infinite, by induction over the construction of formula $F$:

- $A, c \models 1$, no matter what $c$ is.
- $A, c \not\models 0$, no matter what $c$ is.
- $A, c \models P_Y$ if and only if $c = t \cdot c'$ and $t \in T_Y$.
- $A, c \models F \lor F'$ if and only if $A, c \models F$ or $A, c \models F'$.
- $A, c \models F \land F'$ if and only if $A, c \models F$ and $A, c \models F'$.
- $A, c \models \neg F$ if and only if $A, c \not\models F$.
- $A, c \models \nu F$ if and only if $c = t \cdot c'$ and $A, c' \models F$. 
• \( A, c \models F U F' \) if and only if
  
  - \( A, c \models F' \), or
  
  - \( c = t_1 t_2 \cdots t_n \cdot c' \), with \( A, c' \models F' \) and
  
  \[ \forall i \in \{1, \ldots, n\}, A, t_i \cdots t_n \cdot c' \models F. \]

The intuitive interpretation of NF and of \( F U F' \) is therefore: let \( c = t_1 \cdot t_2 \cdots t_n \cdots \) be a path whose transition \( t_i \) is executed at instant \( i \) (which is compatible with the synchronous interpretation of transition systems). This path satisfies NF if the path formed of the transition executed starting from instant 2 (the ‘next’ instant) satisfies \( F \). It satisfies \( F U F' \) if there exists an instant \( k \) such that the path executed starting from that instant satisfies \( F' \) and all the paths executed starting from previous instants satisfy \( F \).

### 4.2.2 Some properties

1. **Path \( c \) is a non-empty path if and only if** \( A, c \models N1 \). Since every path satisfies 1, the above definition of \( A, c \models NF \) becomes: \( A, c \models N1 \) if and only if \( c = t \cdot c' \), which states that \( c \) is non-empty.

2. **If** \( A, c \models \forall_{Y \in Y} \, P_Y \), **then** \( c \) **is not empty**. This property’s reciprocal is not necessarily true: if there exists a transition \( t \) which is not in \( \bigcup_{Y \in Y} \, T_Y \), then for every non-empty path \( c = t \cdot c', A, c \not\models \forall_{Y \in Y} \, P_Y \).

3. **The relation** \( A, c \models \neg(1 U F) \) **is satisfied if and only if**
   
   - if \( c = t_1 \cdots t_n \) is a finite path of length \( n \),
     
     - \( A, \varepsilon_s \models \neg F \), where \( s = \beta(c) = \beta(t_n) \) is the target of the path \( t_1 \cdots t_n \) (\( \varepsilon_s \) designating the empty path of source \( s \)), and
     
     \[ \forall i, 1 \leq i \leq n, \ A, t_i \cdots t_n \models \neg F; \]
   
   - if \( c = t_1 \cdots t_n \cdots \) is an infinite path, \( \forall i \geq 1, \ A, t_i \cdots t_n \cdots \models \neg F. \)

   Since every path satisfies 1, the definition of \( A, c \models F U F' \) yields: \( A, c \models 1 U F \) if and only if \( c = c' \cdot c'' \), with \( A, c' \models F \). By negating each member of that equivalence, the stated property is obtained.

4. **The path \( c \) is infinite if and only if** \( A, c \models \neg(1 U N1) \). According to property 1, the empty path \( \varepsilon_s \) does not satisfy \( \neg N1 \), and therefore, according to property 3, a finite path cannot satisfy \( \neg(1 U N1) \). Conversely, if \( c = t_1 \cdots t_n \cdots \) is an infinite path, then according to property 1, all the non-empty paths \( t_i t_{i+1} \cdots \) satisfy \( \neg N1 \); therefore according to property 3, \( c \) satisfies \( \neg(1 U N1) \).
5. The relation \( A, c \models N(1U F) \) is satisfied if and only if \( A, c \models 1UNF \). Path \( c \) satisfies \( N(1U F) \) if and only if \( c = t \cdot c' \), with \( A, c' \models 1UF \). But \( c' \) satisfies \( 1UF \) if and only if \( c' = c_1 \cdot c_2 \), with \( A, c_2 \models F \).

If \( c' = c_1 \cdot c_2 \), since \( t \cdot c_1 \) is a non-empty path, it can also be written \( c'_1 \cdot t' \) (should \( c_1 \) be empty, \( c'_1 \) is also empty, and \( t = t' \)). If, in addition, \( c_2 \) satisfies \( F \), then \( t' \cdot c_2 \) satisfies \( NF \) and \( c = c'_1 \cdot t' \cdot c_2 \) satisfies \( 1U(NF) \).

Conversely, if \( c \) satisfies \( 1U(NF) \), then \( c = c'_1 \cdot c'_2 \), with \( A, c_2 \models NF \); therefore \( c'_2 = t' \cdot c_2 \), with \( A, c_2 \models F \). By writing \( c'_1 \cdot t' = t \cdot c_1 \) and \( c' = c_1 \cdot c_2 \), the result is obtained.

6. If \( c = t_1 \cdots t_n \cdots \) is an infinite path, there exists an infinite number of \( i \) such that \( A, t_it_{i+1} \cdots \models F \) if and only if \( A, c \models \neg(1U \neg(1UF)) \). According to property 3, \( c \) satisfies formula \( \neg(1U \neg(1UF)) \) if and only if for every \( i \geq 1 \), the path \( t_it_{i+1} \cdots \) satisfies \( 1UF \), i.e. if, still according to property 3, there exists \( j \geq i \) such that \( A, t_j \cdots \models F \).

Abbreviations

The abbreviation \( \diamond F \) is sometimes used for the formula \( 1UF \); the intuitive interpretation of \( A, c \models \diamond F \) is that \( c \) has a suffix which satisfies \( F \). The term \( \square F \) abbreviates \( \neg \diamond \neg F \). Path \( c \) satisfies \( \square F \) if all the suffixes of \( c \) satisfy \( F \) (see property 3). All the suffixes of an infinite path are non-empty, i.e. \( \square \text{N1} \); the formula appearing in property 6 can be written \( \square \diamond F \).

4.2.3 Example

Consider a transition system \( A \) labeled by the alphabet \( A = \{ a, b, e \} \). Let \( P_a \), \( P_b \) and \( P_e \) be the three elementary propositions. Then, for \( x \in A \), \( A, c \models P_x \) if and only if \( c \) is a non-empty path whose first transition is labeled \( x \).

Consider a formula \( F \) that states that a path \( c \) is an infinite path whose trace is in the set of infinite words \( (e^*ae*b)^*e^*b^\omega \cup (e^*ae*b)^\omega \), i.e. removing the letter \( e \) from the trace leaves a finite or infinite word of the form \( ababab \ldots \). For example, \( a \) could mean entering a critical section and \( b \) leaving the critical section and \( e \) any other action.

From property 4, a path is infinite if and only if it satisfies \( \neg(1U \neg \text{N1}) \) or, using the abbreviations, \( \square \text{N1} \).

The trace of a path \( c \) begins with \( e^*b \) if and only if that path satisfies \( P_e UP_b \). The formula \( \neg(P_a \lor N(P_e UP_b)) \) states that if the trace of a non-empty path starts by \( a \), then the \( a \) must be followed by \( e^*b \). From property 3, the fact that in the trace of a path every occurrence of \( a \) is followed by \( e^*b \) can be expressed by

\[
\neg\left(1U\neg(P_a \lor N(P_e UP_b))\right),
\]
\[ \Box(\neg P_a \lor N(P_e \cup P_b)). \]

It must still be stated that each \( b \) must be preceded by an \( a \). Equivalently, at the beginning of a path, or after a \( b \), only \( e \)'s are left (expressed by \( \Box P_e \)), or a finite number of \( e \) followed by an \( a \) (expressed by \( P_e \cup P_a \)). Let therefore

\[ G = (\Box P_e) \lor (P_e \cup P_a); \]

the condition is then expressed by

\[ G \land \Box(\neg P_b \lor NG). \]

The required formula \( F \) is therefore

\[
(\Box N1) \\
\land \Box(\neg P_a \lor N(P_e \cup P_b)) \\
\land G \land \Box(\neg P_b \lor NG).
\]

4.3 Wolper logic

Linear temporal logic cannot express all path properties. The simplest example is as follows: let \( F \) be an arbitrary path property, e.g. that the first transition is labeled by a given letter. Consider the property \( \text{Even}(F) \), satisfied by paths \( c = t_1 t_2 \cdots t_n \cdots \) such that for every even integer \( i \leq |c| \), the path \( t_i t_{i+1} \cdots \) satisfies property \( F \). Even if \( F \) can be expressed using linear temporal logic, it is not necessarily the case for \( \text{Even}(F) \). This is why P. Wolper [89] extended linear temporal logic by introducing new operators from computable languages on finite and infinite words.

Let \( X_n = \{x_1, x_2, \ldots, x_n\} \) be the alphabet formed of \( n \) symbols, interpreted as variables. Let \( L \) be a computable subset of \( X_n^\infty = X_n^* \cup X_n^\omega \). Define an \( n \)-ary operator, also written \( L \), whose meaning is given by the following rule:

- \( A, c \models L(F_1, F_2, \ldots, F_n) \) if and only if there exists a word (finite or infinite, but of the same length as \( c \)) \( u = x_{j_1} x_{j_2} \cdots x_{j_k} \cdots \in L \) such that for every integer \( m \leq |u| \),

\[ A, t_m t_{m+1} \cdots \models F_{j_m}. \]

In other words the successive suffixes of \( c \)—including \( c \) itself—must satisfy, respectively, the formulas \( F_{j_1}, F_{j_2}, \ldots \), obtained by replacing \( x_i \) by \( F_i \) in a word in \( L \).

For example, \( \text{Even}(F) \) can be written \( L(1, F) \), where \( L \) is the language

\[ (x_1 x_2)^* + (x_1 x_2)^* x_1 + (x_1 x_2)^\omega. \]
4.4 Hennessy–Milner logic

Hennessy–Milner logic [54] is used for labeled transition systems. It expresses state properties, but unlike in propositional logic, transitions can also be referred to, through their label. It is also, in a certain sense, a branching-time temporal logic, since, as will be seen, it is possible to take into account, when examining a state, the properties of the states that can be reached by executing certain actions.

4.4.1 Definitions

For a given alphabet \( A \), the formulas of the logic are formed from

- the constants \( 1 \) (true) and \( 0 \) (false),
- the binary operators \( \lor \) (disjunction) and \( \land \) (conjunction),
- the unary operator \( \neg \) (negation),
- unary operators, written \( \langle a \rangle \), for each letter \( a \) of the alphabet \( A \).

Each alphabet \( A \) defines a Hennessy–Milner logic; the formulas of that logic apply only to transition systems labeled by the alphabet \( A \). Henceforth, it is assumed that \( A \) is a given, fixed alphabet.

To define the satisfiability relation \( \mathcal{A}, s \models F \), it suffices to examine what happens when \( F \) is of the form \( \langle a \rangle F' \): a state satisfies \( \langle a \rangle F \) if it is the source of a transition labeled \( a \) whose target satisfies \( F' \), i.e. \( \mathcal{A}, s \models \langle a \rangle F \) if and only if there exists in \( \mathcal{A} \) a transition \( t = s \xrightarrow{a} s' \) such that \( \mathcal{A}, s' \models F' \). The other formulas are handled as in propositional logic.

4.4.2 Examples

1. A state satisfies \( \langle a \rangle 1 \) if and only if it is the source of a transition labeled \( a \).

2. A state satisfies \( \neg(\forall a \in A \langle a \rangle 1) \) if and only if that state is not the source of any transition.

3. A state \( s \) satisfies \( \neg\langle a \rangle \neg F \) if and only if every transition of source \( s \) and label \( a \) has a target satisfying \( F \). In fact, there exists a transition of source \( s \) and label \( a \) whose target satisfies \( \neg F \) if and only if \( s \) satisfies \( \langle a \rangle \neg F \). The formula \( \neg\langle a \rangle \neg F \) can be abbreviated \( [a] F \).

4. A state \( s \) satisfies the formula \( \neg\langle a \rangle \neg\langle b \rangle F \), which can also be written \( [a]\langle b \rangle F \), if and only if every transition of source \( s \) and label \( a \) leads to a state that is the source of a transition, labeled \( b \), whose target satisfies \( F \). Intuitively, action \( a \) has been executed, action \( b \) can always be executed, leading to a state satisfying \( F \).
4.4.3 Some extensions of Hennessy–Milner logic

Hennessy–Milner logic is applicable to labeled transition systems. It can be extended to transition systems that are both labeled and parameterized.

Taking into account state parameters
Let $A$ be an alphabet and let $\mathcal{X}$ be a set of state parameter names. For each parameter name $X$, define an elementary proposition $P_X$, as for propositional logic. The interpretation remains $A, s \models P_X$ if and only if $s \in S_X$, when $A$ is of type $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{Y}$ is arbitrary. These elementary propositions can then be added to the logic’s formulas.

For example, formula $P_X \land \langle a \rangle P_X$ is satisfied by all states in $S_X$ from which it is possible to reach a state in $S_X$ by a transition labeled $a$.

Taking into account transition parameters
In Hennessy–Milner logic, the formula $\langle a \rangle F$ is satisfied by the states that are sources of transitions whose targets satisfy $F$ and that satisfy the property ‘being labeled $a$’. But many other transition properties can be imagined. If a transition system contains transition parameters, it could be interesting to identify the sources of transitions satisfying a particular parameter $T_Y$ whose target satisfies $F$. If each of these transition parameters has an associated elementary proposition $P_Y$, this property is naturally represented by the formula $\langle P_Y \rangle F$.

More generally, consider a set $\mathcal{Y}$ of transition parameter names and the associated propositional logic built up from elementary propositions $P_Y$, for every $Y$ of $\mathcal{Y}$. The term transition formula is used for those formulas of the logic whose satisfiability relation $A, t \models G$ was defined previously, for the transition systems parameterized by $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X}$ is arbitrary.

The state formulas are therefore defined as follows:

- The constants $0$ and $1$ are state formulas.
- If $F$ and $F'$ are state formulas, then $F \lor F'$ and $F \land F'$ are state formulas.
- If $F$ is a state formula, then $\neg F$ is a state formula.
- If $G$ is a transition formula, and $F$ a state formula, then $\langle G \rangle F$ is a state formula.

The satisfiability relation for formulas of the form $\langle G \rangle F$ is given by $A, s \models \langle G \rangle F$ if and only if there exists a transition $t$ such that $\alpha(t) = s$, where $A, t \models G$ and $A, \beta(t) \models F$.

Note that in a formula such as $\langle 1 \rangle 1$, different occurrences of the same symbol may have different ‘meanings’ if they appear in subformulas of different types. This ambiguity can be eliminated by using different symbols, for example by using Dicky logic’s indices, although this is not really necessary since it is always clear from the context whether a symbol appears in a state or a transition subformula.

The two extensions can be combined by allowing elementary propositions in the state formulas. This possibility is not discussed here since the more general Dicky
logic, better adapted to parameterized transition systems, is presented in the next section.

4.5 Dicky logic

In [32], A. Dicky proposed to write state and transition properties in a manner that is particularly well adapted to parameterized transition systems; the ideas are presented here as a logic with state and transition formulas.

For each alphabet $A$, there is a Hennessy–Milner logic; similarly, for each pair $(\mathcal{X}, \mathcal{Y})$ of sets of state and transition parameter names, there is a Dicky logic; the formulas of that logic are applicable to transition systems parameterized by $(\mathcal{X}, \mathcal{Y})$.

A Dicky logic is built up using

- elementary propositions $P_X$ associated with parameters $X$ in $\mathcal{X}$,
- elementary propositions $Q_Y$ associated with parameters $Y$ in $\mathcal{Y}$,
- constants $0_\sigma$, $1_\sigma$, $0_\tau$ and $1_\tau$,
- binary operators $\lor_\sigma$, $\land_\sigma$, $\neg_\sigma$, $\lor_\tau$, $\land_\tau$ and $\neg_\tau$, and
- unary operators src, tgt, in and out.

The state and transition formulas are defined inductively by the following rules:

- The constants $0_\sigma$ and $1_\sigma$ are state formulas.
- The constants $0_\tau$ and $1_\tau$ are transition formulas.
- The elementary propositions $P_X$ associated with state parameter names are state formulas.
- The elementary propositions $P_Y$ associated with transition parameter names are transition formulas.
- If $F$ and $F'$ are state formulas, then $F \lor_\sigma F'$, $F \land_\sigma F'$ and $F \neg_\sigma F'$ are state formulas.
- If $G$ and $G'$ are transition formulas, then $G \lor_\tau G'$, $G \land_\tau G'$ and $G \neg_\tau G'$ are transition formulas.
- If $G$ is a transition formula, then src$(G)$ and tgt$(G)$ are state formulas.
- If $F$ is a state formula, then in$(F)$ and out$(F)$ are transition formulas.

Let $A = (S, T, \alpha, \beta, S_{X_1}, \ldots, S_{X_n}, T_{Y_1}, \ldots, T_{Y_m})$ be a transition system parameterized by $(\mathcal{X}, \mathcal{Y})$. Define the satisfiability relations $A, s \models F$ and $A, t \models G$, where $F$ is a state formula and $G$ a transition formula, by induction over the construction of these formulas.

- $A, s \models P_X$ if and only if $s \in S_X$.
- $A, t \models Q_Y$ if and only if $t \in T_Y$.
- $A, s \models 1_\sigma$.
- $A, s \not\models 0_\sigma$.
- $A, t \models 1_\tau$.
• $A, t \not\models 0_\tau$.
• $A, s \models F \lor_\sigma F'$ if and only if $A, s \models F$ or $A, s \models F'$.
• $A, s \models F \land_\sigma F'$ if and only if $A, s \models F$ and $A, s \models F'$.
• $A, s \models F \rightarrow_\sigma F'$ if and only if $A, s \models F$ and $A, s \not\models F'$.
• $A, t \models G \lor_\tau G'$ if and only if $A, t \models G$ or $A, t \models G'$.
• $A, t \models G \land_\tau G'$ if and only if $A, t \models G$ and $A, t \models G'$.
• $A, t \models G \rightarrow_\tau G'$ if and only if $A, t \models G$ and $A, t \not\models G'$.
• $A, s \models \text{src}(G)$ if and only if there exists a transition $t$ such that $s = \alpha(t)$ and $A, t \models G$.
• $A, s \models \text{tgt}(G)$ if and only if there exists a transition $t$ such that $s = \beta(t)$ and $A, t \models G$.
• $A, t \models \text{in}(F)$ if and only if $A, \beta(t) \models F$.
• $A, t \models \text{out}(F)$ if and only if $A, \alpha(t) \models F$.

Intuitively a state satisfies $\text{src}(G)$ (resp. $\text{tgt}(G)$) if it is the source (resp. the target) of a transition that satisfies $G$. A transition satisfies $\text{in}(F)$ (resp. $\text{out}(F)$) if it leads to (resp. comes from) a state that satisfies $F$.

Some preliminary observations can be made with this logic. First, there is no negation, only difference. The negation of a state formula $F$ is written $1_\sigma \rightarrow_\sigma F$, since $A, s \models 1_\sigma \rightarrow_\sigma F$ if and only if $A, s \not\models F$. Similarly, if $G$ is a transition formula, its negation is written $1_\tau \rightarrow_\tau G$. Second, unlike for Hennessy–Milner logic, the roles played by the source and the target of a transition are completely symmetrical.

The Hennessy–Milner formulas $\langle a \rangle F$ are translated into Dicky logic to illustrate the latter's capabilities. To do this, it is supposed that the labeling of a transition system $A$ is given through parameters: for each letter $a$ of the alphabet, define $T_a = \lambda^{-1}(a)$ and proposition $Q_a$.

Then $\langle a \rangle F$ can be written $\text{src}(Q_a \land_\tau \text{in}(F))$. In fact,

$A, s \models \text{src}(Q_a \land_\tau \text{in}(F))$

if and only if

$\exists t : \alpha(t) = s, A, t \models Q_a \land_\tau \text{in}(F)$

if and only if

$\exists t : \alpha(t) = s, A, t \models Q_a$ and $A, t \models \text{in}(F)$

if and only if

$\exists t : \alpha(t) = s, \lambda(t) = a$ and $A, \beta(t) \models F$.

The advantage of this logic is that it is exactly suited to transition systems, since its basic operators src and tgt correspond to the basic mappings $\alpha$ and $\beta$ used in the definition of transition systems, while $\in$ and out are just their reciprocals.
4.6 CTL

CTL, short for computation tree logic, was introduced by Clarke et al. [24, 35] to express properties serving to be verified about systems of processes represented by transition systems. This logic can be applied to unlabeled transition systems having only state parameters. It contains only state formulas, built up from:

- elementary propositions $P_X$ associated with the state parameter names,
- binary operators $\lor$ (disjunction) and $\land$ (conjunction),
- the unary operator $\neg$ (negation),
- two unary operators $AX$ and $EX$, and
- two binary operators $A[U\cdot]$ and $E[U\cdot]$.

Even though Clarke et al. do not make them appear explicitly,

- the constants $1$ (true) and $0$ (false)

are added here.

For the standard operators, the satisfiability relation is defined as before. Before defining it for the four specific operators, some remarks and a definition are in order.

In the transition systems considered by Clarke et al., a state is always the source of at least one transition. Since the transition systems used here do not necessarily have that property, the definitions given by those authors are changed accordingly. In particular, the definition of the satisfiability relation for the formulas $A[FUF']$ and $E[FUF']$ refers to infinite paths. These are replaced by maximal paths, i.e. paths that are infinite or that end with a state that is the source for no transition. This means lightly modifying the satisfiability relation $A, s \models A[FUF']$ given by Clarke et al. This modification implies adding the condition marked below by a double star. Below, it can be seen that this new definition remains compatible with the CTL verification algorithm, given on page 70, as well as another way of presenting the semantics of the CTL operators, given in section 6.1.2 (page 75).

Here are the definitions:

- $A, s \models EXF$ if and only if there exists a transition of source $s$ and target $s'$ such that $A, s' \models F$.
- $A, s \models AXF$ if and only if every state $s'$ that is the target of a transition of source $s$ is such that $A, s' \models F$.
- $A, s \models E[FUF']$ if and only if there exists a maximal path $c = t_1t_2\ldots t_n\ldots$ of source $s$ such that either
  - $A, s \models F'$, or
  - there exists an integer $k$ such that
    * $A, \beta(t_k) \models F'$ and
    * $\forall i, 1 \leq i \leq k, A, \alpha(t_i) \models F$.  


\( A, s \models A[FUF'] \) if and only if for every maximal path \( c = t_1 \cdots t_n \cdots \) of source \( s \), either

- \( A, s \models F' \),
- there exists an integer \( k \) such that
  - \( A, \beta(t_k) \models F' \) and
  - \( \forall i, 1 \leq i \leq k, A, \alpha(t_i) \models F \), or

** \( c \) is a finite path and for every \( i \), \( A, \alpha(t_i) \models F \) and \( A, \beta(t_i) \models F \).

Note that it follows from the definition of \( A, s \models AXF \) that if a state \( s \) is the source of no transition, then

\( A, s \models AXF \)

always holds, no matter the formula \( F \). This observation is frequently used below.

Note also the analogy between the EX operator and the Hennessy–Milner \( \langle a \rangle \) operator. The state \( s \) satisfies EXF if \( s \) has a successor state \( s' \) that satisfies \( F \). The operator \( \langle a \rangle \) only requires in addition that the transition passing from \( s \) to \( s' \) be labeled \( a \). This addition is clearly irrelevant for unlabeled transition systems.

Note finally that \( A, s \models EXF \) if and only if \( A, s \models \neg AX\neg F \). Therefore \( A, s \models \neg AX\neg F \) if and only if \( A, s \not\models AX\neg F \), if and only if there exists a successor state of \( s \) that satisfies \( F \), if and only if \( A, s \models EXF \).

**Examples**

Consider the formula \( E[1UF] \). It is satisfied by \( s \) if there exists a path starting in \( s \) going through a state satisfying \( F \). The formula \( \neg E[1U\neg F] \) is therefore satisfied by \( s \) if all of the paths starting in \( s \) go through only states satisfying \( F \). The formula \( A[1UF] \) is satisfied by \( s \) if every infinite path from \( s \) passes through at least one state that satisfies \( F \) (because of the ** condition, the finite maximal chains play no rôle here). The formula \( \neg A[1U\neg F] \) is therefore satisfied by \( s \) if that state is the source of at least one infinite path, all of whose states satisfy \( F \). In particular, the formula \( A[1U0] \) is satisfied by the states that are not the source of any infinite path, since if a state satisfies that formula and if it is the source of an infinite path, that path must pass by a state that satisfies \( F \), which is clearly impossible.

If in the CTL formulas the \( A[U.] \) and \( E[U.] \) operators are only used with their first argument equal to 1, as in the preceding examples, the result is the \( UB \) logic, proposed by Ben-Ari et al. [13], compared with CTL in [36].

The \( A[U.] \) and \( E[U.] \) operators are also used by Queille and Sifakis [78] under the names of \( INEV \) and \( POT \).

### 4.7 CTL*

CTL* [37, 40] is a temporal logic that contains state formulas analogous to the state formulas of CTL and path formulas similar to the formulas of linear temporal
logic. Like CTL, it is applicable to unlabeled transition systems having only state parameters. The state formulas are formed under the following rules:

- Every elementary proposition $P_X$ associated with a state parameter name is a state formula.
- The constants 1 and 0 are state formulas.
- If $F$ and $F'$ are state formulas, then $F \lor F'$ and $F \land F'$ are state formulas.
- If $F$ is a state formula, then $\neg F$ is a state formula.
- If $G$ is a path formula, then $EG$ is a state formula.

The path formulas are defined as follows:

- The constants 1 and 0 are path formulas.
- If $G$ and $G'$ are path formulas, then $G \lor G'$ and $G \land G'$ are path formulas.
- If $G$ is a path formula, then $\neg G$ is a path formula.
- If $G$ is a path formula, then $NG$ is a path formula.
- If $G$ and $G'$ are path formulas, then $GUG'$ is a path formula.
- If $F$ is a state formula, then $DF$ is a path formula.

Note that unlike the traditional representations found in [37, 40], the $D$ operator was introduced here to transform state formulas explicitly into path formulas. Furthermore, to be completely rigorous, the constants 1 and 0 and the logical operators $\lor$, $\land$ and $\neg$ should have been distinguished according to their type, as was done in the presentation of Dicky's logic. This was not done to simplify the notation.

The satisfiability relation is defined as follows, by induction over the construction of formulas, where $s$ is a state and $c$ a path:

- $A, s \models P_X$ if and only if $s \in S_X$.
- $A, s \models 1$.
- $A, s \not\models 0$.
- $A, s \models F \lor F'$ if and only if $A, s \models F$ or $A, s \models F'$.
- $A, s \models F \land F'$ if and only if $A, s \models F$ and $A, s \models F'$.
- $A, s \models \neg F$ if and only if $A, s \not\models F$.
- $A, s \models EG$ if and only if there exists a maximal path $c$ of source $s$ such that $A, c \models G$.
- $A, c \models 1$.
- $A, c \not\models 0$.
- $A, c \models G \lor G'$ if and only if $A, c \models G$ or $A, c \models G'$.
- $A, c \models G \land G'$ if and only if $A, c \models G$ and $A, c \models G'$.
- $A, c \models \neg G$ if and only if $A, c \not\models G$.
- $A, c \models NG$ if and only if $c = t \cdot c'$ and $A, c' \models G$.
- $A, c \models GUG'$ if and only if
  - $A, c \models G'$ or
\[ c = t_1 t_2 \cdots t_n \cdot c', \text{ with } A, c' \models G' \text{ and} \]

\[ \forall i \in \{1, \ldots, n\}, A, t_i \cdots t_n \cdot c' \models G, \]

with \( c' \) possibly empty.

- \( A, c \models DF \) if and only if the source \( s \) of \( c \) is such that \( A, s \models F \).

It is clear that this logic is an extension of linear temporal logic, in the sense that all the formulas of linear temporal logic are path formulas of CTL*. CTL* is also an extension of CTL in the following sense: for every formula \( F \) of CTL, there exists a CTL* state formula \( F' \) such that \( A, s \models F \) if and only if \( A, s \models F' \). This formula \( F' \) is easily defined by induction over the construction of CTL formulas:

- If \( F \) is an elementary proposition or a constant, then \( F' = F \).
- If \( F = F_1 \lor F_2 \) (resp. \( F_1 \land F_2, \neg F_1 \)), then \( F' = F'_1 \lor F'_2 \) (resp. \( F'_1 \land F'_2, \neg F'_1 \)).
- If \( F = \text{EX} F_1 \), then \( F' = \text{E} (\text{DF}_1) \).
- If \( F = \text{AX} F_1 \), then \( F' = \neg \text{E}(\text{N} (\text{D}(\neg F_1))) \).
- If \( F = \text{E} [F_1 \cup F_2] \), then \( F' = \text{E}(\text{DF}_1 \cup \text{DF}_2) \).
- If \( F = \text{A} [F_1 \cup F_2] \), then \( F' = \neg \text{E} (\text{DF}_1 \cup \text{DF}_2 \lor (F'_1 \land \neg \text{E} (\text{N} (\text{D}(\neg F_1))) \)).

These translations are justified on page 77. The intuitive meaning can, however, be given here. A state satisfies \( \text{END}(F') \) if and only if it is the source of a maximal path \( s \cdot c \) such that the source of \( c \) satisfies \( F' \), if and only if it has a successor that satisfies \( F' \); hence the translation of \( \text{EX} F \). The translation of \( \text{AX} F \) is in fact the translation of \( \neg \text{EX} \neg F \). The translation of \( \text{E} [F_1 \cup F_2] \) is natural. As for the translation of \( \text{A} [F_1 \cup F_2] \), it must take into account the existence of finite maximal paths and the ** condition: it was already seen that \( \text{N}1 \) characterizes the non-empty paths, hence \( \text{EN}1 \) characterizes the states that are the source of a non-empty path and \( \neg \text{EN}1 \) characterizes the states with no successors. The formula \( (\text{DF}_1 \cup \text{DF}_2 \lor (F'_1 \land \neg \text{E} (\text{N} (\text{D}(\neg F_1)))) \) therefore characterizes the paths appearing in the definition of \( A, s \models \text{A} [F_1 \cup F_2] \), given on page 57.

In addition to its ability to express properties of processes, CTL* has the remarkable property of expressing the same properties as a well-defined subset of another type of logic, second-order monadic logic of trees [50, 85], which will not be elaborated upon here.

### 4.8 Fairness conditions

Logics for transition systems must also be able to express fairness properties (or conditions) about infinite paths.

As a simple example of fairness condition, consider two lanes of cars that merge into a single lane as the road narrows. It is supposed that the car that enters this single lane is the lead car of one of the two lanes. If only the cars from one of the two lanes enter the single lane, the behavior is clearly not fair. For the behavior to
be fair, each lead car must enter the single lane sooner or later, even if the number of cars entering before it from the other lane is arbitrarily large.

A standard computer science example of fairness condition is that of access to a critical section. Consider the variant of Peterson’s algorithm given in section 2.2.6 (page 17). Consider an infinite execution in which a process executes the transition

\[ t : \ 4 \rightarrow (d1 = \text{false}\? \text{or } t = 1?) \equiv \text{false} \rightarrow 4 \]

indefinitely often and only executes the transition

\[ t' : \ 4 \rightarrow (d1 = \text{false}\? \text{or } t = 1?) \equiv \text{true} \rightarrow 5 \]

finitely many times. From some point on, the second process repeatedly tries to enter its critical section, without ever succeeding. This execution is therefore not fair.

Note that the two examples are not analogous. The lead car of a lane remains there until it enters the single lane. However, a process that attempts to enter a critical section does not permanently execute transition

\[ t : \ 4 \rightarrow (d1 = \text{false}\? \text{or } t = 1?) \equiv \text{false} \rightarrow 4, \]

as it can also execute transition

\[ 4 \rightarrow e \rightarrow 4. \]

There are therefore two intuitive concepts of fairness in a transition system, easier to present in terms of what is unfair.

- An infinite path is not weakly fair if something that can always occur after a particular instant no longer occurs.
- An infinite path is not strongly fair if something that can occur infinitely often after a particular instant no longer occurs.

Equivalent positive definitions for fairness are:

- An infinite path is weakly fair if, whenever something can always occur from some point on, it occurs infinitely often.
- An infinite path is strongly fair if, whenever something can occur infinitely often, it occurs infinitely often.

Naturally the expressions ‘something can occur’ and ‘something will occur’ must be formalized. This can be, for example, ‘to be in the source state of a transition \( t \)’ and ‘execute transition \( t \)’. Other examples of those ‘somethings’ are presented in references [76] and [78].

It is supposed here that there exists a property \( P \) characterizing the paths in which ‘something’ can occur and a property \( R \) characterizing the paths in which ‘something’ does occur. Using the \( \Diamond \) and \( \Box \) operators of linear temporal logic (page 50), then \( \Diamond \Box P \) expresses that ‘something’ can always occur starting from
a certain point, □◊P expresses that ‘something’ can occur infinitely often, and □◊R expresses that ‘something’ does occur infinitely often. Weak fairness therefore becomes ¬◊□P ∨ ◊□◊R and strong fairness becomes ¬□◊P ∨ □◊◊R.

Note that ¬◊□P is equivalent to □◊¬P. An elementary fairness condition is any formula of linear temporal logic of the form □◊P, where P is a boolean combination of elementary propositions of that logic, and a fairness condition is any boolean combination of elementary fairness conditions.

Example
Let A = ⟨S, T, α, β, T_Y, T_{Y'}⟩ be a parameterized transition system where T_Y = {t} and T_{Y'} = {t' | α(t) = α(t')}}. An infinite path satisfies the formula □◊P_Y if it contains transition t an infinite number of times; it satisfies □◊P_{Y'} if it goes through t’s source state an infinite number of times. The fairness condition is, ‘a path cannot pass infinitely often through t’s source state and execute t a finite number of times’. The condition is expressed by the formula

¬(□◊P_{Y'} ∧ ¬□◊P_Y),

which is equivalent to

¬□◊P_{Y'} ∨ □◊P_Y.

Note that every finite path satisfies this formula: if a path is finite, it cannot satisfy □◊P_{Y'}.

Unlike other path properties, fairness conditions consider only the infinite behavior of a path; they are independent of its finite prefixes, as is shown by the following proposition:

Proposition 4.1 Let A be a transition system and Φ be a fairness condition. If t · c is a path of A then A, t · c ⊨ Φ if and only if A, c ⊨ Φ.

Proof Any boolean combination of formulas Φ such that

A, t · c ⊨ Φ ⇔ A, c ⊨ Φ

also has that property. It suffices to prove the property for elementary conditions, and this follows immediately from the definition of elementary conditions.

When a transition system’s behavior is examined, one can assume that it follows certain fairness conditions that cannot be modeled in the transition system itself, as for the preceding example. When the definition of the satisfiability relation ⊨ requires paths, as is the case for linear temporal logic, CTL and CTL*, only those paths respecting the fairness conditions are considered.

This poses no problem for linear temporal logic and for CTL*, since they contain path formulas that can express fairness conditions. Hence if Φ is a fairness condition formula, the CTL* formula EG characterizing the sources of maximal paths satisfying G can be rewritten E(G ∧ Φ).
For CTL, which has no path formulas, the definitions of the $\models$ relation for $E[FUF']$ and $A[FUF']$ must be modified as follows:

- $A, s \models E[FUF']$ if and only if there exists a maximal path $c$ of source $s$, $c = t_1 \cdots t_n \cdots$, such that $A, c \models \Phi$ and either
  - $A, s \models F'$ or
  - there exists an integer $k$ such that
    * $A, \beta(t_k) \models F'$,
    * $\forall i, 1 \leq i \leq k$, $A, \alpha(t_i) \models F$.

- $A, s \models A[FUF']$ if and only if for every maximal path $c = t_1 \cdots t_n \cdots$ of source $s$ such that $A, c \models \Phi$ and either
  - $A, s \models F'$,
  - there exists an integer $k$ such that
    * $A, \beta(t_k) \models F'$,
    * $\forall i, 1 \leq i \leq k$, $A, \alpha(t_i) \models F$, or
  ** $c$ is a finite path and for every $i$, $A, \alpha(t_i) \models F$ and $A, \beta(t_i) \models F$.

An extension of CTL, called CFL$^F$, where fairness conditions are of the form $\bigwedge_{i=1}^{n} \square \Diamond P'_i$ and where a path satisfies $P'_i$ if its source state satisfies the elementary proposition $P_i$, was defined by Clarke et al. in [24]. Since the fairness condition $\Phi$ can be considered as a third parameter of $E[FUF']$ and $A[FUF']$, Emerson and Lei [39] proposed an extension of CTL, called FCTL, with, for each fairness condition $\Phi$, the operators $E_{\Phi}[FUF']$ and $A_{\Phi}[FUF']$. The interpretations of these operators are given above; the fairness conditions are boolean combinations of CTL$^F$'s fairness conditions. Queille and Sifakis’s logic [78] is similar to FCTL, with CTL$^F$'s fairness conditions.
In the preceding chapter several logics were presented in the following form:

- A set of formulas is built up inductively from atomic formulas (constants and elementary propositions) and operators ($\lor$, $\land$, $\neg$, $U$, $N$, $\langle a \rangle$, $\text{src}$, $\text{tgt}$, $\text{AX}$, $\text{EX}$, $A[U.]$, $E[U.]$, …). This set of formulas can sometimes be partitioned into formulas of particular types, such as states, transitions or paths.
- Given a transition system $A$, a formula $F$ applicable to $A$ and an object $x$ of the same type, a satisfiability relation $A, x \models F$ is defined by induction over the construction of $F$.

Once a logic is defined, it can be studied as a ‘logic’, i.e. by defining the concepts of valid formula and theorem, by introducing axioms and deduction rules and by comparing the two concepts. References [33, 41] summarize these concepts from the branching-time-temporal-logic point of view. A logic is also a ‘language’ expressing properties: this aspect is developed here, so these logics are presented in a more ‘algebraic’ manner.

5.1 Transition system logics as logics

Consider a logic defined by its formulas and its satisfiability relation. To simplify the presentation, it is assumed that all the formulas are of the same type.

5.1.1 Valid formulas

Let $A$ be a transition system. A formula $F$ is valid in $A$, written $A \models F$, if every state of $A$ satisfies $F$, i.e. $\forall s \in S, A, s \models F$. A formula $F$ is universally valid, written $\models F$, if it is valid in every transition system.

An often asked and much studied question is whether a given formula is universally valid. This problem is decidable if an algorithm can give the answer or,
equivalently, if the set of universally valid formulas is recursive. From a practical point of view, much work has been done to determine the complexity of decision procedures.

One can also define for this logic a deductive system composed of axioms and deduction rules. A theorem is a formula which can be deduced from the axioms using the deduction rules. One can then ask if the deductive system is sound (every theorem is a universally valid formula) and if it is complete (every universally valid formula is a theorem).

If $\mathcal{C}$ is a class of transition systems, a formula $F$ is $\mathcal{C}$-valid, written $\mathcal{C} \models F$, if it is valid in every transition system in $\mathcal{C}$. An example for $\mathcal{C}$ is the class of all finite transition systems.

Class $\mathcal{C}$ is complete if every $\mathcal{C}$-valid formula is universally valid. A particularly interesting case is that of logics for which the class of finite transition systems is complete.

5.1.2 Expressivity of a logic

Two formulas $F$ and $F'$ are equivalent if

$$\forall A, \forall s, A, s \models F \iff A, s \models F'.$$

Should formulas $F$ and $F'$ be written in a logic containing disjunction, conjunction and negation, then $F \equiv F'$ can be defined as the formula $(F \land F') \lor (\neg F \land \neg F')$. Then $F$ and $F'$ are equivalent if and only if $F \equiv F'$ is a universally valid formula.

This concept of equivalence is interesting because it need not be applied to two formulas in the same logic. It can also be applied to two formulas in two different logics, under the sole condition that the two logics are applicable to the same transition systems and that the two formulas are of the same type. Consider then two formulas $F_1$ and $F_2$ of the same type (e.g. state), respectively written in two logics whose satisfiability relations are $\models_1$ and $\models_2$. These formulas are equivalent if

$$\forall A, \forall s, A, s \models_1 F_1 \iff A, s \models_2 F_2.$$

A logic $\mathcal{L}_1$ is uniformly more expressive than a logic $\mathcal{L}_2$ if for every formula of $\mathcal{L}_2$ there exists an equivalent formula of $\mathcal{L}_1$.

These concepts can also be defined relative to a class $\mathcal{C}$ of transition systems. Two formulas $F_1$ and $F_2$ are $\mathcal{C}$-equivalent if

$$\forall A \in \mathcal{C}, \forall s, A, s \models_1 F_1 \iff A, s \models_2 F_2.$$

A logic $\mathcal{L}_1$ is uniformly more expressive, relative to $\mathcal{C}$, than a logic $\mathcal{L}_2$ if for every formula of $\mathcal{L}_2$, there exists a formula of $\mathcal{L}_1$ that is $\mathcal{C}$-equivalent to it.
Finally, the concept of uniform expressivity can be weakened: a logic \( \mathcal{L}_1 \) is more expressive, relative to \( \mathcal{L}_2 \), than a logic \( \mathcal{L}_2 \) if for every formula \( F \) in \( \mathcal{L}_2 \) and for every transition system \( \mathcal{A} \) in \( \mathcal{L}_2 \), there exists a formula \( F_\mathcal{A} \) in \( \mathcal{L}_1 \) such that
\[
\forall s, \mathcal{A}, s \models_1 F_\mathcal{A} \iff \mathcal{A}, s \models_2 F.
\]

5.1.3 Example

Consider, for a given \( n \) \((n \geq 1)\), the labeled transition system \( \mathcal{A}_n \) whose transitions are
\[
egin{align*}
1 & \rightarrow a \rightarrow 2, \\
2 & \rightarrow a \rightarrow 3, \\
& \vdots \\
n - 1 & \rightarrow a \rightarrow n, \\
n & \rightarrow b \rightarrow n + 1,
\end{align*}
\]
and the Hennessy–Milner formula
\[
F_n = \langle b \rangle 1 \lor \langle a \rangle \langle b \rangle 1 \lor \langle a \rangle \langle a \rangle \langle b \rangle 1 \lor \cdots \lor \langle a \rangle^m \langle b \rangle 1.
\]
The set of states \( s \) of \( \mathcal{A}_n \) that satisfy \( F_m \) is equal to
\[
\begin{align*}
\{1, 2, \ldots, n\} & \quad \text{if } n \leq m, \\
\{n - m, n - m + 1, \ldots, n\} & \quad \text{if } n > m.
\end{align*}
\]
Consider the state formula
\[
F_0 = \langle a \rangle 1 \mathbf{U} \langle b \rangle 1
\]
of an arbitrary logic whose satisfiability relation is defined by:
- \( \mathcal{A}, s \models \langle a \rangle 1 \) if there exists a transition of source \( s \) labeled \( a \).
- \( \mathcal{A}, s \models \langle b \rangle 1 \) if there exists a transition of source \( s \) labeled \( b \).
- \( \mathcal{A}, s \models F \lor F' \) if \( \mathcal{A}, s \models F \) or there exists a non-empty finite path \( t_1 \cdot t_2 \cdots t_k \) of source \( s \) such that
  \[- \forall i \in \{1, \ldots, k\}, \mathcal{A}, \alpha(t_i) \models F \text{ and} \]
  \[- \mathcal{A}, \beta(t_k) \models F'. \]

It is clear that for every \( n \) the set of states of \( \mathcal{A}_n \) that satisfy \( F_0 \) is \( \{1, 2, \ldots, n\} \).
It follows that for every \( n \) and for every state \( s \) of \( \mathcal{A}_n \),
\[
\mathcal{A}_n, s \models F_n \iff \mathcal{A}_n, s \models F_0,
\]
hence, over the class \( \{ \mathcal{A}_n \mid n \geq 1 \} \), logic \( \mathcal{L}' \) composed of formulas \( F_n \) is more expressive than logic \( \mathcal{L} \) composed of the sole formula \( F_0 \). But neither of these two logics is uniformly more expressive than the other over \( \{ \mathcal{A}_n \mid n \geq 1 \} \). In fact no formula \( F_n \) is \( \{ \mathcal{A}_n \mid n \geq 1 \} \)-equivalent to \( F_0 \), since \( \mathcal{A}_{n+1}, 1 \models F_0 \) and \( \mathcal{A}_{n+1}, 1 \not\models F_n \).
5.2 Transition system logics as algebras

If a process (or a system of processes) is modeled by a transition system $A$, and a property is expressed by a formula $F$ (in a particular logic), a pragmatic viewpoint might only require the computation of the set $\{x \mid A, x \models F\}$ of objects in $A$ of $F$'s type that satisfy $F$. This process is called the verification of property $F$.

Write $F_A$ for the set $\{x \mid A, x \models F\}$. Since the relation $A, x \models F$ is defined by induction over the construction of $F$, the set $F_A$ is also defined by induction over the construction of $F$. For example one can write

$$(F \lor F')_A = F_A \cup F'_A,$$

and for the Hennessy–Milner $(a)$ operator,

$$(\langle a \rangle F)_A = \{s \mid \exists t : \alpha(t) = s, \lambda(t) = a, \beta(t) \in F_A\}.$$ 

If, for a labeled transition system $A = \langle S, T, \alpha, \beta, \lambda \rangle$, for each $a$ of alphabet $A$, the mapping

$$\langle a \rangle_A : \wp(S) \to \wp(S)$$

is defined by

$$\langle a \rangle_A(X) = \{\alpha(t) \mid t \in T, \lambda(t) = a, \beta(t) \in X\},$$

the result is then

$$(\langle a \rangle F)_A = \langle a \rangle_A(F_A).$$

These equalities suggest that for every transition system $A$, the set $F_A$ is the homomorphic image of the formula $F$ considered as a set of a free algebra. This approach is developed below.

5.2.1 Subset algebras

Let $\Omega$ be a set of operators. Each operator $\omega$ of $\Omega$ has an arity, written $\delta(\omega)$, indicating the number of arguments of $\omega$, and a type of the form

$$\rho_1 \cdots \rho_{\delta(\omega)} \to \rho,$$

where $\rho_i$ and $\rho$ are symbols in a set $R$, the set of base types.

A subset $\Omega$-algebra $A$ is composed

- of a set $A_\rho$ for each base type $\rho$ in $R$; if $\rho \neq \rho'$ then
  $$A_\rho \cap A_{\rho'} = \emptyset;$$
- of a mapping $\omega_A : \wp(A_{\rho_1}) \times \cdots \times \wp(A_{\rho_n}) \to \wp(A_\rho)$, for each operator $\omega$ in $\Omega$, where $n$ is the arity of $\omega$ and $\rho_1 \cdots \rho_n \to \rho$ is its type.
Examples
1. The Hennessy–Milner algebras contain one base type. Their operators are

- \(0\) and \(1\), of arity 0,
- \(\lor\) and \(\land\), of arity 2,
- \(\neg\) of arity 1, and
- \(\langle a \rangle\), of arity 1, for every letter \(a\) in the alphabet \(A\).

For each transition system \(A = \langle S, T, \alpha, \beta, \lambda \rangle\) labeled by an alphabet \(A\), define the subset algebra, also written \(A\), by:

- The set associated with the sole base type is the set \(\wp(S)\) of the subsets of the set of states.
- The mappings associated with the operators are defined by:

\[
\begin{align*}
0_A & = \emptyset, \\
1_A & = S, \\
\lor_A(X, X') & = X \cup X', \\
\land_A(X, X') & = X \cap X', \\
\neg_A(X) & = S - X, \\
\langle a \rangle_A(X) & = \{ a(t) \mid t \in T, \lambda(t) = a, \beta(t) \in X \}.
\end{align*}
\]

Note that the \(\langle a \rangle_A\) mapping is additive:

\[
\langle a \rangle_A(X \cup X') = \langle a \rangle_A(X) \cup \langle a \rangle_A(X').
\]

2. The Dicky algebras have two base types, written \(\sigma\) and \(\tau\), corresponding to the states and transitions of a transition system. They contain the following operators, for a set \(\mathcal{X}\) of state parameter names and a set \(\mathcal{Y}\) of transition parameter names:

- \(0_\sigma, 1_\sigma, 0_\tau, 1_\tau\): 0-ary (constant) operators of types \(\sigma\) and \(\tau\);
- for every state parameter name \(X \in \mathcal{X}\), \(P_X\), 0-ary operator of type \(\sigma\);
- for every transition parameter name \(Y \in \mathcal{Y}\), \(Q_Y\), 0-ary operator of type \(\tau\);
- \(\lor_\sigma, \land_\sigma, -_\sigma\): binary operators of type \(\sigma \sigma \to \sigma\);
- \(\lor_\tau, \land_\tau, -_\tau\): binary operators of type \(\tau \tau \to \tau\);
- \(\text{src, tgt}\): unary operators of type \(\tau \to \sigma\);
- \(\text{in, out}\): unary operators of type \(\sigma \to \tau\).

If \(A = \langle S, T, \alpha, \beta, S_{X_1}, \ldots, S_{X_n}, T_{Y_1}, \ldots, T_{Y_m} \rangle\) is a transition system parameterized by \((\mathcal{X}, \mathcal{Y})\), the subset algebra, also written \(A\), is constructed as follows:

- \(A_\sigma = \wp(S), A_\tau = \wp(T)\),
The mappings associated with the operators are defined by:

\( (0_{\alpha})_A = \emptyset, \)
\( (1_{\sigma})_A = S, \)
\( (0_{\tau})_A = \emptyset, \)
\( (1_{\tau})_A = T, \)
\( (P_X)_A = S_X, \)
\( (Q_Y)_A = T_Y, \)
\( (\forall_{\sigma})_A(E, E') = E \cup E', \)
\( (\wedge_{\sigma})_A(E, E') = E \cap E', \)
\( (\neg_{\sigma})_A(E, E') = E - E', \)
\( (\forall_{\tau})_A(R, R') = R \cup R', \)
\( (\wedge_{\tau})_A(R, R') = R \cap R', \)
\( (\neg_{\tau})_A(R, R') = R - R', \)
\( \text{src}_A(R) = \{\alpha(t) \mid t \in R\}, \)
\( \text{tgt}_A(R) = \{\beta(t) \mid t \in R\}, \)
\( \text{out}_A(E) = \{t \mid \alpha(t) \in E\} = \alpha^{-1}(E), \)
\( \text{in}_A(E) = \{t \mid \beta(t) \in E\} = \beta^{-1}(E). \)

Note that the four mappings src\(_A\), tgt\(_A\), in\(_A\) and out\(_A\) are all additive.

3. Linear temporal algebras contain a single type. Their operators are:

- a number of constants \( P_1, \ldots, P_n, \)
- the constants 0 and 1,
- the binary operators \( \lor, \land \) and \( \cup, \)
- the unary operators \( \neg \) and \( \mathbb{N}. \)

Let \( A = (S, T, \alpha, \beta) \) be a transition system containing \( n \) transition parameters \( T_1, \ldots, T_n \). Define the subset algebra whose set associated with the sole type is the set \( C \) of all paths, finite or infinite. The mappings associated with the operators are:

\( (P_i)_A = \{t \cdot c \mid t \in T_i\} = T_i \cdot C, \)
\( 0_A = \emptyset, \)
\( 1_A = C, \)
\( \lor_A(L, L') = L \cup L', \)
\( \land_A(L, L') = L \cap L', \)
\( \neg_A(L) = C - L, \)
\[ N_A(L) = \{ t \cdot c \mid c \in L \} = T \cdot L, \]
\[ U_A(L, L') = L' \cup \{ t_1 \cdot \ldots \cdot t_n \cdot c \mid c \in L', \forall i \in \{1, \ldots, n\}, t_i \cdot \ldots \cdot t_n \cdot c \in L \}. \]

It is clear that for each transition system of the appropriate type, subset algebras corresponding to the other previously defined logics can also be given.

5.2.2 Terms and their interpretation

Let \( \Omega \) be a set of typed operators. Let \( \vec{x} = \langle x_1, \ldots, x_n \rangle \) be a vector of variables where each variable \( x_i \) has a base type, written \( \rho(x_i) \). Define the set \( T_\rho(\Omega, \vec{x}) \) of terms of type \( \rho \) constructed over \( \Omega \) and over the typed variables \( \{ x_1, \ldots, x_n \} \) by:
- \( x_i \in T_{\rho(x_i)}(\Omega, \vec{x}) \), for every \( i \in \{1, \ldots, n\} \).
- If \( \omega \) is a constant of type \( \rho \), \( \omega \in T_\rho(\Omega, \vec{x}) \).
- If \( \omega \) is a \( k \)-ary operator of type \( \rho_1 \cdot \ldots \cdot \rho_k \to \rho \) and if \( t_i \in T_{\rho_i}(\Omega, \vec{x}) \), \( i \in \{1, \ldots, n\} \), then \( \omega(t_1, \ldots, t_k) \in T_\rho(\Omega, \vec{x}) \).

If \( \vec{x} \) is the empty sequence, the set \( T_\rho(\Omega, \vec{x}) \) is written \( T_\rho(\Omega) \) and its elements are the closed terms of type \( \rho \).

A logic's formulas are the closed terms built up from that logic's operators.

If \( A = \langle (A_\rho)_{\rho \in \mathcal{R}}, (\omega_A)_{\omega \in \Omega} \rangle \) is a subset \( \Omega \)-algebra, for each term \( t \) of \( T_\rho(\Omega, \vec{x}) \) define the mapping
\[ t_A : \varphi(A_{\rho(x_1)}) \times \cdots \times \varphi(A_{\rho(x_n)}) \to \varphi(A_\rho) \]
inductively by
- If \( t = x_i \), then \( t_A(X_1, \ldots, X_n) = X_i \).
- If \( t \) is equal to a constant \( \omega \), then \( t_A(X_1, \ldots, X_n) = \omega_A \).
- If \( t = \omega(t_1, \ldots, t_k) \), then
\[ t_A(X_1, \ldots, X_n) = \omega_A\left((t_1)_A(X_1, \ldots, X_n), \ldots, (t_k)_A(X_1, \ldots, X_n)\right). \]

This mapping \( t_A \) is the interpretation of \( t \) in \( A \). In particular, if \( t \) is a closed term of type \( \rho \), its interpretation \( t_A \) in \( A \) is a subset of \( A_\rho \).

Given the identification of a logical formula \( F \) with a closed \( \Omega \)-term \( t_F \), it can be seen that if a transition system \( A \) is viewed as a subset \( \Omega \)-algebra, then \( \{ x \mid A, x \models F \} = \langle t_F \rangle_A \).

5.3 Verification

The verification problem can now be presented as follows.

Let \( A \) be a transition system and \( F \) be a logical formula viewed as a closed \( \Omega \)-term \( t \). The interpretation \( t_A \) of that term must be computed in the subset \( \Omega \)-algebra associated with the transition system. Given the inductive definition of \( t_A \), it therefore suffices to know how to compute \( \omega_A(X_1, \ldots, X_n) \) for each operator \( \omega \).
Result := \emptyset;
for every transition \( t \) in \( \mathcal{A} \) do
  if \( \lambda(t) = \alpha \) and \( \beta(t) \in X \)
  then Result := Result \cup \{ \alpha(t) \};

Figure 5.1 Algorithm compute\(_{\omega}\)(\( \mathcal{A}, X \)).

Result := \emptyset;
for every state \( s \) in \( \mathcal{A} \) do \( s\text{.visited} := \text{false} \);
for every state \( s \) in \( \mathcal{A} \) do
  if not \( s\\text{.visited} \) then au\(_s\);

Figure 5.2 Function \( \text{AU}_\mathcal{A}(X, X') \).

5.3.1 Verification principle

More precisely and more algorithmically, suppose that there is a data structure representing the transition system \( \mathcal{A} \) and that there are data structures representing the sets \( X_1, \ldots, X_n \). It suffices then to have, for each operator \( \omega \), an algorithm compute\(_{\omega}\)(\( \mathcal{A}, X_1, \ldots, X_n \)) which computes the set \( \omega_\mathcal{A}(X_1, \ldots, X_n) \). If, as it is almost always the case in practice, only the finite transition systems are verified and if logics containing only state and transition formulas are used (the sets of states and transitions are therefore finite, but the sets of paths can be infinite), then it is easy to find data structures implementing these objects and most algorithms computing \( \omega_\mathcal{A} \) become relatively simple, although their efficiency depends on the data structures used. For example, the computation of the interpretations of \( \lor \) and \( \land \) reduces to computing unions and intersections of finite sets, which does not pose any difficulty. The interpretation of the Hennessy–Milner (\( \omega \)) operator can be computed by the algorithm compute\(_{\omega}\)(\( \mathcal{A}, X \)) in Figure 5.1.

If logics containing path formulas are to be used, the data structure problem becomes more difficult, since, as was seen before, these sets can easily be infinite. However, as they are always regular sets, in the formal language sense, they can be finitely represented. Furthermore, when these path formulas only appear as fairness conditions, it is possible to treat them simply by computing the strongly connected components of a transition system, using Tarjan’s well-known algorithm [84]. This question is further studied in section 6.5, page 108.

5.3.2 Example

A less elementary example is the computation of the function \( \text{AU}_\mathcal{A}(X, X') \) of two arguments, shown in Figure 5.2, interpreting the \( \text{A}[\text{U}.] \) operator of CTL.

The recursive procedure au\(_s\)(\( s \)) is shown in Figure 5.3 (page 71). It is shown below that \( \text{AU}_\mathcal{A} \) terminates, and that when it does, the value of Result is precisely
\[ Z = \text{AU}_A(X, X'). \]

Since the execution of \( \text{au}(s) \) is always controlled by the test if not \( s.\text{visited} \), and that execution sets \( s.\text{visited} \) to true, the procedure \( \text{au} \) is executed once and only once for each state \( s \) of \( A \). The states of \( A \) can therefore be numbered in the order in which the executions of \( \text{au}(s) \) terminate.

Let \( R_n \) be the set of states found in \( \text{Result} \) at the end of the \( n \)-th execution of \( \text{au} \), i.e. at the end of the execution of \( \text{au}(s_n) \). It can be shown by induction over \( n \) that for every integer \( n \) less than or equal to the number of states of \( A \), \( R_n = Z \cap \{ s_1, \ldots, s_n \} \), hence, at the end of the algorithm, \( \text{Result} = Z \).

Initially, \( R_0 = Z \cap \emptyset = \emptyset \). Suppose that for every \( i \leq n \), \( R_i = Z \cap \{ s_1, \ldots, s_i \} \) and start the execution of \( \text{au}(s_{n+1}) \). Let \( s_{i_1}, \ldots, s_{i_p} \) be the sequence of states for which the execution of \( \text{au} \) has started without terminating, in the order in which this execution has started (with the exception, of course, of \( s_{n+1} \), also written \( s_{i_{p+1}} \)). Because of the way that the \( \text{au}(s) \) call each other mutually, \( s_{i_{m+1}} \) is a successor of \( s_{i_m} \) for \( 1 \leq m \leq p \). Furthermore, \( i_m \geq n + 1 \) for every \( m \) contained between 1 and \( p \). Finally, all the \( s_{i_m} \) are in \( X - X' \) since, otherwise, the execution of \( \text{au}(s_{i_m}) \) would have necessarily terminated before another execution of \( \text{au} \) had begun.

1. If \( s_{n+1} \) is in \( X' \), then \( s_{n+1} \) is in \( Z \) and \( \text{au} \) puts \( s_{n+1} \) in \( \text{Result} \). If \( s_{n+1} \) is in neither \( X \) nor \( X' \), this state is not in \( Z \) and \( \text{au} \) does not place it in \( \text{Result} \).

2. Consider the case where \( s_{n+1} \) belongs to \( X - X' \). Let \( s_{j_1}, \ldots, s_{j_k} \) be the sequence of successors of \( s_{n+1} \) in the order in which they are examined during the execution of \( \text{au}(s_{n+1}) \).

2.1. If the boolean variable \( b \) is true after having examined all the successors of \( s_{n+1} \), it is because none of its successors appears in the set \( \{ s_{i_1}, \ldots, s_{i_p}, s_{i_{p+1}} \} \), since for such a state \( s' \), \( s'.\text{visited} \) would have been found to be equal to true and so \( \text{au}(s') \)
would not have been executed, and since \( au(s') \) has begun without terminating, 
\( s' \) is not in Result. There therefore remain two possible cases when a successor \( s_{jm} \) of \( s_{n+1} \) is examined:

- either \( s_{jm} \).visited is equal to true, hence \( au(s_{jm}) \) must have terminated, from which \( j_m \leq n \);
- or else \( s_{jm} \).visited is equal to false; then \( au(s_{jm}) \) will be executed and will terminate, and there also \( j_m \leq n \).

In the two cases, \( s_{jm} \in R_{j_m} \), hence, by the inductive hypothesis, \( s_{jm} \in Z \). All of the maximal paths from the \( s_{jm} \) satisfy the desired properties, given on page 57, and hence the same holds for all the maximal paths from \( s_{n+1} \). Hence \( s_{n+1} \in Z \) and \( R_{n+1} = R_n \cup \{ s_{n+1} \} = Z \cap \{ s_1, \ldots, s_n \} \cup \{ s_{n+1} \} = Z \cap \{ s_1, \ldots, s_{n+1} \} \).

2.2. If the boolean variable \( b \) is false, then one of the successors \( s_{jm} \) of \( s_{n+1} \) was not in Result when \( b := b \land (s_{jm} \in \text{Result}) \) was executed:

- If \( j_m \leq n \) then, since \( R_{j_m} = Z \cap \{ s_1, \ldots, s_{j_m} \}, s_{jm} \) was not in \( Z \). There therefore exists a maximal path from \( s_{jm} \) which does not satisfy the desired properties, and there therefore also exists one from \( s_{n+1} \), hence \( s_{n+1} \notin Z \).
- Otherwise \( s_{jm} \) belongs to the set \( \{ s_{i_1}, \ldots, s_{i_p}, s_{i_{p+1}} \} \) of visited states whose execution of \( au \) was not yet terminated. Since \( s_{jm} = s_{i_q} \) is a successor of \( s_{n+1} \), there exists an infinite path

\[
s_{n+1} = s_{i_{p+1}}, s_{i_q}, s_{i_{q+1}}, \ldots, s_{i_{p+1}}, s_{i_q}, s_{i_{q+1}}, \ldots, s_{i_{p+1}}, \ldots
\]

all of whose states are in \( X - X' \). Since this path does not satisfy the desired properties, \( s_{n+1} \notin Z \).

In the two cases, \( R_{n+1} = R_n = Z \cap \{ s_1, \ldots, s_n \} = Z \cap \{ s_1, \ldots, s_{n+1} \} \).
The interpretations of operators in classical temporal logics can, as was noted by Emerson and Clarke [34] and Sifakis [81], be defined as the least solutions of systems of equations. This chapter adds the fixpoint operator to transition system logics to increase their expressivity. A few examples are presented before the precise definition is given.

6.1 Examples

6.1.1 The ‘Until’ operator

Let $A$ be a transition system, containing transition parameters, that is to be considered as a linear temporal logic (see page 68). Let $L$ and $L'$ be two sets of paths. By definition,

$$LU_AL' = L' \cup \{t_1 \cdots t_n \cdot c \mid c \in L', \forall i \in \{1, \ldots, n\}, t_i \cdots t_n \cdot c \in L\}.$$  

Let $L'' = LU_AL'$. The equality

$$L'' = L' \cup (N_A(L'') \cap L)$$

is proven as follows:

1. Show that $L' \cup (N_A(L'') \cap L) \subseteq L''$.

1.1. If $c$ is in $L'$ then it is in $L''$, by the definition of $L''$.

1.2. If $c$ is in $N_A(L'') \cap L$, then $c = t \cdot c'$, with $c \in L$ and, by the definition of $N_A(L'')$, $c' \in L''$. Since $c' \in L''$, either $c' \in L'$ or $c' = t_1 \cdots t_n \cdot c''$, with $c'' \in L'$ and $t_1 \cdots t_n \cdot c'' \in L$. Since $t \cdot c' \in L$, in the two cases, $c \in L''$. 

2. Show that \( L'' \subseteq L' \cup (N_A(L'') \cap L) \). If \( c \in L'' \), then

2.1. either \( c \in L' \) and so \( c \in L' \cup (N_A(L'') \cap L) \),

2.2. or \( c = t_1 \cdots t_n \cdot c' \), with \( c' \in L' \) and \( t_i \cdots t_n \cdot c' \in L \). In that case \( t_2 \cdots t_n \cdot c' \) is also in \( L'' \) and so \( t_1 \cdots t_n \cdot c' \), which is in \( L \), is also in \( N_A(L'') \). \hfill \Box

Furthermore \( L'' \) is the smallest set of paths (under inclusion) that satisfies the equation

\[
X = L' \cup (N_A(X) \cap L).
\]

To show this, let \( X \) be a set of paths such that \( X = L' \cup (N_A(X) \cap L) \). Define

\[
L_0 = L', \\
L_{i+1} = L_i \cup (N_A(L_i) \cap L).
\]

It can easily be shown by induction that, for every \( i \geq 0 \), \( L_i \subseteq X \):

- \( L_0 = L' \subseteq X \).
- If \( L_i \subseteq X \), then \( N_A(L_i) \subseteq N_A(X) \) and \( N_A(L_i) \cap L \subseteq N_A(X) \cap L \), and so

\[
L_{i+1} = L_i \cup (N_A(L_i) \cap L) \subseteq X \cup (N_A(X) \cap L) = X.
\]

It follows that \( \bigcup_{i \geq 0} L_i \subseteq X \). It remains to be shown that \( L'' \subseteq \bigcup_{i \geq 0} L_i \). To do that, for every path \( c \) of \( L'' \), let \( n = n(c) \) be the smallest integer such that \( c = t_1 \cdots t_n \cdot c' \), with \( c' \in L' \) (if \( c \in L' \), then \( n = 0 \)). By the definition of \( L'' \) this integer always exists. Moreover, \( t_i \cdots t_n \cdot c' \in L \). Furthermore, if \( t \cdot c \in L'' \) and \( n(t \cdot c) > 0 \), then \( c \in L'' \) and \( n(c) = n(t \cdot c) - 1 \).

The proof is finished by showing by induction over \( n \) that if \( c \in L'' \) and \( n(c) \leq n \), then \( c \in L_n \):

- If \( n = 0 \) and \( n(c) \leq n \), then \( n(c) = 0 \) and so \( c \in L' = L_0 \subseteq L_n \).
- If \( n(c) \leq n+1 \), then either \( n(c) \leq n \) and \( c \in L_n \subseteq L_{n+1} \), or \( n(c) = n+1 \) and \( c = t \cdot c' \), with \( n(c') = n \), and so \( c' \in L_n \) and \( c = t \cdot c' \in N_A(L_n) \), and since \( c \in L \), it follows that \( c \in L_{n+1} \).

Consider now the set \( \tilde{L} \) of infinite paths \( t_1 \cdots t_n \cdots \) such that

\[
\forall i \geq 1, t_i \cdot t_{i+1} \cdots \in L.
\]

(Note this set can be empty, even if \( L \) contains infinite words; this is the case, for the transition system formed of two transitions \( t \) and \( t' \), with \( \beta(t) = \alpha(t') = \beta(t') \), of the set of paths \( L = t(t')^\omega \).) Then the set \( L'' \cup \tilde{L} \) is still a solution of the equation

\[
X = L' \cup (N_A(X) \cap L),
\]
written $Lu, L'$ in keeping with [20]. These two versions of the 'until' operator are respectively strong and weak until [10, 20, 33].

To show that $Lu, L'$ is still a solution, according to the definition of $N_A$,

$$N_A(L'' \cup \bar{L}) = N_A(L'') \cup N_A(\bar{L}),$$

and so

$$L' \cup (N_A(L'' \cup \bar{L} \cap L)) = L' \cup (N_A(L'') \cap L) \cup (N_A(\bar{L}) \cap L).$$

It therefore suffices to show that $\bar{L} = N_A(\bar{L}) \cap L$, which immediately follows from the definition of $\bar{L}$.

In fact $L'' \cup \bar{L}$ is the greatest solution of the equation $X = L' \cup (N_A(X) \cap L)$. If $X$ is an arbitrary solution, then $L'' \subseteq X$ and it must be shown that $X - L'' \subseteq \bar{L}$. Since $X = L'' \cup (X - L'')$,

$$N_A(X) \cap L = (N_A(L'') \cap L) \cup (N_A(X - L'') \cap L),$$

and so

$$X = L' \cup (N_A(X) \cap L)
= L' \cup (N_A(L'') \cap L) \cup (N_A(X - L'') \cap L)
= L'' \cup (N_A(X - L'') \cap L).$$

It follows that

$$X - L'' \subseteq N_A(X - L'') \cap L.$$ 

Hence, if $c \in X - L''$, then $c \in N_A(X - L'') \cap L$, i.e. $c = t \cdot c'$, with $c \in L$ and $c' \in X - L''$. But, for the same reasons, $c' = t' \cdot c''$, with $c' \in L$ and $c'' \in X - L''$, and so forth, which implies that $c$ is in $\bar{L}$.

### 6.1.2 The CTL operators

Let $A = (S, T, \alpha, \beta)$ be a transition system. It is easy to see that, since

$$(EXF)_A = \{ s \mid \exists t : \alpha(t) = s, \beta(t) \in F_A \} = \alpha(\beta^{-1}(F_A)),$$

$EX_A(X) = \alpha(\beta^{-1}(X))$. Since $(AXF)_A = (\neg EXF)_A$, the operator $AX_A$ is the dual of $EX_A$ and $AX_A(X) = S - \alpha(\beta^{-1}(S - X))$.

It is interesting to note here that if $s$ is a state with no successor, then

$$\forall X \subseteq S, s \in AX_A(X).$$

This is true, for if $s$ is the source of no transition,

$$\forall Y \subseteq T, s \notin \alpha(Y),$$
then
\[ s \in S - \alpha(\beta^{-1}(S - X)) \].

Let \( E \subseteq A \) be the interpretation in \( A \) of the \( E[U] \) operator. As for the 'Until' - operator, it can be shown that \( E \subseteq A(P, P') \) is the least solution of the equation
\[ X = P' \cup (P \cap E \subseteq A(X)) \].

Similarly, it can be shown that \( A \subseteq A(P, P') \) is the least solution of equation
\[ X = P' \cup (P \cap A \subseteq A(X)) \].

If in the definition of \( A \), \( s \models A[FU F'] \) given on page 57, the condition marked with a double asterisk did not appear, then \( A \subseteq A(P, P') \) would not have exactly the same definition and would be the least solution of equation
\[ X = P' \cup (P \cap A \subseteq A(X) \cap E \subseteq A(X)) \].

The difference between the two definitions appears in the case where a state has no successor: this state is always in \( A \subseteq A(X) \), but never in \( E \subseteq A(X) \), whatever the \( X \). If a state has no successor, it is always in \( A \subseteq A(P, P') \) if the first definition is used, and never if the second is used.

### 6.1.3 The CTL* operators

The CTL* operators include N and U, whose interpretations \( N \subseteq A \) and \( U \subseteq A \) over the set of paths of a transition system \( A \) have already been defined. If \( L \) is a set of paths in \( A \) and if \( M \) is the set of all the maximal paths, then the interpretation \( E \subseteq A \) of operator \( E \) is defined by
\[ E \subseteq A(L) = \alpha(L \cap M) \],

i.e. \( E \subseteq A(L) \) is the set of the source states of the maximal paths of \( L \). Conversely, if \( P \) is a set of states,
\[ D \subseteq A(P) = \alpha^{-1}(P) \]
is the set of paths whose sources are in \( P \). These interpretations satisfy the following properties:

**Proposition 6.1** If \( P \) and \( P' \) are sets of states and \( L \) and \( L' \) are sets of paths in a transition system \( A \), then

1. \( D \subseteq A(P \cup P') = D \subseteq A(P) \cup D \subseteq A(P') \);
2. \( D \subseteq A(P \cap P') = D \subseteq A(P) \cap D \subseteq A(P') \);
3. \( D \subseteq A(S - P) = C - D \subseteq A(P) \);
4. \( E \subseteq A(P \cup P') = E \subseteq A(P) \cup E \subseteq A(P') \);
5. \( E \subseteq A(L \cap D \subseteq A(P)) = E \subseteq A(L) \cap P \);
6. \( EX \subseteq A(E \subseteq A(L)) = E \subseteq A(N \subseteq A(L)) \).
Proof. Points (i), (ii), (iii) and (iv) follow immediately from the definitions. Point (v) follows from the fact that a state $s$ is the source of a maximal path of $L \cap D_A(P)$ if and only if it is a state of $P$ that is the source of a maximal path of $L$. Point (vi) can be shown by using the fact that $N_A(L) = T \cdot L$. The sources of the maximal paths of $T \cdot L$ are therefore the sources of the transitions whose target is the source of a maximal path of $L$, i.e. $EX_A(E_A(L))$. \hfill \Box$

It is now possible to justify the translation, given on page 59, of CTL formulas into CTL* formulas. It suffices to show that for every pair $(P, P')$ of sets of states of a transition system $A$,

$$EU_A(P, P') = E_A(D_A(P) \cup D_A(P'))$$

and

$$AU_A(P, P') = S - E_A \left( C - \left( D_A(P) \cup D_A \left( P' \cup (P - E_A(N_A(C))) \right) \right) \right),$$

where $EU_A$ and $AU_A$ were just defined as the least solutions of equations and where $C$ is the set of all paths.

First, $EU_A(P, P')$ is the least set of states that is a solution of the equation

$$X = P' \cup (EX_A(X) \cap P).$$

It is therefore the limit of the increasing sequence

$$X_0 = \emptyset,$$

$$X_{i+1} = P' \cup (EX_A(X_i) \cap P).$$

Second, $D_A(P) \cup D_A(P')$ is the least set of paths that is a solution of

$$L = D_A(P') \cup (N_A(L) \cap D_A(P)).$$

It is therefore the limit of the increasing sequence

$$L_0 = \emptyset,$$

$$L_{i+1} = D_A(P') \cup (N_A(L_i) \cap D_A(P)).$$

It therefore suffices to show, by induction, that for every $i$, $X_i = E_A(L_i)$. Sure enough,

$$X_0 = E_A(L_0) = \emptyset,$$

and, since $P' = E_A(D_A(P'))$,

$$X_{i+1} = P' \cup (EX_A(X_i) \cap P)$$

$$= E_A(D_A(P')) \cup (EX_A(E_A(L_i)) \cap P).$$
\[ E_A(D_A(P')) \cup (E_A(N_A(L_i)) \cap P) \]
\[ = E_A(D_A(P')) \cup E_A(N_A(L_i) \cap D_A(P)) \]
\[ = E_A(D_A(P')) \cup (N_A(L_i) \cap D_A(P)) \]
\[ = E_A(L_{i+1}). \]

Similarly, by replacing \( P' \) with \( P' \cup (P - E_A(N_A(C))) \),
\[ D_A(P) \cup \left( P' \cup \left( P - E_A(N_A(C)) \right) \right) \]
is the limit of the increasing sequence
\[ K_0 = \emptyset, \]
\[ K_{i+1} = D_A \left( P' \cup \left( P - E_A(N_A(C)) \right) \right) \cup (D_A(P) \cap N_A(K_i)). \]

Since \( \mathcal{A} \) is finite, the increasing sequence
\[ M_i = S - E_A(C - K_i) \]
is stationary and its limit is
\[ S - E_A \left( C - \left( D_A(P) \cup D_A \left( P' \cup \left( P - E_A(N_A(C)) \right) \right) \right) \right). \]

The properties stated in proposition 6.1 yield
\[ M_0 = \emptyset \]
\[ M_{i+1} = P' \cup \left( P - E_A(N_A(C)) \right) \cup \left( P \cap \left( S - E_A(C - N_A(K_i)) \right) \right). \]

Furthermore, \( A \cup \mathcal{A} (P, P') \) is the limit of the increasing sequence
\[ Y_0 = \emptyset, \]
\[ Y_{i+1} = P' \cup \left( P \cap AX_A(Y_i) \right) \]
\[ = P' \cup \left( P \cap (S - EX_A(S - Y_i)) \right). \]

It is shown by induction that \( M_i = Y_i \), which is trivially true for \( i = 0 \). Supposing \( M_i = Y_i \) and using the first definition of \( M_i \) yields
\[ S - Y_i = E_A(C - K_i), \]
hence,
\[ Y_{i+1} = P' \cup \left( P \cap \left( S - EX_A \left( E_A(C - K_i) \right) \right) \right) \]
\[ = P' \cup \left( P \cap \left( S - E_A(N_A(C - K_i)) \right) \right). \]
Since
\[ M_{i+1} = P' \cup \left( P - E_A(N_A(C)) \right) \cup \left( P \cap \left( S - E_A(C - N_A(K_i)) \right) \right) \]
\[ = P' \cup \left( P \cap \left( S - E_A(N_A(C)) \right) \right) \cup \left( P \cap \left( S - E_A(C - N_A(K_i)) \right) \right), \]

it suffices to show that
\[ S - E_A(N_A(C - K_i)) = \left( S - E_A(N_A(C)) \right) \cup \left( S - E_A(C - N_A(K_i)) \right), \]
or even
\[ E_A(N_A(C - K_i)) = E_A(N_A(C)) \cap E_A(C - N_A(K_i)). \]

Since \( C - K_i \subseteq C, E_A(N_A(C - K_i)) \subseteq E_A(N_A(C)) \) and it remains to be shown that if \( s \in E_A(N_A(C)) \) then
\[ s \in E_A(N_A(C - K_i)) \text{ if and only if } s \in E_A(C - N_A(K_i)). \]

But if \( s \in E_A(N_A(C)) \), it is because \( s \) is the source of a non-empty maximal path \( t \cdot c \). If \( c \in K_i \), then \( t \cdot c \not\in N_A(C - K_i) \) and \( t \cdot c \in N_A(K_i) \). If \( c \not\in K_i \), then \( t \cdot c \in N_A(C - K_i) \) and \( t \cdot c \not\in N_A(K_i) \). Hence \( t \cdot c \in N_A(C - K_i) \) if and only if \( t \cdot c \in C - N_A(K_i). \)

\[ \square \]

### 6.2 Fixpoints of monotone functions

Before introducing logics containing least and greatest fixpoint operators, some classical results concerning fixpoints, important in computer science, are presented. Instead of considering the general case of ordered sets, as in the comprehensive exposé of I. Gassarian [48], only the case of the lattice of the subsets of a set is considered.

#### 6.2.1 General theorems

Let \( E \) be an arbitrary set and let \( f \) be mapping from \( \wp(E) \) to \( \wp(E) \). The mapping \( f \) is monotone if \( \forall X, X' \subseteq E, \)
\[ X \subseteq X' \Rightarrow f(X) \subseteq f(X'). \]

The mapping \( f \) is additive if \( \forall X, X' \subseteq E, \)
\[ f(X \cup X') = f(X) \cup f(X'). \]

Every additive function is monotone, but, in general, the converse is not true.

The following theorem, often used in theoretical computer science, is known as the Knaster-Tarski theorem.
Theorem 6.1 If \( f \) is a monotone mapping from \( \wp(E) \) to \( \wp(E) \), then the equation
\[
X = f(X)
\]
has a least solution,
\[
\bigcap \{ X \mid f(X) \subseteq X \},
\]
and greatest solution,
\[
\bigcup \{ X \mid X \subseteq f(X) \}.
\]

Proof Let \( \mathcal{X} = \{ X \mid f(X) \subseteq X \} \) and \( Y = \bigcap_{X \in \mathcal{X}} X \), and show that \( Y \) is the least solution of the equation \( X = f(X) \).

Note first that \( \mathcal{X} \) is not empty: since \( f(E) \subseteq E, E \in \mathcal{X} \). Furthermore every solution of the equation \( X = f(X) \) is clearly an element of \( \mathcal{X} \), and therefore contains \( Y \). It therefore suffices to show that \( Y = f(Y) \) to obtain the result.

Let \( X \in \mathcal{X} \). By the definition of \( Y \), \( Y \subseteq X \), and since \( f \) is monotone, \( f(Y) \subseteq f(X) \). Since \( f(X) \subseteq X \) by the definition of \( \mathcal{X} \), \( f(Y) \subseteq X \) also holds, and since this is true for every \( X \in \mathcal{X} \), \( f(Y) \subseteq Y \).

Since \( f \) is monotone, it follows that \( f(f(Y)) \subseteq f(Y) \), i.e. \( f(Y) \in \mathcal{X} \), hence \( Y \subseteq f(Y) \).

The proof is similar for showing that \( \bigcup \{ X \mid X \subseteq f(X) \} \) is the greatest solution of the equation \( X = f(X) \). \( \square \)

The terms least fixpoint and greatest fixpoint of a mapping \( f \) respectively refer to the least and greatest solutions of the equation \( X = f(X) \).

The mapping \( f \) is \( \cup \)-continuous (resp. \( \cap \)-continuous) if for every increasing (resp. decreasing) sequence \( (X_i)_i \), under inclusion, finite or infinite,
\[
f \left( \bigcup_i X_i \right) = \bigcup_i f(X_i) \quad \text{(resp. } f \left( \bigcap_i X_i \right) = \bigcap_i f(X_i) \text{)}.
\]

Note that every \( \cup \)-continuous or \( \cap \)-continuous function is monotone, since
\[
X \subseteq X' \Rightarrow X \cup X' = X', \quad X \cap X' = X,
\]
hence, either \( f(X') = f(X) \cup f(X') \), or \( f(X) = f(X) \cap f(X') \), and in the two cases, \( f(X) \subseteq f(X') \).

In addition if \( f \) is monotone then
\[
\bigcup_i f(X_i) \subseteq f \left( \bigcup_i X_i \right)
\]
and
\[
f \left( \bigcap_i X_i \right) \subseteq \bigcap_i f(X_i)
\]
still hold, but equality does not necessarily.

Theorem 6.2 If \( f \) is \( \cup \)-continuous (resp. \( \cap \)-continuous), the least (resp. greatest) fixpoint of \( f \) is \( \bigcup_{i \geq 0} f^i(\emptyset) \) (resp. \( \bigcap_{i \geq 0} f^i(E) \)).
Proof The result is only proven for the case of \( \cup \)-continuity, the proof being similar in the other case.

It is first shown that the sequence \( (f^i(\emptyset))_{i \geq 0} \) is increasing. Clearly
\[
\emptyset = f^0(\emptyset) \subseteq f^1(\emptyset),
\]
and, since \( f \) is monotone,
\[
\text{if } f^i(\emptyset) \subseteq f^{i+1}(\emptyset) \text{ then } f^{i+1}(\emptyset) \subseteq f^{i+2}(\emptyset).
\]
Since \( f \) is \( \cup \)-continuous, \( f(\bigcup_{i \geq 0} f^i(\emptyset)) \) is equal to \( \bigcup_{i \geq 0} f^{i+1}(\emptyset) \), which is also equal to \( \bigcup_{i \geq 0} f^i(\emptyset) \), and is therefore a fixpoint of \( f \).

It is also the least fixpoint since if \( X \) is a fixpoint of \( f \), then \( \emptyset = f^0(\emptyset) \subseteq X \), and if \( f^i(\emptyset) \subseteq X \), then \( f^{i+1}(\emptyset) \subseteq f(X) = X \).

Finally suppose that \( E \) is finite. Then every monotone function is \( \cup \)- and \( \cap \)-continuous. This is true since if \( (X_i)_i \) is an increasing sequence, where \( n_i \) is the cardinality of \( X_i \), the sequence of integers \( (n_i)_i \) is increasing and bounded above by the cardinality of \( E \), therefore ultimately stationary: there exists an integer \( j \geq 0 \) such that \( \forall i \geq j, n_i = n_j \), i.e. \( \forall i \geq j, X_i = X_j \). Hence, \( \bigcup_{i \geq 0} X_i = X_j \) and \( f(\bigcup_{i \geq 0} X_i) = f(X_j) = \bigcup_{i \geq 0} f(X_i) \).

If \( (X_i)_i \) is a decreasing sequence, the procedure is similar, since the sequence of cardinalities is also ultimately stationary.

**Theorem 6.3** Let \( f \) be a monotone mapping from \( \wp(E) \) to \( \wp(E) \), where \( E \) is a finite set of cardinality \( n \). Then the least fixpoint of \( f \) is \( f^n(\emptyset) \) and its greatest fixpoint is \( f^n(E) \).

Proof Since \( E \) is finite the monotone mapping \( f \) is \( \cup \)- and \( \cap \)-continuous. From the preceding theorem, the least fixpoint of \( f \) is equal to \( \bigcup_{i \geq 0} f^i(\emptyset) \). Let \( n_i \) be the cardinality of \( f^i(\emptyset) \). The sequence \( (n_i)_{i \geq 0} \) is increasing and bounded by \( n \), the cardinality of \( E \). Let \( k \) be the smallest integer such that \( n_k = n_{k+1} \). Then \( 0 = n_0 < n_1 < \cdots < n_k \), and so \( n_k \geq k \) holds, as does \( k \leq n \) since \( n_k \leq n \).

Since \( n_k = n_{k+1} \), \( f^k(\emptyset) = f^{k+1}(\emptyset) \) also holds. It follows that \( \forall j \geq k, f^i(\emptyset) = f^k(\emptyset) \), since \( f^i(\emptyset) = f^{i+1}(\emptyset) \) implies
\[
f^{i+1}(\emptyset) = f(f^i(\emptyset)) = f(f^{i+1}(\emptyset)) = f^{i+2}(\emptyset).
\]
In particular, \( \bigcup_{i \geq 0} f^i(\emptyset) = f^k(\emptyset) = f^n(\emptyset) \).

The greatest fixpoint case is similar. \( \square \)

### 6.2.2 Scalar and vectorial fixpoints

Instead of just trying to solve a single equation
\[
x = f(x),
\]
a system of simultaneous equations
\[ x_1 = f_1(x_1, x_2, \ldots, x_n), \]
\[ x_2 = f_2(x_1, x_2, \ldots, x_n), \]
\[ \vdots \]
\[ x_n = f_n(x_1, x_2, \ldots, x_n), \]
can be solved in the product of the sets of subsets, using the product order:
\[ \langle E_1, E_2, \ldots, E_n \rangle \subseteq \langle E'_1, E'_2, \ldots, E'_n \rangle \quad \text{iff} \quad \forall i \in \{1, \ldots, n\}, E_i \subseteq E'_i. \]
The general results for the existence of least solutions and the manner in which they are computed remain true in this case.

It is shown below that the solution of a system of equations can be obtained by solving each of the equations independently, inserting the obtained results in the other equations.

Consider a mapping \( f : \wp(E)^{k+1} \rightarrow \wp(E) \), monotone over its first argument. With each \( k \)-tuple \( X = \langle X_1, \ldots, X_k \rangle \) of subsets of \( E \), associate the monotone function \( f^X : \wp(E) \rightarrow \wp(E) \) such that \( f^X(X) = f(X, X_1, \ldots, X_k) \). Write
\[ f^\mu : \wp(E)^k \rightarrow \wp(E) \]
for the mapping such that: \( f^\mu(X_1, \ldots, X_k) \) is the least fixpoint of \( f^X \). Write
\[ f^\nu : \wp(E)^k \rightarrow \wp(E) \]
for the mapping such that: \( f^\nu(X_1, \ldots, X_k) \) is the greatest fixpoint of \( f^X \).

Suppose now that \( f \) is also monotone over another argument, for example the \( (i+1) \)-th. Then \( f^\mu \) and \( f^\nu \) are monotone over their \( i \)-th argument. This follows from the following lemma.

**Lemma 6.1** Let \( f : \wp(E)^{k+1} \rightarrow \wp(E) \) be a function monotone over its first argument. If for a certain \( i \) included between 1 and \( k \), and for subsets
\[ X_1, X_2, \ldots, X_{i-1}, X_i, X'_i, X_{i+1}, \ldots, X_k \subseteq E, \]
\[ \forall X \subseteq E, f(X, X_1, \ldots, X_i, \ldots, X_k) \subseteq f(X, X_1, \ldots, X'_i, \ldots, X_k) \]
holds, then
\[ f^\mu(X_1, \ldots, X_i, \ldots, X_k) \subseteq f^\mu(X_1, \ldots, X'_i, \ldots, X_k) \]
and
\[ f^\nu(X_1, \ldots, X_i, \ldots, X_k) \subseteq f^\nu(X_1, \ldots, X'_i, \ldots, X_k). \]
Proof By definition,
\[ f''(X_1, \ldots, X_i, \ldots, X_k) = \bigcap_{X \in \mathcal{X}} X, \]
with
\[ \mathcal{X} = \{X \mid f(X, X_1, \ldots, X_i, \ldots, X_k) \subseteq X \} \]
and
\[ f''(X_1, \ldots, X'_i, \ldots, X_k) = \bigcap_{X \in \mathcal{X}'} X, \]
with
\[ \mathcal{X}' = \{X \mid f(X, X_1, \ldots, X_i', \ldots, X_k) \subseteq X \}. \]
And so if \( X \in \mathcal{X}' \), then
\[ f(X, X_1, \ldots, X'_i, \ldots, X_k) \subseteq X \]
and then
\[ f(X, X_1, \ldots, X_i, \ldots, X_k) \subseteq X, \]
hence \( X \in \mathcal{X} \) and \( \mathcal{X}' \subseteq \mathcal{X} \). It follows that
\[ \bigcap_{X \in \mathcal{X}} X \subseteq \bigcap_{X \in \mathcal{X}'} X. \]

The inclusion of \( f''(X_1, \ldots, X_i, \ldots, X_k) \) in \( f''(X_1, \ldots, X'_i, \ldots, X_k) \) is proven similarly. \( \square \)

Let \( f_1, \ldots, f_n \) be \( n \) functions from \( \wp(E)^{n+k} \) to \( \wp(E) \), monotone over their \( n \) first arguments. The function \( \tilde{f} : \wp(E)^{n+k} \rightarrow \wp(E)^n \) is defined by
\[
\tilde{f}(X_1, \ldots, X_n, Y_1, \ldots, Y_k) = \left\langle f_1(X_1, \ldots, X_n, Y_1, \ldots, Y_k), \ldots, f_n(X_1, \ldots, X_n, Y_1, \ldots, Y_k) \right\rangle.
\]
If the operations of union, intersection and component-wise inclusion are defined over \( \wp(E)^n \), then the equation
\[ \bar{X} = \tilde{f}(\bar{X}, \bar{Y}) \]
also has a least solution,
\[ \bigcap\{\bar{X} \mid \tilde{f}(\bar{X}, \bar{Y}) \subseteq \bar{X}\}, \]
and a greatest solution,
\[ \bigcup\{\bar{X} \mid \bar{X} \subseteq \tilde{f}(\bar{X}, \bar{Y})\}. \]
The proof is similar to the one for theorem 6.1.

These solutions are vectorial fixpoints, as compared to the previously defined scalar fixpoints.

Let $g_i(\vec{Y})$ be the $i$-th component of the least fixpoint $\vec{f}^\mu$ of $\vec{f}$. It is shown below that each $g_i$ can be uniquely defined by its least scalar fixpoints (a similar result is easily shown for greatest fixpoints).

This proof works by induction over the number $n$ of components of $\vec{f}$. For $n = 1$, there is nothing to prove. Consider first the case $n = 2$.

Let $\langle g_1(\vec{Y}), g_2(\vec{Y}) \rangle$ be the least solution of

$$\langle X_1, X_2 \rangle = \langle f_1(X_1, X_2, \vec{Y}), f_2(X_1, X_2, \vec{Y}) \rangle. \tag{6.1}$$

Let $f_1(\vec{Y})$ be the least solution of $X = f_1(X, \vec{Y})$ and let $h_2(\vec{Y})$ be the least solution of $X = f_2(f_1(\vec{Y}), X, \vec{Y})$. Define $h_1(\vec{Y}) = f_1(h_2(\vec{Y}), \vec{Y})$. It is shown that $g_1(\vec{Y}) = h_1(\vec{Y})$ and $g_2(\vec{Y}) = h_2(\vec{Y})$.

By the definition of $f_1^\mu$,

$$f_1^\mu(h_2(\vec{Y}), \vec{Y}) = f_1\left(f_1^\mu(h_2(\vec{Y}), \vec{Y}), h_2(\vec{Y}), \vec{Y}\right),$$

and by the definition of $h_1$,

$$h_1(\vec{Y}) = f_1\left(h_1(\vec{Y}), h_2(\vec{Y}), \vec{Y}\right). \tag{6.2}$$

By the definition of $h_2$,

$$h_2(\vec{Y}) = f_2\left(f_1^\mu(h_2(\vec{Y}), \vec{Y}), h_2(\vec{Y}), \vec{Y}\right),$$

and by the definition of $h_1$,

$$h_2(\vec{Y}) = f_2\left(h_1(\vec{Y}), h_2(\vec{Y}), \vec{Y}\right). \tag{6.3}$$

Equations 6.2 and 6.3 show that $\langle h_1(\vec{Y}), h_2(\vec{Y}) \rangle$ satisfies equation 6.1 and therefore

$$\langle g_1(\vec{Y}), g_2(\vec{Y}) \rangle \subseteq \langle h_1(\vec{Y}), h_2(\vec{Y}) \rangle.$$

Conversely, because

$$g_1(\vec{Y}) = f_1(g_1(\vec{Y}), g_2(\vec{Y}), \vec{Y}) \quad \text{and} \quad g_2(\vec{Y}) = f_2(g_1(\vec{Y}), g_2(\vec{Y}), \vec{Y}),$$

$$f_1^\mu(g_2(\vec{Y}), \vec{Y}) \subseteq g_1(\vec{Y}),$$

since $f_1^\mu(g_2(\vec{Y}), \vec{Y})$ is the least solution of $X = f_1(X, g_2(\vec{Y}), \vec{Y})$. Hence

$$f_2\left(f_1^\mu(g_2(\vec{Y}), \vec{Y}), g_2(\vec{Y}), \vec{Y}\right) \subseteq f_2(g_1(\vec{Y}), g_2(\vec{Y}), \vec{Y}) = g_2(\vec{Y}),$$
and since
\[ h_2(\vec{Y}) = \bigcap \left\{ X \mid f_2\left( f_1^t(X, \vec{Y}), X, \vec{Y} \right) \subseteq X \right\}, \]

\[ h_2(\vec{Y}) \subseteq g_2(\vec{Y}). \]
It follows that
\[ h_1(\vec{Y}) = f_1^t\left( h_2(\vec{Y}), \vec{Y} \right) \subseteq f_2^t\left( g_2(\vec{Y}), \vec{Y} \right) \subseteq g_1(\vec{Y}). \]

Consider the general case. Let \((g_1(\vec{Y}), \ldots, g_n(\vec{Y}))\) be the least solution of equation
\[ \langle X_1, \ldots, X_n \rangle = \langle f_1(X, \vec{Y}), \ldots, f_n(X, \vec{Y}) \rangle. \]

Let \(f_1^t(X_2, \ldots, X_n, \vec{Y})\) be the least solution of \(X = f_1(X, X_2, \ldots, X_n, \vec{Y})\). Consider then the least solution \((h_2(\vec{Y}), \ldots, h_n(\vec{Y}))\) of the vectorial equation
\[ \langle X_2, \ldots, X_n \rangle = \langle f_2(f_1^t(X_2, \ldots, X_n, \vec{Y}), X_2, \ldots, X_n, \vec{Y}), \ldots, f_n(f_1^t(X_2, \ldots, X_n, \vec{Y}), X_2, \ldots, X_n, \vec{Y}) \rangle. \]

By the inductive hypothesis, the \(h_i\) can be stated as scalar fixpoints. So let
\[ h_1(\vec{Y}) = f_1^t(h_2(\vec{Y}), \ldots, h_n(\vec{Y}), \vec{Y}) \]

to show that \(h_i = g_i\), for every \(i\), as was done for the case \(n = 2\). \qed

### 6.2.3 The duality principle

The well-known de Morgan law about equality,
\[ X \cup Y = E - \left( (E - X) \cap (E - Y) \right), \]
is a particular case of a more general situation, already seen with equality:
\[ AX_{\mathcal{A}}(P) = S - EX_{\mathcal{A}}(S - P). \]

These are examples of dual operations. The effect of duality on the computation of fixpoints is given here.

Let \(f\) be a function from \(\wp(E)^n\) to \(\wp(E)\). The dual function, written \(\tilde{f}\), is defined by
\[ \tilde{f}(X_1, \ldots, X_n) = E - f(E - X_1, \ldots, E - X_n). \]

In particular, if \(f\) is a constant, \(\tilde{f} = E - f\). If \(f\) is monotone over its \(i\)-th argument, then \(\tilde{f}\) is also monotone over the same argument.
Consider $n$ functions $f_1, \ldots, f_n : \wp(E)^{n+k} \to \wp(E)$, monotone over their $n$ first arguments. Let $(g_1(\vec{Y}), \ldots, g_n(\vec{Y}))$ be the least solution of the vectorial equation

\[
\langle X_1, \ldots, X_n \rangle = \left\langle f_1(X_1, \ldots, X_n, \vec{Y}), \ldots, f_n(X_1, \ldots, X_n, \vec{Y}) \right\rangle,
\]

and let $(h_1(\vec{Y}), \ldots, h_n(\vec{Y}))$ be the greatest solution of the vectorial equation

\[
\langle X_1, \ldots, X_n \rangle = \left\langle f_1(X_1, \ldots, X_n, \vec{Y}), \ldots, f_n(X_1, \ldots, X_n, \vec{Y}) \right\rangle.
\]

These fixpoints exist because of the monotonicity conditions on $f_i$ and because $g_i = \tilde{h}_i$ holds. In other words, the dual of the least fixpoint of a function is equal to the greatest fixpoint of the dual function.

By defining $\vec{Z} = \langle E - Y_1, \ldots, E - Y_n \rangle$, where $\vec{Y} = \langle Y_1, \ldots, Y_n \rangle$,

\[
g_i(\vec{Y}) = E - h_i(\vec{Z})
\]

must be proven.

Let

\[
\mathcal{X} = \{ \langle X_1, \ldots, X_n \rangle \mid \forall i \in \{1, \ldots, n\}, f_i(X_1, \ldots, X_n, \vec{Y}) \subseteq X_i \},
\]

and

\[
\mathcal{Y} = \{ \langle X_1, \ldots, X_n \rangle \mid \forall i \in \{1, \ldots, n\}, X_i \subseteq f_i(X_1, \ldots, X_n, \vec{Z}) \}.
\]

Then

\[
g(\vec{Y}) = \bigcap_{\vec{X} \in \mathcal{X}} \vec{X}
\]

and

\[
h(\vec{Z}) = \bigcup_{\vec{X} \in \mathcal{Y}} \vec{X}.
\]

Let

\[
\mathcal{Y}' = \{ \langle E - X_1, \ldots, E - X_n \rangle \mid \langle X_1, \ldots, X_n \rangle \in \mathcal{Y} \}.
\]

Then

\[
\langle E - h_1(\vec{Z}), \ldots, E - h_n(\vec{Z}) \rangle = \bigcap_{\vec{X} \in \mathcal{Y}'} \vec{X}.
\]

But

\[
\langle X_1, \ldots, X_n \rangle \in \mathcal{Y}'
\]
if and only if
\[ \langle E - X_1, \ldots, E - X_n \rangle \in \mathcal{Y}, \]
if and only if for every \( i \),
\[ E - X_i \subseteq \tilde{f}_i(E - X_1, \ldots, E - X_n, \tilde{Z}), \]
if and only if for every \( i \),
\[ E - X_i \subseteq E - f_i(X_1, \ldots, X_n, \tilde{Y}), \]
if and only if for every \( i \),
\[ f_i(X_1, \ldots, X_n, \tilde{Y}) \subseteq X_i, \]
if and only if
\[ \langle X_1, \ldots, X_n \rangle \in \mathcal{X}. \]
Hence \( \mathcal{X} = \mathcal{Y}' \).

Example
The ‘Until’ operator was defined, on page 73, by letting \( L \cup A L' \) be the least fixpoint of the equation
\[ X = L' \cup (N_A(X) \cap L). \]
Its dual, \( L \bar{U} A L' \), equal to
\[ C - ((C - L) \cup A (C - L')) \]
where \( C \) is the set of all the paths of \( A \), is therefore the greatest fixpoint of the equation
\[ X = L' \cap (\tilde{N}_A(X) \cup L). \]
So \( \tilde{N}_A(X) \), equal to \( C - N_A(C - X) \), must be calculated, where
\[ N_A(X) = \{ t : c \mid c \in X \} \]
and \( X \) is an arbitrary set of paths. Hence
\[ N_A(C - X) = \{ t : c \mid c \not\in X \}. \]
If a path is empty, it obviously is not in \( N_A(C - X) \). If a path \( t \cdot c \) is not empty, either \( c \in X \) and \( t \cdot c \in N_A(X) \), or \( c \not\in X \) and \( t \cdot c \in N_A(C - X) \). The two sets \( N_A(X) \) and \( N_A(C - X) \) therefore form a partition of the set of non-empty paths, and, since the set of non-empty paths is equal to \( N_A(C) \),
\[ \tilde{N}_A(X) = (C - N_A(C)) \cup N_A(X). \]
6.3 The $\mu$-calculus

The $\mu$-calculus, outlined D. Park [73] to define regular languages, is the extension of a logic by the least and greatest fixpoint operators; these play the rôle of quantifiers. The presentation is simplified by considering a logic containing only state formulas. The Dicky calculus, presented in the next section, illustrates how the definitions presented here can be generalized to define a $\mu$-calculus with several types of formulas.

Let $L$ be a state logic formed of the logical operators union, intersection and negation, and of some monotone operators: for every transition system $A$, the interpretation $\omega_A$ of each operator $\omega$ is monotone: for every subset $P_1, P'_1, \ldots, P_n, P'_n$ of $S$, where $n$ is the arity of $\omega$, if

$$\forall i \in \{1, \ldots, n\}, P_i \subseteq P'_i,$$

then

$$\omega_A(P_1, \ldots, P_n) \subseteq \omega_A(P'_1, \ldots, P'_n).$$

Hennessy–Milner logic is one of these logics. In fact, it was used for the original definition of the $\mu$-calculus [62, 63, 75].

Let $X$ be a set of variables. The set of $\mu$-terms is defined inductively. The variables appearing in a $\mu$-term $\tau$ are partitioned into three pairwise disjoint sets:

- the set $FV^+(\tau)$ of free variables over which the term is syntactically monotone;
- the set $FV^-(\tau)$ of free variables over which the term is syntactically antimonotone; and
- the set $BV(\tau)$ of bound variables.

Write $FV(\tau)$ for the set $FV^+(\tau) \cup FV^-(\tau)$. These sets are (completely) defined at the same time as the terms.

To avoid the usual problems of bound variable renaming, in the definition of $\mu$-terms, strict syntactic conditions are imposed on the choice of term variables. These conditions do not limit the expressivity of the terms.

- The constants $1$ and $0$ are $\mu$-terms and the three sets of variables associated with each of them are empty.
- A variable $x$ in $X$ is a $\mu$-term and

$$FV^+(x) = \{x\},$$
$$FV^-(x) = \emptyset,$$
$$BV(x) = \emptyset.$$
• If $\tau_1$ and $\tau_2$ are $\mu$-terms such that
\[
(BV(\tau_1) \cup BV(\tau_2)) \cap (FV(\tau_1) \cup FV(\tau_2)) = \emptyset
\]
and
\[
(FV^+(\tau_1) \cup FV^+(\tau_2)) \cap (FV^-(\tau_1) \cup FV^-(\tau_2)) = \emptyset,
\]
then $\tau = \tau_1 \lor \tau_2$ and $\tau = \tau_1 \land \tau_2$ are $\mu$-terms and and the sets $FV^+$, $FV^-$ and $BV$ associated with them are
\[
FV^+(\tau) = FV^+(\tau_1) \cup FV^+(\tau_2),
FV^-(\tau) = FV^-(\tau_1) \cup FV^-(\tau_2),
BV(\tau) = BV(\tau_1) \cup BV(\tau_2).
\]

• If $\tau$ is a $\mu$-term, then $\neg \tau$ is a $\mu$-term and
\[
FV^+(\neg \tau) = FV^-(\tau),
FV^-(\neg \tau) = FV^+(\tau),
BV(\neg \tau) = BV(\tau).
\]

• If $\omega$ is an $n$-ary operator, and if $\tau_1, \ldots, \tau_n$ are $\mu$-terms such that
\[
\left(\bigcup_{i=1}^{n} BV(\tau_i)\right) \cap \left(\bigcup_{i=1}^{n} FV(\tau_i)\right) = \emptyset
\]
and
\[
\left(\bigcup_{i=1}^{n} FV^+(\tau_i)\right) \cap \left(\bigcup_{i=1}^{n} FV^-(\tau_i)\right) = \emptyset,
\]
then $\omega(\tau_1, \ldots, \tau_n)$ is a $\mu$-term and
\[
FV^+(\omega(\tau_1, \ldots, \tau_n)) = \bigcup_{i=1}^{n} FV^+(\tau_i),
FV^-(\omega(\tau_1, \ldots, \tau_n)) = \bigcup_{i=1}^{n} FV^-(\tau_i),
BV(\omega(\tau_1, \ldots, \tau_n)) = \bigcup_{i=1}^{n} BV(\tau_i).
\]

• If $\tau$ is a $\mu$-term and if $x \in FV^+(\tau)$, then $\mu x.\tau$ and $\nu x.\tau$ are $\mu$-terms to which are associated the three sets
\[
FV^+(\tau) = FV^+(\tau) - \{x\},
FV^-(\tau) = FV^-(\tau),
BV(\tau) = BV(\tau) \cup \{x\}.
\]
Let $A = (S, T, \alpha, \beta)$ be a transition system and let $X$ be the set of variables used to build up the $\mu$-terms. If $v$ is a mapping from $X$ to $\wp(S)$, if $x$ is a variable and if $P$ is a subset of $S$, then $v[x := P](y)$ is the mapping from $X$ to $\wp(S)$ defined by:

$$v[x := P](y) = \begin{cases} P & \text{if } x = y \\ v(y) & \text{otherwise.} \end{cases}$$

For each $\mu$-term $\tau$ and for each mapping $v$ from $X$ to $\wp(S)$, define a subset of $S$, written $\tau_A \cdot v$, by:

- $1_A \cdot v = S$;
- $0_A \cdot v = \emptyset$;
- $x_A \cdot v = v(x)$;
- $(\tau_1 \lor \tau_2)_A \cdot v = (\tau_1)_A \cdot v \cup (\tau_2)_A \cdot v$;
- $(\tau_1 \land \tau_2)_A \cdot v = (\tau_1)_A \cdot v \cap (\tau_2)_A \cdot v$;
- $(-\tau)_A \cdot v = S - \tau_A \cdot v$;
- $(\omega(\tau_1, \ldots, \tau_n))_A \cdot v = \omega_A((\tau_1)_A \cdot v, \ldots, (\tau_n)_A \cdot v)$;
- $(\mu x. \tau)_A \cdot v$ is the least fixpoint of equation $X = \tau_A \cdot v[x := X]$;
- $(\nu x. \tau)_A \cdot v$ is the greatest fixpoint of equation $X = \tau_A \cdot v[x := X]$.

These least and greatest solutions exist because in $\mu x. \tau$ and $\nu x. \tau$, variable $x$ is in $FV^+(\tau)$ and the logic's operators are monotone. This can be easily shown by induction over the construction of terms.

**Proposition 6.2** Let $P$ and $P'$ be two subsets of $S$ such that $P \subseteq P'$ and let $\tau$ be a $\mu$-term.

- If $x \in FV^+(\tau)$, then $\tau_A \cdot v[x := P] \subseteq \tau_A \cdot v[x := P']$.
- If $x \in FV^-(\tau)$, then $\tau_A \cdot v[x := P'] \subseteq \tau_A \cdot v[x := P]$.

Still by induction over the construction of $\mu$-terms, it can be shown that to determine the value of $\tau_A \cdot v$, only the value of $v(x)$ for the free variables of $\tau$ need be known.

**Proposition 6.3** Let $\tau$ be a $\mu$-term and let $v$ and $v'$ be two mappings from $X$ to $\wp(S)$. If $\forall x \in FV(\tau), v(x) = v'(x)$, then $\tau_A \cdot v = \tau_A \cdot v'$.

In particular, if $\tau$ has no free variables, the value of $\tau_A \cdot v$ is independent of $v$ and can be written $\tau_A$. A $\mu$-term with no free variables is a closed formula of the $\mu$-calculus.
Example
Consider the $\mu$-calculus built up from Hennessy–Milner logic and consider a transition system $A = \langle S, T, \alpha, \beta, \lambda \rangle$ labeled by the alphabet $\{a, b\}$. The set of states that are sources of infinite paths whose traces contain an infinite number of $b$ is characterized by the closed formula

$$\tau = \nu x. \mu y. (\langle a \rangle y \lor \langle b \rangle x).$$

To show this, let $v$ be a mapping from $\{x, y\}$ to $\wp(S)$. The term $\tau_A$ is the greatest fixpoint of

$$X = (\mu y. (\langle a \rangle y \lor \langle b \rangle x))_A \cdot v[x := X]$$

and $(\mu y. (\langle a \rangle y \lor \langle b \rangle x))_A \cdot v[x := X]$ is the least fixpoint of

$$Y = (\langle a \rangle y \lor \langle b \rangle x)_A \cdot v[x := X][y := Y]$$

i.e. the set of states that are sources to paths in $a^*b$ whose targets are in the set $X$, written, by abusing the notation, $\langle a^*b \rangle X$. It must be checked that the set to be characterized is in fact the greatest fixpoint $V$ of the equation

$$X = (\mu y. (\langle a \rangle y \lor \langle b \rangle x))_A \cdot v[x := X]$$

If $s_0 \in V$, then there exists $s_1 \in V$ such that $s_0 \rightarrow a^n b \rightarrow s_1$. There therefore also exists $s_2 \in V$ such that $s_1 \rightarrow a^m b \rightarrow s_2$, and so on. State $s_0$ is therefore the source of a path whose trace contains an infinite number of $b$.

Conversely, consider a path of source $s_0$ whose trace contains an infinite number of $b$. The path can be written

$$s_0 \rightarrow a^n b \rightarrow s_1 \rightarrow a^m b \rightarrow s_2 \ldots$$

Let $W = \{s_i | i \geq 0\}$. It is clear that $W \subseteq \langle a^*b \rangle W$. Therefore, according to the Knaster-Tarski theorem (theorem 6.1, page 80), $W \subseteq V$, in particular $s_0 \in V$. □

6.4 Dicky calculus

The Dicky calculus is an extension of Dicky logic that allows the use of operators defined as the least or greatest fixpoints of systems of equations. Unlike the $\mu$-calculus, the Dicky calculus does not allow, in its equations, operators already
defined as the fixpoints of equations. But this restriction does not prevent it from expressing a large number of properties.

Indeed, as far as we know, the only ‘natural’ examples of $\mu$-formulas with nested least and greatest fixpoint operators are formulas expressing fairness properties (see section 6.5); another method to deal with such properties is presented in this section.

Among others, the advantages of this calculus over the classical $\mu$-calculus are:

- Systems of simultaneous fixpoint equations can be defined. Although such fixpoints can be expressed with combinations of scalar fixpoints in the $\mu$-calculus, it is easier and shorter to use simultaneous fixpoints.
- Sets of transitions and sets of states can be considered in these simultaneous fixpoint equations. This gives the user more flexibility in writing equations (see, for instance, the definition of Inevitable, example 8 below). Moreover, certain properties are more naturally expressed in terms of transitions than in terms of states (see example 6 below).

### 6.4.1 Systems of equations

The equations whose fixpoints are to be calculated are first defined precisely. It is shown in particular that one can avoid the separate consideration of least and greatest fixpoints by giving to the variables *signs* that state whether the value attributed to them is the least or greatest fixpoint.

**Signed terms**

Let $X_n = \{x_1, \ldots, x_n\}$ and $Y_m = \{y_1, \ldots, y_m\}$ be two sets of variables, respectively, of type $\sigma$ and $\tau$. Terms can be built up from these variables and the Dicky logic operators (see page 54). If $t$ is such a term, its interpretation $t_A$ in the parameterized transition system

$$A = \langle S, T, \alpha, \beta, S_{X_1}, \ldots, S_{X_n}, T_{Y_1}, \ldots, T_{Y_m} \rangle$$

is a mapping from $\wp (S)^n \times \wp (T)^m$ to $\wp (S)$ or $\wp (T)$, depending on the type of the term.

Since the difference is not monotone—while all the other operators are—the interpretation of a term $t$ is not necessarily monotone. However, the difference can be seen as monotone if the first argument is considered to be ordered by inclusion and the second argument by the inverse relation.

This leads to the simultaneous consideration of two orders over $\wp (S)$ and $\wp (T)$, inclusion and its inverse. When terms are being formed, variables can take values in $\wp (S)$ and $\wp (T)$ ordered under inclusion (positive variables) or by anti-inclusion (negative variables). It is then possible to construct terms syntactically whose interpretation is monotone (positive terms) or antimonotone (negative terms) under the orders associated with the signed variables of the term.
Let $X^+$ and $X^-$ be two sets of, respectively, positive and negative variables of type $\sigma$; $Y^+$ and $Y^-$ be two sets of, respectively, positive and negative variables of type $\tau$; and $Z_\sigma$ and $Z_\tau$ be two sets of unsigned variables.

The sets of positive and negative terms of type $\sigma$, $T_{\sigma}^+$ and $T_{\sigma}^-$, and positive and negative terms of type $\tau$, $T_{\tau}^+$ and $T_{\tau}^-$, are defined inductively by

- $X^+ \subset T_{\sigma}^+$, $X^- \subset T_{\sigma}^-$, $Y^+ \subset T_{\tau}^+$, and $Y^- \subset T_{\tau}^-$.  
- $\{0_\rho, 1_\rho\} \cup Z_\rho \subset T_{\rho}^+ \cap T_{\rho}^-$, for $\rho \in \{\sigma, \tau\}$.
- Every elementary proposition associated with a state (resp. transition) parameter is in $T_{\sigma}^+ \cap T_{\sigma}^-$ (resp. $T_{\tau}^+ \cap T_{\tau}^-$).
- If $t_1$ and $t_2$ are in $T_\rho^\varsigma$, then $t_1 \lor_\rho t_2$ and $t_1 \land_\rho t_2$ are in $T_\rho^\varsigma$, for $\rho \in \{\sigma, \tau\}$ and $\varsigma \in \{+,-\}$.
- If $t$ is in $T_\rho^\varsigma$, then $\text{src}(t)$ and $\text{tgt}(t)$ are in $T_\rho^\varsigma$, for $\varsigma \in \{+,-\}$.
- If $t$ is in $T_\rho^\varsigma$, then $\text{in}(t)$ and $\text{out}(t)$ are in $T_\rho^\varsigma$, for $\varsigma \in \{+,-\}$.
- If $t_1$ and $t_2$ are in $T_\rho^\varsigma$ and $t_2$ is in $T_{\rho'}^{\varsigma'}$, then $t_1 -_\rho t_2$ is in $T_\rho^\varsigma$, for $\rho \in \{\sigma, \tau\}$ and $(\varsigma, \varsigma') \in \{(+, -), (-, +)\}$.

If $t$ is a term in $T_\rho^\varsigma$, its interpretation $t_A$ is monotone or antimonotone (depending on the value of $\varsigma$) when the positive variables are ordered by inclusion and the negative variables ordered by anti-inclusion.

**Examples**

- If $P$ and $R$ are positive variables, respectively, of type $\sigma$ and $\tau$, then the terms $\text{Pred}(P) = \text{src}(\text{in}(P))$, $\text{tP}(R) = \text{in}(\text{src}(R))$ and $(R)P = \text{src}(R \land_\tau \text{in}(P))$ are positive, respectively of types $\sigma$, $\tau$ and $\sigma$.

- If $P \in X^+$, $Q \in Y^-$ and $R \in Y^+$, then $f(P, Q, R) = (R)P - \text{Pred}(Q)$ is a positive term of type $\sigma$. $\square$

**Dual terms**

If $t$ is a term of $T_\rho^\varsigma$, the dual of $t$ is the term $\bar{t}$ obtained as follows:

- Each variable $z$ is replaced by $1_{\rho_z} - z$, where $\rho_z$ is the type of $z$. The result is a term $t'$ of type $\rho$ whose sign is the opposite of the sign $\varsigma$ of $t$.
- Let $\bar{t} = 1_\rho - t'$. This term $\bar{t}$ is still in $T_\rho^\varsigma$.

For every transition system $A$, the function $t_A$ is the dual, in the Boolean algebra sense, of the function $t_A$.

**Systems of equations**

Consider the sets of variables

$$X^+ = \{x_1^+, \ldots, x_m^+\},$$

$$X^- = \{x_1^-, \ldots, x_{m'}^-\},$$

$$Y^+ = \{y_1^+, \ldots, y_m^+\},$$

$$Y^- = \{y_1^-, \ldots, y_{m'}^-\}.$$
Fixpoints in transition systems

\[ Y^- = \{ y_1^-, \ldots, y_{m'}^- \}, \]
\[ Z_\varpi = \{ z_1, \ldots, z_p \}, \]
\[ Z_\tau = \{ z'_1, \ldots, z'_p \}, \]

and the positive or null integers \( n_1, n'_1, m_1 \) and \( m'_1 \), with
\[ n_1 \leq n, \: n'_1 \leq n', \: m_1 \leq m \: \text{and} \: m'_1 \leq m'. \]

Consider the sets of terms \( T_\varpi^+, T_\varpi_-, T_\tau^+ \) and \( T_\tau^- \) formed over those sets of variables. A system of equations \( \Sigma \) consists of
\[ \{ x_i^+ = u_i^+ \mid 1 \leq i \leq n_1 \} \cup \]
\[ \{ x_i^- = u_i^- \mid 1 \leq i \leq n'_1 \} \cup \]
\[ \{ y_i^+ = v_i^+ \mid 1 \leq i \leq m_1 \} \cup \]
\[ \{ y_i^- = v_i^- \mid 1 \leq i \leq m'_1 \}. \]

where \( u_i^+ \in T_\varpi^+, u_i^- \in T_\varpi_-, v_i^+ \in T_\tau^+ \) and \( v_i^- \in T_\tau^- \).

For a system of equations \( \Sigma \) and a transition system \( A \), let
\[ D_1 = \varphi(S)^{n_1} \times \varphi(S)^{n'_1} \times \varphi(T)^{m_1} \times \varphi(T)^{m'_1} \]
and
\[ D_2 = \varphi(S)^{n-n_1} \times \varphi(S)^{n'-n'_1} \times \varphi(T)^{m-m_1} \times \varphi(T)^{m'-m'_1} \]
be ordered sets whose first and third factors are ordered by inclusion and the others by anti-inclusion. Also let
\[ D_3 = \varphi(S)^p \times \varphi(T)^p' \]
be ordered arbitrarily. The system \( \Sigma \) then defines the mapping
\[ \Sigma_A : D_1 \times D_2 \times D_3 \longrightarrow D_1. \]

This mapping is monotone under the fixed order over \( D_1 \) and \( D_2 \). It has a least fixpoint \( \mu_{\Sigma_A} : D_3 \times D_3 \longrightarrow D_1 \), which is a monotone mapping under the order over \( D_1 \) and \( D_2 \). It also has a greatest fixpoint \( \nu_{\Sigma_A} : D_2 \times D_3 \longrightarrow D_1 \), which is a monotone mapping.

Thanks to the use of variables and of signed terms, only the least fixpoints need be considered. This holds since if \( u \in T_\rho^\varsigma \), by first inverting the sign of variables of \( u \), one obtains \( u' \in T_\rho^{\varsigma'} \), \( \varsigma \neq \varsigma' \). If \( \Sigma' \) is the system obtained by inverting the sign of variables which appear in the left hand side of \( \Sigma \), then \( \nu_{\Sigma_A} = \mu_{\Sigma'_A} \).

6.4.2 Examples

1. Consider the set of states from which there is an infinite path. This is the greatest fixpoint (under inclusion) of the equation \( x = \text{Pred}(x)(= \text{src}(\text{in}(x))) \), hence the greatest fixpoint of the signed equation \( x^+ = \text{Pred}(x^+) \). It is also the least fixpoint of \( x^- = \text{Pred}(x^-) \).
2. It was seen that the interpretation in \( \mathcal{A} \) of the 'Until' operator is the least fixpoint of the equation
\[
x = E' \lor (E \land Pred_{\mathcal{A}}(x)),
\]
and that another interpretation can be given for that equation: the greatest fixpoint. These two interpretations are therefore defined as the least fixpoints of the two signed equations
\[
x^+ = E' \lor \sigma(E \land \sigma Pred(x^+)), \\
x^- = E' \lor \sigma(E \land \sigma Pred(x^-)).
\]

3. Let \( even_{\mathcal{A}}(Q) \) (resp. \( odd_{\mathcal{A}}(Q) \)) be the states where a state in \( Q \) is reachable by an even (resp. odd) number of transitions. The pair \((even_{\mathcal{A}}(Q), odd_{\mathcal{A}}(Q))\) is then the least fixpoint of the system of equations
\[
x^+_1 = Q \lor \text{src}(\text{in}(x^+_2)), \\
x^+_2 = \text{src}(\text{in}(x^+_1)).
\]

4. The dual of the system in the preceding example is
\[
x^+_1 = 1_\sigma - (1_\sigma - Q) \lor \text{src}(\text{in}(1_\sigma - x^+_2)), \\
x^+_2 = 1_\sigma - \text{src}(\text{in}(1_\sigma - x^+_1)).
\]
From above, the dual of \((even(Q), odd(Q))\) is the greatest fixpoint of that system of equations and is also, by changing the signs of variables, the least fixpoint of
\[
x^-_1 = 1_\sigma - (1_\sigma - Q) \lor \text{src}(\text{in}(1_\sigma - x^-_2)), \\
x^-_2 = 1_\sigma - \text{src}(\text{in}(1_\sigma - x^-_1)).
\]
Let \((\bar{even}_{\mathcal{A}}(Q), \bar{odd}_{\mathcal{A}}(Q))\) be the least fixpoint in \( \mathcal{A} \). Then
\[
\bar{even}_{\mathcal{A}}(Q) = S - even_{\mathcal{A}}(S - Q).
\]
In other words \(\bar{even}_{\mathcal{A}}(Q)\) is the set of all the states \( s \) such that all paths of even length with source \( s \) end up in \( Q \).

5. Suppose that \( \mathcal{A} \) is a transition system labeled by an alphabet \( \{a, b\} \). Let \( even'_{\mathcal{A}}(Q) \) (resp. \( odd'_{\mathcal{A}}(Q) \)) be the set of states from which a state in \( Q \) can be reached by an even (resp. odd) number of transitions labeled \( a \). The pair
\[
(\text{even}'_{\mathcal{A}}(Q), \text{odd}'_{\mathcal{A}}(Q))
\]
is therefore the least fixpoint of the system of equations
\[
x^+_1 = Q \lor \text{src}(T_a \cap \text{in}(x^+_2)) \lor \text{src}(T_b \cap \text{in}(x^+_1)), \\
x^+_2 = \text{src}(T_a \cap \text{in}(x^+_1)) \lor \text{src}(T_b \cap \text{in}(x^+_2)),
\]
where \( T_a \) and \( T_b \) are the sets of transitions respectively labeled \( a \) and \( b \).
6. Consider a transition system \( \mathcal{A} \) representing a mutual exclusion algorithm. Suppose that process \( i \) can do the following two actions:

- \( r_i \): Request to enter a critical section, or
- \( e_i \): Entry into a critical section.

The system \( \mathcal{A} \) therefore contains the sets of transitions \( R_i \) and \( E_i \), containing all the transitions in which process \( i \) executes, respectively, action \( r_i \) and action \( e_i \).

The following property is to be expressed: if process \( i \) requests entry into the critical section, it will eventually enter it. This property can be written

\[
R_i \subseteq T\text{coacc}_\mathcal{A}(E_i), \tag{6.4}
\]

where \( T\text{coacc}_\mathcal{A}(E) \) is the set of transitions \( t \) such that once \( t \) has occurred, it is certain that a transition of \( E \) will occur.

\( T\text{coacc}_\mathcal{A}(E) \) is the least fixpoint of the equation

\[
X = T\text{pred}_\mathcal{A}(X \cup E) - T\text{pred}_\mathcal{A}(\emptyset),
\]

where

\[
T\text{pred}_\mathcal{A}(E) = \{ t = q \rightarrow q' \mid \forall t' = q' \rightarrow q'', t' \in E \},
\]

\[
= \text{in}(1_\sigma \rightarrow \text{src}(1_\sigma \rightarrow E)).
\]

\( T\text{pred}_\mathcal{A} \) is clearly the dual of \( T\text{pred}_\mathcal{A} \), defined by

\[
T\text{pred}_\mathcal{A}(E) = \text{src}(\text{in}(E)).
\]

Intuitively, \( t \in T\text{pred}_\mathcal{A}(t') \) if transition \( t \) precedes transition \( t' \). Since \( \text{Pred} \) is defined by \( \text{Pred}(E) = \text{src}(\text{in}(E)) \),

\[
\text{Pred}(\text{src}(E)) = \text{src}(T\text{pred}(E))
\]

also holds.

Property (6.4) is satisfied if and only if the set of transitions

\[
U_i = R_i - T\text{coacc}_\mathcal{A}(E_i)
\]

is empty. If that set is not empty, its elements are requests for entry into the critical section that are not necessarily followed by such an entry.

7. Consider the operator \( Pr \), of type \( \tau \sigma \rightarrow \sigma \), whose interpretation in a transition system \( \mathcal{A} = (S, T, \alpha, \beta) \) is defined by \( s \in Pr\mathcal{A}(R, P) \) if and only if there exists in \( \mathcal{A} \) a non-empty path \( c = t_1 t_2 \cdots t_p \), with \( \alpha(t_1) = s \), \( \beta(t_p) \in P \), and for which exactly one transition is in \( R \). The operator \( Pr \) is the second component of the least solution of the system of equations

\[
x_1^+ = P \lor \sigma \text{src}(\text{in}(x_1^+) - \tau R),
\]

\[
x_2^+ = \text{src}\left(\left(\text{in}(x_1^+) \land \tau R\right) \lor \sigma \left(\text{in}(x_2^+) - \tau R\right)\right).
\]

Like the Hennessy–Milner \( (a) \) operator, \( Pr\mathcal{A}(R, P) \) can be written as \( (R^* RR^*)\mathcal{A} P \).
8. Consider the *Inevitable* operator, of type $\tau \sigma \rightarrow \sigma$, whose interpretation in $A$ is defined by $s \in \text{Inevitable}_A(R, P)$ if and only if every maximal path from $s$ that is composed solely of transitions in $R$ passes through a state in $P$. It is a generalization of Clarke's AU operator, since it can easily be shown that

$$\text{AU}_A(P, P') = \text{Inevitable}(\text{out}(P), P'),$$

by adopting the convention that every maximal finite path always passes through a state in $P$, whatever the subset $P$ of $S$. This convention only appears to be paradoxical: it was seen on page 57 that all maximal finite paths satisfy $A[1U0]$, which means that every maximal finite path passes through a state in the empty set of states, and since the operator is monotone, it must be the case that every maximal finite path passes through every state.

*Inevitable* can be characterized as the first component of the least fixpoint of the system of equations

$$x^+ = P \lor_{\tau} \left( 1_{\tau} -_\sigma \text{src}(z^-) \right),$$
$$z^- = R -_\tau \text{in}(x^+).$$

To show this, let $(X, Z)$ be the least solution of this system of equations in a transition system $A$ and let $Y = \text{Inevitable}_A(R, P)$. The equation $X = Y$ should hold.

By the definition of $\text{Inevitable}_A(R, P)$, $s$ is in $Y$ if and only if it is in $P$ or if all its successors by a transition of $R$ are in $Y$, and hence $s$ is in $Y$ if and only if it is in $P$ or it is not the source of any transition of $Y' = R - \text{in}_A(Y)$. Then

$$Y = P \cup \left( S - \text{src}_A(Y') \right),$$
$$Y' = R - \text{in}_A(Y).$$

Since $Y$ and $Y'$ satisfy the same equations as $X$ and $Z$, it follows that $X \subseteq Y$ and $Y' \subseteq Z$, since $X$ is a least fixpoint and $Z$ a greatest. It remains to be shown that $Y \subseteq X$.

Let $s$ be a state in $Y$ and let $C(s)$ be the set of maximal paths from $s$, all of whose transitions are in $R$. Let $C'(s)$ be the set of finite paths from $s$ such that

- if $c$ is the empty path $\varepsilon_s$, $c \in C'(s)$ if and only if
  $$s \in P \text{ or } \text{out}_A(s) \cap R = \emptyset;$$
- if $c = t_1t_2 \cdots t_n$ is a non-empty path, $c \in C'(s)$ if and only if
  $$- \forall i \in \{1, \ldots, n\}, t_i \in R \text{ and } \alpha(t_i) \notin P, \text{ and}$$
  $$- \beta(t_n) \in P \text{ or } \text{out}_A(\beta(t_n)) \cap R = \emptyset.$$

The fact that $s$ is in $Y$ can be expressed by the fact that every path in $C(s)$ has a prefix in $C'(s)$.
It is first shown that every path in the set \( C'(s) \) is of length less than or equal to the number of transitions in \( R \). Suppose, for contradiction, that \( t_1 \cdots t_n \) is a path in \( C'(s) \) of length strictly greater than the cardinality of \( R \). There therefore exist two integers \( i \) and \( j \) (\( i < j \)) such that \( t_i = t_j \). Consider the infinite path

\[
c = t_1 \cdots t_i t_{i+1} \cdots t_j t_{i+1} \cdots
\]

It is a path in \( C(s) \) and therefore has a prefix \( c' \) in \( C'(s) \). Let \( t \) be the last transition of the prefix \( c' \). Since \( \beta(t) \) is the source of the transition in \( R \) that follows \( t \) in \( c \), \( \text{out}_A(\beta(t)) \cap R \neq \emptyset \) and so, by the definition of \( C'(s) \), \( \beta(t) \in P \). Furthermore, for every other transition \( t' \) of \( c' \), \( \beta(t') \notin P \), which implies that \( c' \) contains its last transition once only and that it is therefore of the form \( t_1 \cdots t_k \), with \( k < j \leq n \). Since its last transition is \( t_k \), \( \beta(t_k) \in P \), and since \( k < n \), \( \beta(t_k) \notin P \), which is a contradiction.

Let \( d(s) \) be the length of the longest path in \( C'(s) \). It will be shown by induction over \( d(s) \) that \( s \in X \).

- If \( d(s) = 0 \), then the only path of \( C'(s) \) is \( \varepsilon_s \), and so either \( s \in P \) or \( \text{out}_A(s) \cap R = \emptyset \). In the first case, \( s \in X \). In the second case, if \( s \notin X \), then \( s \in \text{src}_A(Z) \subseteq \text{src}_A(R) \). But then \( R \cap \text{out}_A(s) \neq \emptyset \), which is a contradiction.
- If \( d(s) = n + 1 \), then \( s \notin P \) and \( \text{out}_A(s) \cap R \neq \emptyset \). Let

\[
\text{out}_A(s) \cap R = \{t_1, \ldots, t_p\}.
\]

Clearly, \( \forall i \in \{1, \ldots, p\}, \beta(t_i) \in Y \) and \( d(\beta(t_i)) \leq n \). By the inductive hypothesis, \( \forall i \in \{1, \ldots, p\}, \beta(t_i) \in X \). Suppose that \( s \notin X \). Since \( s \notin P \), there must exist \( t \in Z \), with \( s = \alpha(t) \). But since \( Z \subseteq R \), \( t = t_i \) for a particular \( i \), and if that transition \( t_i \) is in \( Z \), that is because \( \beta(t_i) \) is not in \( X \), which is a contradiction.

### 6.4.3 Fixpoint computation algorithms in transition systems

The least solution of Dicky calculus systems of equations can be effectively computed. The 'naïve' algorithm, quadratic in the size of the transition system, is presented first. It is followed by Arnold and Crubillé's linear algorithm [4, 7] and, finally, by Cleaveland and Steffen's improvements [26].

These different algorithms are illustrated using the following example of a system of equations with only one equation

\[
X^+ = \overline{\text{Pred}}(X^+),
\]

whose exact writing is

\[
X = 1 - \text{src}(\text{in}(1 - X)).
\]

Consider, therefore, a system of equations, each of whose parameters has a fixed value, here considered to be constants. By adding supplementary variables, any
system can be rewritten so that each equation of the system contains only one Dicky operator, i.e. has one of the following forms:

**State boolean equations** the variable in the left hand side of the equation is a state variable and is defined by one of the four types of equation:
- \( X = P \), where \( P \) is a constant;
- \( X = 1 - X' \);
- \( X = X_1 \cup X_2 \);
- \( X = X_1 \cap X_2 \).

**Transition boolean equations** the variable in the left hand side of the equation is a transition variable and is defined by one of the four types of equation:
- \( Y = Q \), where \( Q \) is a constant;
- \( Y = 1 - Y' \);
- \( Y = Y_1 \cup Y_2 \);
- \( Y = Y_1 \cap Y_2 \).

**Source equations** the variable in the left hand side of the equation is a state variable and is defined by \( X = \text{src}(Y) \), where \( Y \) is a transition variable.

**Target equations** the variable in the left hand side of the equation is a state variable and is defined by \( X = \text{tgt}(Y) \), where \( Y \) is a transition variable.

**Reciprocal equations** the variable in the left hand side of the equation is a transition variable and is defined by \( Y = \text{in}(X) \) or \( Y = \text{out}(X) \), where \( X \) is a state variable.

Naturally, all these equations also respect the rules about signs of variables.

Applied to the example, the addition of supplementary variables gives the system of equations:

\[
\begin{align*}
X^+ & = 1 - X_1^- , \\
X_1^- & = \text{src}(Y^-) , \\
Y^- & = \text{in}(X_2^-) , \\
X_2^- & = 1 - X^+ .
\end{align*}
\]

For each (state \( s \), state variable \( X \)) pair, define a boolean variable \( s.X \). Similarly, for each (transition \( t \), transition variable \( Y \)) pair, define a boolean variable \( t.Y \). The boolean variable associated with a positive (resp. negative) variable is initialized to false (resp. true).

**The quadratic algorithm**

For each (state \( s \), source equation \( X = \text{src}(Y) \)) pair or (state \( s \), target equation \( X = \text{tgt}(Y) \)) pair, define a new auxiliary boolean variable \( s.X_A \).

For each equation \( e \), let \( \text{compute}_e \) be a procedure whose argument is a state or transition, depending on the equation’s type.
for each state $s$ do $s$.visited := false;
for each state $s$ do visit($s$);

Figure 6.1 Procedure pass.

if not $s$.visited then
begin
  $s$.visited := true;
  for every state boolean equation $e$ do compute$_e(s)$;
  for every source equation $e : X = \text{src}(Y)$ do $s.X_A :=$ false;
  for every transition $t$ of source $s$ do
  begin
    for every transition boolean equation $e$ do compute$_e(t)$;
    for every reciprocal equation $e$ do compute$_e(t)$;
    for every source equation $e : X = \text{src}(Y)$ do compute$_e(t)$;
    visit($\beta(t)$);
  end;
  for every source equation $e : X = \text{src}(Y)$ do $s.X := s.X_A$;
  for every target equation $e : X = \text{tgt}(Y)$ do $s.X_A :=$ false;
  for every transition $t$ of target $s$ do
  begin
    for every transition boolean equation $e$ do compute$_e(t)$;
    for every reciprocal equation $e$ do compute$_e(t)$;
    for every target equation $e : X = \text{tgt}(Y)$ do compute$_e(t)$;
    visit($\alpha(t)$);
  end;
  for every target equation $e : X = \text{tgt}(Y)$ do $s.X := s.X_A$;
end;

Figure 6.2 Procedure visit($s$).

- If $e$ is a state boolean equation of the form $X = b(X', X'')$, then the procedure compute$_e(s)$ is $s.X := b(s.X', s.X'')$.
- If $e$ is a transition boolean equation of the form $Y = b(Y', Y'')$, then the procedure compute$_e(t)$ is $t.Y := b(t.Y', t.Y'')$.
- If $e$ is a source equation $X = \text{src}(Y)$, then the procedure compute$_e(t)$ is $\alpha(t).X_A := \alpha(t).X_A$ or $t.Y$.
- If $e$ is a target equation $X = \text{tgt}(Y)$, then the procedure compute$_e(t)$ is $\beta(t).X_A := \beta(t).X_A$ or $t.Y$.
- If $e$ is a reciprocal equation $Y = \text{in}(X)$ (resp. $Y = \text{out}(X)$) then the procedure compute$_e(t)$ is $t.Y := \beta(t).X$ (resp. $t.Y := \alpha(t).X$).

The pass procedure uses, for each state $s$, a boolean variable $s$.visited (see Figure 6.1). It calls the recursive procedure visit($s$), defined in Figure 6.2.
if not s.visited then
begin
s.visited := true;
s.X := not s.X ; s.X2 := not s.X ;
s.X1A := false;
for every transition t of source s do
begin
  t.Y := β(t).X2;
  s.X1A := s.X1A or t.Y ;
  visit(β(t));
end;
s.X1 := s.X1A ;
for every transition t of target s do
begin
  t.Y := s.X2 ;
  visit(α(t));
end;
end;

Figure 6.3 Visiting \( X = 1 - \text{src}(\ln(1 - X)) \).

Substituting the example into the visit(s) procedure gives the program in Figure 6.3.

When pass is executed, each state is visited once and each transition is examined twice, once from its source and once from its target. When a state is visited, the work to do is proportional to \( n_\sigma + n_o + n_b \), where the three integers are, respectively, the number of state boolean equations, the number of source equations and the number of target equations. When a transition is examined from its source, the work to do is proportional to \( n_r + n_o + n_b \), where \( n_r \) and \( n_r \) are, respectively, the number of transition boolean equations and the number of reciprocal equations. Finally, when a transition is examined from its target, the work to do is proportional to \( n_r + n_o + n_b \). The execution time for pass is therefore proportional to

\[
\text{card}(S)(n_\sigma + n_o + n_b) + \text{card}(T)(2n_r + 2n_r + n_o + n_b).
\]

A valuation is any assignment of values true and false to the boolean variables \( s.X \) and \( t.Y \). For each valuation \( V \), define a mapping \( \hat{V} \) from the set of state and transition variables to \( \varphi(S) \) and \( \varphi(T) \) by \( \hat{V}(X) = \{ s \mid V(s.X) = \text{true} \} \) and \( \hat{V}(Y) = \{ t \mid V(t.Y) = \text{true} \} \).

Once all the boolean variables have been initialized, the pass procedure is run until it no longer modifies the valuation. It is easy to show that if a pass over the transition system no longer modifies the valuation \( V \) obtained from the initial valuation, then no other pass will modify the valuation, and so \( \hat{V} \) is the least fixpoint of the system of equations.
Since each variable $s.X$ or $t.Y$ can change value at most once from true to false or from false to true, depending on the sign, and since during each pass, at least one variable must change value, at most

$$\text{card}(S)(n_s + n_o + n_b) + \text{card}(T)(n_s + n_r)$$

possible passes may take place. The algorithm therefore terminates and computes the least fixpoint of the given system of equations. The execution time of the algorithm is proportional to $(\text{card}(S) + \text{card}(T))^2 n_e^2$, where $n_e$ is the total number of equations.

For the example, the above algorithm is applied to the transition system with three transitions and four states:

- $t_1 : 0 \rightarrow 1$,
- $t_2 : 1 \rightarrow 2$,
- $t_3 : 2 \rightarrow 3$.

The sets associated with the valuations obtained after each pass are presented in Table 6.1.

Table 6.1 Values of variables at each iteration.

<table>
<thead>
<tr>
<th>$X^+$</th>
<th>$X_1^-$</th>
<th>$Y^-$</th>
<th>$X_2^-$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td>0,1,2,3</td>
<td>$t_1, t_2, t_3$</td>
</tr>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>0,1,2</td>
<td>$t_1, t_2, t_3$</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0,1,2</td>
<td>$t_1, t_2$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0,1</td>
<td>$t_1, t_2$</td>
</tr>
<tr>
<td>4</td>
<td>2,3</td>
<td>0,1</td>
<td>$t_1$</td>
</tr>
<tr>
<td>5</td>
<td>2,3</td>
<td>0</td>
<td>$t_1$</td>
</tr>
<tr>
<td>6</td>
<td>1,2,3</td>
<td>0</td>
<td>$t_1$</td>
</tr>
<tr>
<td>7</td>
<td>1,2,3</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>8</td>
<td>0,1,2,3</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>9</td>
<td>0,1,2,3</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

**Remark**

The quadratic algorithm implicitly assumes that a particular state or a transition with a particular source or target can be accessed in constant time, which is possible if appropriate data structures are used. However, the standard representations of directed graphs allow the access in constant time to edges leaving a vertex, but not to edges entering a vertex. If the transitions with a given target cannot be reached in constant time, the algorithm can be modified by defining $\text{visit}(s)$ as in Figure 6.4 (page 103).
if not s.visited then
begin
    s.visited := true;
    for every state boolean equation e do compute\_e(s);
    for every source equation e : X = src(Y) do begin
        s.X := s.X\_A;
        s.X\_A := false;
    end;
    for every source equation e : X = tgt(Y) do begin
        s.X := s.X\_A;
        s.X\_A := false;
    end;
    for every transition t of source s do begin
        for every transition boolean equation e do compute\_e(t);
        for every reciprocal equation e do compute\_e(t);
        for every source equation e : X = src(Y) do compute\_e(t);
        for every target equation e : X = tgt(Y) do compute\_e(t);
        visit(\( \beta(t) \));
    end;
end;

Figure 6.4 Revised procedure visit(s).

As previously, the pass procedure is repeated, but here it must initialize the variables s.X\_A to true or false depending on whether variable X is negative or positive, and repeat the pass until the valuation is not modified by two successive passes. The new algorithm's proof is left to the reader. The essential difference with the previous algorithm is that here the computation of s.X for the source and target equations is not made in the same pass but by straddling two consecutive passes.

The linear algorithm
The major problem with the quadratic algorithm is that it is much more costly than necessary: each time that a state is visited, the values of all the variables associated with that state are recomputed. But if none of the variables used in a computation has been modified by the previous computation, the computation will yield the same result as the previous one. So an attempt is made to determine if the variables used in a computation have been modified since the previous computation.

For each state s (resp. transition t), define a boolean variable s.modified (resp. t.modified). These variables are initialized to true. For each (state s, negative variable X) pair, where X is defined by a source equation e : X = src(Y) or target equation e : X = tgt(Y), define a counter s.c\_X initialized to the number of
transitions of source $s$ for the source equations, and to the number of transitions of target $s$ for the target equations. If the initial value of $s.c_X$ is zero, then $s.X$ is initialized to false instead of being initialized to true: $s$ can never be the source (or the target) of a transition in $Y$ if it is not the source (or the target) of any transition.

The compute$_e$ procedures are modified as follows:

- If $e$ is a state boolean equation of the form $X = b(X', X'')$, then the procedure compute$_e(t)$ computes $s.X := b(s.X', s.X'')$. If the new value is different from the old one, then $s.modified := true$ is also executed.
- If $e$ is a transition boolean equation of the form $Y = b(Y', Y'')$, then the procedure compute$_e(t)$ computes $t.Y := b(t.Y', t.Y'')$. If the new value of $t.Y$ is different from the old one, then $t.modified := true$ is also executed, and, furthermore, for each equation $e'$ of source or target having $Y$ in the right hand side, compute$_e(t)$ is executed.
- If $e$ is a source equation $X = src(Y)$ then the procedure compute$_e(t)$ is as follows:
  - If $X$ is a positive variable and if $a(t).X$ is false, then set $a(t).X$ and $a(t).modified$ to true.
  - If $X$ is a negative variable, subtract 1 from $a(t).c_X$; if this quantity becomes zero, set $a(t).X$ to false and $a(t).modified$ to true.
- If $e$ is a target equation $X = tgt(Y)$, then the procedure compute$_e(t)$ is defined as for $X = src(Y)$, but by replacing $a(t)$ by $b(t)$.
- If $e$ is a reciprocal equation $Y = in(X)$ or $Y = out(X)$, then the procedure compute$_e(t)$ computes $t.Y := b(t).X$ or $t.Y := a(t).X$. If the value of $t.Y$ was modified, $t.modified := true$ is also executed and, for each equation $e'$ of source or target having $Y$ in its right hand side, compute$_e(t)$ is executed.

Finally, for each state $s$ define a boolean variable $s.active$ initialized to false instead of the boolean variable $s.visited$, as they play different roles.

Once initializations of the variables have been made, the algorithm reads:

\[
\text{for each state } s \text{ do visit}(s);
\]

where recursive procedure visit$(s)$ is given in Figure 6.5 and procedure treat$(t)$ is given in Figure 6.6 (page 105).

It is not too difficult to show that this algorithm still computes the least fixpoint of the system of equations, although a rigorous proof would be quite long (see [7], although the algorithm presented here differs slightly).

With the same notations as previously, and by writing $d^+(s)$ and $d^-(s)$ for the number of transitions of source and target $s$, it can be seen that each time that $s.modified$ changes from false to true, the $n_s$ state boolean equations will be
if not \(s\).active then
begin
\(s\).active := true;
while \(s\).modified do
begin
\(s\).modified := false;
for every state boolean equation \(e\) do compute\(_e\)(\(s\));
for every transition \(t\) of source \(s\) do
begin
\(\text{treat}(\beta(\text{t}));\)
\(\text{visit}(\beta(\text{t}));\)
end;
for every transition \(t\) of target \(s\) do
begin
\(\text{treat}(\alpha(\text{t}));\)
\(\text{visit}(\alpha(\text{t}));\)
end;
\(s\).active := false;
end;
\]

**Figure 6.5** Final version of \(\text{visit}(s)\).

for every reciprocal equation \(e\) do compute\(_e\)(\(\text{t}\));
while \(t\).modified do
begin
\(t\).modified := false;
for every transition boolean equation \(e\) do compute\(_e\)(\(\text{t}\));
end;

**Figure 6.6** Procedure \(\text{treat}(s)\).

recomputed, as will the \(n_r\) reciprocal equations \(d^+(s) + d^-(s)\) times. Since this variable can change value at most \(n_\sigma + n_o + n_b\) times, a computation time of
\[(n_\sigma + n_o + n_b) \sum_{s \in S} (n_\sigma + d^+(s) + d^-(s)),\]
which is equal to
\[(n_\sigma + n_o + n_b)(n_\sigma \text{card}(S) + 2\text{card}(T)),\]
is obtained. Similarly, the variable \(t\).modified can change value at most \(n_r + n_r\) times. Each of these modifications provokes the recomputation of the \(n_r\) transition boolean equations and also, when necessary, the source and target transitions, whose maximal number is \(n_o + n_b\), giving a total of
\[\text{card}(T)(n_r + n_r)(n_r + n_o + n_b)\]
visit(0)
    visit(1)
    visit(2)
    visit(3)
        3.X^+ := true; 3.X^- := false;
        backtrack(t_3)
        t_3.Y := false;
        2.c_{X_1^-} := 2.c_{X_1^-} - 1;
        2.X^- := false; 2.modified := true;
        2.X^+ := true; 2.X^- := false;
        backtrack(t_2)
        t_2.Y := false;
        1.c_{X_1^-} := 1.c_{X_1^-} - 1;
        1.X^- := false; 1.modified := true;
        1.X^+ := true; 1.X^- := false;
        backtrack(t_1)
        t_1.Y := false;
        0.c_{X_1^-} := 0.c_{X_1^-} - 1;
        0.X^- := false; 0.modified := true;
        0.X^+ := true; 0.X^- := false;
visit(1)
visit(2)
visit(3)

Figure 6.7 The algorithm applied to \( X = 1 - \text{src}(\text{in}(1 - X)) \).

and a complexity for the algorithm of

\[
\text{card}(S)(n_\sigma + n_o + n_b)n_\sigma \\
+ \text{card}(T)(2(n_\sigma + n_o + n_b) + (n_r + n_r)(n_r + n_o + n_b)).
\]

The behavior of the algorithm, on the example given on page 98, is presented in Figure 6.7. Only the modifications to the main variables are shown.

The improved linear algorithm

Cleaveland and Steffen [26] noticed that the algorithm can still be improved: when \( s.\text{modified} \) or \( t.\text{modified} \) become true, it is not necessary to recompute all the variables associated with \( s \) or \( t \), but, rather, only those that are effectively going to be modified. The latter can always be determined. In the above example, the modification of \( s.X_1^- \) implies a modification of \( s.X^+ \), which itself implies the modification of \( s.X_2^- \). The modification of \( s.X_2^- \) implies the modification of \( t.Y^- \), for every transition \( t \) of target \( s \), and so forth. The result is an algorithm whose complexity is \( \text{card}(S)(n_\sigma + n_o + n_b) + \text{card}(T)(n_r + n_r) \). The converse technique, used to treat the source and target equations, can also be applied to the boolean equations.
while to_compute non-empty do
begin
extract an element $z.Z$ from to_compute;
invert the value of $z.Z$;
$(z,Z).finished := true$;
consequences_of($z.Z$);
end;

Figure 6.8 Improved linear algorithm

Let $X = X' \cup X''$ be a state boolean equation and let $s$ be a state. If the three variables are positive, the boolean variables $s.X$, $s.X'$ and $s.X''$ are initialized to false and $s.X$ becomes true as soon as one of the variables $s.X'$ or $s.X''$ becomes true. If the three variables are negative, the boolean variables $s.X$, $s.X'$ and $s.X''$ are initialized to true and $s.X$ becomes false when the two variables $s.X'$ and $s.X''$ become false.

As in the preceding algorithms, boolean variables $s.X$ and $t.Y$, correctly initialized, are used. The counters $s.c_X$ used in the previous linear algorithm are also used. Furthermore, for each state of transition variable $Z$ defined by a boolean equation of the form $Z' \cap Z''$ (if $Z$ is positive) or of the form $Z' \cup Z''$ (if $Z$ is negative), there are counters $z.c_Z$ ($z$ is a state or a transition, depending on whether $Z$ is a state or transition variable) initialized to 2.

A set to_compute of variables $z.Z$ is also used. Initially this set contains

- all the $z.Z$, where $Z$ is a boolean equation of the form $Z = \mathrm{constant}$ and where the value of that constant in $z$ is different from the initial value of $z.Z$; and

- all the $z.Z$ having a counter $z.c_Z$ whose initial value is 0.

Finally a boolean variable $z.Z.finally$, initialized to false, is associated with each pair $(z, Z)$.

The algorithm is given in Figure 6.8. Procedure consequences_of($z.Z$) puts in to_compute all the variables for which the final value can be known. Therefore all the equations where $Z$ appears in the right hand side are examined. For each such equation $e$, evaluate$_e(z)$, which is defined according to $e$'s form, is executed.

- If $e$ is a boolean equation $Z = b(Z', Z'')$,
  - if a counter is associated with $Z$, then subtract 1 from the counter; if it becomes zero, put $z.Z$ in to_compute;
  - otherwise, if $z.Z.finally$ is false put $z.Z$ in to_compute.

- If $e$ is a source equation $X = \mathrm{src}(Z)$, $z$ must be a transition because $Z$ is a transition variable, so
  - if $X$ is negative, then subtract 1 from $\alpha(z).c_X$; if it becomes zero, put $\alpha(z).X$ in to_compute;
otherwise, if $\alpha(z).X.\text{finished}$ is false put $\alpha(z).X$ in to_compute.

- If $e$ is a target equation, proceed similarly, replacing $\alpha$ by $\beta$.
- If $e$ is a reciprocal equation $Y = \text{in}(Z)$ (resp. $Y = \text{out}(Z)$), and in that case, since $z$ is a state variable, $z$ is a state, put all the variables $t.Y$ in to_compute, where $t$ is a transition of target $z$ (resp. of source $z$).

### 6.5 Strongly connected components and fairness

As was seen in section 4.8, a fairness condition is a boolean combination of elementary conditions of the form $\square \Diamond P$, where $P$ is a boolean combination of elementary propositions. In the general case, the latter express path properties, but in most cases they express state properties. In those conditions, a path satisfies $\square \Diamond P$ if it passes infinitely often through states satisfying $P$. The sources of these paths, i.e. the states from which there exists a path passing infinitely often through states satisfying $P$, can be characterized by a new unary operator $\text{Liv}$. A transition system $\mathcal{A}$’s states satisfying $\text{Liv}(P)$ form the set $\text{Liv}_{\mathcal{A}}(P_{\mathcal{A}})$, where $\text{Liv}_{\mathcal{A}}(E)$ is the set of states $s$ of $\mathcal{A}$ that are the sources of paths passing infinitely often through $E$.

The discussion below pertains to the sources of maximal paths satisfying a fairness condition, which changes nothing when the paths are infinite. In the first instance, the set of states that are sources of maximal paths satisfying

$$\square \Diamond P_1 \land \square \Diamond P_2 \land \cdots \land \square \Diamond P_n$$

is equal to $\text{Liv}_{\mathcal{A}}(\langle P_1 \rangle_{\mathcal{A}}, \langle P_2 \rangle_{\mathcal{A}}, \ldots, \langle P_n \rangle_{\mathcal{A}})$, where $\text{Liv}_{\mathcal{A}}(E_1, E_2, \ldots, E_n)$ is the set of sources of paths that contain an infinite number of states in sets $E_i$, $i = 1 \ldots n$.

For $n = 1$, the result is the original definition $\text{Liv}_{\mathcal{A}}$.

Note further that

$$\square \Diamond P_1 \lor \square \Diamond P_2 \lor \cdots \lor \square \Diamond P_n$$

is equivalent to

$$\square \Diamond (P_1 \lor P_2 \lor \cdots \lor P_n),$$

because, since the set of states of $\mathcal{A}$ is finite, a path passes infinitely often through states satisfying $P_1 \lor P_2 \lor \cdots \lor P_n$ if and only if there exists an $i$ such that the path passes infinitely often through states satisfying $P_i$. Hence

$$\neg \square \Diamond Q_1 \land \neg \square \Diamond Q_2 \land \cdots \land \neg \square \Diamond Q_m$$

is equivalent to

$$\neg \square \Diamond (Q_1 \lor Q_2 \lor \cdots \lor Q_m).$$

Every finite path satisfies this property. Since the infinite paths that satisfy $\neg \square \Diamond Q$ are exactly the paths that satisfy $\square \Diamond 1 \land \neg \square \Diamond Q$, the set of maximal paths satisfying
\neg \Box \Diamond Q \text{ is therefore formed of all the finite maximal paths and of all the paths satisfying } \Box \Diamond \mathbf{1} \land \neg \Box \Diamond Q.

The set of sources of paths satisfying

\[ \Box \Diamond P_1 \land \Box \Diamond P_2 \land \ldots \land \Box \Diamond P_n \land \neg \Box \Diamond Q \]

is equal to

\[ \mathit{Liv}_{A}^{n-}(P_1, A, P_2, A, \ldots, P_n, A, Q, A), \]

where \( \mathit{Liv}_{A}^{n-}(E_1, E_2, \ldots, E_n, E') \) is the set of sources of paths that contain an infinite number of states in sets \( E_i, i = 1 \ldots n, \) and a finite number of states in \( E' \).

Since every fairness condition can be transformed into a disjunction of conditions of the form

\[ \Box \Diamond P_1 \land \Box \Diamond P_2 \land \ldots \land \Box \Diamond P_n \land \neg \Box \Diamond Q_1 \land \neg \Box \Diamond Q_2 \land \ldots \land \neg \Box \Diamond Q_m, \]

with \( n \geq 1 \) and \( m \geq 0, \) or

\[ \neg \Box \Diamond Q, \]

the set of sources of maximal paths satisfying a fairness condition \( \Phi \) is equal to a union of sets of the form

\[ \mathit{Liv}_{A}^{n-}(E_1, E_2, \ldots, E_n, E'), \]

or

\[ \mathit{Liv}_{A}^{1-}(S, E') \cup S', \]

where \( S' \) is the set of sources of maximal finite paths.

Before showing how these sets can be computed, it is shown how they are used for the evaluation of CTL formulas subject to fairness conditions. Since, according to proposition 4.1, for every path \( c \) and every transition \( t \) such that \( \beta(t) = \alpha(c) \), the path \( c \) satisfies the fairness condition \( \Phi \) if and only if the path \( t \cdot c \) satisfies the same condition, it can be seen that

\[ E_{\Phi}[FUF']_{A} = EU_{A}(F_A, E \cap F'_A) \]

and

\[ A_{\Phi}[FUF']_{A} = AU_{A}(F_A, (S - E) \cup (E \cap F'_A)), \]

where \( E \) is the set of states that are the sources of paths satisfying \( \Phi \).
6.5.1 Higher-order systems of equations

One way to define the functions $L_{iv^n}$ and $L_{iv^n}^-$ is to use higher-order equations, defined below.

If $\bar{v}_1 = f(\bar{v}_1, \bar{v}_2, \bar{z})$ is a system of equations, where $\bar{v}_1$ and $\bar{v}_2$ are vectors of signed variables of type $\sigma$ and $\tau$ and $\bar{z}$ is a vector of unsigned variables, its least fixpoint $f^n(\bar{v}_2, \bar{z})$ is a vector of monotone functions. These functions can therefore be used in the terms of a new system of equations. The result is a system of second-order equations, for which most properties of systems of first-order equations remain true. By solving these systems, new functions can be constructed, allowing third-order systems, etc. The result is a hierarchy similar to the one proposed for the $\mu$-calculus by Emerson and Lei [38].

Example

Consider the system

$$x^+ = \text{Pred}(x^+ \lor \sigma z).$$

Its least fixpoint in $\mathcal{A}$ is a monotone function $Coacc^+_{\mathcal{A}}(z)$. Consider then a new unary operator $Coacc^+$ whose interpretation in $\mathcal{A}$ is $Coacc^+_{\mathcal{A}}$.

This $Coacc^+$ operator can be used to compute the set of the sources of maximal finite paths: it is easy to check that in a transition system $\mathcal{A}$ this set is equal to $Coacc^+_{\mathcal{A}}(S - \text{Pred}_\mathcal{A}(S))$.

The system of equations

$$x^+ = Coacc^+(x^+ \land \sigma z)$$

has a least fixpoint $F_{\mathcal{A}}(z)$ in $\mathcal{A}$, as well as a greatest fixpoint, easily shown to be $L_{iv_{\mathcal{A}}}(z)$. The fixpoint $L_{iv_{\mathcal{A}}}(z)$ is therefore also the least fixpoint in $\mathcal{A}$ of the system

$$x^- = Coacc^+(x^- \land \sigma z).$$

Given the preceding results for scalar and vectorial fixpoints, it would not be surprising that $F_{\mathcal{A}}(z)$ is the least fixpoint in $\mathcal{A}$ of a system of ‘ordinary’ first-order equations. Sure enough, $F_{\mathcal{A}}(z)$ is the least fixpoint in $\mathcal{A}$ of $x_2^+ = Coacc^+(x_2^+ \land \sigma z)$, and $Coacc^+(x_2^+ \land \sigma z)$ is the least fixpoint of $x_1^+ = \text{Pred}(x_1^+ \lor \sigma (x_2^+ \land \sigma z))$. Hence $(Coacc^+_{\mathcal{A}}(F_{\mathcal{A}}(z) \land z), F_{\mathcal{A}}(z))$ is the least fixpoint of the system

$$x_1^+ = \text{Pred}(x_1^+ \lor \sigma (x_2^+ \land \sigma z)),$$

$$x_2^+ = x_1^+,$$

and that fixpoint is $(\emptyset, \emptyset)$.

This construction does not apply to the case of $L_{iv_{\mathcal{A}}}(z)$ because of the incompatibility of the signs of variables. In fact, it can be shown that $L_{iv_{\mathcal{A}}}(z)$ is the component of no solution of a first-order system.
More generally $\Liv^n_A(E_1, E_2, \ldots, E_n)$ is the first component of the greatest fixpoint of
\[
x_1^+ = \Coacc_A^+(x_2^+ \cap E_1),
\]
\[
x_2^+ = \Coacc_A^+(x_3^+ \cap E_2),
\]
\[
\vdots
\]
\[
x_n^+ = \Coacc_A^+(x_1^+ \cap E_n),
\]
and $\Liv^n_A^{-}(E_1, E_2, \ldots, E_n, E')$ is the first component of the greatest fixpoint of
\[
x_0^+ = \Coacc_A^+(x_1^+ \cap E_1),
\]
\[
x_1^+ = \Coacc_{\text{restricted}}(x_2^+ \cap E_1, E'),
\]
\[
x_2^+ = \Coacc_{\text{restricted}}(x_3^+ \cap E_2, E'),
\]
\[
\vdots
\]
\[
x_n^+ = \Coacc_{\text{restricted}}(x_1^+ \cap E_n, E'),
\]
where $\Coacc_{\text{restricted}}^+(E, E')$ is the least fixpoint of
\[
x^+ = \text{Pred}_A\left(x^+ \cup (E - E')\right) - E'.
\]

6.5.2 Strongly connected components

Since the examined transition systems are finite, the $\Liv$ functions can be computed differently, using the non-trivial strongly connected components of a transition system.

Since $A$ has only a finite number of states, $\Liv_A(E)$ is also equal to the set of states from which there exists a path that passes infinitely often through the same state $s$ of $E$. This state $s$ therefore belongs to a non-trivial strongly connected component of $A$, i.e. there exists in $A$ a path of positive length from $s$ to $s$. Write $CC_A(E)$ for the set of all states belonging to non-trivial strongly connected components containing states in $E$. In other words, $s \in CC_A(E)$ if and only if there exists $s' \in E$ and a non-empty path from $s$ to $s'$ and a non-empty path from $s'$ to $s$.

It is then easy to see that
\[
Liv_A(E) = \Coacc_A^+(CC_A(E)).
\]
This is true since if $s \in \Coacc_A^+(CC_A(E))$, then there exist $s' \in CC_A(E)$, $s'' \in E$, and three paths from $s$ to $s'$, from $s'$ to $s''$ and from $s''$ to $s'$. The state $s$ is therefore the source of a path that passes infinitely often through $s'' \in E$. Conversely, if there exists a path of source $s$ that passes infinitely often through $E$, that path passes infinitely often through $s' \in E$. There therefore exists a non-empty path from $s'$ to $s'$ and hence $s' \in CC_A(E)$ and $s \in \Coacc_A^+(CC_A(E))$. 

By similar reasoning it can be shown that
\[ \operatorname{Liv}_{\lambda}^n(E_1, \ldots, E_n) = \operatorname{Coacc}_{\lambda}^+(\operatorname{CC}_{\lambda}^n(E_1, \ldots, E_n)), \]
where \( \operatorname{CC}_{\lambda}^n(E_1, \ldots, E_n) \) is the set of states belonging to non-trivial strongly connected components containing states in sets \( E_i, i = 1 \ldots n \), i.e.
\[ \operatorname{CC}_{\lambda}^n(E_1, \ldots, E_n) = \operatorname{CC}_{\lambda}(E_1) \cap \cdots \cap \operatorname{CC}_{\lambda}(E_n). \]
Finally,
\[ \operatorname{Liv}_{\lambda}^n-(E_1, \ldots, E_n, E') = \operatorname{Coacc}_{\lambda}^-(\operatorname{CCC}_{\lambda}^n(E_1, \ldots, E_n, E')), \]
where \( \operatorname{CCC}_{\lambda}^n(E_1, \ldots, E_n, E') \) is the set of states belonging to non-trivial strongly connected components containing states in \( E_i, i = 1 \ldots n \), and no state in \( E' \). It is easily shown that
\[ \operatorname{CCC}_{\lambda}^n(E_1, \ldots, E_n, E') = \operatorname{CCC}_{\lambda}(E_1, E') \cap \cdots \cap \operatorname{CCC}_{\lambda}(E_n, E'), \]
where \( \operatorname{CCC}_{\lambda}(E, E') \) is the set of states belonging to non-trivial strongly connected components containing states in \( E \) and no state in \( E' \). A slight modification of Tarjan's algorithm can compute \( \operatorname{CCC}_{\lambda}(E, E') \), but it is not needed since another generalization of the algorithm is given below.

### 6.5.3 Loops

Instead of considering a non-trivial strongly connected component as a set of states, it shall be considered as a set of transitions.

In fact, more generally, if \( R \) and \( R' \) are sets of transitions, the set \( \operatorname{Loop}_{\lambda}(R, R') \) can easily be computed. It is the set of transitions belonging to a closed path (cycle) composed of transitions in \( R' \) and containing at least one transition in \( R \).

Hence \( \operatorname{CC}_{\lambda}(E) \) is equal to
\[ \operatorname{src}_{\lambda}\left(\operatorname{Loop}_{\lambda}(\operatorname{in}_{\lambda}(E), T)\right). \]
This follows since if \( s \) is the source of a transition of \( \operatorname{Loop}_{\lambda}(\operatorname{in}_{\lambda}(E), T) \), there exists a cycle passing through \( s \) that contains a transition from \( \operatorname{in}_{\lambda}(E) \) and that therefore passes through \( E \), hence \( s \in \operatorname{CC}_{\lambda}(E) \). Conversely, if there exists a path from \( s \) to \( s' \) and a path from \( s' \) to \( s \), with \( s' \in E \), the concatenation of these two paths is a cycle containing a transition of \( s' \).

It turns out that in \( \operatorname{src}_{\lambda}(\operatorname{Loop}_{\lambda}(\operatorname{in}_{\lambda}(E), T)) \), \( \operatorname{src} \) can be independently replaced by \( \operatorname{tgt} \), and in by \( \operatorname{out} \) without changing the value.

Similarly \( \operatorname{CCC}_{\lambda}(E, E') \) is equal to
\[ \operatorname{src}_{\lambda}\left(\operatorname{Loop}_{\lambda}(\operatorname{in}_{\lambda}(E), T - (\operatorname{in}_{\lambda}(E') \cup \operatorname{out}_{\lambda}(E'))\right). \]
Clearly *Loop* is a stronger operator than *CC* or *CCC*, since *CC* and *CCC* can be expressed in terms of *Loop*, but the converse is not true, as is shown by the following example, due to A. Dicky [32]:

Consider \( A \), with states \( \{1, 2\} \) and transitions \( t_1 : 1 \rightarrow 1, t_2 : 1 \rightarrow 2 \) and \( t_3 : 2 \rightarrow 2 \). Then

\[
\begin{align*}
CC_A(\emptyset) &= \emptyset, \\
CC_A(S) &= S, \\
CCC_A(\emptyset, \emptyset) &= \emptyset, \\
CCC_A(\emptyset, S) &= \emptyset, \\
CCC_A(S, \emptyset) &= S, \\
CCC_A(S, S) &= \emptyset, \\
Loop_A(T, T) &= \{t_1, t_3\}.
\end{align*}
\]

But no function defined by a system of equations using \( CC_A \) and \( CCC_A \) can compute the set \( \{t_1, t_3\} \). Each of these functions can only take one of two values, \( \emptyset \) or \( S \) (or \( \emptyset \) or \( T \) depending on the type of the function). To see this, note that the logical operators, as well as src, tgt, in and out, take only those values when they are given those values as arguments. By composition, the terms constructed using those operators still have that property. Finally if \( x_i = t_i(\bar{x}), i = 1, \ldots, n \), is a system of equations whose terms have this property, the successive values that these variables take during the fixpoint computation are still \( \emptyset, S \) or \( T \).

**Example**

Consider the sets of transitions \( R_i \) and \( E_i \) defined in example 6 of page 96, for \( i = 1, 2 \). Suppose that it is possible that the first process enters its critical section infinitely often and that the second only does so a finite number of times; this is equivalent to stating that there exists a cycle containing a transition of \( E_1 \) and no transition of \( E_2 \). The set of transitions belonging to such cycles is precisely \( Loop_A(E_1, T - E_2) \). That set is empty if and only if it is impossible that the first process enters its critical section infinitely often without the second doing the same.

### 6.5.4 Tarjan’s algorithm and its variants

Computing the loops is quite easy and can be done in linear time by adapting Tarjan’s famous algorithm [1, 84] for computing the strongly connected components of a directed graph.

The algorithm is presented here in a slightly modified version, proposed by Cрубиллé [27]. For each state \( s \) of the transition system, define two integer variables \( s\text{-number} \) and \( s\text{-link} \), initialized to 0, and a boolean variable \( s\text{-visited} \), initialized to false. It is assumed that a state \( s \) can be added to a stack by \( \text{push}(s) \) and that it
if not s.visited then
begin
    s.visited := true;
    n := n+1;
    s.number := n;
    s.link := n;
    push(s);
    for every transition t of source s do
    begin
        visit(β(t));
        if β(t) is pushed then s.link := inf(s.link, β(t).link);
    end;
    if s.link = s.number then pop until s is included;
    (the pushed states form a strongly connected component, possibly trivial)
end;

Figure 6.9 Tarjan’s visit(s).

is possible to determine whether a state is in a stack.

The algorithm is

n := 0; for every state s do visit(s)

where the recursive procedure visit(s) is presented in Figure 6.9.

If one wishes to restrict oneself to the sub-transition system formed solely of the transitions belonging to a subset R' of T, it suffices to replace

for every transition t of source s do

by

for every transition t ∈ R' of source s do

The function \(\text{Loop}_A(R, R')\) can now be written. This is done by modifying the above visit(s) procedure to also use a boolean variable s.valid, initialized to false, for each state s, and a boolean variable t.loop, also initialized to false, for each transition t. The resulting algorithm is presented in Figure 6.10 (page 115).

The proof of this algorithm is based on the three following remarks (the algorithm shows the places where the remarks are applicable).

Remark 1. After the execution of visit(β(t)), β(t) is pushed if and only if \(s = α(t)\) and β(t) belong to the same strongly connected component of the sub-transition systems by \(R'\) and therefore if and only if t is a transition of \(\text{Loop}_A(T, R')\).
if not s.visited then
begin
    s.visited := true;
n := n+1;
s.number := n;
s.link := n;
push(s);
end;
for every transition $t \in R'$ of source $s$ do
begin
    visit($\beta(t)$);
    if $\beta(t)$ is pushed then
begin
        s.link := inf(s.link, $\beta(t).link$);
t.loop := true; (see Remark 1)
        if ($t \in R$ or $\beta(t).valid$) then s.valid := true; (see Remark 2)
end;
end;
if s.link = s.number then
begin
    if s.valid then pop until $s$ included;
else begin (see Remark 3)
    pop until $s$ included;
    (for all the popped $s'$, do:
    for every transition $t \in R'$ of source $s'$ do t.loop := false;)
end
end;

Figure 6.10 Tarjan’s revised algorithm

Remark 2. Variable $s.valid$ is set to true if and only if there exists a transition $t \in R$ whose source $\alpha(t)$ is the target of a possibly empty path of transitions in $R'$ whose source is $s$, such that $\alpha(t)$ and $\beta(t)$ are in the same strongly connected component of the sub-transition system defined by $R'$.

Remark 3. If a strongly connected component of the sub-transition system defined by $R'$ contains a transition of $R$, then its state $s$ of minimal number is such that $s.valid$ is equal to true. If this is not the case, the variables $t.loop$, associated with the transitions $t$ whose source belongs to the strongly connected component which is being popped, must be reinitialized to false; these variables were incorrectly set to true (see Remark 1).
In the chapter dealing with verification, the following question was addressed: what are the states (or transitions, or paths) of a transition system $\mathcal{A}$ that satisfy a property expressed by a formula $F$ of a particular logic? This chapter examines the converse question: given a logic, what are the formulas satisfied by a given state (or transition, or path) of a transition system $\mathcal{A}$?

This is done by introducing the concept of *indistinguishability* under a logic $L$: two objects $x$ and $x'$ of the same type in a transition system $\mathcal{A}$ are *indistinguishable* if they satisfy the same set of formulas of that logic:

$$\forall F, \ A, x \models F \iff A, x' \models F.$$ 

### 7.1 Indistinguishable states and transitions

#### 7.1.1 Definitions

Let $L$ be a logic having only state formulas, whose $\Omega$-terms are built up from the set $\Omega$ of operators, containing the standard operators $0$, $1$, $\lor$, $\land$ and $\neg$. Write $\Omega'$ for the set of non-standard operators.

Let $\mathcal{A} = (S, T, \alpha, \beta)$ be a transition system. Define over the states of $\mathcal{A}$ the indistinguishability relation $\sim_L$ associated with the logic $L$ by $s \sim_L s'$ if and only if for every formula $F$ of $L$,

$$\mathcal{A}, s \models F \iff \mathcal{A}, s' \models F,$$

or, equivalently, for every closed $\Omega$-term $t$,

$$s \in t_\mathcal{A} \iff s' \in t_\mathcal{A}.$$ 

This relation is obviously an equivalence relation.
Assume as well that if \( \mathcal{A} \) is a parameterized transition system, then the set \( \Omega \) of operators from which the logic \( L \) is built up contains the elementary propositions \( P_X \) associated with the state parameters, satisfying \((P_X)_\mathcal{A} = S_X\), so that

\[
s \sim_L s' \Rightarrow \mu_\sigma(s) = \mu_\sigma(s'),
\]

where \( \mu_\sigma \) is the marking of the states of a parameterized transition system, defined on page 8.

### 7.1.2 Quotients under indistinguishability

The indistinguishability relation over the states of a transition system \( \mathcal{A} \) induces an indistinguishability relation over the transitions, also written \( \sim_L \), defined by

\[
t \sim_L t'
\]

if and only if

\[
\alpha(t) \sim_L \alpha(t') \text{ and } \beta(t) \sim_L \beta(t').
\]

If \( \mathcal{A} \) is a labeled transition system, the condition \( \lambda(t) = \lambda(t') \) is added, and if \( \mathcal{A} \) is parameterized, the condition \( \mu_r(t) = \mu_r(t') \) is added, where \( \mu_r \) is the marking of the transitions, defined on page 8. This relation is also an equivalence relation.

The quotient of a transition system \( \mathcal{A} = \langle S, T, \alpha, \beta \rangle \) under the indistinguishability relation \( \sim_L \) is the transition system \( \mathcal{A}/\sim_L = \langle S', T', \alpha', \beta' \rangle \), where

- \( S' = S/\sim_L \) is the set of indistinguishability classes of states;
- \( T' = T/\sim_L \) is the set of indistinguishability classes of transitions;
- \( \alpha'(t/\sim_L) = \alpha(t)/\sim_L \);
- \( \beta'(t/\sim_L) = \beta(t)/\sim_L \).

The definition of \( \alpha' \) and \( \beta' \) is meaningful since it does not depend on the transition chosen as representative of an indistinguishability class: if \( t \sim_L t' \), then, by the definition of the extension of the indistinguishability relation to transitions, \( t \sim_L t' \), hence \( \alpha(t) \sim_L \alpha(t') \) and \( \beta(t) \sim_L \beta(t') \), from which \( \alpha(t/\sim_L) = \alpha(t'/\sim_L) \) and \( \beta(t/\sim_L) = \beta(t'/\sim_L) \).

If in addition \( \mathcal{A} \) is labeled, its quotient can be labeled by \( \lambda'(t/\sim_L) = \lambda(t) \), since it was stipulated that in that case, \( t \sim_L t' \) implies \( \lambda(t) = \lambda(t') \). If \( \mathcal{A} \) is parameterized by \( \langle \mathcal{X}, \mathcal{Y} \rangle \), then \( \mathcal{A}/\sim_L \) can be parameterized by

\[
S'_X = \{ s/\sim_L \mid s \in S_X \},
\]

\[
T'_X = \{ t/\sim_L \mid t \in T_X \}.
\]

Here again, the conditions imposed on the indistinguishability relation between parameterized systems ensure that the definition is meaningful.
The mapping $h = (h_\sigma, h_\tau)$ defined by

$$h_\sigma : S \to S/\sim_L,$$
$$s \sim h_\sigma(s/\sim_L),$$
$$h_\tau : T \to T/\sim_L,$$
$$t \sim h_\tau(t/\sim_L),$$
is clearly a surjective homomorphism of (labeled, parameterized) transition systems, since

$$\alpha'(h_\tau(t)) = h_\sigma(\alpha(t)),$$
$$\beta'(h_\tau(t)) = h_\sigma(\beta(t)),$$
$$\lambda'(h_\tau(t)) = \lambda(t),$$

$s \in S_X \iff h_\sigma(s) \in S'_X,$
$t \in T_Y \iff h_\tau(t) \in T'_Y.$

The existence of this homomorphism justifies the term ‘quotient’ for the transition system $A/\sim_L$.

### 7.1.3 Logics adequate for indistinguishability

Consider a logic containing state and transition formulas. Assume that if this logic is applicable to labeled transition systems, then it contains elementary propositions $T_a$, for each letter $a$ of the alphabet, such that if $A = (S, T, \alpha, \beta, \lambda)$, then $t \in (T_a)_A \iff \lambda(t) = a$. If this logic is applicable to transition systems parameterized by $(X, Y)$, assume it contains elementary (state) propositions $P_X$ for each $X$ in $X$ and elementary (transition) propositions $P_Y$ for each $Y$ in $Y$, with $(P_X)_A = S_X$ and $(P_Y)_A = T_Y$.

For this logic define an indistinguishability relation over the states, written $\sim_L$, by $s \sim_L s'$ if and only if for every state formula $F$,

$$A, s \models F \iff A, s' \models F,$$

and an indistinguishability relation over the transitions, also written $\sim_L$, defined by $t \sim_L t'$ if and only if for every transition formula $G$,

$$A, t \models G \iff A, t' \models G.$$

The conditions imposed on the existence of certain propositions in logic $L$ allow the deduction that if $A$ is a labeled transition system, then

$$t \sim_L t' \Rightarrow \lambda(t) = \lambda(t'),$$

and if $A$ is a parameterized transition system, then

$$s \sim_L s' \Rightarrow \mu_\sigma(s) = \mu_\sigma(s'),$$
$$t \sim_L t' \Rightarrow \mu_\tau(t) = \mu_\tau(t').$$
The logic $L$ is adequate if
\[ t \sim_L t' \Rightarrow \alpha(t) \sim_L \alpha(t'), \beta(t) \sim_L \beta(t'). \]

It is then possible to define, as was done previously, the transition system $\mathcal{A}/\sim_L$, quotient of $\mathcal{A}$ under the indistinguishability relation associated with $L$.

**Examples**

1. Dicky logic is adequate. Suppose that there exist two transitions $t$ and $t'$ in a transition system $\mathcal{A}$, with $\alpha(t) \not\sim_L \alpha(t')$. There then exists a state formula $F$ such that $\alpha(t) \in F_\mathcal{A}$ and $\alpha(t') \not\in F_\mathcal{A}$. Then $t \in (\text{out}(F))_\mathcal{A}$ and $t' \notin (\text{out}(F))_\mathcal{A}$, hence $t \not\sim_L t'$. Similarly, if $\beta(t) \not\sim_L \beta(t')$, there exists a formula $F$ such that $\text{in}(F)$ can differentiate $t$ and $t'$.

2. Consider a logic containing only state formulas, as in the beginning of this section. Transition formulas are added to it as follows:

- $(F_1 \rightsquigarrow F_2)$, where $F_1$ and $F_2$ are state formulas;
- $G_1 \lor G_2$, $G_1 \land G_2$, and $\neg G_1$, where $G_1$ and $G_2$ are transition formulas;

and, possibly,

- $P_Y$, where $Y$ is a transition parameter name; and
- $T_a$, where $a$ is a label.

The interpretation of formula $(F \rightsquigarrow F')$ is
\[ (F \rightsquigarrow F')_{\mathcal{A}} = \{ t \mid \alpha(t) \in F_\mathcal{A}, \beta(t) \in F'_\mathcal{A} \}, \]

and the other formulas are interpreted as usual.

The indistinguishability relation $\sim_L$ over transitions induced by these transition formulas is the same as the indistinguishability relation over transitions, here written $\sim_L$, induced by the previously defined indistinguishability relation over states.

In fact, if $t \sim_L t'$, then $\lambda(t) = \lambda(t')$ or $\mu_r(t) = \mu_r(t')$. At the same time,
\[ \alpha(t) \in F_\mathcal{A} \text{ and } \beta(t) \in F'_\mathcal{A} \]

if and only if
\[ t \in (F \rightsquigarrow F') \]

if and only if
\[ t' \in (F \rightsquigarrow F') \]

if and only if
\[ \alpha(t') \in F_\mathcal{A} \text{ and } \beta(t') \in F'_\mathcal{A}. \]

Hence $\alpha(t) \sim_L \alpha(t')$ and $\beta(t) \sim_L \beta(t')$. By definition it follows that $t \sim_L t'$.

Conversely, if $t \not\sim_L t'$, then there exists a transition formula $G$ that distinguishes $t$ and $t'$ in the sense that $t \in G_\mathcal{A}$ and $t' \notin G_\mathcal{A}$, or the converse:
• If $G = P_Y$ (or if $G = T_a$), then $\mu_r(t) \neq \mu_r(t')$ (or $\lambda(t) \neq \lambda(t')$), hence $t \not \sim_L t'$.

• If $G = (F \sim F')$, then (admitting that $t \in G_A$, the proof would be similar in the other case) $\alpha(t) \in F_A$, $\beta(t) \in F'_A$ and, since $t' \not \in G_A$, $\alpha(t') \not \in F_A$ or $\beta(t') \not \in F'_A$. Hence $\alpha(t) \not \sim_L \alpha(t')$ or $\beta(t) \not \sim_L \beta(t')$, and so $t \not \sim_L t'$.

• If $G = G' \land G''$, still assuming that $t \in G_A$, then $t \in G'_A \cap G''_A$ and $t' \not \in G'_A \cap G''_A$, hence one of the formulas $G'$ or $G''$ can differentiate $t$ and $t'$.

• Finally, if $\neg G$ can distinguish $t$ and $t'$, so can $G$, and if $G = G' \lor G''$ can distinguish $t$ and $t'$, then one of the two formulas $G$ or $G''$ can as well.

The logic is therefore adequate.

7.2 Indistinguishability classes

In accordance with the construction of example 2 on page 119, an indistinguishability relation over states and transitions is used. Assume that there is an associated logic containing state and transition formulas.

This logic must have two base types, $\sigma$ and $\tau$, and is formed from a set $\Omega$ of operators containing

• constants of types $\sigma$ and $\tau$, including $0_\sigma$, $1_\sigma$, $0_\tau$ and $1_\tau$;

• operators $\lor_\sigma$, $\land_\sigma$, $\neg_\sigma$, $\lor_\tau$, $\land_\tau$ and $\neg_\tau$ ($\neg_\sigma$ and $\neg_\tau$ can be used instead of $\neg_\sigma$ and $\neg_\tau$); and

• the elements of a set $\Omega'$ of typed operators.

7.2.1 Indistinguishability and fixpoints

Suppose that logic $L$ is augmented with operators definable as the least or greatest solutions of systems of equations. Then, for finite transition systems, the indistinguishability relation for the extended logic $L'$ is the same as the one for the logic $L$.

To avoid abusing notation, this result will be proven for a very simple case. The general case is handled similarly and is left as an exercise to the reader.

Let $u(x, y)$ be a $\Omega$-term syntactically monotone over variable $x$. Let $\omega$ be a new operator of type $\rho_1 \cdots \rho_n \rightarrow \rho$ such that for every transition system $A$, and for every vector $\vec{Y} = (Y_1, \ldots, Y_n)$ of sets $Y_i$ of type $\rho_i$, $\omega_A(\vec{Y})$ is the least solution of the equation

\[ X = u_A(X, \vec{Y}), \]

or of $X = f_{\vec{Y}}(X)$, where $f_{\vec{Y}}(X) = u_A(X, \vec{Y})$.

According to theorem 6.3, there exists an integer $k$ such that for every $\vec{Y}$,

\[ \omega_A(\vec{Y}) = f_{\vec{Y}}^k(\emptyset). \]

Define then the sequence $u_i$ of terms of $T_\rho(\Omega, \vec{y})$:
\[ u_0 = 0_\rho, \]
\[ u_{i+1} = u(u_i, i). \]

It is easily shown by induction that
\[ (u_i)_A(\vec{v}) = f^i_P(\emptyset). \]

Consider a formula \( F \), i.e. a closed \( \Omega \)-term, containing occurrences of the operator \( \omega \) and let \( F' \) be the formula obtained from \( F \) by replacing all the occurrences of \( \omega \) by \( u_k \). It is easy to show by induction over the construction of \( F \) that \( F_A = F'_A \).

The stated result follows immediately from this property:

- Since each formula of \( L \) is also a formula of \( L' \), \( x \sim_{L'} x' \Rightarrow x \sim_L x' \).
- If \( x \not\sim_{L'} x' \), there exists a formula \( F \) of \( L' \) such that \( x \in F_A \) and \( x' \not\in F_A \), and since there exists a formula \( F' \) of \( L \) such that \( F_A = F'_A \), the relation \( x \not\sim_L x' \) holds.

### 7.2.2 Characteristic formulas

A (state or transition) formula \( F \) is characteristic of an indistinguishability class \( x/\sim_L \) (composed of states or transitions) of a transition system \( A \) if

\[
\forall x', \ x' \in x/\sim_L \Leftrightarrow x' \in F_A.
\]

**Theorem 7.1** If \( A \) is a finite transition system, every indistinguishability class has a characteristic formula.

**Proof** Let \( x \) be an object of \( A \) (a state or a transition). For every object \( x' \) (of the same type) that is distinguishable from \( x \), there exists a formula \( F_{x'} \) such that \( x' \not\in (F_{x'})_A \). Consider then the formula

\[
F = F_{x_1} \land \cdots \land F_{x_n},
\]

where \( \{x_1, \ldots, x_n\} \) is the set—obviously finite—of all the objects distinguishable from \( x \). Then, by definition, \( x \in F_A = (F_{x_1})_A \cap \cdots \cap (F_{x_n})_A \), and if \( x' \sim_L x \), then \( x' \in F_A \), by the definition of the indistinguishability relation. Conversely, if \( x' \not\sim_L x \), then there exists an \( i \) such that \( x' = x_i \), and, since \( x' \not\in (F_{x_i})_A \), such that \( x' \not\in F_A. \quad \square \)

The existence of a characteristic formula in CTL, for which the indistinguishability classes have a particular characterization (see section 7.3.3, page 126), was constructively proven in [21].

Should \( A \) be infinite, characteristic formulas can also be defined if the logic allows the construction of formulas that are infinite conjunctions of other formulas [8].
7.3 Saturating homomorphisms

It was seen that the mapping that takes objects (states or transitions) of a transition system to their indistinguishability class is a surjective homomorphism. This homomorphism has other properties that are studied here. To do this, saturating homomorphisms of transition systems are presented. These homomorphisms were introduced in [8] in a more general context.

7.3.1 Definitions and properties

Let $\mathcal{A} = \langle S, T, \alpha, \beta \rangle$ and $\mathcal{A}' = \langle S', T', \alpha', \beta' \rangle$ be two transition systems, and let $h = (h_{\sigma}, h_{\tau})$ be a homomorphism from $\mathcal{A}$ to $\mathcal{A}'$. Let $\omega$ be an operator of type $\rho_1 \cdots \rho_n \rightarrow \rho$, where $\rho, \rho_i \in \{\sigma, \tau\}$.

The homomorphism $h$ saturates $\omega$ if for every sequence $Z_1', \ldots, Z_n'$ of subsets of $S'$ or $T'$, depending on whether $\rho_i = \sigma$ or $\tau$,

$$h_{\rho_i}^{-1}(\omega_{\mathcal{A}'}(Z_1', \ldots, Z_n')) = \omega_{\mathcal{A}}(h_{\rho_1}^{-1}(Z_1'), \ldots, h_{\rho_n}^{-1}(Z_n')),$$

where $h_{\rho_i}^{-1}(Z) = \{z \mid h_{\rho}(z) \in Z\}$.

**Proposition 7.1**

(i) Every homomorphism saturates the operators $0, 1, \lor, \land, \neg$ and $-$ of type $\sigma$ or $\tau$.

(ii) If $h$ is a homomorphism of labeled transition systems, it saturates the constants of type $\tau$ associated with the letters of the alphabet. If $h$ is a homomorphism of parameterized transition systems, it saturates the constants associated with the parameter names.

**Proof** Point (i) follows from the general properties of reciprocal functions. Point (ii) is an immediate consequence of the definitions of these homomorphisms: for example, if $P_X$ is the elementary proposition associated with the state parameter name $X$, as in $(P_X)_A = S_X$, $(P_X)_{A'} = S'_X$ and $h_{\sigma}(s) \in S'_X \Leftrightarrow s \in S_X$, then $s \in h_{\rho}^{-1}((P_X)_{A'})$ if and only if $h_{\sigma}(s) \in S'_X$ if and only if $s \in S_X = (P_X)_A$.

A homomorphism saturates logic $L$ if it saturates all that logic's operators. According to the above proposition, this is equivalent to requiring that the homomorphism saturates the operators in $\Omega'$.

**Theorem 7.2** Let $h : \mathcal{A} \rightarrow \mathcal{A}'$ be a homomorphism saturating a logic $L$. Then for every formula $F$ in $L$ of type $\rho \in \{\sigma, \tau\}$, and for every object of type $\rho$,

$$x \in F_A \Leftrightarrow h_{\rho}(x) \in F_{A'}.$$
Proof What is to be proven can be reformulated as
\[ h^{-1}_\rho(F_{A'}) = F_A, \]
which is easily proven by induction over the construction of \( F \). Let
\[ F = \omega(F_1, \ldots, F_n), \]
where \( \omega \) is an arbitrary operator of the logic, of type \( \rho_1 \cdots \rho_n \to \rho \), \( n \) possibly zero if \( \omega \) is a constant, and \( F_i \) is a formula of type \( \rho_i \). Since \( h \) is saturating,
\[ h^{-1}_\rho(F_{A'}) = h^{-1}_\rho(\omega_{A'}((F_1)_{A'}, \ldots, (F_n)_{A'})) = \omega_{A'}(h^{-1}_\rho_1(F_1)_{A'}, \ldots, h^{-1}_\rho_n(F_n)_{A'}) \]
and by the inductive hypothesis,
\[ \omega_{A'}(h^{-1}_\rho_1(F_1)_{A'}, \ldots, h^{-1}_\rho_n(F_n)_{A'}) = \omega_{A'}((F_1)_{A'}, \ldots, (F_n)_{A'}) = F_A. \]
\[ \square \]

Note that if \( h \) is surjective, then \( h^{-1}_\rho(F_{A'}) = F_A \) implies \( F_{A'} = h_\rho(F_A) \), since if \( h \) is a surjective mapping, \( h(h^{-1}(X)) = X \).

This property of saturating homomorphisms can be used to prove the following theorem for surjective homomorphisms saturating an adequate logic.

Theorem 7.3 Let \( h_i : A_i \to A'_i, i \in \{1, 2\} \), be the canonical homomorphism from \( A_i \) to its quotient under indistinguishability. If \( h \) is a saturating surjective homomorphism from \( A_1 \) to \( A_2 \), then \( A'_1 \) and \( A'_2 \) are isomorphic and \( h_1 = h_2 \circ h \), up to isomorphism.

Proof It is first shown that two objects \( x \) and \( x' \) of \( A_1 \) are indistinguishable if and only if
\[ (h_2)_\rho(h_\rho(x)) = (h_2)_\rho(h_\rho(x')), \]
which is equivalent to
\[ h_\rho(x) \sim_L h_\rho(x'). \]
In fact, for every formula \( F, x \in F_{A_1} \Leftrightarrow h_\rho(x) \in F_{A_2} \), from which
\[ x \sim_L x' \Leftrightarrow h_\rho(x) \sim_L h_\rho(x'). \]
It must also be shown that the mapping \( g = (g_\sigma, g_\tau) \) from \( A'_1 \) to \( A'_2 \), defined by
\[ g_\rho(x/\sim_L) = (h_2)_\rho(h_\rho(x)), \]
is a transition system isomorphism. Note first that \( g \) is well defined, from above. Since \( h_2 \) and \( h \) are surjective homomorphisms, \( h_2 \circ h \) is a surjective homomorphism and so \( g \) is too: if \( y \) is an object of \( A'_2 \), there exists an object \( x \) of \( A_1 \) such that
y = (h_2)_\rho(h_\rho(x))$, hence $g_\rho(x / \sim_L) = y$. If $g_\rho(x / \sim_L) = g_\rho(x' / \sim_L)$, then, from above, $x$ and $x'$ are indistinguishable and $x / \sim_L = x' / \sim_L$, so $g$ is injective.

Finally, it must be shown that $g$ is a transition system homomorphism. Let $t$ be a transition of $A_1$ and let $s = \alpha_1(t)$ (the proof is the same for $\beta$). Since $A'_1$ is the quotient of $A_1$,

$$\alpha'_1(t / \sim_L) = s / \sim_L$$

and

$$g_\sigma(s / \sim_L) = (h_2)_\sigma(h_\sigma(s)) = (h_2)_\sigma(h_\sigma(\alpha_1(t))) = (h_2)_\sigma(\alpha_2(h_\tau(t))) = \alpha'_2((h_2)_\tau(h_\tau(t))) = \alpha'_2(g_\tau(t / \sim_L)).$$

If the $A_i$ are labeled,

$$\chi'_2(g_\tau(t / \sim_L)) = \chi'_2((h_2)_\tau(h_\tau(t))) = \chi_1(t) = \chi'_1(t / \sim_L).$$

If the $A_i$ are parameterized, then for every state parameter name $X$,

$$s / \sim_L \in (S'_1)_X \iff s \in (S_1)_X \iff g_\sigma(s) \in (S'_2)_X,$$

and similarly for transition parameters. \qed

### 7.3.2 The diamond property

Let $\mathcal{H}$ be the family of all the surjective homomorphisms saturating a given adequate logic. It is clear that this family contains the identities and that it is closed under composition: if $h$ and $h'$ are in $\mathcal{H}$, then $h' \circ h$ is also in $\mathcal{H}$.

The family $\mathcal{H}$ satisfies the diamond property if for every pair $h' : A \to A'$ and $h'' : A \to A''$ of homomorphisms in $\mathcal{H}$, there exists a pair $g' : A' \to B$ and $g'' : A'' \to B$ of homomorphisms in $\mathcal{H}$ such that $g' \circ h' = g'' \circ h''$.

Define the symmetric relation $\approx_\mathcal{H}$ between two transition systems $A$ and $A'$ by $A \approx_\mathcal{H} A'$ if and only if there exists a transition system $A''$ and two homomorphisms $h : A \to A''$ and $h' : A' \to A''$ in $\mathcal{H}$.

**Proposition 7.2** If $\mathcal{H}$ satisfies the diamond property, then $\approx_\mathcal{H}$ is an equivalence relation.
Proof Since the identities are in $\mathcal{H}$, $A \approx_{\mathcal{H}} A$. Since $\approx_{\mathcal{H}}$ is symmetric by definition, it remains to show that it is transitive. Let $A_1 \approx_{\mathcal{H}} A_2$ and $A_2 \approx_{\mathcal{H}} A_3$. Then $h_1 : A_1 \to B_1$, $h_2 : A_2 \to B_1$, $h_3 : A_2 \to B_2$ and $h_4 : A_3 \to B_2$ exist. Since $\mathcal{H}$ satisfies the diamond property, there exist $C$ and $g_1 : B_1 \to C$, $g_2 : B_2 \to C$ such that $g_1$ and $g_2$ are in $\mathcal{H}$. But then $g_1 \circ h_1 : A_1 \to C$ and $g_2 \circ h_4 : A_3 \to C$ are also in $\mathcal{H}$ and so $A_1 \approx_{\mathcal{H}} A_3$.

Proposition 7.3 Two transition systems equivalent under $\approx_{\mathcal{H}}$ have the same quotient under indistinguishability, up to isomorphism.

Proof If there exist two surjective saturating homomorphisms, from $A_1$ and $A_2$ to $A_3$, then, according to theorem 7.3, their quotients under indistinguishability, $A_1'$, $A_2'$ and $A_3'$, are isomorphic.

Suppose that $\mathcal{H}$ satisfies the diamond property. A transition system $A$ is $\mathcal{H}$-irreducible if the image of $A$ under a homomorphism of $\mathcal{H}$ is isomorphic to $A$.

Proposition 7.4 Each equivalence class of $\approx_{\mathcal{H}}$ contains exactly one $\mathcal{H}$-irreducible transition system, up to isomorphism.

Proof Suppose that a class contains two irreducible transition systems $A$ and $A'$. Since $A \approx_{\mathcal{H}} A'$, there exist $A''$, and $h : A \to A''$ and $h' : A' \to A''$. Since $A$ and $A'$ are irreducible, they are both isomorphic to $A''$.

Consider now an equivalence class of transition systems, under relation $\approx_{\mathcal{H}}$, and consider a transition system $A$ of that class for which the sum of the number of states and the number of transitions is minimal. This is always possible since only finite transition systems are being considered. Let $h : A \to A'$ be a homomorphism in $\mathcal{H}$. Since $h$ is surjective, and the sum of the number of states and the number of transitions of $A'$ is the same as for $A$, homomorphism $h$ is necessarily injective, therefore is an isomorphism. Hence $A$ is irreducible.

Proposition 7.5 If $A'$ is an irreducible transition system equivalent under $\approx_{\mathcal{H}}$ to $A$, then $A'$ is the image of $A$ under a homomorphism of $\mathcal{H}$.

Since $A \approx_{\mathcal{H}} A'$, there exist $h : A \to A''$ and $h' : A' \to A''$. Since $A'$ is irreducible, $A''$ is isomorphic to $A'$ and so $A'$ is the image of $A$ under the composition of $h$ and that isomorphism.

It follows from proposition 7.5 and from theorem 7.3 that the canonical homomorphism from a transition system to its quotient under indistinguishability can be factored using the irreducible transition system equivalent to it.

Proposition 7.6 Let $h : A \to B$ be the homomorphism from $A$ to the irreducible transition system of its equivalence class under $\approx_{\mathcal{H}}$, and let $h_1 : A \to A'$ and $h_2 : B \to B'$ be the canonical homomorphisms from $A$ and $B$ to their quotient under indistinguishability. Then, up to isomorphism, $h_1 = h_2 \circ h$. 
7.3.3 Strongly adequate logics

Note that the canonical homomorphism from a transition system to its quotient under indistinguishability, for a given logic, is not necessarily in the family $\mathcal{H}$ of homomorphisms saturating that logic, and so it is not always true that the $\mathcal{H}$-irreducible transition system equivalent to a transition system is its quotient under indistinguishability. If such were the case, the set of formulas satisfied by an object $x$ of a transition system $A$ would be, in accordance with theorem 7.2, the same as the set of formulas satisfied by the image of $x$ under the quotient of $A$ under indistinguishability.

Example
Let $A$ be a transition system. It can be augmented with a single 0-ary operator $\text{Inf}$ interpreted by: $s \in \text{Inf}_A$ if and only if $s$ is the source of an infinite path. For the transition system $A_2$ having two states $s_1$ and $s_2$ and a single transition of source $s_1$ and target $s_2$, the operator $\text{Inf}_{A_2} = \emptyset$. The two states $s_1$ and $s_2$ are indistinguishable by $\text{Inf}$. The quotient under indistinguishability of $A_2$ is the transition system $A_1$ with one state $s_0$ and a single transition whose source and target are $s_0$. But $\text{Inf}_{A_2} = \{s_0\}$, hence $s_i$ does not satisfy $\text{Inf}$ in $A_2$ although its image $s_0$ satisfies it in the quotient $A_1$.

An adequate logic is strongly adequate if for every transition system $A$, the (surjective) canonical homomorphism from $A$ to its quotient under indistinguishability saturates that logic. It is then in $\mathcal{H}$.

Proposition 7.7 If a logic is strongly adequate, the family $\mathcal{H}$ of its surjective saturating homomorphisms satisfies the diamond property and the $\mathcal{H}$-irreducible transition system equivalent to a transition system $A$ under $\approx_{\mathcal{H}}$ is the quotient of $A$ under indistinguishability.

Proof If $h' : A \rightarrow A'$ and $h'' : A \rightarrow A''$ are in $\mathcal{H}$, from theorem 7.3, up to isomorphism, $g = g' \circ h' = g'' \circ h''$. If $A$ is irreducible, since the homomorphism that takes it to its quotient under indistinguishability is in $\mathcal{H}$, the quotient is isomorphic to it.

The strongly adequate logics can be characterized by the following property:

Proposition 7.8 Let $h : A \rightarrow A'$ be the surjective homomorphism from $A$ to its quotient under indistinguishability. Then $h$ is saturating if and only if for every formula $F$ of type $\rho$,

$$h^{-1}_\rho(F_A) = F_{A'}.$$

In that case, two objects of the quotient are always distinguishable.
Proof If $h$ is saturating, then $h^{-1}(F_{A'}) = F_A$, according to theorem 7.2. Conversely, suppose that for every formula $F$,

$$h^{-1}(F_{A'}) = F_A,$$

or,

$$x \in F_A \iff h_\rho(x) \in F_{A'}.$$

It is first shown that two objects of $A'$ are always distinguishable. If $A'$ contains two indistinguishable objects $y$ and $y'$, then, for every $F$,

$$y \in F_{A'} \iff y' \in F_{A'}.$$

Since $h$ is surjective, there exist $x$ and $x'$ in $A$ such that $y = h_\rho(x)$ and $y' = h_\rho(x')$. It follows that for every $F$,

$$x \in F_A \iff h_\rho(x) = y \in F_{A'}$$

$$\iff h_\rho(x') = y' \in F_{A'}$$

$$\iff x' \in F_A.$$

The objects $x$ and $x'$ are therefore indistinguishable and

$$y = h_\rho(x) = h_\rho(x') = y'.$$

It follows that for each object $y$ of $A'$, a characteristic formula $F_y$ can be defined so that $y = y'$ if and only if $y' \in (F_y)_{A'}$.

So, for each set $Y = \{y_1, \ldots, y_k\}$ of objects in $A'$, define the formula $F_Y = F_{y_1} \lor \cdots \lor F_{y_k}$. Then $y \in Y \iff y \in (F_Y)_{A'}$, and so $Y = (F_Y)_{A'}$.

Let $\omega$ be an operator of type $\rho_1 \cdots \rho_n \rightarrow \rho$ and let $Y_i$, $i = 1 \ldots n$, be a set of objects of $A'$ of type $\rho_i$. It must be shown that

$$h^{-1}_{\rho}(\omega_{A'}(Y_1, \ldots, Y_n)) = \omega_A(h^{-1}_{\rho_1}(Y_1), \ldots, h^{-1}_{\rho_n}(Y_n)).$$

Consider the formula $F = \omega(F_{Y_1}, \ldots, F_{Y_n})$. By assumption, $F_A = h^{-1}(F_{A'})$. But

$$F_A = \omega_A((F_{Y_1}), \ldots, (F_{Y_n}))$$

$$= \omega_A(h^{-1}_{\rho_1}((F_{Y_1})), \ldots, h^{-1}_{\rho_n}((F_{Y_n})))$$

$$= \omega_A(h^{-1}_{\rho_1}(Y_1), \ldots, h^{-1}_{\rho_n}(Y_n)).$$

Furthermore,

$$h^{-1}_{\rho}(F_{A'}) = h^{-1}_{\rho}(\omega_{A'}((F_{Y_1}), \ldots, (F_{Y_n})))$$

$$= h^{-1}_{\rho}(\omega_{A'}(Y_1, \ldots, Y_n)),$$

which is the required result. □
It was seen in section 7.2.1 that the extension of an adequate logic by operators definable as least and greatest fixpoints does not change the indistinguishability relation, nor the quotient under indistinguishability. It is shown here that the canonical homomorphism to the quotient also saturates this extended logic. It follows that the extension by fixpoint of a strongly adequate logic is still strongly adequate.

**Proposition 7.9** Let \( u(x, y) \) be an \( \Omega \)-term syntactically monotone over variable \( x \). Let \( \omega \) be a new operator of type \( \rho_1 \cdots \rho_n \rightarrow \rho \) such that for every transition system \( A \), and for every vector \( \vec{Y} = (Y_1, \ldots, Y_n) \) of sets \( Y_i \) of type \( \rho_i \), \( \omega_A(\vec{Y}) \) is the least solution of equation

\[
X = u_A(X, \vec{Y}),
\]

or of \( X = f_\omega(X) \), where \( f_\omega(X) = u_A(X, \vec{Y}) \). Let \( h : A \rightarrow A' \) be the canonical homomorphism from \( A \) to its quotient under indistinguishability. If \( F \) is a formula containing occurrences of \( \omega \), then \( h^{-1}_\rho(F_{A'}) = F_A \).

**Proof** The term \( u \) is defined by

- \( u_0 = 0_\rho \),
- \( u_{i+1} = u(u_i, \vec{Y}) \).

According to theorem 6.3, there exist \( k' \) and \( k'' \) such that for every \( k \geq k' \), \( (u_k)_A = \omega_A \), and for every \( k \geq k'' \), \( (u_k)_A = \omega_A \). Take \( k \) greater than \( k' \) and \( k'' \). Then

\[
h^{-1}_\rho(\omega_A(Y_1, \ldots, Y_n)) = h^{-1}_\rho((u_k)_A(Y_1, \ldots, Y_n)) = (u_k)_A \cdot (h^{-1}_{\rho_1}(Y_1), \ldots, h^{-1}_{\rho_n}(Y_n)),
\]

which is the desired result.

**Examples**

Hennessy-Milner logic and Dicky logic are shown here to be strongly adequate. It is already known that they are adequate. It suffices to show by induction over the construction of formulas that \( h^{-1}_\rho(F_{A'}) = F_A \).

1. Let \( A = (S, T, \alpha, \beta, \lambda) \) be a labeled transition system and, for its quotient, let \( A' = (S', T', \alpha', \beta', \lambda') \). Let \( h \) be the homomorphism from \( A \) to \( A' \). It suffices to show that

\[
h^{-1}_\rho((\langle a \rangle F)_A) = (\langle a \rangle F)_A,
\]

knowing that \( h^{-1}(F_{A'}) = F_A \).

If \( s \in h^{-1}_\rho((\langle a \rangle F)_A) \) then \( h_\sigma(s) \in (\langle a \rangle F)_{A'} \) and so there exists in \( A' \) a transition \( \tilde{t}' = h_\sigma(s) \mapsto a \rightarrow s'_1 \), with \( s'_1 \in F_{A'} \). Since \( h \) is surjective there exists in \( A \)
a transition \( t = s' \mapsto a \rightarrow s_1 \), with \( h_\tau(t) = t' \), from which \( h_\sigma(s) = h_\sigma(s') \) and \( h_\sigma(s_1) = s_1' \). Since \( s_1' \in F_{A'} \), \( s_1 \in F_A \) and hence \( s' \in ((a)F)_A \). But since \( h_\sigma(s) = h_\sigma(s') \), \( s \) and \( s' \) are indistinguishable, hence \( s \in ((a)F)_A \).

Conversely, if \( s \in ((a)F)_A \), there exists in \( A \) a transition \( t = s \mapsto a \rightarrow s' \) such that \( s' \in F_A \). The transition \( h_\tau(t) = h_\sigma(s) \mapsto a \rightarrow h_\sigma(s') \) is in \( A' \) and, since \( s' \in F_A \) implies \( h_\sigma(s') \in F_{A'} \), the state \( h_\sigma(s) \in ((a)F)_{A'} \).

2. A similar result can be shown for Dicky logic. The result is only shown for formulas of the form \( \text{src}(G) \) and \( \text{in}(F) \), the proof being the same for formulas \( \text{tgt}(G) \) and \( \text{out}(F) \).

- It is given that \( (\text{src}(G))_A = \alpha(G_A) \) and \( (\text{src}(G))_{A'} = \alpha'(G_{A'}) \). But, since \( G_A = h^{-1}(G_{A'}) \) and \( h \) is surjective, \( h_\tau(G_A) = G_{A'} \) and so

\[
(\text{src}(G))_{A'} = \alpha'(G_{A'}) \\
\quad = \alpha'(h_\tau(G_A)) \\
\quad = h_\sigma(\alpha'(G_{A'})) \\
\quad = h_\sigma((\text{src}(G))_A).
\]

It follows that if \( s \in (\text{src}(G))_A \), then \( h_\sigma(s) \in (\text{src}(G))_{A'} \).

Conversely, if \( h_\sigma(s) \in (\text{src}(G))_{A'} \), there exists \( s' \in (\text{src}(G))_A \) such that \( h_\sigma(s) = h_\sigma(s') \). Since \( s \) and \( s' \) are indistinguishable, \( s \in (\text{src}(G))_A \).

- It is also given that \( (\text{in}(F))_A = \beta^{-1}(F_A) \) and \( (\text{in}(F))_{A'} = \beta'^{-1}(F_{A'}) \). Since \( \beta' \circ h_\tau = h_\sigma \circ \beta \), \( h_\tau^{-1} \circ \beta'^{-1} = \beta^{-1} \circ h_\sigma^{-1} \). Hence

\[
h_\tau^{-1}((\text{in}(F))_{A'}) = h_\tau^{-1}(\beta'^{-1}(F_{A'})) \\
\quad = \beta^{-1}(h_\sigma^{-1}(F_{A'})) \\
\quad = \beta^{-1}(F_A) \\
\quad = (\text{in}(F))_A.
\]
The major problem for the semantics of systems of processes, and the basis for much of the literature, is that of their equivalence. Given two systems, written in a particular formalism (CSP, CCS, \ldots), can they be considered to be two different ways of writing the same system? This question actually has two parts:

- What are the criteria to be taken into consideration to affirm that two systems are equivalent?
- How is this equivalence relation between systems defined formally?

The first of these questions is the fundamental question of the semantics of systems of processes. Two systems are equivalent if they have the same semantics, i.e. from the ‘text’ describing them (the syntax), the same information—judged relevant—can be derived from their ‘behavior’ (the semantics). The definition of an equivalence relation therefore presupposes a semantics. Conversely an equivalence relation between systems implicitly induces a semantics: the relevant information about the behavior of a system is precisely that which is common to all equivalent systems. The semantics of a system can then be defined as its equivalence class. This way of defining semantics is standard in mathematics and computer science. Consider the case of integer arithmetic expressions. The semantics of such an expression can be defined as the ‘value’ obtained by ‘computing’ it. Hence the semantics of $2 \times (3 + 4)$ is 14. Two expressions are equivalent if they have the same value: $2 \times (3 + 4)$ and $2 + 3 \times 4$ are two equivalent expressions. Conversely an equivalence relation can be defined between expressions and the ‘value’ of an expression can be defined to be its equivalence class. If each equivalence class contains exactly one canonical element (here the canonical element is the expression formed of a single integer), that element can be used as a value instead of the class, since there is a bijection between the classes and their canonical representatives. A less trivial example is that of Cauchy sequences in a metric space; the semantics of a sequence is its limit, but if the space is not complete this limit might not exist; an equivalence relation is then defined between sequences: two sequences
are equivalent if the distance between two elements of an arbitrarily large index of each sequence is arbitrarily small. The limit of a sequence is then defined as its equivalence class and that is how metric spaces are completed. As for normal deterministic sequential programs, the semantics of a program is the partial function taking the data to the results of the program; but the concept of equivalent programs can be defined a priori (for example two programs are equivalent if a sequence of elementary transformations can change one into the other), thereby inducing a semantics. In the case of these programs it is easy to define what its semantics must be, or the equivalence relation between programs, but this is not the case for systems of processes, where it is possible to define several equivalences without one appearing to be more natural than another; the choice of a 'good' equivalence is based on pragmatic considerations.

As for the formal definition of equivalences, the syntax in which a system of processes is written clearly plays an important rôle and formal definitions of equivalence are given for each description language. However, since it is given here that systems of processes can be written using transition systems, only equivalences of transition systems are examined.

8.1 Equivalences induced by saturating homomorphisms

One natural equivalence concept is to compare quotients under particular kinds of homomorphism that capture the essential properties of transition systems.

The preceding chapter defined such equivalences, starting from saturating homomorphisms, under the condition that the family of saturating homomorphisms satisfies the diamond property.

This definition can be generalized, by making it independent of the logic, as follows: let \( \mathcal{H} \) be a family of homomorphisms such that:

- \( \mathcal{H} \) contains the identity mapping from every transition system to itself.
- \( \mathcal{H} \) is closed under composition.
- \( \mathcal{H} \) satisfies the diamond property: for every pair of homomorphisms of \( \mathcal{H} \), \( h' : A \to A' \) and \( h'' : A \to A'' \), there exists a pair \( g' : A' \to B \) and \( g'' : A'' \to B \) of homomorphisms of \( \mathcal{H} \) such that \( g' \circ h' = g'' \circ h'' \).

Then the relation \( \approx_{\mathcal{H}} \), defined by \( A_1 \approx_{\mathcal{H}} A_2 \) if and only if there exists a transition system \( A_3 \) and two surjective homomorphisms \( h_1 : A_1 \to A_3 \) and \( h_2 : A_2 \to A_3 \), is itself an equivalence relation and each equivalence class contains exactly one irreducible element.

As an example, it is shown below that the family of all the (simple, labeled or parameterized) homomorphisms has these properties, in particular the diamond property, since the other properties are immediate.

Consider two surjective homomorphisms \( h_1 : A \to A_1 \) and \( h_2 : A \to A_2 \), where \( A = (S, T, \alpha, \beta) \). For homomorphism \( h_i \) define the equivalence relations \( \approx_{E_i} \) and
\( \sim^i \) respectively over \( S \) and \( T \) by \( s \sim^i_s s' \) if and only if \((h_i)_{\sim}(s) = (h_i)_{\sim}(s')\) and by \( t \sim^i_t t' \) if and only if \((h_i)_t(t) = (h_i)_t(t')\). Let \( \sim_{\sigma} \) be the least equivalence relation containing \( \sim^1_{\sigma} \) and \( \sim^2_{\sigma} \) and let \( \sim_{\tau} \) be the least equivalence relation containing \( \sim^1_{\tau} \) and \( \sim^2_{\tau} \), and \( A' = \langle S', T', \alpha', \beta' \rangle \), where

- \( S' = S/\sim_{\sigma} \);
- \( T' = T/\sim_{\tau} \);
- \( \alpha'(t/\sim_{\tau}) = \alpha(t)/\sim_{\sigma} \);
- \( \beta'(t/\sim_{\tau}) = \beta(t)/\sim_{\sigma} \).

This definition is meaningful since the definition of \( \alpha' \) and \( \beta' \) is independent of the choice of \( t \) in its equivalence class under \( \sim_{\tau} \). In particular if \( t \sim_{\tau} t' \), there exists

\[ t = t_1, t_2, \ldots, t_m = t', \]

with \( t_i \sim^i_{\tau} t_{i+1} \), and so \( \alpha(t_i) \sim^i_{\sigma} \alpha(t_{i+1}) \) and \( \beta(t_i) \sim^i_{\sigma} \beta(t_{i+1}) \), hence \( \alpha(t) \sim_{\sigma} \alpha(t') \) and \( \beta(t) \sim_{\sigma} \beta(t') \). It also follows that the mapping \( h = (h_{\sigma}, h_{\tau}) \), where \( h_{\sigma}(s) = s/\sim_{\sigma} \) and \( h_{\tau}(t) = t/\sim_{\tau} \), is a surjective homomorphism. Let \( h_i' = ((h_i')_{\sigma}, (h_i')_{\tau}) \), where \( (h_i')_{\sigma}(h_i)_{\sigma}(s) = s/\sim_{\sigma} \) and \( (h_i')_{\tau}(h_i)_{\tau}(t) = t/\sim_{\tau} \). It is clear that \( h_i' \) is still a surjective homomorphism and that \( h = h_i' \circ h_i \).

If in addition \( A \) is a labeled transition system, then

\( t \sim_{\tau} t' \Rightarrow \lambda(t) = \lambda(t') \),

since

\[ t \sim^i_{\tau} t' \Rightarrow (h_i)_t(t) = (h_i)_{t'}(t') \Rightarrow \lambda(t) = \lambda(t') = \lambda_t((h_i)_t(t)). \]

By defining \( \lambda'(t/\sim_{\tau}) = \lambda(t) \), \( A' \) becomes a labeled transition system and so \( \lambda'(h_{\tau}(t)) = \lambda(t) \). So \( h \) and \( h_i' \) become labeled transition system homomorphisms.

If \( A \) is a transition system parameterized by \( (X, Y) \), then \( A' \) is also a transition system parameterized by \( (X, Y) \), by letting \( S'_X = S_X/\sim_{\sigma} \) and \( T'_Y = T_Y/\sim_{\tau} \). The homomorphisms \( h \) and \( h_i \) are then parameterized transition system homomorphisms.

The equivalence relation induced by the family of all homomorphisms is very coarse. To see this, it suffices to examine the transition systems irreducible under the equivalence.

In the case of simple transition systems (no labels or parameters), there are only two:

- the transition system with one state and no transitions; and
- the transition system with one state and one transition.

The first of these transition systems is the homomorphic image of every transition system without transitions; the second is the homomorphic image of every transition system having at least one transition.

In the case of labeled transition systems, with each (possibly empty) subset \( B \) of the alphabet \( A \) is associated the labeled transition system with one state and
containing, for each letter \( a \) in \( B \), exactly one transition labeled \( a \) (hence if \( B \) is empty, this transition system has no transition). This transition system is the homomorphic image of every transition system for which the set of transition labels is \( B \) (i.e. \( B = \lambda(T) \)).

In the case of transition systems parameterized \((\mathcal{X}, \mathcal{Y})\), given a parameterized transition system

\[
\mathcal{A} = \langle S, T, \alpha, \beta, S_{X_1}, \ldots, S_{X_n}, T_{Y_1}, \ldots, T_{Y_m} \rangle,
\]

the equivalent irreducible parameterized transition system is defined in the following manner. For each subset \( \mathcal{X}' \) of \( \mathcal{X} \), let \( S_{\mathcal{X}'} \) be the set of states \( s \) of \( S \) such that

\[
\forall X \in \mathcal{X}', s \in S_X \text{ and } \forall X \notin \mathcal{X}', s \notin S_X.
\]

Note that if \( \mathcal{X}' \neq \mathcal{X}'' \) then \( S_{\mathcal{X}'} \cap S_{\mathcal{X}''} = \emptyset \). Then let

\[
S' = \{ \mathcal{X}' \subseteq \mathcal{X} \mid S_{\mathcal{X}'} \neq \emptyset \}
\]

and

\[
h_s(s) = \mathcal{X}' \iff s \in S_{\mathcal{X}'}.
\]

Also let \( S_X' = \{ \mathcal{X}' \mid X \in \mathcal{X}' \} \). To define \( T' \), given a triplet \( \langle \mathcal{X}', \mathcal{Y}', \mathcal{X}'' \rangle \) where \( \mathcal{X}' \) and \( \mathcal{X}'' \) are in \( S' \) and where \( \mathcal{Y}' \) is a subset of \( \mathcal{Y} \), define the set of transitions

\[
T_{(\mathcal{X}', \mathcal{Y}', \mathcal{X}'')} = \left\{ t \in T \mid \alpha(t) \in S_{\mathcal{X}'}, \beta(t) \in S_{\mathcal{X}''}, \forall Y \in \mathcal{Y}'(t \in T_Y), \forall Y \notin \mathcal{Y}'(t \notin T_Y) \right\}.
\]

Then let

- \( T' \) be equal to the set of triplets such that the associated set of transitions is not empty;
- \( \alpha'(\langle \mathcal{X}', \mathcal{Y}', \mathcal{X}'' \rangle) = \mathcal{X}' \);
- \( \beta'(\langle \mathcal{X}', \mathcal{Y}', \mathcal{X}'' \rangle) = \mathcal{X}'' \);
- \( T'_Y = \{ \langle \mathcal{X}', \mathcal{Y}', \mathcal{X}'' \rangle \mid Y \in \mathcal{Y}' \} \).

8.2 Bisimulation

The concept of bisimulation, due to Park [73], and used by Milner [54, 66, 67], is without question the simplest way to define the equivalence of two transition systems. In particular, it is shown below that this equivalence is none other than the \( \mathcal{H} \)-equivalence of Hennessy–Milner logic, which is strongly adequate, as was seen on page 128.

Let \( \mathcal{A}_1 = < S_1, T_1, \alpha_1, \beta_1, \lambda_1 > \) and \( \mathcal{A}_2 = < S_2, T_2, \alpha_2, \beta_2, \lambda_2 > \) be two transition systems labeled by the same alphabet \( A \). A **bisimulation** between \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) is a binary relation \( R \) between \( S_1 \) and \( S_2 \) such that
For every state $s_1$ in $S_1$, there exists $s_2$ in $S_2$ such that $s_1 R s_2$.

For every state $s_2$ in $S_2$, there exists $s_1$ in $S_1$ such that $s_1 R s_2$.

For every transition $t_1$ in $T_1$ and for every state $s_2$ in $S_2$ such that $\alpha_1(t_1) R s_2$, there exists $t_2$ in $T_2$ such that $s_2 = \alpha_2(t_2)$, $\lambda_1(t_1) = \lambda_2(t_2)$ and $\beta_1(t_1) R \beta_2(t_2)$.

For every transition $t_2$ in $T_2$ and for every state $s_1$ in $S_1$ such that $s_1 R \alpha_2(t_2)$, there exists $t_1$ in $T_1$ such that $s_1 = \alpha_1(t_1)$, $\lambda_1(t_1) = \lambda_2(t_2)$ and $\beta_1(t_1) R \beta_2(t_2)$.

A relation $R$ that only satisfies conditions (i a) and (ii a) is a simulation of $A_1$ by $A_2$. If $s_1 R s_2$ then $s_2$ is a state that simulates $s_1$ since for every transition $t_1$ of source $s_1$ and label $a$ there exists a transition $t_2$ of source $s_2$ and label $a$ whose target also simulates the target of $t_1$. If $R$ is a bisimulation, $s_1 R s_2$ means that $s_1$ simulates $s_2$, and conversely.

**Example**

Consider the following two labeled transition systems graphically represented in Figure 8.1 (page 153):

- $A$:
  
  \[
  t_1 : 1 \xrightarrow{a} 2,
  t_2 : 1 \xrightarrow{a} 3,
  t_3 : 2 \xrightarrow{b} 4,
  t_4 : 3 \xrightarrow{c} 5.
  \]

- $A'$:

  \[
  t'_1 : 1' \xrightarrow{a} 2',
  t'_2 : 2' \xrightarrow{b} 3',
  t'_3 : 2' \xrightarrow{c} 4'.
  \]

Even though these two systems can execute exactly the same sequences of actions, Milner [54, 66] does not consider them equivalent, since in state $2'$ of the second, $b$ or $c$ can be executed, while in no state of the first is it possible to execute both actions. It is that intuitive idea that bisimulation formalizes, and it turns out that $A$ and $A'$ are not in bisimulation.

In fact, if a bisimulation existed between those two transition systems, intuitively, it would necessarily be the following relation:

\[
1R1', 2R2', 2R2', 4R3', 5R4'.
\]

But this relation cannot be a bisimulation, for although it satisfies conditions (i a), (i b) and (ii a), it does not satisfy condition (ii b). To show this, it suffices to consider transition $t'_2$ and state 3. This state is in relation $R$ with state $2'$, which is the source of $t'_2$, but it is the source of no transition labeled $b$, which is the label of $t'_2$.  

8.2.1 A few properties

A few obvious properties of the bisimulation relation are given here.

**Proposition 8.1**

- If $R$ is a bisimulation relation between $\mathcal{A}$ and $\mathcal{A}'$, $R^{-1}$ is a bisimulation relation between $\mathcal{A}'$ and $\mathcal{A}$.
- If $R$ is a bisimulation relation between $\mathcal{A}$ and $\mathcal{A}'$ and if $R'$ is a bisimulation relation between $\mathcal{A}'$ and $\mathcal{A}''$, then the composition $R \cdot R'$ is a bisimulation relation between $\mathcal{A}$ and $\mathcal{A}''$.
- If $(R_i)_{i \in I}$ is a family of bisimulations between $\mathcal{A}$ and $\mathcal{A}'$, then the relation $R = \bigcup_{i \in I} R_i$ is a bisimulation relation between $\mathcal{A}$ and $\mathcal{A}'$.

**Proof** The first point follows immediately from the symmetry of the definition. The second point follows from these remarks:

- If $s \in S$, there exists $s' \in S'$ such that $sRs'$; there also exists $s'' \in S''$ such that $s'R's''$, and hence $sRs''$. For the same reasons, if $s'' \in S''$, there exists $s \in S$ such that $sR's''$.
- If $t$ is a transition such that $\alpha(t)R \cdot R's''$, then there exists a state $s'$ such that $\alpha(t)Rs'$ and $s'R's''$. Therefore there exists a transition $t'$ such that

\[ s' = \alpha'(t), \quad \lambda(t) = \lambda'(t'), \quad \beta(t)R\beta'(t'); \]

there also exists a transition $t''$ such that

\[ s'' = \alpha''(t''), \quad \lambda'(t') = \lambda''(t''), \quad \beta'(t')R\beta''(t''). \]

Hence $s'' = \alpha''(t''), \lambda(t) = \lambda''(t'')$ and $\beta(t)R \cdot R'\beta''(t'')$. The symmetric property is shown in the same manner.

For the third point, let $R = \bigcup_{i \in I} R_i$, and note:

- If $s \in S$, there exists $s' \in S'$ such that $sRs'$, hence $sRs'$.
- If $\alpha(t)Rs'$, there also exists $i$ such that $\alpha(t)R_is'$, hence there exists $t'$ such that $s' = \alpha'(t')$, $\lambda(t) = \lambda'(t')$ and $\beta(t)R_i\beta'(t')$, and so $\beta(t)R\beta'(t')$. \(\Box\)

In particular it follows that the relation, 'there exists a bisimulation relation between $\mathcal{A}$ and $\mathcal{A}'$' is an equivalence relation.

It will now be shown that there exists a link between bisimulation and the equivalence $\approx_\mathcal{H}$ induced by the family of surjective homomorphisms saturating Hennessy–Milner logic. This link was shown by Castellani [23] in certain cases, and by Arnold and Dicky [8] and independently by Ferrari and Montanari [46] in the general case.

To do this, the homomorphisms introduced in [8], known as *transition preserving homomorphisms* in [31, 45, 46], must be characterized.
Proposition 8.2 A labeled transition system homomorphism

\[ h : \mathcal{A} = (S, T, \alpha, \beta, \lambda) \rightarrow \mathcal{A}' = (S', T', \alpha', \beta', \lambda') \]

saturates Hennessy–Milner logic if and only if for every state \( s_1 \in S \) and for every transition \( t' = h_\sigma(s_1) \mapsto a \rightarrow s'_2 \in T' \), there exists \( t = s_1 \mapsto a \rightarrow s_2 \in T \) such that \( h_\sigma(s_2) = s'_2 \).

Proof The homomorphism \( h \) saturates Hennessy–Milner logic if and only if for every letter \( a \) and for every subset \( X \) of \( S' \),

\[ h_\sigma^{-1}(\langle a \rangle_{\mathcal{A}'}(X)) = \langle a \rangle_{\mathcal{A}}(h_\sigma^{-1}(X)). \]

Since the mappings \( h_\sigma^{-1}, \langle a \rangle_{\mathcal{A}} \) and \( \langle a \rangle_{\mathcal{A}'} \) are additive,

\[ h_\sigma^{-1}(X) = \bigcup_{s' \in X} h_\sigma^{-1}(s'), \]
\[ \langle a \rangle_{\mathcal{A}}(h_\sigma^{-1}(X)) = \bigcup_{s' \in X} \langle a \rangle_{\mathcal{A}}(h_\sigma^{-1}(s')), \]
\[ \langle a \rangle_{\mathcal{A}'}(X) = \bigcup_{s' \in X} \langle a \rangle_{\mathcal{A}'}(s'), \]
\[ h_\sigma^{-1}(\langle a \rangle_{\mathcal{A}'}(X)) = \bigcup_{s' \in X} h_\sigma^{-1}(\langle a \rangle_{\mathcal{A}'}(s')). \]

The fact that a homomorphism is saturating can be written

\[ \forall s' \in S', \ h_\sigma^{-1}(\langle a \rangle_{\mathcal{A}'}(s')) = \langle a \rangle_{\mathcal{A}}(h_\sigma^{-1}(s')), \]

or

\[ \forall s' \in S', \forall s \in S, \ h_\sigma(s) \in \langle a \rangle_{\mathcal{A}'}(s') \Leftrightarrow s \in \langle a \rangle_{\mathcal{A}}(h_\sigma^{-1}(s')). \]

But \( h_\sigma(s) \in \langle a \rangle_{\mathcal{A}'}(s') \) if and only if there exists \( t' = h_\sigma(s) \mapsto a \rightarrow s' \in T' \) and \( s \in \langle a \rangle_{\mathcal{A}}(h_\sigma^{-1}(s')) \) if and only if there exists \( t = s \mapsto a \rightarrow s'' \in T \), with \( h_\sigma(s'') = s' \). Since \( h \) is a homomorphism, it is always the case that if \( t = s \mapsto a \rightarrow s'' \in T \) then \( t' = h_\sigma(t) = h_\sigma(s) \mapsto a \rightarrow h_\sigma(s'') \in T' \). It follows that \( h \) is saturating if and only if \( \forall s \in S, \forall t' = h_\sigma(s) \mapsto a \rightarrow s' \in T' \), \( \exists t = s \mapsto a \rightarrow s'' \in T : h_\sigma(s'') = s' \), which is exactly the sought characterization.

The next proposition follows immediately.

Proposition 8.3 If a surjective labeled transition system homomorphism

\[ h : \mathcal{A} = (S, T, \alpha, \beta, \lambda) \rightarrow \mathcal{A}' = (S', T', \alpha', \beta', \lambda') \]
saturates Hennessy–Milner logic, then the relation

\[ R = \left\{ (s, h_\sigma(s)) \mid s \in S \right\} \]

is a bisimulation relation.
Proof If \( s \in S \), then \( s R h_\sigma(s) \). Conversely, since \( h \) is surjective, for every \( s' \in S' \), there exists \( s \in S \) such that \( s' = h_\sigma(s) \) and \( s R s' \).

If \( s_1 R s'_1 \), then \( s'_1 = h_\sigma(s_1) \). If \( s_1 \mapsto a \rightarrow s_2 \in T \), then \( s'_1 = h_\sigma(s_1) \mapsto a \rightarrow h_\sigma(s_2) \in T' \) and \( s_2 R h_\sigma(s_2) \). If \( h_\sigma(s_1) \mapsto a \rightarrow s'_2 \in T' \), there exists \( s_1 \mapsto a \rightarrow s_2 \in T \), with \( s'_2 = h_\sigma(s_2) \), hence \( s_2 R s'_2 \).

The next result follows from the fact that a family of bisimulation relations is closed under product and inversion.

**Proposition 8.4** If \( \mathcal{A} \) and \( \mathcal{A}' \) are labeled transition systems such that \( \mathcal{A} \cong_{\mathcal{H}} \mathcal{A}' \), then there exists a bisimulation between \( \mathcal{A} \) and \( \mathcal{A}' \).

This bisimulation relation is \( h_\sigma \cdot (h')^{-1}_\sigma \), where \( h \) and \( h' \) are the two saturating surjective homomorphisms taking \( \mathcal{A} \) and \( \mathcal{A}' \) to the same image.

This proposition has a converse.

**Proposition 8.5** If there exists a bisimulation relation \( R \) between two labeled transition systems \( \mathcal{A} = \langle S, T, \alpha, \beta, \lambda \rangle \) and \( \mathcal{A}' = \langle S', T', \alpha', \beta', \lambda' \rangle \), then there exists a labeled transition system \( \mathcal{B} \) and two surjective homomorphisms, saturating over Hennessy–Milner logic, \( h : \mathcal{A} \rightarrow \mathcal{B} \) and \( h' : \mathcal{A}' \rightarrow \mathcal{B} \), such that \( R \subseteq h_\sigma \cdot (h')^{-1}_\sigma \).

**Proof** Construct the labeled transition system

\[
\mathcal{A}'' = \langle S'', T'', \alpha'', \beta'', \lambda'' \rangle
\]

defined by

\[
\begin{align*}
\cdot & \quad S'' = \{(s, s') \mid s R s'\}; \\
\cdot & \quad T'' = \{(t, t') \mid \lambda(t) = \lambda'(t'), \alpha(t) R \alpha'(t'), \beta(t) R \beta'(t')\}; \\
\cdot & \quad \lambda''((t, t')) = \lambda(t) = \lambda'(t'); \\
\cdot & \quad \alpha''((t, t')) = \langle \alpha(t), \alpha'(t') \rangle; \\
\cdot & \quad \beta''((t, t')) = \langle \beta(t), \beta'(t') \rangle.
\end{align*}
\]

Since \( \langle t, t' \rangle \in T'' \) implies \( \alpha(t) R \alpha'(t') \) and \( \beta(t) R \beta'(t') \), \( \alpha''((t, t')) \) and \( \beta''((t, t')) \) are in \( S'' \). Consider then the two projections \( \pi = (\pi_\sigma, \pi_\tau) \) and \( \pi' = (\pi'_\sigma, \pi'_\tau) \) defined by

\[
\begin{align*}
\cdot & \quad \pi_\sigma((s, s')) = s; \\
\cdot & \quad \pi'_\sigma((s, s')) = s', \\
\cdot & \quad \pi_\tau((t, t')) = t; \\
\cdot & \quad \pi'_\tau((t, t')) = t'.
\end{align*}
\]

These mappings should be surjective homomorphisms saturating Hennessy–Milner logic. The proof is only given for \( \pi \), the \( \pi' \) case being similar because of the definition of \( \mathcal{A}'' \).
\* \( \pi \) is a homomorphism. By definition,

\[
\lambda''(\langle t, t' \rangle) = \lambda(t) = \lambda(\pi'_T(\langle t, t' \rangle)),
\]

\[
\pi_\sigma(\alpha''(\langle t, t' \rangle)) = \pi_\sigma(\langle \alpha(t), \alpha'(t') \rangle) = \alpha(t) = \alpha(\pi_T(\langle t, t' \rangle)),
\]

\[
\pi_\sigma(\beta''(\langle t, t' \rangle)) = \pi_\sigma(\langle \beta(t), \beta'(t') \rangle) = \beta(t) = \beta(\pi_T(\langle t, t' \rangle)).
\]

\* \( \pi \) is surjective. If \( s \in S \), there exists \( s' \in S' \) such that \( sRs' \) and so \( \langle s, s' \rangle \in S'' \), with \( s = \pi_\sigma(\langle s, s' \rangle) \). If \( t = s_1 \mapsto a \mapsto s_2 \in T \), there exists \( s'_1 \in S' \) such that \( s_1Rs'_1 \); there also exists \( t' = s'_1 \mapsto a \mapsto s'_2 \in T' \) such that \( s_2Rs'_2 \), hence \( \langle t, t' \rangle \in T'' \).

\* \( \pi \) is saturating over Hennessy–Milner logic. Let \( \langle s_1, s'_1 \rangle \in S'' \) and \( t = s_1 \mapsto a \mapsto s_2 \in T \). Since \( s_1Rs'_1 \), there exists \( t' = s'_1 \mapsto a \mapsto s'_2 \in T' \), with \( s_2Rs'_2 \), and so \( \langle t, t' \rangle = \langle s_1, s'_1 \rangle \mapsto a \mapsto \langle s_2, s'_2 \rangle \in T'' \). Therefore \( \pi_\sigma(\langle s_2, s'_2 \rangle) = s_2 \).

Since Hennessy–Milner logic is strongly adequate, the family of surjective homomorphisms saturating over this logic satisfies the diamond property. There therefore exist \( h : A \to B \) and \( h' : A' \to B \) such that \( h \circ \pi = h' \circ \pi' \). If \( sRs' \), then \( \langle s, s' \rangle \in S'' \) and \( \pi_\sigma(s) = \pi_\sigma(\langle s, s' \rangle) = h'_\sigma(\pi'_\sigma(\langle s, s' \rangle)) = h'_\sigma(s') \), i.e. \( \langle s, s' \rangle \in h_\sigma \cdot h'_\sigma^{-1} \). \( \Box \)

These two propositions can be combined.

**Theorem 8.1** There exists a bisimulation relation between two labeled transition systems if and only if these two transition systems are equivalent under the equivalence \( \approx_\{\} \) induced by the family of surjective homomorphisms saturating Hennessy–Milner logic.

### 8.2.2 Greatest bisimulation

Since the arbitrary union of bisimulation relations between \( A \) and \( A' \) is still a bisimulation relation, if there exists a bisimulation relation between two transition systems, there also exists a greatest bisimulation.

That greatest bisimulation is characterized by the following theorem, known as the Hennessy–Milner theorem.

**Theorem 8.2** Let \( A \) and \( A' \) be two labeled transition systems. Let \( R \) be the included relation \( S \times S' \) defined by \( sRs' \) if and only if for every Hennessy–Milner formula \( F \), \( s \in F_A \Leftrightarrow s' \in F_{A'} \). Then the three following properties are equivalent:

(i) \( \forall s \in S, \exists s' \in S' : sRs' \) and \( \forall s' \in S' \after\exists s \in S : sRs' \).

(ii) There exists a bisimulation relation between \( A \) and \( A' \).

(iii) \( R \) is the greatest bisimulation between \( A \) and \( A' \).
Proof. The implication (iii) \( \Rightarrow \) (i) is an immediate consequence of the definition of a bisimulation relation.

To show the implication (i) \( \Rightarrow \) (ii), R is shown to be a bisimulation. Since (i) is none other than the conjunction of conditions (i a) and (i b) of the definition of a bisimulation relation, only (ii a) and (ii b) remain to be shown. For reasons of symmetry only the first of the two conditions is proven. Consider then \( s_1Rs_1' \) and a transition \( t' = s_1' \rightarrow a \rightarrow s_2' \) of \( T' \). Let \( F \) be a characteristic formula of the indistinguishability class of \( s_2' \). Then \( s_1' \in (\langle a \rangle F)_{A'} \) and, since \( s_1Rs_1', s_1 \in (\langle a \rangle F)_{A} \). There therefore exists a transition \( s_1 \rightarrow a \rightarrow s_2 \), with \( s_2 \in F_A \). At the same time, \( s_2Rs_2' \) must hold. If this were not the case, there would exist a formula \( F' \) such that \( s_2 \in F_A' \) and \( s_2' \notin F_A' \). Then \( s_1 \in (\langle a \rangle (F \cap F'))_{A} \) would hold and therefore, since \( s_1Rs_1', s_1 \in (\langle a \rangle (F \cap F'))_{A} \). There would therefore exist in \( A' \) a transition \( s_1' \rightarrow a \rightarrow s' \), with \( s' \in (F \cap F')_{A'} \). Since \( s' \in F_A' \) and \( s_2' \notin F_A' \), \( s' \) and \( s_2' \) would not be indistinguishable, and since \( s' \) and \( s_2' \) are both in \( F_A' \), \( F \) would not be a characteristic formula of the indistinguishability class of \( s_2' \), which is a contradiction.

Last is (ii) \( \Rightarrow \) (iii). Let \( R' \) be a bisimulation relation between \( A \) and \( A' \). According to proposition 8.5 there exist two surjective homomorphisms that saturate Hennessy–Milner logic, \( h : A \rightarrow B \) and \( h' : A' \rightarrow B \), such that \( R' \subseteq h_{\sigma} \cdot h'_{\sigma}^{-1} \).

According to theorem 7.3, the three transition systems \( A, A' \) and \( B \) have the same quotient \( D \) under indistinguishability. Let \( g \) be the canonical homomorphism from \( B \) to the quotient. The relation \( h_{\sigma} \cdot h'_{\sigma} s^{-1} \) is itself included in \( h_{\sigma} \cdot g_{\sigma} \cdot g_{\sigma}^{-1} \cdot h'_{\sigma} s^{-1} \). Since Hennessy–Milner logic is strongly adequate, the homomorphism \( g \) saturates this logic and the last relation is therefore a bisimulation. It remains to show that it is equal to \( R \). Now, \( g(s, s') \in h_{\sigma} \cdot g_{\sigma} \cdot g_{\sigma}^{-1} \cdot h'_{\sigma} s^{-1} \) if and only if \( g_{\sigma}(h_{\sigma}(s)) = g_{\sigma}(h'_{\sigma}(s')) \).

Since \( g \circ h \) and \( g \circ h' \) are saturating, \( s \in F_A \) if and only if \( g_{\sigma}(h_{\sigma}(s)) \in F_B \) and \( s' \in F_A' \) if and only if \( g_{\sigma}(h'_{\sigma}(s')) \in F_B' \). Hence \( g_{\sigma}(h_{\sigma}(s)) = g_{\sigma}(h'_{\sigma}(s')) \) implies \( s \in F_A \) if and only if \( s' \in F_A' \), for every formula \( F \), and so \( sRs' \). Conversely, if \( sRs' \), then \( g_{\sigma}(h_{\sigma}(s)) \in F_B \) if and only if \( g_{\sigma}(h'_{\sigma}(s')) \in F_B' \), for every formula \( F \). Since \( D \) is, according to proposition 7.8, a transition system where each object is distinguishable from another, this implies \( g_{\sigma}(h_{\sigma}(s)) = g_{\sigma}(h'_{\sigma}(s')) \).

An important application of this theorem is that if \( R \) is a bisimulation and \( sRs' \) then for every Hennessy–Milner formula \( F \), \( s \in F_A \) if and only if \( s' \in F_A' \). Furthermore, combined with theorem 8.1, it gives another characterization of the equivalence \( \approx_{H} \) induced by the family of surjective homomorphisms saturating Hennessy–Milner logic.

Theorem 8.3 Let \( A \) and \( A' \) be two labeled transition systems.

\[ A \approx_{H} A' \]

if and only if for every Hennessy–Milner formula \( F \),

\[ F_A \neq \emptyset \Leftrightarrow F_{A'} \neq \emptyset . \]
Example
A simple way to show that the two transition systems \( \mathcal{A} \) and \( \mathcal{A}' \) of Figure 8.1, page 153, cannot be related by a bisimulation relation is, given the preceding theorems, to note that state \( 2' \) of \( \mathcal{A}' \) satisfies formula \( \langle a \rangle (\langle b \rangle 1 \wedge \langle c \rangle 1) \) while no state of \( \mathcal{A} \) satisfies that formula.

Another way to define the greatest bisimulation, if it exists, between two labeled transition systems \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) is, given a relation \( R \), to define the mapping \( E : \wp(S_1 \times S_2) \rightarrow \wp(S_1 \times S_2) \) by \( \langle s_1, s_2 \rangle \in E(R) \) if and only if the three following conditions are satisfied:

\[(i) \quad \langle s_1, s_2 \rangle \in R.
(ii) \quad \forall t_1 = s_1 \Rightarrow a \rightarrow s'_1 \in T_1, \exists t_2 = s_2 \Rightarrow a \rightarrow s'_2 \in T_2 \text{ such that } \langle s'_1, s'_2 \rangle \in R.
(iii) \quad \forall t_2 = s_2 \Rightarrow a \rightarrow s'_2 \in T_2, \exists t_1 = s_1 \Rightarrow a \rightarrow s'_1 \in T_1 \text{ such that } \langle s'_1, s'_2 \rangle \in R.\]

By definition \( E(R) \subseteq R \) and \( E \) is monotone.

\[\square\]

Proposition 8.6 \( R \) is a bisimulation if and only if

\[\pi_1(R) = S_1, \quad \pi_2(R) = S_2 \quad \text{and} \quad R = E(R).\]

Proof Conditions \( \pi_1(R) = S_1 \) and \( \pi_2(R) = S_2 \) are none other than formulations of conditions \((i\ a)\) and \((i\ b)\) of the definition of a bisimulation relation. It is easy to see that conditions \((ii\ a)\) and \((ii\ b)\) are equivalent to \( R \subseteq E(R) \), hence to \( R = E(R) \), since \( E(R) \subseteq R \).

Since the mapping \( E \) is monotone it has a greatest fixpoint \( E_\nu, \cap_{n \geq 0} E^n(S_1 \times S_2) \).

Proposition 8.7 The following properties are equivalent:

\[(i) \quad \text{There exists a bisimulation } R \text{ between } \mathcal{A} \text{ and } \mathcal{A}'.
(ii) \quad \pi_1(E_\nu) = S_1 \text{ and } \pi_2(E_\nu) = S_2.
(iii) \quad E_\nu \text{ is the greatest bisimulation between } \mathcal{A} \text{ and } \mathcal{A}'.\]

Proof If \( R \) is a bisimulation, then \( E(R) = R \), hence \( R \subseteq E_\nu \). Hence \( E_\nu \) is the greatest bisimulation if and only if it is a bisimulation, and according to the preceding proposition, if and only if condition \((ii)\) is satisfied. So \((ii) \iff (iii)\) and \((iii) \Rightarrow (i)\). The implication \((i) \Rightarrow (ii)\) remains to be shown, but since \( R \) is a bisimulation, \( \pi_i(R) = S_i \), and since \( R \subseteq E_\nu \), \( \pi_i(E_\nu) = S_i \).

\[\square\]

8.2.3 Bisimulation generated by a relation

Consider an arbitrary relation \( R \) included in \( S_1 \times S_2 \) and let \( R^\omega = \cap_{n \geq 0} E^n(R) \). Since the sequence \( (E^n(R))_{n \geq 0} \) is decreasing, \( R^\omega \subseteq E^n(R) \) for every \( n \), and in particular, since \( E^0(R) = R \), \( R^\omega \subseteq R \).
Proposition 8.8 The following properties are equivalent:

(i) There exists a bisimulation $R'$ between $A$ and $A'$ included in $R$.

(ii) $\pi_1(R') = S_1$ and $\pi_2(R') = S_2$.

(iii) $R'$ is the greatest bisimulation between $A$ and $A'$ included in $R$.

Proof First show that $R'$ contains all the bisimulations included in $R$. If $R'$ is a bisimulation included in $R$, then $E(R') = R'$. Then by induction, $R' \subseteq E^n(R)$ for every $n$, which implies

$$R' \subseteq \bigcap_{n \geq 0} E^n(R) = R'. $$

The inequalities $R' \subseteq R = E^0(R)$ and $R' \subseteq E^n(R)$ imply

$$R' = E(R') \subseteq E^{n+1}(R).$$

Hence, since $R' \subseteq R'$, (iii) $\Rightarrow$ (i) $\Rightarrow$ (ii). To show (ii) $\Rightarrow$ (iii), it suffices to show, according to proposition 8.6 and since $R' \subseteq R$, that $R' = E(R')$.

By definition $E(R') \subseteq R'$. The converse inclusion is shown. Let $(s, s') \in R'$, and let $t = s \mapsto a \mapsto s_1$ be a transition of $A$. By the definition of $R'$, $(s, s') \in E^{n+1}(R)$, for every $n \geq 0$; there therefore exists a transition $t'_n = s' \mapsto a \mapsto s'_n$ of $A'$, with $(s_1, s'_n) \in E^n(R)$. Since $A'$ is finite there exists a state $s''$ such that the set $\{n \mid s'' = s'_n\}$ of integers is infinite. Hence $(s_1, s'') \in E^n(R)$ for an infinite number of $n$. Since the sequence $(E^n(R))_{n \geq 0}$ is decreasing, it follows that $(s_1, s'') \in E^n(R)$ for every $n$, and so $(s_1, s'') \in R'$. By doing the same thing to prove the symmetric condition, the result is $(s, s') \in E(R')$, which finishes the proof.

The concept of bisimulation generated by an arbitrary relation is of interest because of point (iii) of the previous proposition. To ask that a bisimulation relation be included in a given relation $R$ means adding to the conditions defining a bisimulation relation the supplementary condition $sRs'$. Therefore constraints can be imposed upon pairs of states so that they are in the bisimulation relation, for example, for semantic reasons, thereby obtaining a finer relation. An example is presented below.

In particular, if labeled transition systems having state parameters are to be compared, the starting relation $R$ could be the relation defined by $sRs'$ if and only if

$$\forall X \in \mathcal{X}, \ s \in S_X \Leftrightarrow s' \in S'_X.$$

In that case the greatest bisimulation included in $R$ is the relation defined by $sRs'$ if and only if for every formula $F'$ of Hennessy–Milner logic extended by the elementary propositions associated with the state parameters, $s \in F_A \Leftrightarrow s' \in F'_{A'}$. 
8.2.4 Autobisimulation

An autobisimulation is a bisimulation relation between a labeled transition system $A$ and itself. In particular, if $R$ is a bisimulation between $A$ and $A'$, then the relation $R \cdot R^{-1}$, which is a bisimulation, is an autobisimulation. If $R$ is an autobisimulation, the least equivalence relation containing $R$ is $(R \cup R^{-1})^*$, which is still an autobisimulation.

Since the identity relation is an autobisimulation, there exists a greatest autobisimulation $R$, which is an equivalence relation since it contains $R$, $R^{-1}$ and $R \cdot R$. According to theorem 8.2 the equivalence classes are precisely the indistinguishability classes of Hennessy–Milner logic. This gives a method to compute the quotient under indistinguishability (for Hennessy–Milner logic) of a given transition system: it suffices to compute its greatest autobisimulation as the limit of the decreasing sequence $E^n(S \times S)$. This construction is examined below.

Consequently, and still as an application of theorem 8.2, to know if there exists a bisimulation between two labeled transition systems, it suffices to verify that their quotients under indistinguishability, easy to construct, are isomorphic.

In the general case, the autobisimulation generated by a relation $R$ can also be considered. It is given a priori between the states of a transition system. If it exists it is the greatest bisimulation included in $R$ and is an equivalence relation. It follows that $R$ must be a reflexive relation. If such were not the case, $R$ could not contain an autobisimulation.

8.2.5 Example

A labeled transition system describing a simplified connection–disconnection procedure is considered. The transition system has three states:

- **ready** the user is not connected nor waiting for connection.
- **waiting** the user has requested a connection and identification is expected, for example through a password, before the connection can be made.
- **connected** the identification is valid and the connection is established.

The four possible actions labeling the transitions are:

- C request for connection.
- D request for disconnection.
- Iyes send a valid identification.
- Ino send an invalid identification.

The main transitions of this system are therefore:

- **ready** $\mapsto$ C $\mapsto$ waiting,
- **waiting** $\mapsto$ Iyes $\mapsto$ connected,
waiting ⊳ Ino ⊳ ready,
connected ⊳ D ⊳ ready.

To take into account erratic behavior, the user must provide for the possibility that the system might react with unexpected messages. The following transitions are therefore added:

\begin{align*}
\text{ready} & \rightarrow D \rightarrow \text{ready}, \\
\text{ready} & \rightarrow \text{Iyes} \rightarrow \text{ready}, \\
\text{ready} & \rightarrow \text{Ino} \rightarrow \text{ready}, \\
\text{waiting} & \rightarrow D \rightarrow \text{ready}, \\
\text{waiting} & \rightarrow C \rightarrow \text{waiting}, \\
\text{connected} & \rightarrow C \rightarrow \text{connected}, \\
\text{connected} & \rightarrow \text{Iyes} \rightarrow \text{connected}, \\
\text{connected} & \rightarrow \text{Ino} \rightarrow \text{connected}.
\end{align*}

The greatest autobisimulation of this transition system is the equivalence relation where all the states are equivalent. It is easily checked that this relation is a bisimulation relation. The quotient transition system, itself a quotient under indistinguishability for Hennessy–Milner logic, is the transition system with one state.

More generally, the following result is easily shown.

**Proposition 8.9** Let $A = \langle S, T, \alpha, \beta, \lambda \rangle$ be a labeled transition system and let $h$ be the surjective homomorphism from $A$ to the labeled transition system with one state $A' = \langle S', T', \alpha', \beta', \lambda' \rangle$. This homomorphism $h$ is saturating over Hennessy–Milner logic if and only if

$$\forall s \in S, \lambda(\alpha^{-1}(s)) = \lambda'(T').$$

It follows, due to proposition 8.4, that the greatest autobisimulation of $A$ is the 'complete' relation.

A necessary and sufficient condition for the existence of a surjective homomorphism from $A$ to a labeled transition system with one state satisfying

$$\forall s \in S, \lambda(\alpha^{-1}(s)) = \lambda'(T')$$

is simply

$$\forall s, s' \in S, \lambda(\alpha^{-1}(s)) = \lambda(\alpha^{-1}(s')),$$

i.e. if one action can be executed by a state of $A$, it can be executed by any other state of $A$.

Returning to the example, it is difficult to admit that the transition system with a single state is equivalent to the transition system of the example. The intuitive
reason is that the connected clearly plays a special rôle. However, nothing in the proposed model made that special rôle apparent. To distinguish this state from the others, just add the relation

\[ R = \{\text{ready, waiting}\}^2 \cup \{\text{connected}\}^2, \]

which states that the connected state can only be in relation with itself, and consider the greatest autobisimulation included in \( R \).

That greatest autobisimulation is the identity, since if it related ready and waiting, and since the Iyes action goes from ready to ready and from waiting to connected, the two states ready and connected would have to be related, which is impossible.

Another way to distinguish the connected from the others is to introduce a U action that represents an ordinary communication between the user and the system and that can only be executed in the connected state, by adding the transition

\[ \text{connected} \rightarrow \text{U} \rightarrow \text{connected}. \]

It is clear that every autobisimulation of this new transition system is included in the relation \( R \) defined above and that its greatest autobisimulation is the identity.

8.2.6 Algorithm

To compute the greatest autobisimulation included in a given relation \( R \), it suffices to compute the decreasing sequence \( E^n(R) \) until it becomes stationary.

Note first that if a relation \( R \) is symmetric, then the definition of \( E \), given on page 140, becomes \( (s_1, s_2) \in E(R) \) if and only if the two following conditions are satisfied:

(i) \( (s_1, s_2) \in R. \)
(ii) \( \forall t_1 = s_1 \rightarrow a \rightarrow s'_1 \in T, \exists t_2 = s_2 \rightarrow a \rightarrow s'_2 \in T \text{ such that } (s'_1, s'_2) \in R. \)

Note further that if \( R \) is an equivalence relation then \( E(R) \) is still an equivalence relation and the sequence \( E^n(R) \) is a decreasing sequence of equivalence relations. The assumption that the first relation \( R \) be an equivalence relation is not very restrictive. As was already mentioned, in most cases this first relation is an equivalence relation grouping in the same class states having similar properties, for example properties expressed by state parameters. So the presentation is restricted to how to compute the greatest autobisimulation included in a given equivalence relation.

Each of the equivalence relations in the sequence \( E^n(R) \) can be given in the form of a partition of the set of states. The computation of that decreasing sequence becomes a problem of partition `refinement'. It is a well-known problem that appears in the minimization of deterministic automata [1] and in the refinement
of partitions [72]. Reference [44] gives a detailed description of the algorithm; only the salient points are given here.

Let \( R \) be an equivalence relation over the set \( S \) of states of a transition system \( A = (S, T, \alpha, \beta, \lambda) \) labeled by an alphabet \( A \). Let \( B \) and \( B' \) be two equivalence classes of \( R \), not necessarily distinct, and let \( a \) be a letter in alphabet \( A \). Let \( R' = \Theta(R, B, B', a) \) be a new equivalence relation defined by \( sR's' \) if and only if

- \( s, s' \in S - B \) and \( sRs' \), or
- \( s, s' \in B \) and \( \beta(\lambda^{-1}(a) \cap \alpha^{-1}(s)) \cap B' \neq \emptyset \iff \beta(\lambda^{-1}(a) \cap \alpha^{-1}(s')) \cap B' \neq \emptyset \).

The second condition means that two states \( s \) and \( s' \) in \( B \) are equivalent under \( R' \) when there exists a transition of source \( s \) and label \( a \) whose target is in \( B' \) if and only if there exists a transition of source \( s' \) and label \( a \) whose target is also in \( B' \). It is easy to show the following properties.

**Lemma 8.1** Let \( s \) and \( s' \) be two states in \( S \) equivalent under \( E(R) \). For every equivalence class \( B' \) of \( R \) and for every letter \( a \) of \( A \),

\[
\beta(\lambda^{-1}(a) \cap \alpha^{-1}(s)) \cap B' \neq \emptyset \iff \beta(\lambda^{-1}(a) \cap \alpha^{-1}(s')) \cap B' \neq \emptyset.
\]

**Proof** The lemma is an immediate consequence of the definition of \( E(R) \). \( \square \)

**Proposition 8.10**

(i) The equivalence \( R' = \Theta(R, B, B', a) \) is finer than \( R \).

(ii) The equivalence \( E(R) \) is finer than \( R' \).

(iii) If the equivalence \( E(R) \) is strictly finer than \( R \), there exist \( B, B' \) and \( a \) such that \( \Theta(R, B, B', a) \) is strictly finer than \( R \).

**Proof** The first part of this proposition follows directly from the definition. To show the second part, consider two states \( s \) and \( s' \) equivalent under \( E(R) \). Since \( E(R) \) is finer than \( R \), the two states are also equivalent under \( R \). They are either both in \( S - B \), hence equivalent under \( R' \), or both in \( B \), hence, according to the lemma, equivalent under \( R' \).

Finally suppose that the two states are equivalent under \( R \) and are not under \( E(R) \). There therefore exist a letter \( a \) and a transition \( t = s_1 \rightarrow a \rightarrow s'_1 \), where \( s_1 \) is one of the two states, such that for every transition \( t' = s_2 \rightarrow a \rightarrow s'_2 \), where \( s_2 \) is the other of the two states, \( s'_2 \) is not equivalent to \( s'_1 \) under \( R \). If \( B \) is the equivalence class under \( R \) of \( s_1 \), and \( B' \) that of \( s'_1 \), then \( s_1 \) and \( s_2 \) are not equivalent under \( \Theta(R, B, B', a) \). \( \square \)

Let then \( R^n \) be the greatest autobisimulation included in \( R \). It is known that there exists an integer \( n \) such that \( R^n = E^n(R) \). Consider a decreasing sequence of equivalence relations \( R_0 = R, R_1, R_2, \ldots, R_m \) such that for every \( i \in \{0, \ldots, m-1\} \), there exists a letter \( a_i \) and equivalence classes \( B_i, B'_i \) of \( R_i \) such that

\[
R_{i+1} = \Theta(R_i, B_i, B'_i, a_i).
\]
Thanks to the previous proposition, it can be shown by induction that
\[ R^\omega \subseteq E^m(R) \subseteq R_m. \]

If in addition \( R_m \) is not equal to \( R^\omega \), then \( E^m(R) \) is not equal to \( R_m \) (for otherwise \( R_m \) would be a fixpoint of \( E \) included in \( R \), strictly containing the greatest fixpoint of \( E \) included in \( R \)) and one can find \( B_m, B'_m \) and \( a_m \) such that
\[ R_{m+1} = \Theta(R_m, B_m, B'_m, a_m) \]
is strictly finer than \( R_m \). Since \( R^\omega \) has only a finite number of classes, \( R^\omega \) is ultimately obtained by iteration of the operation \( \Theta \).

The operation that consists of constructing \( R' = \Theta(R, B, B', a) \) is not difficult to implement. The successive choices of \( B_i, B'_i \) and \( a_i \), which naturally play an important role in the performance of the algorithm, form a difficult problem. Fernandez [44] showed that it is possible to obtain the greatest bisimulation in time \( O(n.m) \), where \( n \) is the number of states and \( m \) the number of transitions of the transition system. In the case of an alphabet with a single letter, Paige and Tarjan [72] showed that it is possible to obtain the result in time \( O(m \log n) \).

### 8.2.7 Bidirectional bisimulation

It was shown in proposition 8.2 that a labeled transition system homomorphism
\[ h : A = \langle S, T, \alpha, \beta, \lambda \rangle \rightarrow A' = \langle S', T', \alpha', \beta', \lambda' \rangle \]
saturates Hennessy–Milner logic if and only if for every state \( s_1 \in S \) and for every transition \( t' = h_\sigma(s_1) \leftarrow a \rightarrow s'_2 \in T' \), there exists \( t = s_1 \leftarrow a \rightarrow s_2 \in T \) such that \( h_\sigma(s_2) = s'_2 \).

In fact this property can be expressed more simply by stating that \( h \) saturates the src operator of Dicky logic.

**Proposition 8.11** A labeled transition system homomorphism
\[ h : A = \langle S, T, \alpha, \beta, \lambda \rangle \rightarrow A' = \langle S', T', \alpha', \beta', \lambda' \rangle \]
saturates Hennessy–Milner logic if and only if it saturates the src operator.

**Proof** According to proposition 8.2 it suffices to show that \( h \) saturates src if and only if for every state \( s_1 \in S \) and for every transition
\[ t' = h_\sigma(s_1) \leftarrow a \rightarrow s'_2 \in T' \]
there exists a transition
\[ t = s_1 \leftarrow a \rightarrow s_2 \in T \]
such that $h_\sigma(s_2) = s'_2$.

Since $h_\sigma^{-1}$, $h_\tau^{-1}$, src$_A$ and src$_A'$ are additive, $h$ saturates src if and only if $\forall t' \in T'$,

\[
  h_\tau^{-1}\left(\text{src}_A\left(\{t'\}\right)\right) = \text{src}_A\left(h_\tau^{-1}(t')\right).
\]

Now src$_A\left(\{t'\}\right) = \alpha'(t')$, so $h$ saturates src if and only if $\forall t' = s'_1 \mapsto a \mapsto s'_2 \in T'$,

\[
  h_\sigma^{-1}(s'_1) = \text{src}_A\left(h_\tau^{-1}(t')\right).
\]

Let $t = s_1 \mapsto a \mapsto s_2 \in T$ and $t' = s'_1 \mapsto a' \mapsto s'_2 \in T'$. Then $t' = h_\tau(t)$ if and only if

\[
  h_\sigma(s_1) = s'_1, \quad h_\sigma(s_2) = s'_2 \quad \text{and} \quad a = a',
\]

since

\[
  h_\tau(s_1 \mapsto a \mapsto s_2) = h_\sigma(s_1) \mapsto a \mapsto h_\sigma(s_2).
\]

It follows that if $t = s_1 \mapsto a \mapsto s_2 \in h_\tau^{-1}(t')$ then $h_\sigma(s_1) = s'_1$ and so

\[
  \text{src}_A\left(h_\tau^{-1}(t')\right) \subseteq h_\sigma^{-1}(s'_1).
\]

Furthermore, the condition

\[
  h_\sigma^{-1}(s'_1) \subseteq \text{src}_A\left(h_\tau^{-1}(t')\right)
\]

is equivalent to: for every $s_1$ such that $h_\sigma(s_1) = s'_1$, there exists $t \in T$ such that $h_\tau(t) = t'$ and $\alpha(t) = s_1$, which can also be written: for every $s_1$ such that $h_\sigma(s_1) = s'_1$, there exists $t \in T$ such that $\alpha(t) = s_1$, $\lambda(t) = a$ and $h_\sigma(\beta(t)) = s'_2$, and the proof is finished. \hfill $\square$

The fact that a homomorphism saturates tgt can be translated similarly, by exchanging the rôles of the source and the target of a transition. Since every homomorphism saturates in and out (see page 129), it is possible to characterize the parameterized transition system homomorphisms that saturate Dicky logic. A bisimulation relation can then be defined, called bidirectional because of the symmetric rôle played by the source and target of transitions, and it is related to Dicky logic as bisimulation is to Hennessy–Milner logic. Everything that has been shown for bisimulation could be shown similarly for bidirectional bisimulation, whose definition is given here.

**Definition**

Let

\[
  \mathcal{A} = (S, T, \alpha, \beta, S_{X_1}, \ldots, S_{X_n}, T_{Y_1}, \ldots, T_{Y_m})
\]

and

\[
  \mathcal{A}' = (S', T', \alpha', \beta', S'_{X_1}, \ldots, S'_{X_n}, T'_{Y_1}, \ldots, T'_{Y_m})
\]

be two transition systems parameterized by $(\mathcal{X}, \mathcal{Y})$. A relation $R$ included in $S \times S'$ is a bidirectional bisimulation if:
(i a) For every state \( s \) in \( S \), there exists \( s' \) in \( S' \) such that \( sRs' \).

(i b) For every state \( s' \) in \( S' \), there exists \( s \) in \( S \) such that \( sRs' \).

(i c) If \( sRs' \), then for every \( X \in X \), \( s \in S_X \Leftrightarrow s' \in S'_X \).

(ii a) For every transition \( t \) in \( T \) and for every state \( s' \) in \( S' \) such that \( \alpha(t)Rs' \), there exists \( t' \) in \( T' \) such that \( s' = \alpha'(t') \), \( \beta(t)R\beta'(t') \) and, for every \( Y \in Y \), \( t \in T_Y \Leftrightarrow t' \in T'_Y \).

(ii b) For every transition \( t' \) in \( T' \) and for every state \( s \) in \( S \) such that \( sR\alpha'(t') \), there exists \( t \) in \( T \) such that \( s = \alpha(t) \), \( \beta(t)R\beta'(t') \) and, for every \( Y \in Y \), \( t \in T_Y \Leftrightarrow t' \in T'_Y \).

(iii a) For every transition \( t \) in \( T \) and for every state \( s' \) in \( S' \) such that \( \beta(t)Rs' \), there exists \( t' \) in \( T' \) such that \( s' = \beta'(t') \), \( \alpha(t)R\alpha'(t') \) and, for every \( Y \in Y \), \( t \in T_Y \Leftrightarrow t' \in T'_Y \).

(iii b) For every transition \( t' \) in \( T' \) and for every state \( s \) in \( S \) such that \( sR\beta'(t') \), there exists \( t \) in \( T \) such that \( s = \beta(t) \), \( \alpha(t)R\alpha'(t') \), and for every \( Y \in Y \), \( t \in T_Y \Leftrightarrow t \in T'_Y \).

Consider again the example of section 8.2.5. The greatest bidirectional autobisimulation of \( A \) is the identity: if the sets \( \lambda(\alpha^{-1}(s)) \) are all equal, this is not the case for the sets \( \lambda(\beta^{-1}(s)) \), since

\[
\begin{align*}
\lambda(\beta^{-1}(\text{ready})) & = \{D, Iyes, Ino\}, \\
\lambda(\beta^{-1}(\text{waiting})) & = \{C\}, \\
\lambda(\beta^{-1}(\text{connected})) & = \{C, Iyes, Ino\},
\end{align*}
\]

which suffices to show that two distinct states cannot be related by a bidirectional autobisimulation.

### 8.3 Trace equivalences

The equivalences presented here are not defined over transition systems, as before, but rather over \((\text{state, transition system})\) pairs. The reason is that in general it can be assumed that a transition system representing a process or a system of processes has a unique initial state. The two transition systems can then be compared by comparing the different evolutions that they can undergo starting from their respective initial states. These evolutions can be represented by the traces of the paths from the initial state, augmented with other information. Depending on the supplementary information’s form, different kinds of equivalence are obtained. Reference [86] offers an interesting overview of the various kinds of equivalence.
8.3.1 Trace equivalence

The simplest information that one can have about the evolution of a labeled transition system is the set of traces of its paths.

Let $\mathcal{A} = \langle S, T, \alpha, \beta, \lambda \rangle$ be a transition system labeled by an alphabet $A$. For each state $s$ of $S$ let

$$C_A(s) = \{ c \in T^* \mid \alpha(c) = s \}$$

be the set of finite paths from $s$ and let the language $L_A(s) = \lambda(C_A(s))$, included in $A^*$, be the set of all the sequences of actions that can be executed from $s$.

**Proposition 8.12** If $\mathcal{A}$ is finite then $L_A(s)$ is a regular language.

**Proof** $L_A(s)$ is the language recognized by the nondeterministic automaton whose transitions are given by $\mathcal{A}$, whose initial state is $s$ and whose states are all final. $\Box$

Two states $s$ and $s'$ of two transition systems $\mathcal{A}$ and $\mathcal{A}'$ labeled by the same alphabet are *trace-equivalent* if $L_A(s) = L_{A'}(s')$.

**Proposition 8.13** If $R$ is a bisimulation relation between $\mathcal{A}$ and $\mathcal{A}'$, and if $sRs'$, then $s$ and $s'$ are trace-equivalent.

**Proof** For each word $u = a_1 \cdots a_n$ of $A^*$, define the Hennessy–Milner logic formula $\langle a_1 \rangle \cdots \langle a_n \rangle 1$, written $\langle u \rangle 1$. Then $u \in L_A(s)$ if and only if $s \in (\langle u \rangle 1)_A$ and $u \in L_{A'}(s')$ if and only if $s' \in (\langle u \rangle 1)_{A'}$. Since $sRs'$, according to theorem 8.2, $s \in (\langle u \rangle 1)_A$ if and only if $s' \in (\langle u \rangle 1)_{A'}$. $\Box$

The converse is not true in general, as is shown by the example on page 134 (see Figure 8.1). There $L_A(1) = L_{A'}(1') = \{ \varepsilon, a, ab, ac \}$, yet there exists no bisimulation between 1 and 1'. But it is true for deterministic transition systems, i.e. those that satisfy $\alpha(t) = \alpha(t')$ and $\lambda(t) = \lambda(t') \Rightarrow \beta(t) = \beta(t')$.

**Proposition 8.14** Let $\mathcal{A} = \langle S, T, \alpha, \beta, \lambda \rangle$ and $\mathcal{A}' = \langle S', T', \alpha', \beta', \lambda' \rangle$ be deterministic transition systems labeled by alphabet $A$, such that $\{L_A(s) \mid s \in S\} = \{L_{A'}(s') \mid s' \in S'\}$. Then the relation $R$, defined by $sRs'$ if and only if $L_A(s) = L_{A'}(s')$, is a bisimulation.

**Proof** Since $\{L_A(s) \mid s \in S\} = \{L_{A'}(s') \mid s' \in S'\}$,

$$\forall s, \exists s' : sRs' \text{ and } \forall s', \exists s : sRs'.$$

The idea is to show that if $\mathcal{A}$ is deterministic, $a^{-1}L_A(s_1) \neq \emptyset$ if and only if there exists $s_2$ such that $t = s_1 \mapsto a \mapsto s_2$ and $a^{-1}L_A(s_1) = L_A(s_2)$, where, for every subset $L$ of $A^*$, $a^{-1}L = \{ u \mid au \in L \}$. Now, if $\mathcal{A}$ is deterministic and if $s_1 \mapsto a \mapsto s_2$, then $a^{-1}L_A(s_1) = L_A(s_2)$ and $L_A(s_2) \neq \emptyset$, since $\varepsilon \in L_A(s_2)$. Conversely, if
\[ a^{-1}L_A(s_1) \neq \emptyset, \] there exists a non-empty path of source \( s_1 \), whose first transition is labeled by \( a \). There therefore exists \( s_2 \) such that \( s_1 \xrightarrow{a} s_2 \).

It follows that if \( s_1 \xrightarrow{a} s_2 \) is a transition of \( A \) then \( L_A(s_2) = a^{-1}L_A(s_1) \neq \emptyset \), and if furthermore \( s_1Rs_1' \), then \( a^{-1}L_A(s_1) = a^{-1}L_A(s_1') \neq \emptyset \). There therefore exists \( s_2' \) such that \( s_1' \xrightarrow{a} s_2' \) and \( L_{A'}(s_2') = a^{-1}L_{A'}(s_1') = L_A(s_2) \), hence \( s_2Rs_2' \). \( \square \)

### 8.3.2 Multitrace equivalence

In trace equivalence all the paths emanating from a state \( s \) are considered. This equivalence can be refined by distinguishing paths according to properties held by the target state. To do this a general definition is given; this is then adapted to particular cases.

Let \( A \) be a labeled transition system having state parameters \( S_X \) for \( X \in \mathcal{X} \). Let \( L_X^A(s) = \{ \lambda(c) \mid \alpha(c) = s, \beta(c) \in S_X \} \).

Let \( A' \) be another transition system, labeled by the same alphabet and having the same set \( \mathcal{X} \) of state parameter names. Then \( s \in S \) and \( s' \in S' \) are multitrace-equivalent if for every \( X \) in \( \mathcal{X} \), \( L_X^A(s) = L_X^{A'}(s') \).

Using the sole parameter of the set of all states gives the previous definition of trace equivalence.

### Accepting trace equivalence

To study the semantics of CSP, Hoare introduced in [57] the concept of channel state. At a given instant, the state of a channel is the sequence of messages that have passed through the channel and the set of messages that can enter it. This concept, readiness semantics [71, 86], is easily applied to transition systems. Two states are equivalent if every executable sequence of actions starting from one of the states leading to a state where certain actions are executable is also executable starting from the other state and leads to a state where the same actions are executable.

This is a multitrace equivalence: for a state \( s \) of a labeled transition system \( A \), the set of letters

\[ \text{Poss}_A(s) = \{ a \in A \mid \exists t : \alpha(t) = s, \lambda(t) = a \} = \lambda(a^{-1}(s)), \]

is the set of actions that \( A \) can perform in state \( s \). For each subset \( A' \) of \( A \), define the state parameter name \( A' \) and the corresponding parameter

\[ S_{A'} = \{ s \mid \text{Poss}_A(s) = A' \}. \]

For this equivalence, states 1 and 1' of Figure 8.1 (page 153) are no longer
equivalent:
\[ L^a_A(1) = \{\varepsilon\}, \quad L'^a_{A'}(1') = \{\varepsilon\}, \]
\[ L^a_A(1) = \{ab, ac\}, \quad L^0_{A'}(1') = \{ab, ac\}, \]
\[ L^a_A(1) = \{a\}, \quad L^0_{A'}(1') = \{\emptyset\}, \]
\[ L^a_A(1) = \{a\}, \quad L^0_{A'}(1') = \{\emptyset\}, \]
\[ L^{b,c}_A(1) = \{\emptyset\}, \quad L^{b,c}_{A'}(1') = \{a\}. \]

Refusing trace equivalence

In [19], Brookes et al. introduced another equivalence concept, also to study the semantics of communicating processes, that is coarser than accepting trace equivalence. It is another multitrace equivalence associated with the set of parameter names \( \{A' \mid A' \subseteq A\} \), but the parameter corresponding to \( A' \) is now \( S_{A'} = \{s \mid \text{Poss}_A(s) \cap A' = \emptyset\} \).

For this equivalence, the states 1 and 1' of Figure 8.1 (page 153) are no longer equivalent: the set of traces of the paths of source 1 whose target \( s \) is such that \( c \in \text{Poss}_A(s) \) is \( \{\varepsilon, a, ab, ac\} \), while for state 1', the set is \( \{\varepsilon, ab, ac\} \).

Maximal finite trace equivalence

A finite path is maximal if in its target state no action is possible, i.e. if
\[ \text{Poss}_A(\beta(c)) = \emptyset. \]

To compare two states, if only the traces of maximal finite paths are to be considered, this can be done by using the multitrace equivalence defined by the parameter \( \{s \mid \text{Poss}_A(s) = \emptyset\} \).

Now the states 1 and 1' of Figure 8.1 (page 153) are equivalent since the traces of maximal finite paths are in both cases \( \{ab, ac\} \).

General properties

Let \( X \) be a set of state parameter names, and suppose that for every parameter name \( X \), there exists a Hennessy–Milner formula \( F_X \) such that for every transition system \( A \), labeled by \( A \), containing the parameters \( S_X, S_X = (F_X)_A \). Then the following proposition applies.

**Proposition 8.15** If \( R \) is a bisimulation between \( A \) and \( A' \), and if \( sRs' \), then \( s \) and \( s' \) are equivalent under the multitrace equivalence induced by this set of parameters.

**Proof** A word \( u = a_1 \cdots a_n \) is in \( L^X_A(s) \) if and only if \( s \) is in \( (\langle a_1 \rangle \cdots \langle a_n \rangle F_X)_A \). If \( sRs' \) then \( s \) and \( s' \) are indistinguishable and
\[ s \in (\langle a_1 \rangle \cdots \langle a_n \rangle F_X)_A \Leftrightarrow s' \in (\langle a_1 \rangle \cdots \langle a_n \rangle F_X)_A. \]

\[ \square \]
All the previously introduced parameters can be characterized by Hennessy–Milner formulas and the previous proposition then applies.

To show this, \( \text{Poss}_A(s) = A' \) if and only if

\[
s \in \left( \bigwedge_{a \in A'} \langle a \rangle 1 \wedge \bigwedge_{b \in A - A'} \neg \langle b \rangle 1 \right)_A,
\]

and \( \text{Poss}_A(s) \cap A' = \emptyset \) if and only if

\[
s \in \left( \bigwedge_{a \in A'} \neg \langle a \rangle 1 \right)_A.
\]

Consider now two sets of state parameter names \( \mathcal{X} \) and \( \mathcal{Y} \). Suppose that for each \( X \in \mathcal{X} \), a subset \( \mathcal{Y}_X \) of \( \mathcal{Y} \) has been defined, and suppose that for every transition system \( A \) having the two sets of parameters, \( \forall X \in \mathcal{X}, \ S_X = \bigcup_{Y \in \mathcal{Y}_X} S_Y \). Then if two states of two transition systems are multitrace-equivalent under the equivalence induced by \( \mathcal{Y} \), they are also multitrace-equivalent under the equivalence induced by \( \mathcal{X} \):

\[
L^X_A(s) = \lambda \left( \left\{ c \mid \alpha(c) = s, \beta(c) \in S_X \right\} \right)
\]

\[
= \lambda \left( \left\{ c \mid \alpha(c) = s, \beta(c) \in \bigcup_{Y \in \mathcal{Y}_X} S_Y \right\} \right)
\]

\[
= \bigcup_{Y \in \mathcal{Y}_X} \lambda \left( \left\{ c \mid \alpha(c) = s, \beta(c) \in S_X \right\} \right)
\]

\[
= \bigcup_{Y \in \mathcal{Y}_X} L^Y_A(s).
\]

It follows for example that:

1. Refusing trace equivalence implies the maximal trace equivalence since the parameter used in the second case is only one of the parameters used in the first case: \( \text{Poss}_A(s) = \emptyset \) if and only if \( \text{Poss}_A(s) \cap A = \emptyset \).

2. Refusing trace equivalence implies trace equivalence since the only parameter used for the latter is \( S = \left\{ s \mid \text{Poss}_A(s) \cap \emptyset = \emptyset \right\} \).

3. Accepting trace equivalence implies refusing trace equivalence:

\[
\left\{ s \mid \text{Poss}_A(s) \cap A' = \emptyset \right\} = \bigcup_{A'' \subseteq A - A'} \left\{ s \mid \text{Poss}_A(s) = A'' \right\}.
\]

It has already been seen in the example of Figure 8.1 that refusing trace equivalence could be strictly finer than maximal trace equivalence.

In the example of Figure 8.2 (page 153), the traces of paths of source 1 and 1' are respectively \( a^* + b \) and \( b \), while the traces of maximal paths are \( \{ b \} \) in the two
Figure 8.1 The transition systems of the example on page 134.

Figure 8.2 Two transition systems having the same maximal traces, yet different traces.

Figure 8.3 Two transition systems having the same traces, yet different maximal traces.
cases. On the other hand, in the example of Figure 8.3 (page 153), the traces of the paths are \{\varepsilon, a, ab\} in the two cases, while the traces of the maximal paths are respectively \{a, ab\} and \{ab\}. These two concepts are not comparable.

In the example of Figure 8.4 (page 155), states 1 and 1' are refusing trace-equivalent but not accepting trace-equivalent.

Finally the example of Figure 8.5 (page 155) shows that states 1 and 1' are not in bisimulation since the Hennessy-Milner formula \((a)(\langle b \rangle(c)1 \land \langle b \rangle(d)1)\) distinguishes them, yet they are accepting trace-equivalent.

### 8.3.3 Extended trace equivalence

Instead of only using the state parameters to distinguish the target states of paths, they can also be used to distinguish the intermediary states of those paths.

Let \(\mathcal{A}\) be a labeled transition system with parameters \(S_X\) for \(X \in \mathcal{X}\). For each finite path \(c = t_1 \cdots t_n\), define the set of sequences

\[
\hat{\lambda}(c) = \{X_0a_1X_1a_2 \cdots a_nX_n \mid \lambda(t_i) = a_i, \alpha(t_i) \in S_{X_{i-1}}, \beta(t_i) \in S_{X_i}\},
\]

and let \(\hat{L}_\mathcal{A}(s) = \bigcup\{\hat{\lambda}(c) \mid \alpha(c) = s\}\).

Two states \(s\) of \(\mathcal{A}\) and \(s'\) of \(\mathcal{A}'\) are equivalent if \(\hat{L}_\mathcal{A}(s) = \hat{L}_\mathcal{A}''(s')\).

**Examples**

The parameters \(S_{\mathcal{A}'} = \{s \mid \text{Poss}_\mathcal{A}(s) = \mathcal{A}'\}\) yield ready trace semantics equivalence and the parameters \(S_{\mathcal{A}'} = \{s \mid \text{Poss}_\mathcal{A}(s) \cap \mathcal{A}' = \emptyset\}\) yield failure trace semantics equivalence, according to van Glabbeek's terminology [86].

Note that if for every transition system \(\mathcal{A}\), whatever the path \(c\), \(\hat{\lambda}(c)\) is never empty. This means, assuming that \(S = \bigcup_{X \in \mathcal{X}} S_X\) (which is the case in the preceding examples), extended trace equivalence under these parameters implies multitrace equivalence under the same parameters:

\[
L^X_\mathcal{A}(s) = \{a_1 \cdots a_n \mid \exists X_0, \ldots, X_{n-1} \in \mathcal{X} : X_0a_1X_1 \cdots X_{n-1}a_nX \in \hat{L}_\mathcal{A}(s)\}.
\]

It follows in particular that:

4. Ready trace semantics equivalence implies accepting trace equivalence.

5. Failure trace semantics equivalence implies refusing trace semantics.

Furthermore, in the same way that accepting trace equivalence implies refusing trace equivalence:

6. Ready trace semantics equivalence implies failure trace semantics equivalence.

The example of Figure 8.4 (page 155) shows that accepting trace equivalence and ready trace semantics equivalence can be strictly finer than failure trace semantics equivalence.
Figure 8.4  Two transition systems having the same refusing traces, yet different accepting traces.

Figure 8.5  Two transition systems having the same accepting traces, yet not equivalent under bisimulation.
The example of Figure 8.6 (page 158) shows that:

- Ready trace semantics equivalence can be strictly finer than accepting trace equivalence.
- Failure trace semantics equivalence can be strictly finer than refusing trace equivalence.
- Failure trace semantics equivalence can be strictly finer than accepting trace equivalence.

Finally, the example of Figure 8.5 (page 155) shows that bisimulation can be strictly finer than any of these equivalences.

Another way to define \( \hat{L}_A(s) \) is to define inductively for a sequence \( u \) of the form \( X_0 a_1 X_1 a_2 \cdots a_n X_n \) a formula \( F_u \) of Hennessy–Milner logic extended by elementary propositions \( P_X \) for the parameters \( X \) in \( \mathcal{X} \), whose interpretation \( (P_X)_A \) is precisely \( S_X \).

If \( u = X \), then \( F_u = P_X \). If \( u = Xa \), then \( F_u = P_X \land (a) F_v \).

It is clear that \( u \in \hat{L}_A(s) \) if and only if \( s \in (F_u)_A \). If the parameters are definable by Hennessy–Milner formulas, i.e. if for every \( X \) in \( \mathcal{X} \), there exists \( F_X \) such that for every \( A \), \( S_X = (F_X)_A \), then \( P_X \) can be replaced by \( F_X \) in \( F_u \) which becomes a Hennessy–Milner formula. In that case extended trace equivalence is coarser than bisimulation.

### 8.3.4 A generalization of trace equivalences

The extended trace of a finite path \( c = t_1 \cdots t_n \) was defined on page 154 as a sequence \( X_0 a_1 X_1 a_2 \cdots a_n X_n \) such that \( \lambda(t_i) = a_i \), \( \alpha(t_i) \in S_{X_{i-1}} \) and \( \beta(t_i) \in S_{X_i} \). This sequence can also be written in the form

\[
(X_0, a_1, X_1)(X_1, a_2, X_2) \cdots (X_{n-1}, a_n, X_n),
\]

each triplet \( (X_{i-1}, a_i, X_i) \) being a property of transition \( t_i \).

This definition can be generalized by considering a set \( \mathcal{Y} \) of transition parameters. A sequence \( Y_1 Y_2 \cdots Y_n \) of parameter names is the generalized trace of the finite path \( c = t_1 \cdots t_n \) of a parameterized transition system if for every \( i \in \{1, \ldots, n\} \), \( t_i \in T_{Y_i} \). Two states are equivalent if the set of traces of the paths of which they are the respective sources are equal.

Furthermore nothing prevents the examination of traces of infinite paths. But it is easy to see that this brings nothing if transitions whose targets are sources of infinite paths cannot be distinguished. (It was seen in the examples of page 57 that the set of states of a transition system \( A = (S, T, \alpha, \beta) \) that were not the source of an infinite path is equal to \( AU_A(S, \emptyset) \).)

In fact, if \( A \) is a parameterized transition system, the transition parameters can be redefined by \( T_{Y'} = \{ t \mid t \in T_Y, \beta(t) \not\in AU_A(S, \emptyset) \} \) so that if a finite path is not the prefix of any infinite path the set of traces becomes empty, otherwise it is
unchanged. Let $L(s)$ (resp. $L\omega(s)$) be the set of traces, for this new definition of transition parameters, of finite (resp. infinite) paths of source $s$.

If $K$ is an arbitrary set of finite or infinite words over the alphabet $\mathcal{Y}$, then let

\[
\text{Pref}(K) = \{ u \in \mathcal{Y}^* \mid \exists v \in \mathcal{Y}^\infty : uv \in K \},
\]
\[
\text{Adh}(K) = \left\{ w \in \mathcal{Y}^\omega \mid \text{Pref}(\{w\}) \subseteq \text{Pref}(K) \right\}.
\]

The fact that taking into account the traces of infinite paths gives nothing more follows from the next lemma:

**Lemma 8.2**

\[
L(s) = \text{Pref}(L\omega(s)),
\]
\[
L\omega(s) = \text{Adh}(L(s)).
\]

**Proof** It is clear that if $w$ is the trace of an infinite path of source $s$, every prefix of that path has a trace that is a prefix of $w$ and that can be found in $L(s)$. Conversely, if $w$ is in $L(s)$, it is the trace of a finite path $c$ of source $s$ that is prefix of an infinite path $c'$. There exists a trace $w$ of $c'$ of which $u$ is a prefix. And so $L(s) = \text{Pref}(L\omega(s))$.

It follows immediately that $L\omega(s) \subseteq \text{Adh}(L(s))$, and the converse inclusion remains to be shown. So let $w \in \text{Adh}(L(s))$. For every $n \geq 0$ there exists a path $c_n$ of length $n$, of source $s$, which is prefix to an infinite path and whose trace is the prefix of $w$ of length $n$. Let $C_n$ be the set of those paths. Let $C' = \bigcup_{n \geq 0} C_n$. Define a binary relation $\prec$ by $c \prec c'$ if and only if there exists a transition $t$ such that $c \cdot t = c'$. Every path $c$ of $C'$ has only a finite number of successors under $\prec$, and if it is non-empty, a unique predecessor under $\prec$ in $C'$, since if $c' \cdot t$ has for a trace a prefix of $w$, so does $c'$. Furthermore $C'$ is infinite. König's lemma can be applied and so there exists an infinite sequence $c_0 \prec c_1 \prec \ldots$ that defines an infinite path one trace of which is necessarily $w$. \(\square\)

But instead of considering all the traces of infinite paths, it can be of interest to consider only those traces satisfying given fairness conditions. A fairness condition being representable by a language $K$ of $\mathcal{Y}^\omega$, the associated equivalence is defined by $L\omega(s) \cap K = L\omega(s') \cap K$. Things can be complicated just a bit more by simultaneously using several fairness conditions.\ldots

The example of Figure 8.7 (page 158) shows two labeled transition systems whose traces of infinite paths are not the same (here, it is once again the label that is associated with a transition), but become equal if, as a fairness condition, the traces must contain infinitely often the letter $b$. 
Figure 8.6 Two transition systems having the same accepting and refusing traces, yet not equivalent under ready trace or failure trace semantics.

Figure 8.7 Two transition systems having different infinite traces and the same fair traces.
The observation of transition systems

Up until now all actions or events appearing in a transition system have been treated similarly. Allusion was made in the examples of subsections 2.2.1, 2.2.5 and 2.2.6 to the use of a null action to indicate explicitly that a process could not do anything and still remain in the same state. Process algebras such as Milner’s contain an unobservable action, normally written \( \tau \), as was presented in the example in section 2.4. Milner explains in [66] the existence of such an action: it represents the fact that a system can ‘spontaneously’ change state, without any apparent, or observable, reason. This can occur, for example, in a system made up of two components that can exchange messages either between themselves or with an outside ‘observer’. When messages are exchanged between the two components the system can change state even though the action that provoked the change of state is not ‘visible’ to the observer. Equivalences of transition systems can be defined so that the unobservable actions are not taken into account.

9.1 Observational equivalence

Example
When transition system \( A'' \) of Figure 9.1 (page 161) is in state \( 1'' \) and executes action \( a \), it goes into state \( 1a'' \). Then, spontaneously and in a nondeterministic manner, it goes into state \( 2'' \) or \( 3'' \). Depending on the case, it can either execute action \( b \) or action \( c \). Everything appears as if action \( a \) made the system pass nondeterministically from state \( 1'' \) to state \( 2'' \) or \( 3'' \). Transition system \( A'' \) is therefore equivalent, in a certain sense, to transition system \( A \) of the example on page 134 (see Figure 8.1, page 153).

However these two transition systems are not equivalent under bisimulation. A new equivalence relation must be defined, taking into account the existence of invisible transitions, and only considering the ‘observable’ behaviors of the transition systems to be compared. This equivalence, observational equivalence, is defined
like bisimulation, but instead of considering only transitions, ‘paths’ of the same trace are also considered.

In a labeled transition system \( A = (S, T, \alpha, \beta, \lambda) \), whose alphabet \( A \) of actions contains an invisible action, written \( \tau \), the concept of transition can be generalized by defining another set of transitions \( T_\tau \), included in \( S \times A \times S \), defined by: \( (s, a, s') \in T_\tau \) if and only if

- there exists in \( A \) a path, from \( s \) to \( s' \), whose trace is of the form \( \tau^n a \tau^m \), \( n, m \geq 0 \); or
- \( a = \tau \) and \( s = s' \).

Two transition systems are observationally equivalent if there exists a bisimulation between transition systems obtained by replacing their sets of transitions by the \( T_\tau \) sets defined above. This definition can also be written as follows, where \( s \xrightarrow{a} s' \) denotes that \( (s, a, s') \) is in \( T_\tau \).

An observationally equivalence between two transition systems \( A_1 \) and \( A_2 \) is a relation \( R \) between their sets \( S_1 \) and \( S_2 \) of states that satisfies:

(i a) For every state \( s_1 \) in \( S_1 \), there exists \( s_2 \) in \( S_2 \) such that \( s_1 R s_2 \).

(i b) For every state \( s_2 \) in \( S_2 \), there exists \( s_1 \) in \( S_1 \) such that \( s_1 R s_2 \).

(ii a) For every \( s_1 \) in \( S_1 \) and for every \( s_2 \) in \( S_2 \) such that \( s_1 R s_2 \), if \( s_1 \xrightarrow{a} s'_1 \), there exists \( s'_2 \) in \( S_2 \) such that \( s'_1 R s'_2 \) and \( s_2 \xrightarrow{a} s'_2 \).

(ii b) For every \( s_1 \) in \( S_1 \) and for every \( s_2 \) in \( S_2 \) such that \( s_1 R s_2 \), if \( s_2 \xrightarrow{a} s'_2 \), there exists \( s'_1 \) in \( S_1 \) such that \( s'_1 R s'_2 \) and \( s_1 \xrightarrow{a} s'_1 \).

Example

Consider the transition systems \( A \) and \( A' \) of Figure 8.1 (page 153), \( A'' \) of Figure 9.1 (page 161) and \( B \) of Figure 9.2 (page 161).

In \( A \):

\[
\begin{align*}
1 & \xrightarrow{\tau} 1, \\
1 & \xrightarrow{a} 2, \\
1 & \xrightarrow{a} 3, \\
2 & \xrightarrow{\tau} 2, \\
2 & \xrightarrow{b} 4, \\
3 & \xrightarrow{\tau} 3, \\
3 & \xrightarrow{c} 5, \\
4 & \xrightarrow{\tau} 4, \\
5 & \xrightarrow{\tau} 5.
\end{align*}
\]

In \( A' \):

\[
\begin{align*}
1' & \xrightarrow{\tau} 1', \\
1' & \xrightarrow{a} 2', \\
2' & \xrightarrow{\tau} 2', \\
2' & \xrightarrow{b} 3', \\
2' & \xrightarrow{c} 4', \\
3' & \xrightarrow{\tau} 3', \\
4' & \xrightarrow{\tau} 4'.
\end{align*}
\]
Figure 9.1 A transition system with invisible transitions.

Figure 9.2 Another transition system with invisible transitions.
The observation of transition systems

In $A''$:  

In $B$:  

\[ 1'' \xrightarrow{\tau} 1'', \quad 1b \xrightarrow{\tau} 1b, \]

\[ 1'' \xrightarrow{a} 1a'', \quad 1b \xrightarrow{a} 2b, \]

\[ 1'' \xrightarrow{a} 2'', \quad 1b \xrightarrow{a} 2b', \]

\[ 1'' \xrightarrow{a} 3'', \quad 2b \xrightarrow{\tau} 2b, \]

\[ 1a'' \xrightarrow{\tau} 1a'', \quad 2b \xrightarrow{\tau} 2b', \]

\[ 1a'' \xrightarrow{\tau} 2'', \quad 2b \xrightarrow{b} 4b, \]

\[ 1a'' \xrightarrow{\tau} 3'', \quad 2b \xrightarrow{c} 3b, \]

\[ 1a'' \xrightarrow{b} 4'', \quad 2b' \xrightarrow{\tau} 2b', \]

\[ 1a'' \xrightarrow{c} 5'', \quad 2b' \xrightarrow{b} 4b, \]

\[ 2'' \xrightarrow{\tau} 2'', \quad 3b \xrightarrow{\tau} 3b, \]

\[ 2'' \xrightarrow{b} 4'', \quad 4b \xrightarrow{\tau} 4b. \]

There is therefore no observational equivalence relation between any two of these transition systems.  

Example  
Consider the transition system $C$ of Figure 9.3 (page 166):  

\[ 1c \xrightarrow{\tau} 1c, \]

\[ 1c \xrightarrow{a} 1c', \]

\[ 1c \xrightarrow{a} 2c, \]

\[ 1c' \xrightarrow{\tau} 1c', \]

\[ 1c' \xrightarrow{\tau} 2c, \]

\[ 1c' \xrightarrow{b} 3c, \]

\[ 1c' \xrightarrow{c} 4c, \]

\[ 2c \xrightarrow{\tau} 2c, \]

\[ 2c \xrightarrow{b} 3c, \]

\[ 2c \xrightarrow{c} 4c, \]

\[ 3c \xrightarrow{\tau} 3c, \]

\[ 4c \xrightarrow{\tau} 4c. \]

This transition system is observationally equivalent to transition system $A'$ in
Figure 8.1 (page 153), and the relation $R$ defined below satisfies the required properties.

\[
\begin{align*}
(1c, 1') \\
(1c', 2') \\
(2c, 2') \\
(3c, 3') \\
(4c, 4')
\end{align*}
\]

\[\square\]

### 9.2 Observation criteria

Observational equivalence essentially means applying bisimulation once paths have been replaced by transitions. One can do the same in other situations. Suppose, for example, that one wishes to model a program in which a subprogram is executed. An action $a$ can represent calling the subprogram, $r$ the return to the calling program, and $i$ any action within the subprogram. An execution of the subprogram will then be a path whose trace is a word in the regular language $ai^*r$. Should abstraction from the details of this subprogram be required, that path can be replaced by a single transition labeled by an action $p$. In the obtained transition system, the actions internal to the subprogram are no longer observable. What is observable is the execution of the subprogram, seen as an atomic action.

It was in this perspective that Boudol [16] invented the concept of observation criteria. Let $A$ and $B$ be two alphabets. An observation criterion is a partial mapping $\phi$ from $A^*$ to $B$. If $b \in B$, every word in $\phi^{-1}(b)$ can be seen as an unobservable implementation of the observable action $b$.

In the case of observational equivalence, the two alphabets $A$ and $B$ are identical and the observation criterion $\phi_r$ to be used is defined by:

\[
\begin{align*}
\phi_r^{-1}(r) &= r^*, \\
\phi_r^{-1}(a) &= r^*ar^*.
\end{align*}
\]

If $A = (S, T, \alpha, \beta, \lambda)$ is a transition system labeled by an alphabet $A$ and if $\phi$ is an observation criterion, the observation $\phi(A)$ of $A$ is the transition system $A' = (S', T', \alpha', \beta', \lambda')$ labeled by the alphabet $B$, defined by:

- $S' = S$.
- $T'$ is the set of triples $(s, b, s')$ such that there exists a finite path $c$ with source $s$ and target $s'$ and whose label $\lambda(c)$ is in $\phi^{-1}(b)$.
- $\alpha'(s, b, s') = s, \beta'(s, b, s') = s'$ and $\lambda'(s, b, s') = b$.

All of the manipulations applicable to an ordinary transition system can be applied to $\phi(A)$. In particular, two transition systems $A$ and $A'$ labeled by the same alphabet $A$ are equivalent up to an observation criterion $\phi$, if $\phi(A)$ and $\phi(A')$ are equivalent. Hence, observational equivalence can be defined as bisimulation modulo $\phi_r$. 
9.3 Other equivalences

It is possible to define many other equivalences, which can be characterized by logical and algebraic criteria, e.g. the future perfect logic of Hennessy and Stirling [55], also studied in [8]. Two of these equivalences are presented here.

9.3.1 Stuttering equivalence

Stuttering equivalence was introduced in [21] for CTL*, minus the N operator. Its relation to that logic is analogous to that of bisimulation for Hennessy–Milner logic (see [21, 29]). Like CTL*, it is designed for unlabeled transition systems with state parameters. The definition given here comes from [29].

Let $\mathcal{A} = \langle S, T, \alpha, \beta \rangle$ be a transition system with state parameters, and let $\mu_\sigma$ be a state marking, defined on page 8, which associates with each state the set of parameters that contain it.

A stuttering equivalence is an equivalence relation $R$ over $S$ such that for every pair $s, s'$ of states, if $sRs'$ then

- $\mu_\sigma(s) = \mu_\sigma(s')$;
- for every transition $t$ of source $s$,
  - either $\beta(t)Rs'$,
  - or there exists a non-empty path $c = t_0t_1 \cdots t_n$, such that $\beta(t)R\beta(t_n)$ and for all $i \in \{0, \ldots, n\}$, $sR\alpha(t_i)$.

To compare two transition systems using this equivalence, it suffices to define a relation $R$ over their disjoint union.

In an intuitive manner, this equivalence allows the ‘merging’ of consecutive states having the same properties. For labeled transition systems, this definition can be adapted by allowing a single transition to be observed when several transitions of the same label follow one another, yielding the observation criterion $\phi_{\text{stat}}$ defined by

$$\forall a \in A, \phi_{\text{stat}}^{-1}(a) = a^+.$$

9.3.2 Branching bisimulation

For transition systems labeled by an alphabet containing the invisible action $\tau$, a relation close to stuttering equivalence, branching equivalence, was introduced by van Glabbeek and Weijland [87]. Reference [29] presents several logics that can characterize this equivalence.

An equivalence relation $R$ over the states of a labeled transition system $\mathcal{A} = \langle S, T, \alpha, \beta, \lambda \rangle$ is a branching bisimulation if for every pair $(s, s')$ of states such that $sRs'$, and if for every transition $t$ of source $s$, at least one of the two following conditions holds:
• $\lambda(t) = \tau$ and $\beta(t)R$s';

• there exists a non-empty path $t_0t_1\cdots t_n$ of source $s'$ such that
  
  - $\lambda(t_n) = \lambda(t)$ and for every $i \in \{0, \ldots, n - 1\}$, $\lambda(t_i) = \tau$,
  
  - $sR\alpha(t_n)$ and $\beta(t)R\beta(t_n)$.

As before, two states in two different transition systems are equivalent if they are in such a relation $R$ defined over their disjoint union.

It is easy to see that if two states are equivalent under a branching bisimulation $R$, then they are observationally equivalent. The inverse is not true, as can be seen in the example in Figure 9.4 (page 166). By constructing the image of $\phi_r$ of this transition system, it is clear that states 1 and 1' are observationally equivalent. But they cannot be equivalent under a branching bisimulation. Suppose that there did exist a branching bisimulation $R$ such that $1R1'$. Since there exists a transition $1 \rightarrow a \rightarrow 5$, there must exist a path whose source is 1' and whose last transition $t$ with label $a$ satisfies: $1R\alpha(t)$ and $5R\beta(t)$. Hence $1R2'$. But since there exists a transition whose source is 1, labeled $b$, state 2' must be the source of a path labeled $\tau b$, which it is not.

Section 8.2 presented the close relation between bisimulation and the equivalence induced by the family of homomorphisms saturating Hennessy–Milner logic. A similar relation exists between branching bisimulation and the equivalence induced by a family of homomorphisms defined below.

Recall that in proposition 8.11 the homomorphisms saturating over Hennessy–Milner logic are characterized as those that saturate the src operator.

The $\text{src}^\tau$ operator of type $\tau \rightarrow \sigma$ is the least fixpoint of equation

$$X = \text{src}(Y) \cup (\tau)X,$$

where $(\tau)X$ is an abbreviation of $\text{src}(T_\tau \cap \text{in}(X))$ and $T_\tau$ is the set of transitions labeled by the invisible action $\tau$.

In a labeled transition system $A = \langle S, T, \alpha, \beta, \lambda \rangle$, $\text{src}^\tau_A(Y)$ is equal to the set of states $s$ that are the source of a (possibly empty) path, composed only of transitions labeled $\tau$ and whose target is the source of a transition in $Y$.

The homomorphisms that saturate src$^\tau$ then play, for branching bisimulation, the rôle that homomorphisms saturating Hennessy–Milner logic play for bisimulation. In particular is the following result, analogous to proposition 8.3. The proof is immediate.

**Proposition 9.1** If a surjective homomorphism of labeled transition systems

$$h : A = \langle S, T, \alpha, \beta, \lambda \rangle \rightarrow A' = \langle S', T', \alpha', \beta', \lambda' \rangle$$

saturates src$^\tau$, then the relation

$$R = \{ \langle s, h_\sigma(s) \rangle \mid s \in S \}$$

is a branching bisimulation.
Figure 9.3 A third transition system with invisible transitions.

Figure 9.4 States 1 and 1' are not observationally equivalent and are not equivalent under branching bisimulation.
Chapter 10

Software tools

Since the verification of transition systems properties and the comparison of transition systems are algorithmically feasible, it is natural that software verification tools have been developed to implement these algorithms.

There are a large number of existing tools. The purpose here is not to catalogue them (an annual international colloquium, called Computer Aided Verification focuses on this subject, the first held in 1979 in Grenoble, France [82]), but to discuss some basic choices that must be made.

Some tools allow the verification of the validity, over a given transition system, of formulas in a particular temporal logic. One of the oldest, EMC, developed by Clarke and his team, uses CTL [24]; Cesar [77] and its successors, developed at Grenoble, use Sifakis’s logic, and MEC [5], developed at Bordeaux, France, implements Dicky’s calculus. Others focus on the computation of equivalence, as does AUTO [30], developed at Sophia-Antipolis, France, by Boudol and his team, or ALDEBARAN [44] of Fernandez at Grenoble. Others offer the two possibilities, such as the ‘Concurrency Workbench’ [25].

Another choice that must be made when designing such tools is the language used to describe processes. It can be simply transition systems, or higher-level languages, such as process algebras, protocol description languages such as ESTELLE [22] or LOTOS [18], real-time languages such as ESTEREL [14] or LUSTRE [51], or special-purpose languages. It is not difficult, using compilation techniques, to generate transition systems from programs written in these languages.

Should these tools be used for non-trivial systems, it is important to know the size of the transition systems that these tools can manipulate. This question is fundamental: for example, a simple exclusion algorithm with four processes is represented by a transition system with 65 000 states and 260 000 transitions [47]. A system used in an industrial situation would obviously be much larger. Even with sophisticated representation techniques for transition systems, the size limits imposed by existing machines with large memories are easily reached. This essential problem is being addressed by much current work in formal verification methods.
It is important to relate the possibilities offered by these tools—their functionality, not their performance—with the real verification needs in the domains where they are used: communication protocols, reactive systems, logical circuits, automata and control systems, etc. It is the only way to ensure that theoretical concepts being developed about transition systems and implemented in these tools actually have practical value, and, if need be, to develop from these experiments other, better-suited, theoretical concepts. For example, here are two points in which work is still being done, but where theory is going to have to advance before it can contribute to real verification problems:

- the explicit introduction of time in transition systems (length of actions, temporal intervals of the execution of a transition) in order to validate not only the ‘logical’ properties of a program, but also its performance through time (execution time, reaction delays to events);
- the definition of new types of equivalences generalizing the equivalences modulo an observation criterion, allowing the comparison of transition systems labeled by different alphabets in order to take into account the concept of implementation: what kind of relation between transition systems formalizes the intuitive concept of the implementation of a specification?

Finally, throughout this work it was supposed that the hypothesis of an execution is a sequence, finite or infinite, of occurrences of actions, local or global. In ‘truly’ parallel systems, the set of occurrences of actions making up an execution is no longer necessarily totally ordered. Theoretical models other than transition systems are then necessary. Of course, some of these models already exist, others are being developed, but to discuss them seriously would require another book. This transition brings us to the state labeled End.
Bibliography


Index

$[a]F$, 52
$\langle a \rangle F$, 52
$\square F$, 50, 60, 108
$\Diamond F$, 50, 60, 108
$\approx_{\nu}$, 124
$A \models F$, see satisfiability relation

A[U], 56, 76
AX, 56
action, 7–8
  global, 27–8
  invisible, 21, 159
  null, 12, 25
  observable, 159
additive
  function, 79
algebra
  Dicky, 67
  Hennessy–Milner, 67
process, 20
subset, 66, 88
algorithm
  bisimulation, 144
  fixpoint computation, 98–108
    improved linear, 106
  linear, 103
  quadratic, 99
  Peterson, 14, 17, 29
  Tarjan, 70, 113–5
$\alpha$, 5–6
alphabet, 7
alternating bit, 35–46
asynchronous, see interpretation
$\beta$, 5–6
bisimulation, 133–49
  algorithm, 144
  autobisimulation, 142
  bidirectional, 146
  branching, 164
  generated by a relation, 140
  greatest, 138
calculus
  $\mu$-calculus, 88
  Dicky, 91–4
CCS, 20
connected component, 111
continuous function, 80
CSP, 34
CTL, see logic
CTL*, see logic
deadlock, 46
duality, 85, 93, 96
E[U], 56, 76
EX, 56
$\varepsilon$, 6
equivalence (transition system)
  bisimulation, see bisimulation
branching bisimulation, 164
homomorphism, 124, 131
observational, 159–63
trace, 148–54
event, 7–8
in invisible, 21
null, 10, 12
fairness, 59
strong, 60
weak, 60
fairness condition, 59, 61, 70, 108
fixpoint, 80, 90, 92
and indistinguishability, 120, 128
dual, 86
of a monotone function, 80
scalar, 82
vectorial, 84
formula, 47, 52, 56, 63, 69
characteristic, 121
closed, 90
path, 48, 58, 70
state, 53–4, 58
transition, 53–4
function
additive, 67–8, 79
continuous, 80
monotone, 79
homomorphism, 8, 135
saturating, 122, 131, 165
surjective, 9
in, 54
indistinguishability, 116
and fixpoint, 120
interaction, 27, 33
interpretation
asynchronous, 24
of a μ-term, 90
of a term, 69
synchronous, 24
isomorphism, 9
label, 7–8

λ, 7
Liv, 108, 110
livelock, 46
logic
adequate, 118
CTL, 56–7, 70, 75
CTL*, 57–9, 76, 164
Dicky, 54–5, 119, 128
Hennessy–Milner, 52–4, 91, 128, 133, 138, 165
linear temporal, 48, 60
propositional, 47
Sifakis, 57
statement of properties, 46
strongly adequate, 126
Wolper, 51
marking, 8–9
modeling, 10, 34
monotone
function, 79
operator, 88, 92
μ-term, 88–90
μ-calculus, 88
μσ, 8
μτ, 8
mutual exclusion, 14, 17, 30
N, see next
next, 48
observation, 159
criterion, 163
operator, 63, 66, 75–6, 116
monotone, 88, 92
out, 54
parameter, 8, 35–7
state, 8, 47, 53
transition, 8, 16, 36–8, 48, 53
path, 6
empty, 6, 49
infinite, 6, 49
maximal, 56, 108
Petri net, 17
product (transition system)
  free, 26
  synchronous, 28, 33

quotient, 9
  indistinguishability, 117, 125, 128

regular expression, 7
rendezvous, 34

satisfiability relation, 47–8, 51–4, 56, 58, 63
saturation, 122
semantics, 130
source (of a transition), 5
src, 54, 146, 165
state, 5
  global, 26
  initial, 10, 13, 14, 30, 36–7
synchronization, see product
  constraint, 28
  vector, 27–8
synchronous, see interpretation
system
  interacting, 27
  synchronous, 27
system of equations, 74, 76–7, 82, 94–8, 110

\( T_a \), 8
\( T^\omega \), 6
\( T^+ \), 6
\( T^* \), 6

target (of a transition), 5
\( \tau \), see action, invisible
term, 69
  \( \mu \)-term, 88–90
  closed, 69
  interpretation of a, 69
  signed, 92
tgt, 54
theorem
  Knaster-Tarski, 79
trace, 148–54
  accepting, 150
  extended, 154
  maximal, 151
  multitrace, 150
  of a path, 7
  refusing, 151
transition, 5, 168
  global, 26
transition system, 5
  irreducible, 125
  labeled, 7
  parameterized, 8
  simple, 8

U, see until
until, 48, 73, 95
  strong, 75
  weak, 75

verification, 70
C.A.R. Hoare, Series Editor

RYDEHEARD, D.E. and BURSTALL, R.M., Computational Category Theory
SLOMAN, M. and KRAMER, J., Distributed Systems and Computer Networks
SPIVEY, J.M., The Z Notation: A reference manual (2nd edn)
TENNENT, R.D., Principles of Programming Languages
TENNENT, R.D., Semantics of Programming Languages
WATT, D.A., Programming Language Concepts and Paradigms
WATT, D.A., Programming Language Processors
WATT, D.A., WICHMANN, B.A. and FINDLAY, W., ADA: Language and methodology
WELSH, J. and ELDER, J., Introduction to Modula 2
WELSH, J. and ELDER, J., Introduction to Pascal (3rd edn)
WELSH, J., ELDER, J. and BUSTARD, D., Sequential Program Structures
WELSH, J. and HAY, A., A Model Implementation of Standard Pascal
WELSH, J. and McKEAG, M., Structured System Programming
WIKSTRÖM, Å., Functional Programming Using Standard ML
Finite Transition Systems
André Arnold

Presenting a synthesis of recent works in the field, this book covers transition systems and the derived theoretical tools dealing with the semantics of systems of processes. Finite transition systems constitute one of the formalisms used to describe systems of processes. This formalism, although mathematically simple, can model most of the properties of such systems, and so plays an important role in the study of their semantics. Several theoretical tools, based on this formalism, have been developed, including equivalence relations between transition systems and languages stating their properties.

Originating from a course in France, the book will be suitable for use on advanced undergraduate courses in Computer Science, and also by researchers and professional software designers.