Structural Operational Semantics for Synchronous Languages

Simone Tini

March 2000

Address: Corso Italia 40, 56125, Pisa, Italy
Tel: +39-050-887267 — Fax: +39-050-887226
E-mail: tini@di.unipi.it
Home page: http://www.di.unipi.it/~tini
Thesis supervisor:
Prof. Andrea Maggiolo-Schettini
Abstract

In this thesis we demonstrate that Plotkin’s Structural Operational Semantics applies well to the class of synchronous languages.

We provide interpretations in terms of labeled transition systems (LTSs) for the imperative language Esterel, the constraint based paradigm Timed Default Concurrent Constraint Programming (tdccp) and the visual formalism Statecharts. These LTS interpretations describe the behavior of programs in terms of reactions from *global* configurations to global configurations. We prove that from our LTSs we can recover existing operational semantics of the considered languages. We prove the property of congruence of some well know behavioral equivalences, by exploiting the format of our semantics rules when this is possible. Moreover, we give an axiomatization over Esterel and we prove that it is sound and complete modulo bisimulation.

We provide also “distributed” interpretations for Esterel and tdccp. Namely, we provide semantics describing the behavior of programs in terms of reactions from *distributed* configurations to distributed configurations. We show that distributed interpretations may be used for improving debugging of programs, because they allow to isolate parts of programs causing the violation of given properties. Finally, we show that distributed interpretations for Esterel may be used to optimize the hardware implementation of the language, namely to remove redundant latches from circuits synthesized from programs.
Acknowledgments

First of all, I wish to thank my supervisor Andrea Maggiolo-Schettini. His guidance and suggestions have been essential during my PhD studies. Andrea supported me in all possible ways.

Thanks go also to Adriano Peron for collaboration and friendship.

I thank also my external reviewers, Luca Aceto and Rance Cleaveland, for their precious comments, that helped me to improve the thesis.

I would also like to thank my roommates Chiara Bodei, Davide Marchignoli and Fabio Martinelli.

Finally, let me switch to Italian ... per ringraziare i miei genitori, Artemia e Lucio, mia sorella, Serena, e mia nonna, Maria, per avermi sempre sostenuto in questi anni.
Contents

1 Introduction ................................................................. 1
  1.1 Structural operational semantics ..................................... 1
     1.1.1 Equivalences .................................................... 2
     1.1.2 Rule formats .................................................... 3
     1.1.3 Interleaving versus true concurrency ....................... 4
  1.2 Reactive systems and synchronous languages ..................... 5
     1.2.1 Implementation ................................................ 6
     1.2.2 Verification ................................................... 7
     1.2.3 Causality versus modularity ................................ 8
  1.3 SOS and synchronous languages .................................... 9
     1.3.1 Classical interpretations .................................... 9
     1.3.2 Distributed interpretations ................................. 12
  1.4 Organization of the thesis ......................................... 14

2 Background ................................................................. 17
  2.1 Labelled transition systems ....................................... 17
  2.2 Equivalences and rule formats .................................... 19

3 A constructive SOS for Esterel ......................................... 25
  3.1 An overview of Esterel ............................................. 25
     3.1.1 Syntax and informal semantics ............................. 26
     3.1.2 The circuit semantics ...................................... 28
  3.2 The labeled transition system .................................... 32
  3.3 Correspondence between SOS and circuit semantics ............. 45

4 An axiomatization of Esterel .......................................... 55
  4.1 The axiomatization ................................................. 55
  4.2 The proof of completeness ....................................... 69

5 A distributed interpretation of Esterel ............................ 79
  5.1 The proved transition system .................................... 80
  5.2 Locality trees and enabling trees ................................ 86
  5.3 An application to circuit optimization .......................... 91
5.4 An application to verification .............................................. 101

6 SOS and causal semantics for tdccp ........................................ 107
  6.1 An overview of tdccp ...................................................... 108
  6.2 The labeled transition system ........................................... 112
  6.3 Correspondence with the original semantics ....................... 116
  6.4 The causal semantics .................................................... 120
  6.5 Local variables .......................................................... 127

7 SOS for Statecharts ......................................................... 135
  7.1 An overview of Statecharts .............................................. 136
  7.2 Statechart terms .......................................................... 138
  7.3 The labeled transition system ......................................... 144
  7.4 Correspondence between SOS and step semantics ................ 149
  7.5 Equivalences .............................................................. 151

8 Conclusions ........................................................................... 157

Bibliography ............................................................................. 161
Chapter 1

Introduction

Synchronous languages have been proposed for programming and specifying reactive systems, namely systems which maintain an ongoing interaction with their environment at a rate controlled by this. Structural operational semantics (SOS) provides a framework to give semantics to programming and specification languages. In particular, because of its intuitive appeal and flexibility, SOS has found considerable application in the study of the semantics of concurrent languages. In this thesis we apply the SOS approach to synchronous languages.

1.1 Structural operational semantics

The term operational semantics appeared in the literature in [61, 68]. Operational semantics describes the behavior of programs in terms of transitions between computation states. A computation state (or configuration) consists of the program itself and some auxiliary data which represent the store or the data structures on which the program works. A transition indicates the move from one configuration to another one. Behaviors may be represented graphically by means of transition systems, namely oriented graphs with nodes representing configurations and arcs representing transitions between them. An arc may be labeled by additional information on the activity performed by the underlying transition. In this case, transition systems are called labeled transition systems (LTSs). Finally, a run of a program (computation) is represented through a sequence of nodes where each node is connected to the next one by an arc.

Operational semantics is close to intuition and mathematically simple, and provides a guideline to language implementation, because operational definitions provide abstract machines and highlight implementation issues like distribution of resources or allocation of data structures. The main criticism to operational semantics is that it does not provide the meaning to programs directly, in the sense that the execution sequence of a program is needed to give its semantics. So, the definition of the semantics of a given language may become unstructured, and compositionality
may be lost.

A semantics is compositional if the meaning of a program is a suitable combination of the meaning of its components (subprograms), so that only the meaning of the basic constructs of a language and the meaning of their compositions are needed to completely define the semantics of the language. In general, if compositionality is lost, then the semantics of a language cannot be finitely expressed, because any program requires an ad hoc definition.

The original idea to define compositionally the operational semantics of languages is in [8]. The main novelty of Plotkin’s structural operational semantics (SOS) [78] is the logically based way in which transitions are deduced, by inducing on the syntactic structure of the language itself. Configurations are programs, expressed according to the abstract syntax of the considered language defined by means of a BNF-like grammar, plus, possibly, auxiliary data which may or may not affect the deduction of transitions. Inductive definitions are given by sets of rules of the form

\[
\text{Premises} \\
\text{Conclusion}
\]

whose intuitive meaning is that whenever the premises are satisfied, the conclusion is satisfied as well. In an operational framework, premises are sets of transitions, related by logical connectives like and and not, and the conclusion is a transition.

1.1.1 Equivalences

Labeled transition systems may describe the operational behavior of programs in great detail. In order to abstract away from irrelevant information on the way in which programs compute, a wealth of notions of behavioral preorder and behavioral equivalence have been studied in the literature. We recall that a preorder is a relation that is reflexive and transitive, while an equivalence is a relation that besides being reflexive and transitive is also symmetric.

Preorders and equivalences are relations defined over LTS nodes, which correspond to programs. Intuitively, two programs are related by a preorder if one is an approximation of the other. Two programs are related by an equivalence if they cannot be distinguished. More precisely, almost all equivalences proposed are based on the idea that two programs are equivalent whenever no external observer can distinguish them. That is, equivalences abstract away from aspects of the internal structure of programs that cannot be observed. The existence of many notions of equivalence is due to the large number of properties of programs which may be relevant, and therefore observable, or negligible.

One of the most known notions of behavioral equivalence is that of bisimulation equivalence [71, 74]. Bisimulation relates programs having exactly the same branching structure. That is, two programs are bisimilar if, whenever the first may perform an activity, the other may do as well and reached configurations are still bisimilar,
and conversely. Coarser notions of equivalence have been defined to release this requirement to various degrees. We refer to [36, 37, 38, 39] for a systematic treatment of these notions.

Equivalences have provided a fairly successful method, sometimes called *equivalence reduction*, for verifying program behaviors. Actually, one writes a program to realize a given specification. Verifying a program means to show that the program is behaviorally equivalent to, or in some sense a suitable approximation of, the specification.

A very interesting property of equivalences is that of *congruence*: an equivalence is a congruence if the constructs of the language preserve the equivalence, i.e. if equivalent programs cannot be distinguished by any context. If an equivalence is a congruence then equivalent programs can be freely exchanged in whatsoever context.

An interesting issue is to provide axiomatizations of equivalences. Given a language, an axiomatization consists of a set of *conditional axioms* of the form

\[ \text{Premises} \implies \text{Conclusion} \]

defining an equality relation over programs. This relation is usually denoted with \("\equiv\", is reflexive, symmetric and transitive, and is preserved by all language constructs. Namely, \("\equiv\"\) is a congruence. Every rule of the form described above permits to deduce that two programs are related by \("\equiv\"\) from the fact that other programs are. An axiomatization if *sound* modulo an equivalence if all programs related by \("\equiv\"\) are equivalent. An axiomatization is *complete* modulo an equivalence if all equivalent programs are related by \("\equiv\"\). Axiomatizations sound and complete modulo a given equivalence may be used both for transformation of programs and for proof by rewriting.

### 1.1.2 Rule formats

SOS has been applied as a formal tool to establish results that hold for classes of process description languages, like CCS [71], \(\pi\)-calculus [72], CSP [53], PA [7]. The idea was to generalize well-known results in the field of process algebras, so to develop a meta theory for process description languages. In fact, there exist results in the field of process description languages that depend on general semantic properties of language constructs, and do not depend on the particular language chosen. Since the pioneering work of de Simone [92], the concept of a *rule format* has played a major rôle in developing a meta theory for process description languages. A survey on the main rule formats proposed in the literature can be found in [3].

In practice, a rule format imposes syntactic constraints on the form of the rules that are allowed to deduce transitions. A central issue is to define rule formats ensuring that some important properties hold, such as that a behavioral equivalence is a congruence. As examples, both *de Simone format* [92] and the so called *GSOS format* [19] guarantee that bisimulation is a congruence.
Another application of rule formats has been showed in [2] and [1], where algorithms have been proposed to derive from an arbitrary language belonging to a particular subclass of GSOS languages, a superset of this language and an axiomatization sound and complete modulo bisimulation on the language obtained (with GSOS language we mean a language whose semantics is defined by means of rules in GSOS format).

1.1.3 Interleaving versus true concurrency

Asynchronous models of concurrency are based on the assumption that system components running in parallel proceed at different rates. In general, one cannot predict the relative temporal ordering between actions of different components. (In this setting, an action is the activity performed during a transition.)

Two different approaches to asynchronous concurrency have emerged: *interleaving* and *true concurrency*. Interleaving simulates concurrency of actions with their arbitrary interleaving, thus reducing concurrency to sequentiality plus nondeterminism. True concurrency interprets concurrency as a basic phenomenon, following the idea that some properties of a concurrent system should be described by clearly distinguishing between concurrency and nondeterminism. In the truly concurrent approach, properties of systems expressed by relations over their actions, such as relations of *locality* and *causality*, have been extensively studied. The relation of locality relates two actions only if these are performed by a sequential component of the system. The relation of causality relates two actions only if performing one of them is a necessary condition for performing the other.

Interleaving describes system behavior in terms of *global* transitions taking global configurations to global configurations, while true concurrency describes system behavior in terms of *distributed* transitions taking distributed configurations to distributed configurations.

As thoroughly argued in [30], SOS is well suited to define both interleaving and truly concurrent semantics for concurrent languages. Semantics for asynchronous process description languages are usually given in SOS style, so that behaviors of processes are described by LTSs. In the interleaving approach, labels of arcs are actions representing either interaction with the environment or internal computational tasks, and concurrency of two actions is reflected by the existence in the LTS of the possibility of performing the actions in both the possible orders. A possible choice to describe transitions as distributed transitions is to enrich labels of arcs with encodings of their deduction trees (*proofs*) from which one can determine which components of a system are involved in performing actions. This is the approach of [23, 26]. LTSs with labels enriched with encodings of transition proofs are called proved transition systems (PTSs) in [28].
1.2 Reactive systems and synchronous languages

In [45] computing systems have been classified into transformational, interactive and reactive [50], on the basis of the way in which they interact with their external environment. A transformational system accepts inputs, performs transformations on them and produces outputs at the end of a terminating computation. Its behavior can be viewed as a function from an initial configuration to a final configuration or result. On the contrary, the rôle of interactive and reactive systems is not to produce a final result, but to maintain an ongoing interaction with their environment, by continuously responding to external prompts. This process is not expected to terminate, and some inputs may depend on intermediate outputs.

A transformational system and its environment act sequentially, so that their interaction consists in three successive activities: first the environment prompts an input, then the system performs its computation (independently from possible changes of the environment in the meantime) and, when the computation terminates, the environment uses the output. Interactive and reactive systems act concurrently with their environment which may present new inputs and attempt to use outputs of the system at the same time the system tries to read and write them. While interactive systems determine the pace of the interaction with the environment, namely they are the leader of the interaction, reactive systems react to their environment at a speed determined by this. Typical interactive systems are data bases, operating systems, networking, etc. Reactive systems are prominent in industrial process control, airplane or automobile control, signal processing, audio or video protocols, embedded systems, etc.

The life of a reactive system is divided into instants, namely moments in which the system is stimulated by the environment and must react. Every reaction of the system produces a response which is expected by the environment within a bounded time, at least within the next instant, so that reactions do not overlap. So, reactive systems are required to be faster than their environment, otherwise they may miss some deadlines, or fail to respond to, or sense important events. For reactive systems it is therefore relevant to argue about their responsiveness, namely their ability to react in a bounded amount of time to prompts from the environment. This implies that formalisms for programming and specification of reactive systems cannot completely abstract time of observable actions.

Synchronous languages [10, 17] are formalisms of this kind. They are based on Berry's synchronous hypothesis which states that reactions of a system are instantaneous, in the sense that outputs from the system are available as soon as inputs from the environment are. This hypothesis is indeed an abstraction and relies on the assumption that the system is faster than its environment, namely that the environment is invariant w.r.t. the system when this reacts.

The synchronous hypothesis simplifies reasoning about reactive systems, since it presents at least the following advantages:
• The synchronous hypothesis reconciles concurrency and determinism. The construct of parallel composition does not give rise to nondeterminism, namely the various components of a system act synchronously, and their actions cannot arbitrarily interleave.

• Time is treated as any other external event, namely no special construct to deal with physical time is needed. In fact, the notion of physical time is replaced by a notion of ordering among events.

• The reaction time is known and does not depend on the implementation.

• The timed behavior is abstract, allowing further refinement without having to redo the timing, that is, if a certain reaction is refined to include several subreactions, the timed behavior is not changed since the subreactions take no time, and so the overall reaction takes no time.

Synchronous languages are more powerful than the classical synchronous models, like SCCS [69] and Meije [22], because they offer primitives to deal with negative information, namely to instantaneously test absence of inputs from the environment (see [17]).

Most reactive systems involve two kinds of activities: control handling and data handling. Control intensive and data intensive applications call for different programming and specification techniques. Synchronous languages may be classified on the basis of the kind of application they are tailored to:

• State based languages, like the imperative language Esterel [18], the visual languages Argos [66] and SynchCharts [4], and the visual nondeterministic formalism Statecharts [48], are tailored to applications where the control handling aspects are predominant.

• Data flow languages, like the declarative languages Lustre [24] and Signal [41], are tailored to applications where the data handling aspects are predominant.

• Constraint based languages, like the paradigms Timed Default Concurrent Constraint Programming [86] and Timed Concurrent Constraint Programming [84], are tailored to applications where both control handling and data handling aspects are relevant.

1.2.1 Implementation

In order to have reaction times very close to the theoretical zero of the synchronous hypothesis, synchronous languages require very efficient implementations. Implementations in input/output Finite State Machines (i/o FSMs) and implementations in hardware circuits have been proposed.
Implementations in i/o FSMs are realized by mapping programs to i/o FSMs with FSM states corresponding to program configurations, and FSM transitions corresponding to program reactions. The execution times of apparently complex tasks may really be very low, since both the internal control transmission and communication among components running in parallel corresponding to the treatment of an external prompt are compiled into a single FSM transition. The synchronous hypothesis avoids the state explosion which, as it is well known, is unavoidable in asynchronous formalisms.

Also implementation of synchronous programs in circuits guarantees short reaction times. Among the papers following this approach, we mention [32, 82, 14, 16, 77]. States of circuits correspond to program configurations, and one-clock executions of circuits correspond to program reactions. Circuits can be either synthesized from FSMs corresponding to programs by means of classical techniques, as in [77], or synthesized compositionally w.r.t. to the structure of programs, as in [14, 16]. Compositional synthesis means that circuits implementing programs are obtained by suitable compositions of circuits implementing subprograms. In [89, 90] it has been proved that compositional synthesis compares favorably w.r.t. classical synthesis as regards the tradeoff between number of latches and size of the logic of the obtained circuits.

1.2.2 Verification

As regards the verification of synchronous programs, we recall that, as argued in [79], most interesting properties of reactive systems are safety properties. Intuitively, a safety property expresses a specification like “something will never happen”. Liveness properties, expressing specifications like “something will eventually happen”, are not interesting in the field of reactive systems, where responsiveness is a central issue. Interesting properties like “something will eventually happen within $n$ units of time” can be translated into safety properties.

Classical verification techniques, like bisimulation reduction and developing of proof systems, can be adapted to synchronous formalisms, as demonstrated in [20] and [59, 60], respectively. A technique explicitly developed for verifying safety properties of synchronous programs is that of observer monitoring [46]. The main idea of the method consists in translating a safety property $\phi$, expressing logic relations over inputs and outputs of a given program $P$, into an observer program $\Omega_\phi$. (As an example, a compositional translation of a very large class of propositional linear-time logic safety properties into Ésterel has been proposed in [55].) The observer program $\Omega_\phi$ runs in parallel with $P$ and, at each instant, observes its behavior. Namely, at each instant $\Omega_\phi$ observes communications between $P$ and the environment. Program $\Omega_\phi$ is able to detect when $P$ has violated $\phi$ and, in this case, it emits an “alarm” signal. So, $P$ violates $\phi$ if and only if the parallel composition of $P$ and $\Omega_\phi$ emits this alarm signal.
1.2.3 Causality versus modularity

Synchronous languages offer both primitives to test for the instantaneous presence/absence of binary signals and primitives to produce such signals. Both communication among program components running in parallel and communication between program and external environment are achieved through these signals. So, at a given instant, the status of a given signal may depend on the status assumed, at the same instant, by other signals.

In [34] it has been proved that a compositional semantics of a synchronous language cannot enforce both causality and modularity. Causality is the property that every output produced by a program is causally justified by inputs prompted by the environment, namely outputs cannot justify each other. Modularity is the property that programs can be viewed as “black boxes”, in the sense that their behavior is described by observing their input/output interface and by disregarding their internal structure. In particular, a modular semantics does not keep track of causal dependencies between input and output signals. To see the reason for which synchronous languages cannot be endowed with a causal and modular semantics, let us consider the following Esterel program $P_1$:

$$\text{present } a \text{ then emit } b \text{ else nothing end}$$

$$\text{||}$$

$$\text{present } c \text{ then emit } d \text{ else nothing end}$$

which is the parallel composition of two programs: one tests for the presence of the binary signal $a$ and, if $a$ is present, then it produces the signal $b$, while the other tests for the presence of the signal $c$ and, if $c$ is present, then it produces the signal $d$. The reaction to the environment that prompts both $a$ and $c$ is described by a modular semantics as the observation of the pair $(\{a, c\}, \{b, d\})$, where the first component describes the input, namely the set of signals received by $P_1$, and the second component describes the output, namely the set of signals produced by $P_1$.

Let us consider now the following program $P_2$:

$$\text{present } a \text{ then (present } c \text{ then (emit } b \text{ || emit } d) \text{ else nothing end)}$$

$$\text{else nothing end}$$

Also the reaction of $P_2$ to the environment prompting $a$ and $c$ is described by the pair $(\{a, c\}, \{b, d\})$. In practice, modular semantics do not take into account that $P_1$ requires $a$ to produce $b$ and $c$ to produce $d$, while $P_2$ requires both $a$ and $c$ to produce $b$ and $d$.

Now, let us assume the program $P_3$:

$$\text{present } b \text{ then emit } c \text{ else nothing end}$$

which produces $c$ if $b$ is present.
According to a causal semantics, \( P_3 \) discriminates between \( P_1 \) and \( P_2 \). In fact, let us consider program \texttt{signal\ b,\ c in\ P_1 \parallel\ P_3\ end} where both \( b \) and \( c \) are local signals, namely they can be neither produced nor received by the environment. If the environment produces \( a \) then a parallel component of \( P_1 \) produces \( b \), \( b \) is received by \( P_3 \) which produces \( c \), \( c \) is received by the other component of \( P_1 \) which produces \( d \). Let consider now program \texttt{signal\ b,\ c in\ P_2 \parallel\ P_3\ end} and the same input from the environment. Neither \( P_2 \) nor \( P_3 \) are able to produce any signal.

A modular semantics describes the reaction of \( P_3 \) to an environment prompting \( b \) as the observation of the pair \((\{b\}, \{c\})\). Moreover, by joining observations \((\{a\}, \{b\})\) and \((\{b\}, \{c\})\), one obtains the observation \((\{a\}, \{d\})\). This means that both \texttt{signal\ b,\ c in\ P_1 \parallel\ P_3\ end} and \texttt{signal\ b,\ c in\ P_2 \parallel\ P_3\ end} are able to produce \( d \) when receiving \( a \). Namely, signal causality of \texttt{signal\ b,\ c in\ P_2 \parallel\ P_3\ end} is lost.

Since causality is needed to have behaviors close to intuition, semantics of synchronous languages sacrifice modularity.

### 1.3 SOS and synchronous languages

In this thesis we are interested in giving semantics is SOS style for synchronous languages. We consider the imperative language Esterel, the constraint based paradigm Timed Default Concurrent Constraint Programming (\texttt{tdccp}) and the visual formalism Statecharts.

We will consider both “classical interpretations” of synchronous languages, i.e. semantics interpreting program configurations as global configurations, and “distributed interpretations”, i.e. semantics interpreting program configurations as distributed configurations.

#### 1.3.1 Classical interpretations

According to our classical interpretations of synchronous languages, LTS nodes correspond to program configurations, LTS arcs correspond to reactions, and LTS labels carry information on interaction between programs and their environment. More precisely, labels give information on the status of communication signals and on causal dependencies among them. Information on the status of signals permits to observe the input/output behavior of programs, while information on signal causality is needed to achieve compositionality.

Semantics in SOS style for Esterel have been already proposed by other authors in [18, 93, 16]. While both semantics of [18] and [93] deal with the non constructive version of Esterel, we treat here the constructive version of the language. Also in [16] the constructive version of the language is considered. We will describe the differences between our SOS approach and that of [16] in Chapter 3.
The notion of constructiveness has been introduced by Berry in [16]. Intuitively, a program is constructive if the status of local and output signals can be determined without making any assumption on them. Namely, the status of local and output signals must be computed by a fact-to-fact propagation starting from the status of input signals. As an example, let us consider the Esterel statement $P$:

\[
\text{present } a \text{ then emit } a \text{ else emit } a \text{ end}
\]

which produces the output signal $a$ if either the input signal $a$ is present or the input signal $a$ is absent. The statement $P' = \text{signal } a \text{ in } P \text{ end}$ is obtained by identifying the input signal $a$ and the output signal $a$ of $P$. Namely, $a$ is local to $P'$, and $P'$ can sense $a$ if and only if $P'$ produces it. In the nonconstructive version of Esterel, $P'$ is a program with a well defined semantics: $P'$ emits $a$, since $a$ is emitted both if it is present and if it is absent. In the constructive version of Esterel, $P'$ has no meaning. In fact, we cannot determine whether $a$ is produced, unless we assume the status of $a$. In the nonconstructive version, the production of $a$ by the then branch of \text{present} justifies the choice of this branch. In the constructive version, this behavior is rejected, because it seems to be counterintuitive. In fact, it seems that some information flows backward w.r.t. the control flow.

In practice, in the nonconstructive version of Esterel, the status of signals is computed by interpreting signals as boolean variables and programs as systems of equations of formulas of the boolean logic. So, $P'$ corresponds to the equation $a = a \lor \neg a$, where $a \lor \neg a$ evaluates to \text{true}. In the constructive version of Esterel, programs are interpreted as systems of equations of formulas of the constructive boolean logic [16]. This logic rejects the middle law $a \lor \neg a = \text{true}$, so that we can infer that $a \lor \neg a$ evaluates to \text{true} only if we are able to prove that $a$ is either \text{true} or \text{false}. Since $a$ is not an input signal, we do not know the status of $a$, and we can infer neither that it is \text{true}, nor that it is \text{false}. So, the value of $a \lor \neg a$ remains unknown.

In [16] the so called circuit semantics of Esterel has been provided. This semantics interprets programs as sequential circuits, which could be viewed also as the hardware implementation of the language. Sequential circuits are synthesized compositionally w.r.t. the structure of programs. In [16] and [91] it is explained that, when a circuit is considered, the electrical status at which internal and output wires of the circuit stabilize can be determined only by a fact-to-fact propagation, starting from the known electrical status at which input wires are kept stable. In practice, circuit wires (like Esterel signals) can be interpreted as boolean variables, and circuit gates (like Esterel programs) can be interpreted as equations of formulas of the constructive boolean logic.

Our structural operational semantics for Esterel agrees with the circuit semantics of [16]. In fact, we prove that from our LTS describing the behavior of programs we are able to deduce the input/output behavior of circuits corresponding to such programs.
1.3. **SOS AND SYNCHRONOUS LANGUAGES**

Since our SOS rules defining the operational semantics for Esterel are in GSOS format, the bisimulation on Esterel programs is a congruence.

The agreement between LTS and circuit semantics and the fact that bisimulation is a congruence imply that bisimilar programs are distinguished neither by the external environment nor by any Esterel context. For this reason, bisimulation is a reasonable notion of equivalence on Esterel programs.

We characterize the classes of equivalence modulo bisimulation. In fact, we present an axiomatization sound and complete modulo bisimulation on Esterel. This axiomatization could be useful for transformation of programs and for proof by rewriting. To prove the completeness of our axiomatization, we introduce a notion of normal form of programs, we prove that every program can be transformed into a bisimilar normal form by applying axioms, thus reducing ourselves to prove the completeness of our axiomatization over normal forms. The choice of introducing normal forms to prove completeness of axiomatizations is well established in the field of asynchronous languages. Our normal forms differ from those introduced for asynchronous languages because our normal forms admit the construct of parallel composition. In fact, we cannot reduce concurrency to arbitrary interleaving, as it is done in the case of asynchronous languages.

Note that, to our knowledge, no attempt to give axiomatic semantics to synchronous languages has appeared in the literature up to now.

The paradigm tdccp is endowed with an operational semantics given in [86]. We show the agreement between our structural operational semantics and the original semantics of [86], namely we prove that from our LTS we can infer the input/output behavior of programs.

Since our SOS rules defining the operational semantics for tdccp are in de Simone format, the bisimulation on tdccp programs is a congruence.

The agreement between our LTS semantics and the semantics of [86] and the fact that bisimulation is a congruence imply that bisimilar programs are distinguished neither by the external environment nor by any tdccp context. For this reason, bisimulation is a reasonable notion of equivalence on tdccp programs.

We plan to provide an axiomatization for tdccp in the future, by extending the approach adopted to axiomatize Esterel.

We briefly recall here that a Statecharts program is either an i/o FSM, or an FSM with states refined by other FSMs, or a parallel composition of FSMs. FSMs running in parallel may communicate by exchanging signals. Unlike Esterel and tdccp, Statecharts admits nondeterministic behaviors.

Many definitions of semantics for several dialects of Statecharts have been proposed in the literature, after the original one of [51]. Most of them are described and compared in [9]. These semantics describe the behavior of Statechart programs in terms of sequences of sets of fired FSM transitions, called steps.

According to the original operational semantics of [51], a binary signal $a$ may
be both absent and present at a given instant, provided that the step performed at such instant can be divided into a sequence of substeps, called microsteps, and a microstep producing a follows (in the sequence) all microsteps assuming the absence of a, and precedes all microsteps assuming the presence of a. In [80] a semantics has been provided as an improvement w.r.t. to the semantics of [51]. The semantics of [80] guarantees global consistency of signals, in the sense that signals cannot be considered both absent and present at an instant.

Here we present a “step semantics” for Statecharts that slightly differs w.r.t. to the step semantics of [80]. Programs having no behavior according to the semantics of [80] have a well defined behavior according to our semantics. As in [80] we give a procedure to compute steps. While the procedure of [80] does not terminate in some cases, our procedure always terminate and, therefore, guarantees that every program has a well defined meaning.

Then we provide a structural operational semantics in terms of an LTS for Statecharts and we prove the agreement between this semantics and our step semantics. Namely, we prove that from our LTS describing the operational behavior of programs we deduce the input/output behavior of these programs, as determined by the step semantics.

Since Statecharts admits nondeterminism, it may happen that two LTS arcs having the same LTS node as source node have also the same label. For this reason, we believe that it is interesting to consider a hierarchy of preorders and equivalences, coarser than bisimulation, over Statecharts programs.

We exploit the format of the rules defining our LTS for Statecharts to prove that bisimulation is a congruence and to prove that the simulation preorder, the ready simulation preorder and the ready trace preorder are precongruences, namely they are preserved by Statecharts constructs. Finally, we prove by ad hoc reasonings that also the failure preorder and the trace preorder are precongruences.

We will discuss also the interpretations in LTSs of variants of Statecharts given in [97, 98, 58, 62].

### 1.3.2 Distributed interpretations

In general, synchronous programs can be perceived as purely sequential. Parallelism is a logical concept that simplifies programming, since it permits modular development of programs. Semantic models for synchronous languages reflect this idea, since they describe behaviors of programs in terms of reactions from global configurations to global configurations.

In this thesis we advocate semantic interpretation for synchronous languages where concurrency does not disappear, namely program configurations are interpreted as distributed configurations, and program reactions are interpreted as distributed reactions. We do not speak about “truly concurrent” semantics of synchronous languages because, as we have briefly explained in the previous sections,
true concurrency is usually viewed as a counterpart of interleaving, and speaking about interleaving in synchronous languages seems to be a nonsense.

Semantics interpreting programs as distributed programs should not compete with the classical ones, but should offer very concrete views of programs, useful for tasks as the following ones:

- Improvement of hardware implementation. As argued in the previous sections, programs can be implemented in sequential circuits which are synthesized compositionally w.r.t. to their syntactic structure. A semantics interpreting program configurations as distributed configurations could give sufficient information to recover the latch encoding of program configurations and to discover redundancy of latches. So, such semantics could support removal of redundant latches from circuits.

- Improvement of the verification phase. Let us consider the model checking problem for a program $P$ and a safety formula $\phi$. According to the observer monitoring verification technique, an observer program $\Omega_\phi$ is obtained from $\phi$. Now, $\Omega_\phi$ runs in parallel with $P$, observes its behavior, and, when it detects when $P$ has violated $\phi$, it emits an alarm signal $\alpha$. In cases where $\alpha$ is produced, we expect to be able to deduce what actions of $P$ are actually responsible of the violation of $\phi$ from the causality between actions of $P$ and the action of $\Omega_\phi$ producing $\alpha$. This could simplify the debugging. Causality between actions may be induced both by the fact that these actions are performed by sequential components, namely they are sequentialized, or by communications between program components running in parallel (intuitively, an action consisting in producing a signal causes an action that requires the presence of this signal).

- Support to the design of model-based schedulers. The technique of model-based computing [33, 34] has been proposed to develop software for electromechanical systems. The central idea is that models of the various physical components of a system are described by means of declarative or constraint based formalisms (as examples, see [43, 44]) and are used to derive information for tasks such as scheduling, diagnosis and simulation. Declarative synchronous languages, such as Lustre and Signal, and constraint based synchronous languages, such as tdcpp, are well suited for component modeling. Each system component is modeled as a transducer which accepts input and control signals, operates in a given set of modes, and produces output signals. The task of scheduling is to determine input and control signals which must be supplied to the system to cause a certain output. To do this one is interested in knowing the minimal cause of the output considered, namely the minimal set of input and control signals causing it. This causality between inputs and outputs can be recovered from a semantics of the program modeling the system where causality relations over actions performed by the various program
components are highlighted.

In this thesis we propose semantics for Esterel and \texttt{tdccp} which interpret programs as distributed ones, in terms of proved transition systems. We show how the semantics for Esterel supports the removal of latches from circuits compositionally synthesized from programs. Since we are able to remove latches which are not removed by the existing optimization techniques, and, conversely, we do not remove latches removed by means of these techniques, we discuss possible integration between our method and the existing ones. We demonstrate by means of an example how the information given by our semantics for Esterel and \texttt{tdccp} can be exploited to improve the verification phase when the observer monitoring verification technique is adopted. In particular, we show that we are able to isolate program actions causing the violation of a considered property, and that this cannot be done by considering classical semantics.

1.4 Organization of the thesis

In this section we explain how we have organized this thesis.

In Chapter 2 we recall some well known definitions and results developed in the literature of process description languages and of structural operational semantics.

In Chapters 3, 4 and 5 we deal with the imperative language Esterel. In particular, in Chapter 3 we recall the syntax of the language, its informal semantics and its circuit semantics. Then we define our labeled transition system for Esterel and we prove the correspondence between our LTS semantics and the circuit semantics of [16].

In Chapter 4 we present our axiomatization of Esterel and we prove that this is sound and complete modulo bisimulation.

In Chapter 5 we present two distributed interpretations of Esterel. We show how one of them supports the removal of latches generated by the Esterel compiler, and we show by an example how the other supplies a user with information for debugging programs.

In Chapter 6 we deal with the constraint based paradigm \texttt{tdccp}. We recall its syntax and the operational semantics given in [86]. Then, we define our labeled transition system for \texttt{tdccp} and we prove the correspondence between the LTS semantics and the original semantics of [86].

We provide also a distributed interpretation of \texttt{tdccp} which leads to define a causal semantics. We show by an example how this causal semantics supplies an user with useful information for debugging programs. Since a causal semantics for the paradigm Concurrent Constraint Programming [83] (\texttt{ccp}) has been proposed in [42], and since \texttt{ccp} and \texttt{tdccp} have a non empty intersection, we prove that our causal semantics and the causal semantics of [42] coincide on programs belonging to such intersection.
1.4. ORGANIZATION OF THE THESIS

Finally, in Chapter 7 we consider the visual language Statecharts. We recall its syntax, we introduce our step semantics and we compare it with the step semantics of [80]. Then, we define our labeled transition system for Statecharts and we prove the correspondence between the LTS semantics and our step semantics. Finally, we consider a hierarchy of preorders and equivalences, and we prove their precongruence and congruence property, respectively.

In Chapters 5, 6 and 7 we develop ideas expressed originally in [65], [95, 96] and [64], respectively.
Chapter 2

Background

In this chapter we present some basic notions from process description languages theory that are needed in the remainder of this thesis.

We do not introduce in this chapter the languages Esterel, $\text{tdccp}$ and Statecharts which will be introduced in Chapters 3, 6 and 7, respectively.

2.1 Labelled transition systems

We begin by reviewing the model of labelled transition systems [56, 78] which is used to express the operational semantics of many languages.

Definition 2.1.1 A labeled transition system (LTS) is a tuple $\langle S, \text{Act}, \{ \xrightarrow{a} | a \in \text{Act} \} \rangle$ such that:
- $S$ is a set of states;
- $\text{Act}$ is a set of actions;
- $\xrightarrow{a} \subseteq S \times S$ is a transition relation for every $a \in \text{Act}$.

The operational semantics of a language can be naturally expressed in terms of an LTS, where LTS states correspond to configurations of programs, LTS transition relations correspond to atomic evolution steps of programs, and LTS actions describe activities underlying transitions.

States are sometimes called configurations. Actions are sometimes called labels or operations. Sets $S$ and $\text{Act}$ are ranged over by $s$ and $a$, respectively.

Following standard notations, we write $s_1 \xrightarrow{a} s_2$ for $\langle s_1, s_2 \rangle \in \xrightarrow{a}$, and we say that $s_1 \xrightarrow{a} s_2$ is a transition having $s_1$ and $s_2$ as source state the target state, respectively. The state $s_2$ is called the derivative of $s_1$ w.r.t. $a$. We will sometimes identify an LTS with the collection of its transitions.

Given a state $s_1$, we write $s_1 \xrightarrow{a}$ if there exists a state $s_2$ such that $s_1 \xrightarrow{a} s_2$, while we write $s_1 \xleftarrow{a}$ otherwise. Moreover, we denote with $\text{initials}(s_1)$ the set of actions $\{a | s_1 \xrightarrow{a}\}$. 

A sequence $\sigma = a_1 \ldots a_n \in \text{Act}^*$ is a trace of a state $s_0$ if there exist states $s_1, \ldots, s_n$ such that $s_0 \xrightarrow{a_1} s_1 \ldots \xrightarrow{a_n} s_n$. In this case, we write $s_0 \xrightarrow{\sigma} s_n$. Traces describe computations of programs, namely sequences of atomic evolution steps.

The set of derivatives of a state $s_0$ is the set of states that can be reached from $s_0$ by $\xrightarrow{}$, namely $\{ s \mid s_0 \xrightarrow{a} s \text{ for some } a \in \text{Act} \}$. It will be denoted with $ds(s_0)$. The oriented graph $dg(s_0)$ having $ds(s_0)$ as set of nodes and $\{ s \xrightarrow{a} s' \mid s \in ds(s_0) \}$ as set of labelled arcs will be referred to as the derivation graph of $s_0$. Sometimes, we will refer to $dg(s_0)$ as the “part of LTS reachable from $s_0$”.

The graph $dg(s_0)$ describes all computations starting at $s_0$. Sometimes it is useful to have a linearization of all computations that a program configuration may engage in. A possibility is to get the unfolding of the derivation graph of the LTS state representing the configuration, thus yielding a tree of computations. So, the derivation tree of a state $s_0$ is the tree $dt(s_0)$ having a node labelled with $s_0$ as root, and an arc labelled with $a$ from this node to the root of $dt(s)$ for every arc $s_0 \xrightarrow{a} s$ in $dg(s_0)$.

Labelled transition systems are usually defined by means of transition system specifications. Before introducing the notion of transition system specification, we recall some preliminary notions.

Let us consider a countably infinite set of variables $\text{Var}$, ranged over by $x, y$. A signature consists of a set of function symbols, disjoint from $\text{Var}$, together with an arity mapping that assigns a natural number $ar(f)$ to each function symbol $f$. Functions of arity zero are called constants.

**Definition 2.1.2** The set of (open) terms $T(\Sigma)$ over a signature $\Sigma$ is the least set such that:

- each $x \in \text{Var}$ is a term;
- if $f$ is a function symbol and $t_1, \ldots, t_{ar(f)}$ are terms, then $f(t_1, \ldots, t_{ar(f)})$ is a term.

Terms that do not contain variables are called closed terms. The set of closed terms is denoted with $T(\Sigma)$.

The set of open terms $T(\Sigma)$ is ranged over by $t, u$. Given a constant $f$, we write $f$ for $f(\ )$.

The set of closed terms over $\Sigma$ gives the term algebra of $\Sigma$. We recall that, given a signature $\Sigma$, a $\Sigma$-algebra is a pair $(A, \Sigma_A)$, where $A$ is a set called carrier and $\Sigma_A$ is a set of functions $\{ f_A : A^n \to A \mid f \in \Sigma \text{ and } ar(f) = n \}$. Essentially, $(A, \Sigma_A)$ is an interpretation of $\Sigma$. Now, the term algebra of $\Sigma$ is the $\Sigma$-algebra having $T(\Sigma)$ as carrier and, for each $f \in \Sigma$ with $ar(f) = n$, a function mapping closed terms $t_1, \ldots, t_n$ to term $f(t_1, \ldots, t_n)$.

The syntax of process description languages, such as CCS [71], $\pi$-calculus [72], CSP [53], PA [7], is usually expressed in terms of a signature $\Sigma$. The term algebra of $\Sigma$ is called a process algebra and terms in $T(\Sigma)$ are called processes.
2.2. EQUIVALENCES AND RULE FORMATS

A substitution is a mapping $\sigma : \text{Var} \rightarrow T(\Sigma)$. A substitution extends to a mapping from terms to terms as usual, namely, $\sigma(t)$ is the term obtained by replacing occurrences of variables $x$ in term $t$ by $\sigma(x)$. A substitution is closed if it maps variables to closed terms.

A context $C[x_1, \ldots, x_n]$ denotes an open term in which at most the distinct variables $x_1, \ldots, x_n$ may appear. The term $C[t_1, \ldots, t_n]$ is obtained by replacing all occurrences of variables $x_i$ in $C[x_1, \ldots, x_n]$ by $t_i, 1 \leq i \leq n$.

We can introduce now the notion of transition system specification. Transition system specifications are collections of rules which permit to infer LTS transitions from possibly empty sets of LTS transitions. LTSs are interpreted as the meaning of transition system specifications.

Definition 2.1.3 Let us assume a signature $\Sigma$ and a set of actions $\text{Act}$. A transition rule (with positive premises) $\rho$ is of the form $H/\alpha$, where $H$ is a collection of premises of the form $t \xrightarrow{a} t'$, $\alpha$ is a conclusion of the form $t \xrightarrow{a} t'$, $t, t'$ range over $T(\Sigma)$, and $a$ ranges over $\text{Act}$.

A transition system specification (TSS) is a collection of transition rules.

In general, also negative premises of the form $t \not\xrightarrow{a}$ may appear in transition rules. If $t \xrightarrow{a} t'$ is the conclusion of a transition rule $\rho$, then $t$ and $t'$ are called the source and the target of $\rho$, respectively.

A transition rule $H/\alpha$ can be denoted also by $\frac{H}{\alpha}$.

A transition $t \xrightarrow{a} t'$ is a closed transition if both $t$ and $t'$ are closed terms.

We give now the definition of transition provable from a TSS [40].

Definition 2.1.4 Let $T$ be a TSS. A proof of a closed transition of the form $t \xrightarrow{a} t'$ is a well-founded, upwardly branching tree whose nodes are labelled by closed transitions, the root is labelled by $t \xrightarrow{a} t'$, and, if $K$ is the set of labels of the nodes directly above a node labelled by $\beta$, then $K/\beta$ is a closed substitution instance of a transition rule in $T$.

A closed transition of the form $t \xrightarrow{a} t'$ is provable from $T$ if there exists a proof of $t \xrightarrow{a} t'$ from $T$.

The meaning of a TSS with positive premises $T$ is the LTS having as transitions the set of the closed transitions provable from $T$.

Given an LTS induced by a TSS, if actions labelling transitions contain encodings of transition proofs, then the LTS is called a proved transition system (PTS). In this case, derivation trees of PTS states are called proved trees.

2.2 Equivalences and rule formats

Labelled transition systems may describe the operational behavior of programs in great detail. In order to abstract away from irrelevant information on the way that
processes compute, a wealth of notions of behavioral preorder (namely, a relation that is reflexive and transitive) and behavioral equivalence (namely, a symmetric preorder) have been studied in the literature of process theory. A preorder relates two processes if one approximates the other, while an equivalence relates two processes if they are not distinguished by any external observer, according to a given notion of observation.

The kernel of a given preorder is the equivalence relating every pairs of processes such that each process of the pair approximates the other.

We recall here the notion of bisimulation equivalence [71, 74] which relates two states in an LTS precisely when they have the same branching structure. In Chapter 7 we will consider coarser equivalences. Given a relation $R$ and a pair of states $(s_1, s_2)$, we write $s_1 R s_2$ for $(s_1, s_2) \in R$.

**Definition 2.2.1** Given an LTS $\langle S, \text{Act}, \{\stackrel{a}{\rightarrow} | a \in \text{Act} \} \rangle$, a relation $R \subseteq S \times S$ is a bisimulation if whenever $s_1 R s_2$:

- if $s_1 \stackrel{a}{\rightarrow} s'_1$ then there exists a transition $s_2 \stackrel{a}{\rightarrow} s'_2$ such that $s'_1 R s'_2$;
- if $s_2 \stackrel{a}{\rightarrow} s'_2$ then there exists a transition $s_1 \stackrel{a}{\rightarrow} s'_1$ such that $s'_1 R s'_2$.

It is immediate to prove that a bisimulation is an equivalence. In general, one is interested in considering the union of all bisimulation on $S$. This relation is a bisimulation and is denoted with $\approx$.

Sometimes, an equivalent definition of bisimulation is given. Given a relation $R \subseteq S \times S$, let us denote with $\mathcal{F}(R)$ the relation on $S$ containing all pairs $(s_1, s_2)$ such that:

- if $s_1 \stackrel{a}{\rightarrow} s'_1$ then there exists a transition $s_2 \stackrel{a}{\rightarrow} s'_2$ such that $s'_1 R s'_2$;
- if $s_2 \stackrel{a}{\rightarrow} s'_2$ then there exists a transition $s_1 \stackrel{a}{\rightarrow} s'_1$ such that $s'_1 R s'_2$.

Now, $R$ is a bisimulation if and only if $R \subseteq \mathcal{F}(R)$.

An equivalence relation on terms of a signature is a congruence if it is preserved by all function symbols.

**Definition 2.2.2** Given a signature $\Sigma$, a relation of equivalence $R$ over $T(\Sigma)$ is a congruence if, for every function symbol $f \in \Sigma$,

$$t_i R u_i \text{ for } 1 \leq i \leq \text{ar}(f) \text{ implies } f(t_1, \ldots, t_{\text{ar}(f)}) R f(u_1, \ldots, u_{\text{ar}(f)})$$

When considering a behavioral equivalence over processes of a given process description language, one is generally interested to prove that this equivalence is a congruence w.r.t. constructs of the language. In fact, in this case, equivalent
processes can be freely exchanged in whatsoever context while preserving the equivalence.

A central issue in the area of SOS is to define rule formats ensuring that a given behavioral equivalence is a congruence. A rule format imposes syntactic constraints on the form of the allowed transition rules.

We introduce here the notion of de Simone format [92] and the notion of de Simone language adopted in [3].

**Definition 2.2.3** A transition rule \( \rho \) is in de Simone format if it has the form

\[
\frac{x_i \xrightarrow{a_i} y_i \quad i \in I}{f(x_1, \ldots, x_{ar(f)}) \xrightarrow{a} t}
\]

where variables \( x_i \) and \( y_i \) are all distinct and the only variables that occur in \( \rho \), the target \( t \in T(\Sigma) \) does not contain variables \( x_i \) for \( i \in I \) and has no multiple occurrences of variables.

We will say that a rule in de Simone format is a de Simone rule.

Let us assume a signature \( \Sigma \) and a countably infinite set of recursion variables, disjoint from \( \Sigma \) and \( \text{Var} \), ranged over by \( X \). The recursive terms over \( \Sigma \) are those generated by the following BNF grammar:

\[
t ::= X \mid f(t_1, \ldots, t_{ar(f)}) \mid \text{rec}(X = t)
\]

where \( X \) is any recursion variable, \( f \) is any function symbol in \( \Sigma \) and \( \text{rec} \) is a binding construct.

Given a recursive term \( t \), an occurrence of a recursive variable \( X \) is free in \( t \) if it does not appear in any subterm \( \text{rec}(X = u) \) of \( t \). Occurrences of recursion variables that are not free in a term are bound in such term. A recursion variable \( X \) is free in \( t \) if there exists an occurrence of \( X \) free in \( t \).

For every recursive term \( \text{rec}(X = t) \) we introduce a transition rule:

\[
\frac{t[u/X] \xrightarrow{a} y}{\text{rec}(X = t) \xrightarrow{a} y}
\]

where \( t[u/X] \) denotes the recursive term \( t \) in which each free occurrence of the variable \( X \) has been replaced by \( u \).

A de Simone language is a collection of de Simone rules, extended with the transition rules for recursion.

We introduce here the notion of GSOS format [19] for transition rules allowing negative premises.
Definition 2.2.4 A transition rule $\rho$ is in GSOS format if it has the form

$$\frac{x_i \xrightarrow{a_{ij}} y_{ij} \mid 1 \leq i \leq \text{ar}(f), 1 \leq j \leq m_i} {\bigcup \{x_i \xrightarrow{b_{ik}} y_{ik} \mid 1 \leq i \leq \text{ar}(f), 1 \leq k \leq n_i\}} f(x_1, \ldots, x_{\text{ar}(f)}) \xrightarrow{a} t$$

where $m_i, n_i \geq 0$, and the variables $x_i$ and $y_{ij}$ are all distinct and the only variables that occur in $\rho$.

We will say that a rule in GSOS format is a GSOS rule.

An (infinitary) GSOS language is a pair consisting of a countable signature and a countable set of GSOS rules.

Note that a de Simone rule is a GSOS rules. In general, the converse is not true. In fact, the GSOS format allows for negative premises, as well as for multiple occurrences of variables in the left hand sides of premises and in the target. Moreover, a variable occurring in the left hand side of a premise may appear also in the target.

A transition rule is in positive GSOS format if it is in GSOS format and it does not contain negative premises.

Among the properties that are guaranteed to hold by de Simone and GSOS formats, we recall that bisimulation over de Simone languages and over infinitary GSOS languages is a congruence.

A GSOS rule is simple if its target contains at most one function symbol. A GSOS language is simple if each of its transition rules is.

An interesting issue in the area of programming languages if to provide axiomatizations sound and complete modulo a given equivalence.

A (conditional) axiomatization over a signature $\Sigma$ consists of a set of (conditional) equations, called (conditional) axioms, of the form

$$t_0 = u_0 \iff t_1 = u_1, \ldots, t_n = u_n$$

with $t_i, u_i \in T(\Sigma)$, $0 \leq i \leq n$, and $n \geq 0$.

An axiomatization gives rise to a binary equality relation $=$ on $T(\Sigma)$ such that:

- if $t_0 = u_0 \iff t_1 = u_1, \ldots, t_n = u_n$ is an axiom and $\sigma$ is a substitution such that $\sigma(t_i) = \sigma(u_i)$, $1 \leq i \leq n$, then $\sigma(t_0) = \sigma(u_0)$;

- the relation $=$ is closed under reflexivity, symmetry and transitivity;

- if $f \in \Sigma$ is a function symbol and $t_i = u_i$, $1 \leq i \leq \text{ar}(f)$, then

$$f(t_1, \ldots, t_{\text{ar}(f)}) = f(u_1, \ldots, u_{\text{ar}(f)})$$

Note that relation $=$ is an equivalence and a congruence. We will denote with

$$\frac{t_1 = u_1, \ldots, t_n = u_n}{t_0 = u_0}$$

the axiom

$$t_0 = u_0 \iff t_1 = u_1, \ldots, t_n = u_n$$
**Definition 2.2.5** Given a signature $\Sigma$, let us assume a relation of equivalence $\sim$ on $T(\Sigma)$ and an axiomatization $\mathcal{A}$ over $\Sigma$.

- $\mathcal{A}$ is *sound* modulo $\sim$ if and only if $t = u$ implies $t \sim u$ for all $t, u \in T(\Sigma)$.
- $\mathcal{A}$ is *complete* modulo $\sim$ if and only if $t \sim u$ implies $t = u$ for all $t, u \in T(\Sigma)$.

Axiomatizations sound and complete modulo a given equivalence characterize the equivalence.
Chapter 3

A constructive SOS for Esterel

In this chapter we give a structural operational semantics for Esterel in terms of a labeled transition system. Then, we introduce the notion of bisimulation on Esterel programs and we prove that it is a congruence. The proof of congruence is immediate, since our transition system specification consists of a countable set of rules in GSOS format.

We show the correspondence between our SOS semantics and the so called circuit semantics of [16]. More precisely, the information in our labeled transition system permits both to establish whether circuits corresponding to programs electrically stabilize for every input, and to deduce the input/output behavior of these circuits.

The results in this chapter imply that bisimulation is a reasonable notion of behavioral equivalence over Esterel programs. In fact, since bisimulation is a congruence, equivalent programs cannot be distinguished by any Esterel context. Moreover, the correspondence between SOS and circuit semantics implies that equivalent programs are mapped to circuits that cannot be distinguished by the external environment.

This chapter is organized as follows. In Section 3.1 we recall Esterel and the circuit semantics of [16]. In Section 3.2 we present our SOS semantics. In Section 3.3 we prove the correspondence between SOS and circuit semantics.

3.1 An overview of Esterel

We begin with introducing the syntax of Esterel and its informal semantics. We consider the syntax of [15, 16] which slightly differs w.r.t. the original one of [18]. Then, we recall the circuit semantics of the language. We consider the translation of Esterel in hardware of [16] that has been implemented by the Esterel v5 compiler. It improves the original translation of [14] which turns out to be inadequate in the constructive setting.
3.1.1 Syntax and informal semantics

An Esterel program (module) has a body, consisting of an imperative statement, and an interface w.r.t. the environment, consisting of a set of input signals, denoted with \(I\), and of a set of output signals, denoted with \(O\). Signals local to the body may be defined for internal communications. Signals are pure, namely they carry only their presence/absence status.

Here we interpret Esterel as a process algebra. Terms (statements) of this algebra are those generated by the following BNF-like grammar

\[
E ::= \text{nothing} \mid \text{emit } s \mid \text{pause} \mid \text{present } s \text{ then } E \text{ else } E \text{ end} \mid E \parallel E \\
E ; E \mid \text{signal } s \text{ in } E \text{ end} \mid \text{loop } E \text{ end} \mid \text{suspend when } s \mid \\
\text{trap } T \text{ in } E \text{ end} \mid \text{exit } T
\]

where \(s\) ranges over a finite set of signal names \(S\), and \(T\) ranges over a finite set of trap names \(T\).

By default, \(";\) binds tighter than \("\parallel\). One can use brackets to group statements in arbitrary ways.

In the following we will refer to statements \text{nothing}, \text{pause}, \text{emit } s\) and \text{exit } T\) as basic statements.

We will write \text{signal } s_1, \ldots, s_n \text{ in } E \text{ end} for \text{signal } s_1 \text{ in } (\ldots (\text{signal } s_n \text{ in } E \text{ end}) \ldots) \text{ end}.

We will denote with \(s_1, s_2, \ldots, s_{|S|}\) the signals in \(S\).

A module behaves cyclically: at each cycle it reads the status of input signals and reacts by executing the current statement, so that both the status of output signals and the statement to be executed at the subsequent cycle are determined. According to the synchronous hypothesis, Esterel constructs take no time, except for the delay statement \text{pause} which takes precisely one unit of time. So, when a statement \text{starts}, either it executes a statement \text{exit } T\) in its body and \text{exits the trap } T\), so that the body of \text{trap } T\) terminates immediately, or it executes a statement \text{pause} in its body and it \text{pauses}, or it \text{terminates} immediately. A pausing statement will be \text{resumed} at the subsequent cycle.

Informally, \text{nothing} does nothing and terminates immediately. This means that \text{nothing} terminates in the cycle in which it starts.

Statement \text{emit } s\) sets the status of the signal \(s\) to “present” and terminates immediately. At every execution cycle, local signals and output signals are present only if some corresponding statement \text{emit} is executed.

Statement \text{pause} pauses for one cycle. It will be resumed at the subsequent one and it will terminate immediately.

Statement \text{present } s \text{ then } E_1 \text{ else } E_2 \text{ end} behaves either as \(E_1\), if \(s\) is present, or as \(E_2\), otherwise.

Statement \(E_1 \parallel E_2\) is the synchronous parallel composition of \(E_1\) and \(E_2\).

Statement \(E_1 ; E_2\) is the sequencing of \(E_1\) and \(E_2\). As \(";\) takes no time, if \(E_1\) terminates then \(E_2\) starts immediately.
Statement signal \( s \) in \( E \) end behaves as \( E \) with a local occurrence of \( s \). The output signal \( s \) of \( E \) is fed back to input signal \( s \) of \( E \). This feedback is instantaneous, namely if \( E \) emits \( s \) then \( s \) is immediately sensed by \( E \).

Statement loop \( E \) end executes infinitely \( E \). In [18, 16] it is required that the body of a loop_end cannot terminate immediately. This restriction is mandatory since it ensures that the computation in every cycle is finite.

Statement suspend \( E \) when \( s \) behaves as \( E \) at the first execution cycle. At subsequent cycles, if \( s \) is present then the execution of \( E \) is suspended for one cycle, else \( E \) receives the control.

Finally, trap \( T \) in \( E \) end defines the scope of trap \( T \), and exit \( T \) causes its body to terminate immediately.

It is well known that instantaneous feedback may originate paradoxes of causality between signals, so that programs do not satisfy the requirements of reactivity and determinism.

Reactivity is the ability of a program to react to any input from the environment. As an example, the statement

\[
signal s \in (\text{present } s \text{ then nothing else emit } s \text{ end}) \text{ end}
\]

is nonreactive (namely, it cannot perform any reaction), since the local signal \( s \) is emitted if and only if it is absent, so that it cannot assume any status.

Determinism is the ability of a program to have a unique reaction to any input from the environment. As an example, the statement

\[
signal s \in (\text{present } s \text{ then emit } s \text{ else nothing end}) \text{ end}
\]

is nondeterministic, since the local signal \( s \) is emitted if and only if it is present, so that it could coherently assume either the status present or absent.

Esterel rejects both nonreactive and nondeterministic statements. In [16] the notion of constructiveness has been introduced. Constructiveness is the ability to determine the status of local and output signals by a fact-to-fact propagation, namely without making any assumption on them. As an example, let us consider the statement

\[
signal s \in (\text{present } s \text{ then emit } s \text{ else emit } s \text{ end}) \text{ end}
\]

where the local signal \( s \) is emitted either if it is present or if it is absent, so that it can only be present. It follows that this statement is reactive and deterministic, but it is nonconstructive. In fact, in order to deduce that \( s \) is present we must be sure that some emit \( s \) is executed. Now, to infer that the emit \( s \) in the then branch of \( \text{present } s \) is executed, we must assume that \( s \) is present. So, in practice, we assume \( s \) present and then we check that this assumption is correct. This is counterintuitive, since it seems that what happens in a branch of a \( \text{present } s \) determines the choice of such branch, namely, some information flows backward w.r.t. the sequential control. This is the reason for which nonconstructive statements are rejected in [16].
3.1.2 The circuit semantics

The circuit semantics maps each Esterel statement to a sequential circuit. This
mapping is compositional w.r.t. the structure of statements, namely the circuit
(corresponding to a given statement is obtained as a suitable composition of cir-
cuits corresponding to its substatements. Circuits may be interpreted either as the
semantics model of Esterel or as the implementation of the language.

We denote with $C_E$ the circuit implementing the Esterel statement $E$. A latch
in $C_E$ is associated with each statement pause in the body of $E$ and it is set in cor-
respondence with the execution of the pause considered, while other constructs are
translated into combinatorial logic. States of circuit $C_E$ correspond to configurations
of statement $E$, namely to sets of pause in which $E$ is pausing.

A wire $s$ is associated with each signal $s$ and is set when $s$ is present. Causal-
ity between signals, reactivity, determinism and constructiveness have a physical
interpretation. Signal causality in $E$ corresponds to wire connections in $C_E$. In
fact, if a statement emit $s$ is in the then branch of statements “present $s_1$”,
..., “present $s_n$” and in the else branch of statements “present $s_{n+1}$”, ..., “present $s_m$”, then the anding of the wires $s_1,...,s_n$ and of the negation of the
wires $s_{n+1},...,s_m$ is connected to $s$.

Now, according to [91] a circuit is constructive in a reachable state if and only
if it electrically stabilizes for any input, and a circuit is constructive if and only if
it is constructive in any reachable state. A statement $E$ is reactive, deterministic
and constructive if and only if the circuit $C_E$ is constructive. As an example, in the
circuit corresponding to statement

$$\text{signal } s \text{ in (present } s \text{ then emit } s \text{ else emit } s \text{ end) end}$$

the orring of the wire $s$ and of its negation is connected to $s$ itself, so that this wire
cannot stabilize and $C_E$ is nonconstructive.

All circuits corresponding to statements have the same input/output interface.
A circuit $C_E$ has a set of input pins $I$ corresponding to input signals $I$, and a
set of output pins $O$ corresponding to output signals $O$. A wire $i \in I$ is set by the
environment in correspondence with the communication of signal $i$. Analogously,$C_E$ sets a wire $o \in O$ in correspondence with the communication of signal $o$.

The input interface of $C_E$ consists also of pins $\mathcal{G}_0$, RES, SUSP, KILL, while the
output interface consists also of pins Sel, $\mathcal{K}_i$, $0 \leq i \leq |\mathcal{T}| + 1$.

Input pin $\mathcal{G}_0$ is used to activate $C_E$ in correspondence with the starting of state-
ment $E$.

Input pin RES is used to reactivate $C_E$ in correspondence with the resuming of
statement $E$.

Input pin SUSP is used to suspend the activity of $C_E$ in correspondence with the
suspension of statement $E$. 

3.1. AN OVERVIEW OF ESTEREL

Input pin KILL is used to unset latches of $C_E$ in correspondence with a trap exit, namely in correspondence with the preemption of $E$.

Output pin $K_0$ is set by $C_E$ in correspondence with the termination of $E$.

Output pin $K_1$ is set by $C_E$ in correspondence with the pausing of $E$.

Output pin $K_i$, $i \geq 2$, is set by $C_E$ in correspondence with the the fact that $E$ exits the $(i-2)^{th}$ outermost trap. We will refer to $K_0$, $K_1$, ..., as the termination pins of $C_E$. If $K_i$ refers to trap $T_i$, we will denote $K_i$ also with $K_T$.

Finally, output pin SEL is set by $C_E$ to indicate that $E$ is selected for resumption, namely that some internal latch has been set. Pin SEL is simply the orring of all internal latches.

In order to start execution of $C_E$, pin GO is set. At subsequent cycles, pin RES is set to resume $C_E$. At each cycle, control propagates within $C_E$, wires corresponding to output signals are set in correspondence with executions of statements emit, a termination wire is set, latches corresponding to executed occurrences of pause are set.

To suspend $E$ for an execution cycle, pin SUSP is set, instead of pin RES. If $E$ is preempted by some internal or concurrent trap exit, then pin KILL is set to unset latches.

The wire SEL is propagated upwards in compound statements and it remains set as long as some latch is set. The wire SEL is necessary since RES may also be sent to currently unselected statements. When RES is set, unselected statements do not react. This is implemented by anding RES and SEL.

From now on, we denote with $\equiv$ the syntactic identity between statements. We explain now the general idea of the construction of circuit $C_E$ corresponding to statement $E$.

- $E \equiv \text{nothing}$: pin GO is connected to pin $K_0$. This implements the immediate termination of $E$.

- $E \equiv \text{emit } s$: pin GO is connected both to pin $K_0$ and to output pin $s$. So, when $E$ starts, it sets signal $s$ to “present” and it terminates immediately.
- $E \equiv \text{pause}$: the circuit implementing $E$ is described in Fig. 3.1. Pin $G0$ is connected to pin $K_1$, as $E$ surely pauses when it is started. The latch is set when $\text{KILL}$ is not set and either $G0$ is set or both $\text{SEL}$ and $\text{SUSP}$ are set. So, if $E$ is preempted by some trap exit then the latch is unset. Otherwise, the latch is set when $E$ is started or when $E$ is both selected for resumption and suspended. In the second case, $E$ is in the body of a statement $\text{suspend}$. Output pin $K_0$ is set when both the latch and $\text{RES}$ are set. This means that $E$ terminates when it is resumed after the pausing. Output pin $\text{SEL}$ is set when the latch is set. This means that $E$ must be resumed when the latch is set.

- $E \equiv \text{present } s \text{ then } E_1 \text{ else } E_2 \text{ end}$: input pins $s$ and $G0$ of $C_E$ are anded and the result is connected to pin $G0$ of $C_{E_1}$, so that $E_1$ is started only if $s$ is present. Pin $G0$ of $C_E$ and the negation of input pin $s$ of $C_E$ are anded and the result is connected to pin $G0$ of $C_{E_2}$, so that $E_2$ is started only when $s$ is absent. Input pins $\text{KILL}$, $\text{RES}$, $\text{SUSP}$ and $I$ of $C_E$ are connected to the respective pins of $C_{E_1}$ and $C_{E_2}$, since if $E_i$ is activated then it views the same input interface of $E$ and it is preempted, resumed and suspended when $E$ is, $1 \leq i \leq 2$. Output pins of $C_E$ are obtained by orring the respective output pins of $C_{E_1}$ and $C_{E_2}$. This correspond to the fact that the whole statement terminates, pauses, exits a trap and emits a signal iff either $E_1$ or $E_2$ does it.

- $E \equiv E_1 \parallel E_2$: all input pins of $C_E$ are connected to the respective input pins of $C_{E_1}$ and $C_{E_2}$. This corresponds to the fact that $E_1$ and $E_2$ run synchronously and view the same input interface. Output pins of $C_E$ but termination pins are obtained as orring of the respective output pins of $C_{E_1}$ and $C_{E_2}$. A logic synchronizer sets output pin $K_i$ of $C_E$ iff either pin $K_i$ of $C_{E_1}$ is set and no pin $K_j$, with $i < j$, of $C_{E_2}$ is set, or conversely. This means that $E$ terminates iff both $E_1$ and $E_2$ terminate, $E$ exits trap $T$ iff $T$ is the outermost trap exited by $E_1$ and $E_2$, $E$ pauses iff either $E_1$ pauses and $E_2$ does not exit any trap, or conversely.

- $E \equiv E_1; E_2$: input pin $G0$ of $C_E$ is connected to pin $G0$ of $C_{E_1}$, so that the starting of $E$ coincides with the starting of $E_1$. Output pin $K_0$ of $C_{E_1}$ is connected to input pin $G0$ of $C_{E_2}$, so that $E_2$ is started when $E_1$ terminates. Output pin $K_0$ of $C_{E_1}$ is connected to output pin $K_0$ of $C_E$, so that $E$ terminates when $E_2$ terminates. Input pins $\text{KILL}$, $\text{RES}$, $\text{SUSP}$ and $I$ of $C_E$ are connected to the respective pins of $C_{E_1}$ and $C_{E_2}$, since if $E_i$ is activated then it views the same input interface of $E$ and it is preempted, resumed and suspended when $E$ is, $1 \leq i \leq 2$. Output pins $\text{SEL}$, $\text{G0}$ and $K_i$, $i \geq 1$, of $C_E$ are obtained as orring of the respective output pins of $C_{E_1}$ and $C_{E_2}$. This correspond to the fact that the whole statement pauses, exits a trap and emits a signal iff either $E_1$ or $E_2$ does it.

- $E \equiv \text{signal } s \text{ in } E_1 \text{ end}$: $C_E$ is obtained from $C_{E_1}$ by connecting output pin
s to input pin $s$. Cycles of wires giving rise to nonconstructiveness may be created.

- $E \equiv \text{loop } E_1 \text{ end}$: since $E$ executes infinitely $E_1$, it would seem to be natural to connect pin $k_0$ of $C_{E_1}$ to pin $\phi_0$ of $C_{E_1}$, so that $C_{E_1}$ is reactivated in correspondence with the restarting of $E_1$. This solution is rejected in [16] since it originates schizophrenia. Namely, if $E_1$ may terminate in the cycle subsequent that of its starting, it may happen that cycles of wires internal to $C_E$ do not stabilize. A possible solution is to duplicate the body of a loop, namely to replace $\text{loop } E_1 \text{ end}$ by $\text{loop } E_1; E_1 \text{ end}$. In fact, schizophrenia cannot arise if the body of a loop needs at least two reactions to terminate. Here, we assume that this solution to schizophrenia is adopted, namely we assume that the body of a loop is of the form $F; F$, where $F$ cannot terminate immediately.

- $E \equiv \text{suspend } E_1 \text{ when } s$: since at the first execution cycle $E_1$ cannot be suspended, pin $\phi_0$ of $C_E$ is connected to pin $\phi_0$ of $C_{E_1}$. The anding of pins $\text{SEL}$ of $C_{E_1}$, $\text{RES}$ of $C_E$ and of the negation of pin $s$ of $C_E$ is connected to pin $\text{RES}$ of $C_{E_1}$, so that $E_1$ is resumed when it is selected for resumption and $s$ is absent. The anding of pins $\text{SEL}$ of $C_{E_1}$, $\text{RES}$ of $C_E$ and $s$ of $C_E$ is connected to pin $\text{SUSP}$ of $C_{E_1}$, so that $E_1$ is suspended when it is selected for resumption and $s$ is present.

- $E \equiv \text{trap } T \text{ in } E_1 \text{ end}$: input pins of $C_E$ but $\text{KILL}$ are connected to the respective pins of $C_{E_1}$. As $E$ terminates when $E_1$ either terminates or exits trap $T$, then the oring of pins $k_0$ and $k_2$ of $C_{E_1}$ is connected to pin $k_0$ of $C_E$. As $E$ exits the $i^{th}$ outermost trap if $E_1$ exits the $(i+1)^{th}$ outermost trap, then pin $k_{i+1}$ of $C_{E_1}$ is connected to pin $k_i$ of $C_{E_1}$, for $i \geq 2$. Pin $\text{KILL}$ of $C_E$ and pin $k_2$ of $C_{E_1}$ are composed in or and the result is connected to pin $\text{KILL}$ of $C_{E_1}$, so that latches of $C_{E_1}$ are unset when either $E_1$ exits trap $T$ or when latches of $C_E$ must be unset.

- $E \equiv \text{exit } T$: let $i$ be the cardinality of the set of $\text{trap}$ declarations of the form $\text{trap } T' \text{ in } E_T \text{ end}$ such that:
  
  - exit $T$ is in the body of $\text{trap } T'$;
  - declaration $\text{trap } T'$ is in the body of $\text{trap } T$.

Then pin $\phi_0$ of $C_E$ is connected to pin $k_{i+2}$ of $C_{E_1}$.

Given a module $M$, we denote with $E_M$ its body and with $C_M$ the circuit corresponding to $M$. Circuit $C_M$ is obtained by connecting a so called boot latch to pin $\phi_0$ of $C_{E_M}$. This latch is set only at the first execution cycle, so that $C_{E_M}$ is activated only at the first execution cycle. Input pin $\text{RES}$ of $C_{E_M}$ is always set, so that $C_{E_M}$ is resumed at each execution cycle. Note that, since $\text{SEL}$ is unset at the first execution
cycle, the fact that \texttt{RES} is set does not affect the behavior of \( C_{E_M} \). Input pins \( I \) of \( C_{E_M} \) are set by the environment at each execution cycle. Finally, input pins \texttt{KILL} and \texttt{SUSP} of \( C_{E_M} \) are never set.

### 3.2 The labeled transition system

In this section we propose a labeled transition system as an operational semantic model for Esterel. LTS states correspond to Esterel statements, LTS transitions correspond to statement reactions, and LTS labels carry information on the status of input/output signals, on signal causality, and on the termination of statements.

We begin with introducing some notations.

Let \( S^\pm_\gamma \) be the set \( \{ s^+, s^- \mid s \in S \} \). Given a signal \( s \in S \), the symbol \( s^+ \) denotes the presence of \( s \), while \( s^- \) denotes the absence of \( s \). In the following, \( \gamma \) will range over \( S^\pm_\gamma \). We assume a function \( \neg : S^\pm_\gamma \rightarrow S^\pm_\gamma \) such that \( s^+ = s^- \) and \( \overline{s^-} = s^+ \), for every \( s \in S \).

An event \( S \) (over \( S \)) is a subset of \( S^\pm_\gamma \) such that, for no signal \( s \), both \( s^+ \in S \) and \( s^- \in S \). An event \( S \) is interpreted as an assumption over the status of signals, namely a signal \( s \) is assumed to be present if \( s^+ \in S \), while it is assumed to be absent if \( s^- \in S \).

Two events \( S \) and \( S' \) are consistent, written \( S \uparrow S' \), if and only if there exists no \( \gamma \in S^\pm_\gamma \) with \( \gamma \in S \) and \( \overline{\gamma} \in S' \), namely, \( S \) and \( S' \) are coherent assumptions over \( S \). Note that the union of two consistent events is an event.

An ordered event \( \vartheta \) (over \( S \)) is a string in \((S^\pm_\gamma)^*\). We let \( \vartheta, \phi, \psi \) range over ordered events. Following the usual convention, we denote with \( \varepsilon \) the empty string. Given an ordered event \( \vartheta \), we denote with \(|\vartheta|\) the event s.t.: \(|\vartheta| = \begin{cases}\emptyset & \text{if } \vartheta = \varepsilon \\ \{\gamma\} \cup |\phi| & \text{if } \vartheta = \gamma\phi. \end{cases}\)

**Definition 3.2.1** Given an ordered event \( \vartheta \) and a symbol \( \mu \in \{n, p\} \cup S \cup T \), \( \vartheta \mu \) is a causality term with \( \vartheta \) as cause and \( \mu \) as action.

A causality term \( \vartheta \mu \) refers to an atomic action performed be some statement if input signals have status as assumed by \( \vartheta \). Atomic actions may be: termination, denoted with \( n \), pausing, denoted with \( p \), production of a signal \( s \), denoted with \( s \), exiting a trap \( T \), denoted with \( T \). Action \( s \) subsumes the action of termination, in the sense that when a statement produces a signal \( s \) then it terminates.

**Definition 3.2.2** A label is a tuple \( l = (S_t, \mathcal{E}_t, \mathcal{N}_t, \mathcal{T}_t) \) such that:

- \( S_t \) is an event over \( S \);
- \( \mathcal{E}_t \) is a set of causality terms such that, for each \( \vartheta \mu \in \mathcal{E}_t \), \(|\vartheta| \subseteq S_t \);
- \( \mathcal{N}_t \) is a set of causality terms such that, for each \( \vartheta \mu \in \mathcal{N}_t \), \(|\vartheta| \not\subseteq S_t \);
3.2. THE LABELED TRANSITION SYSTEM

- \( \mathcal{T}_t \in \{0, 1\} \cup 2^T \);
- if \( \vartheta_\mu \in \mathcal{E}_t \cup \mathcal{N}_t \), \( \vartheta = \gamma_1 \ldots \gamma_m \), \( \mu = n \), then \( \gamma_m = s^+ \) for some \( s \in S \).

We will denote with \( \mathcal{L} \) the set of labels as in Def. 3.2.2.

Given statements \( E \) and \( F \), an LTS transition \( E \xrightarrow{I} F \) will represent the reaction of \( E \) to an environment that supplies every input signal \( s \) such that \( s^+ \in S_t \) and does not supply any input signal \( s \) such that \( s^- \in S_t \).

Causality terms in \( \mathcal{E}_t \) refer to atomic actions that are performed during the reaction represented by \( E \xrightarrow{I} F \). In particular, during this reaction \( E \) emits the set of signals \( \bigcup_{\vartheta \in \mathcal{E}_t \cup \mathcal{N}_t} \vartheta \), which will be denoted with \( E m(l) \).

A causality term \( \vartheta_\mu \) is in \( \mathcal{N}_t \) if it refers to an atomic action that is not performed, since either some input signal \( s \) with \( s^+ \in |\vartheta| \) is absent or some input signal \( s \) with \( s^- \in |\vartheta| \) is present. Now, if no \( \vartheta_\mu \) with \( \mu = s \) is in \( \mathcal{E}_t \) then the output signal \( s \) is not emitted. In this case, if \( \vartheta' \in \mathcal{N}_t \) then \( s \) would be emitted if input signals would have the status as given by \( \vartheta' \). This means that a statement \( \text{emit} \ s \) is in a discarded branch of some statement \( \text{present} \). We will see that we need information in \( \mathcal{N}_t \) to have the correspondence between our SOS semantics and the circuit semantics of [16].

The component \( \mathcal{T}_t \) carries information about the termination of \( E \), namely \( \mathcal{T}_t = 0 \) if \( E \) terminates, \( \mathcal{T}_t = 1 \) if \( E \) pauses, \( \mathcal{T}_t \subseteq 2^T \) if \( E \) exits the outermost trap in \( \mathcal{T}_t \).

\[
\begin{array}{l}
\text{nothing} \xrightarrow{\varnothing \{\text{en}\} \varnothing} \text{nothing} \quad \text{(nothing)} \\
\text{emit} \ s \xrightarrow{\varnothing \{\text{en}\} \varnothing} \text{nothing} \quad \text{(emit)} \\
\text{pause} \xrightarrow{\varnothing \{\text{en}\} \varnothing} \text{nothing} \quad \text{(pause)} \\
\text{exit} \ T \xrightarrow{\varnothing \{\text{en}\} \varnothing} \text{nothing} \quad \text{(exit)} \\
\end{array}
\]

\[
\begin{array}{l}
\text{trap} \ T \text{ in } E \text{ end} \xrightarrow{\text{tr}(T)} \text{nothing} \quad \mathcal{T}_t = 0 \lor \mathcal{T}_t = \{T\} \quad \text{(trap-1)} \\
\end{array}
\]

\[
\begin{array}{l}
\text{trap} \ T \text{ in } E \text{ end} \xrightarrow{\text{tr}(T)} \text{trap} \ T \text{ in } F \text{ end} \quad \mathcal{T}_t = 1 \quad \text{(trap-2)} \\
\text{trap} \ T \text{ in } E \text{ end} \xrightarrow{\text{tr}(T)} \text{nothing} \quad \mathcal{T}_t \subseteq \mathcal{T}, \mathcal{T}_t \neq \{T\} \quad \text{(trap-3)}
\end{array}
\]

The LTS for Esterel.
The LTS for Esterel.

\[
E \xrightarrow{l} F\quad E' \xrightarrow{l} F' \quad s'^{(l)} \notin S_l \quad \text{present}_1
\]

present \( s \) then \( E \) else \( E' \) end \( \xrightarrow{s^{(l)}(F')} F \) \( \text{present}_2 \)

\[
E \xrightarrow{l} F\quad E' \xrightarrow{l} F' \quad s'^{(l)} \notin S'_v \quad \text{parallel}_1
\]

\[
E \parallel E' \xrightarrow{l \oplus l'} F \parallel F' \quad S_l \uparrow S'_v, T_l, T'_v \in \{0,1\} \quad \text{parallel}_2
\]

\[
E \xrightarrow{l} F\quad E' \xrightarrow{l} F' \quad T_l = 0, S_l \uparrow S'_v \quad \text{seq}_1
\]

\[
E \xrightarrow{l} F\quad E' \xrightarrow{l} F' \quad T_l = 1 \quad \text{seq}_2
\]

\[
E \xrightarrow{l} F\quad E' \xrightarrow{l} F' \quad T_l \subseteq T \quad \text{seq}_3
\]

\[
E \xrightarrow{l} F \quad \text{loc}(s,l) \in \mathcal{L} \quad \text{signal}_1
\]

\[
E \xrightarrow{l} F \quad \text{signal}_2
\]

\[
E \xrightarrow{l} F \quad \text{loop}_1
\]

\[
E \xrightarrow{l} F \quad \text{loop}_2
\]

\[
E \xrightarrow{l} F \quad \text{suspend}_1
\]

\[
E \xrightarrow{l} F \quad \text{suspend}_2
\]

Table 3.1: The labeled transition system for Esterel.
3.2. THE Labeled TRANSITION SYSTEM

The LTS giving the operational semantics for Esterel is defined by the transition system specification in Table 3.1.

Rule nothing states that nothing terminates immediately. Label $\langle \emptyset, \{en\}, \emptyset, 0 \rangle$ emphasizes that a statement terminates, independently of the status of input signals.

Rule emit states that emit $s$ emits signal $s$ and terminates immediately. Label $\langle \emptyset, \{es\}, \emptyset, 0 \rangle$ emphasizes that signal $s$ has been emitted, independently of the status of input signals.

Rule pause states that pause pauses for one cycle and will behave as nothing at the subsequent one. Label $\langle \emptyset, \{ep\}, \emptyset, 1 \rangle$ emphasizes that a statement pauses, independently of the status of input signals.

Rule exit states that exit $T$ exits trap $T$. Label $\langle \emptyset, \{eT\}, \emptyset, \{T\} \rangle$ emphasizes that the body of the trap $T$ must terminate, independently of the status of input signals.

We will denote by $δ$ the label $\langle \emptyset, \{en\}, \emptyset, 0 \rangle$ representing the reaction of nothing.

Before explaining the rules for statement present $s$ then $E$ else $E'$ end, we need some notations. Given a set of causality terms $Θ$ and an ordered event $φ$, we denote with $Θ^φ$ the set of causality terms $\{φθμ | θμ ∈ Θ\}$. Namely, we obtain $Θ^φ$ by prefixing with $φ$ the cause of each causality term in $Θ$.

Given labels $l, l' ∈ L$ and a symbol $γ ∈ S_+^+$ such that $γ ∉ S_l$, we denote by $γ(l, l')$ the label:

$$γ(l, l') = \begin{cases} 
\langle S_l ∪ \{γ\}, E_l^γ, N_l^γ ∪ N_l^{γ+} ∪ N_l^{γ-}, T_l \rangle & \text{ if } l ≠ δ ≠ l' \\
\langle S_l ∪ \{γ\}, E_l^{γ+}, N_l^{γ+}, T_l \rangle & \text{ if } l ≠ δ = l' \\
\langle S_l ∪ \{γ\}, 0, E_l^{γ+} ∪ N_l^{γ+}, T_l \rangle & \text{ if } l = δ ≠ l' \\
\langle \{s^+\}, \{s^+n\}, 0, 0 \rangle & \text{ if } l = δ = l' \text{ and } γ = s^+ \\
\langle \{s^-\}, \emptyset, \{s^-n\}, 0 \rangle & \text{ if } l = δ = l' \text{ and } γ = s^-.
\end{cases}$$

Rule present $s$ states that if $s$ is present then present $s$ then $E$ else $E'$ end behaves as $E$. We have that $S_{l^+} = S_l ∪ \{s^+\}$ since the reaction represented by the transition labeled by $s^+(l, l')$ is caused by the presence of $s$ and by the input causing the reaction of $E$.

Assume that $l ≠ δ ≠ l'$. If $φμ$ refers to an atomic action performed by a statement in the body of $E$, then $s^+φμ$ appears in $s^+(l, l')$ and reflects that this atomic action requires the presence of $s$. If $φμ$ refers to an atomic action performed by a statement in the body of $E'$, then $s^-φμ$ appears in $s^+(l, l')$ and reflects that this atomic action requires the absence of $s$. If $l ≠ δ = l'$ then we forget the causality term $s^-n$, since causality terms of the form $s^+φμ$ implicitly keep track that if $s$ is absent then a nothing is executed. The case with $l = δ ≠ l'$ is analogous. In both labels $s^+(δ, δ)$ and $s^-(δ, δ)$, the causality term $s^+n$ appears. Another possible choice is to allow causality terms of the form $δs^-n$ and to have $s^-δ, δ = \langle \{s^-\}, \{s^-n\}, 0, 0 \rangle$. Our choice permits to have $E_l ∪ N_l = E_l' ∪ N_l'$, for $l$ and $l'$ labels of two arbitrary transitions having the same LTS state as source state.
We have $T_{s^+(l,l')} = T_l$ since the whole statement terminates, pauses or exits a trap if $E$ does it. Note that the label $s^+(l,l')$ keeps track of signal causality arising from both branches of present $s$. We will see that this is needed to have correspondence between our SOS semantics and the circuit semantics of [16].

Rule $\text{present}_2$ is symmetric to rule $\text{present}_1$ and states that if $s$ is absent then present $s$ then $E$ else $E'$ end behaves as $E'$.

**Example 3.2.3** Assume $E \equiv \text{present } s \text{ then emit } s_1 \text{ else emit } s_2 \text{ end.}$

By rule $\text{present}_1$, we have $E \xrightarrow{l_1} \text{nothing}$, $l_1 = \langle \{s^+\}, \{s^+s_1\}, \{s^-s_2\}, 0 \rangle$. Label $l_1$ reflects that $s_1$ is caused by the presence of $s$, while $s_2$ would be caused by the absence of $s$. Moreover, $l_1$ reflects that the presence of $s$ causes the immediate termination of $E$.

By rule $\text{present}_2$, we have $E \xrightarrow{l_2} \text{nothing}$, $l_2 = \langle \{s^-\}, \{s^-s_2\}, \{s^+s_1\}, 0 \rangle$.

As will be stated by Prop. 3.2.14, $\mathcal{E}_l \cup \mathcal{N}_l = \mathcal{E}_l \cup \mathcal{N}_l$ for $l_1$ and $l_2$ labels of two arbitrary transitions having the same state as source state. So, rule $\text{present}_1$ does not depend on the choice of $l'$ and rule $\text{present}_2$ does not depend on the choice of $l$.

Rules $\text{parallel}_1$ and $\text{parallel}_2$ state that $E \parallel E'$ performs both reactions of $E$ and $E'$. We assume a partial function $\otimes : \mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L}$ such that, given labels $l$ and $l'$ such that $S_l \uparrow S_{l'}$, we have:

1. $S_{l \otimes l'} = (S_l \cup S_{l'}) \setminus \{\gamma \mid \{\gamma, \bar{\gamma}\} \cap \emptyset = \emptyset \text{ for each } \partial \mu \in \mathcal{E}_{l \otimes l'} \cup \mathcal{N}_{l \otimes l'}\};$

2. $\mathcal{E}_{l \otimes l'} = (\mathcal{E}_l \cup \mathcal{E}_{l'}) \setminus (\{\partial \phi \rho, \partial \phi \rho | \partial \phi \rho \in \mathcal{E}_l \cup \mathcal{E}_{l'}\} \cup \{\partial \phi \mu | \text{ either } \partial \phi \mu \in \mathcal{E}_l \cup \mathcal{E}_{l'} \cup \mathcal{N}_l \cup \mathcal{N}_{l'} \text{ or } \partial \phi \bar{\mu} \in \mathcal{E}_l \cup \mathcal{E}_{l'} \cup \mathcal{N}_l \cup \mathcal{N}_{l'}\});$

3. $\mathcal{N}_{l \otimes l'} = (\mathcal{N}_l \cup \mathcal{N}_{l'}) \setminus (\{\partial \phi \rho, \partial \phi \rho | \partial \phi \rho \in \mathcal{E}_l \cup \mathcal{E}_{l'} \cup \mathcal{N}_l \cup \mathcal{N}_{l'}\} \cup \{\partial \phi \mu | \text{ either } \partial \phi \mu \in \mathcal{E}_l \cup \mathcal{E}_{l'} \cup \mathcal{N}_l \cup \mathcal{N}_{l'} \text{ or } \partial \phi \bar{\mu} \in \mathcal{E}_l \cup \mathcal{E}_{l'} \cup \mathcal{N}_l \cup \mathcal{N}_{l'}\});$

4. $T_{l \otimes l'} = \begin{cases} \max(T_l, T_{l'}) & \text{if } T_l, T_{l'} \in \{0, 1\} \\ (T_l \cup T_{l'}) \cap T & \text{otherwise.} \end{cases}$

**Example 3.2.4** Assume $E \equiv E_1 \parallel E_2$, where

$E_1 \equiv \text{present } s_1 \text{ then emit } s_2 \text{ else nothing end}$.

$E_2 \equiv \text{present } s_3 \text{ then emit } s_4 \text{ else nothing end}$.

By transition rule $\text{parallel}_1$, we have $E \xrightarrow{l} \text{nothing } \parallel \text{nothing}$, where $l = \langle \{s^+_1, s^+_3\}, \{s^+_1s_2, s^+_3s_4\}, \emptyset, 0 \rangle$.

Function $\otimes$ is such that the label $l \otimes l'$ carries information given by both $l$ and $l'$. We could define function $\otimes$ by imposing $S_{l \otimes l'} = S_l \cup S_{l'}$, $\mathcal{E}_{l \otimes l'} = \mathcal{E}_l \cup \mathcal{E}_{l'}$ and $\mathcal{N}_{l \otimes l'} = \mathcal{N}_l \cup \mathcal{N}_{l'}$. Our choice permits to remove redundant information from labels and, as a consequence, to have a finer notion of bisimulation on statements.
3.2. THE Labeled TRANSITION SYSTEM

If both $\vartheta \phi p$ (resp. $\vartheta \phi n$) and $\vartheta p$ are in $\mathcal{E}_i \cup \mathcal{E}_f \cup \mathcal{N}_i \cup \mathcal{N}_f$, then we forget $\vartheta \phi p$ (resp. $\vartheta \phi n$) which carries redundant information. In fact, when each signal $s$ such that $s^+ \in |\vartheta|$ is present and each signal $s$ such that $s^- \in |\vartheta|$ is absent, an action of pausing is performed by a statement in the body of $E \parallel E'$. So, the action of pausing (resp. termination) referred by $\vartheta \phi p$ (resp. $\vartheta \phi n$) is useless.

**Example 3.2.5** Assume $E \equiv \text{pause} \parallel (E_1 \parallel E_2)$, where 
$E_1 \equiv \text{present } s_1 \text{ then pause else nothing end},$
$E_2 \equiv \text{present } s_1 \text{ then nothing else nothing end}.$
We have $E \xrightarrow{\ell} \text{nothing}$, where $\ell = (\emptyset, \{\epsilon \vartheta\}, \emptyset, 1)$. Note that $E \approx \text{pause}.$

Note that we forget a causality term $\vartheta \phi p$ or $\vartheta \phi n$ such that $\vartheta \mu \in \mathcal{E}_i \cup \mathcal{E}_f \cup \mathcal{N}_i \cup \mathcal{N}_f$ only if $\mu = p$. To see the reasons, consider the statement $E'$ obtained by replacing all occurrences of $\text{pause}$ in the statement $E$ of Example 3.2.5 by $\text{nothing}$. In this case, either the presence or the absence of $s_1$ is needed to have the termination of $E'$. So, given an arbitrary statement $E''$, if we consider $E'; E''$, either the presence or the absence of $s_1$ is needed to start $E''$. If, as an example, $E'' \equiv \text{emit } z$, then we must keep track of the causality between $s_1$ and $z$.

If both $\vartheta \gamma n$ and $\vartheta \gamma \phi \mu$ (resp. $\vartheta \gamma \phi \mu$) are in $\mathcal{E}_i \cup \mathcal{E}_f \cup \mathcal{N}_i \cup \mathcal{N}_f$ then we forget $\vartheta \gamma n$, since $\vartheta \gamma \phi \mu$ (resp. $\vartheta \gamma \phi \mu$) keeps track of the fact that if each signal $s$ such that $s^+ \in |\vartheta \gamma|$ is present and each signal $s$ such that $s^- \in |\vartheta \gamma|$ is absent then an action is performed by some statement, so that the action of termination referred by $\vartheta \gamma n$ is useless.

**Example 3.2.6** Assume $E \equiv E_1 \parallel E_2$, where
$E_1 \equiv \text{present } s_1 \text{ then emit } s_2 \text{ else nothing},$
$E_2 \equiv \text{present } s_1 \text{ then nothing else nothing}.$
We have $E \xrightarrow{l_1} \text{nothing}$ and $E \xrightarrow{l_2} \text{nothing}$, where $l_1 = (\{s_1^+\}, \{s_1^+s_2\}, \emptyset, 0)$ and $l_2 = (\{s_1^-\}, \emptyset, \{s_1^-s_2\}, 0)$. Note that $E \approx E_1$.

Since $E \parallel E'$ terminates if both $E$ and $E'$ do, if $T_i = T_f = 0$ then $T_{i\otimes f} = 0$. Since $E \parallel E'$ pauses if either $E$ pauses and $E'$ does not exit any trap, or conversely, if $T_i, T_f \in \{0, 1\}$ and either $T_i = 1$ or $T_f = 1$ then $T_{i\otimes f} = 1$. Since $E \parallel E'$ exits the outermost trap among those exited by $E$ and $E'$, if either $T_i \subseteq T$ or $T_f \subseteq T$ then $T_{i\otimes f} \subseteq T$. In the last case, the derivative of $E \parallel E'$ is $\text{nothing}$. Note that this derivative will never be executed, since $E \parallel E'$ is in the body of a $\text{trap}$ that terminates immediately.

The following proposition implies that $E \parallel F \approx F \parallel E$, for $E$ and $F$ arbitrary Esterel statements.

**Proposition 3.2.7** Given labels $l_1$ and $l_2$, we have $l_1 \otimes l_2 = l_2 \otimes l_1.$
Proof. Directly by the definition of $\otimes$.

The following proposition implies that $E \parallel (F \parallel G) \approx (E \parallel F) \parallel G$, for $E$, $F$, and $G$ arbitrary Esterel statements.

Proposition 3.2.8 Given labels $l_1$, $l_2$ and $l_3$, we have $l_1 \otimes (l_2 \otimes l_3) = (l_1 \otimes l_2) \otimes l_3$.

Proof. We prove that $E_{l_1 \otimes (l_2 \otimes l_3)} = E_{(l_1 \otimes l_2) \otimes l_3}$. The proof that $N_{l_1 \otimes (l_2 \otimes l_3)} = N_{(l_1 \otimes l_2) \otimes l_3}$ is analogous, while $S_{l_1 \otimes (l_2 \otimes l_3)} = S_{(l_1 \otimes l_2) \otimes l_3}$ and $T_{l_1 \otimes (l_2 \otimes l_3)} = T_{(l_1 \otimes l_2) \otimes l_3}$ are immediate.

We begin by proving that $E_{l_1 \otimes (l_2 \otimes l_3)} \subseteq E_{(l_1 \otimes l_2) \otimes l_3}$ by showing that causality terms in $E_{l_1 \otimes (l_2 \otimes l_3)}$ are in $E_{(l_1 \otimes l_2) \otimes l_3}$ for an arbitrary $1 \leq i \leq 3$.

If $\gamma_1 \ldots \gamma_i \otimes \gamma_j \in E_{l_1 \otimes (l_2 \otimes l_3)}$ then $\gamma_1 \ldots \gamma_j \otimes \gamma_i \in E_{l_1 \otimes l_2 \otimes l_3}$ for some $j < i$. If $\gamma_1 \ldots \gamma_{j-1} \in E_{l_1 \otimes l_2 \otimes l_3}$ then $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \notin E_{l_1 \otimes (l_2 \otimes l_3)}$. In fact, either $\gamma_1 \ldots \gamma_i \otimes p, \gamma_1 \ldots \gamma_j \otimes p \in E_{l_2 \otimes l_3}$ and $\gamma_1 \otimes p$ is removed when computing $l_2 \otimes l_3$, or $\gamma_1 \otimes p$ is removed when computing $l_1 \otimes (l_2 \otimes l_3)$. Otherwise, there exists $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \in E_{l_1 \otimes l_2 \otimes l_3}$ for some $h < j$, and, also in this case, $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \notin E_{l_1 \otimes (l_2 \otimes l_3)}$.

If $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \in E_{l_1 \otimes (l_2 \otimes l_3)}$ then we have one of the following cases:

- $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \in E_{l_1 \otimes l_2 \otimes l_3}$ for some $j < i$. If $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \in E_{l_1 \otimes l_2 \otimes l_3}$ then $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \notin E_{l_1 \otimes l_2 \otimes l_3}$. Otherwise, there exists $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \in E_{l_1 \otimes l_2 \otimes l_3}$ for some $h < j$, and, also in this case, $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \notin E_{l_1 \otimes l_2 \otimes l_3}$.

- either $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \in E_{l_1 \otimes l_2 \otimes l_3}$ or $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \notin E_{l_1 \otimes l_2 \otimes l_3}$ or $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \in E_{l_1 \otimes l_2 \otimes l_3}$ or $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \notin E_{l_1 \otimes l_2 \otimes l_3}$. Let us assume the first case. The other is analogous. If $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \in E_{l_1 \otimes l_2 \otimes l_3}$ then $\gamma_1 \ldots \gamma_{j-1} \otimes \gamma_j \notin E_{l_1 \otimes l_2 \otimes l_3}$.

The proof that $E_{l_1 \otimes (l_2 \otimes l_3)} \supseteq E_{(l_1 \otimes l_2) \otimes l_3}$ is analogous because, from Prop. 3.2.7, we have that $E_{l_1 \otimes (l_2 \otimes l_3)} = E_{(l_2 \otimes l_3) \otimes l_1}$ and $E_{(l_1 \otimes l_2) \otimes l_3} = E_{l_3 \otimes (l_1 \otimes l_2)}$.

This completes the proof.

Rule seq.1 states that if $E$ terminates then $E \Rightarrow E'$. Rules seq.2 and seq.3 state that if $E$ either pauses or exits a trap then $E \Rightarrow E'$ reacts as $E$. If $E$ exits a trap then the derivative of $E \Rightarrow E'$ is nothing. This will be never executed, since $E \Rightarrow E'$ is in the body of a trap that terminates immediately.

Given an ordered event $\vartheta = \gamma_1 \ldots \gamma_m$, we denote with $\overline{\vartheta}$ the set of ordered events $\{\gamma_1 \ldots \gamma_{m-i} \mid 1 \leq i \leq m\}$.

Given a statement $E$ and a label $l$ such that $E_l \cup N_l = \{\vartheta_1 \mu_1, \ldots, \vartheta_n \mu_n\}$ and $E \rightarrow l$, we denote with $I(E)$ the set of all ordered events $\phi$ of the form $\phi_1 \ldots \phi_n$ such that, for every $1 \leq j \leq n$, $\phi_{ij} \in \{\vartheta_{ij} \mid \vartheta_{ij} \notin \{p\} \cup I\}$ otherwise.
3.2. THE Labeled Transition System

For each ordered event \( \phi \in \mathcal{I}(E) \), if the environment supplies each signal \( s \) such that \( s^+ \in |\phi| \) and does not supply any signal \( s \) such that \( s^- \in |\phi| \), then \( E \) terminates immediately. In fact, no statement \texttt{pause} or \texttt{exit} is executed. So, for a causality term \( \phi' \mu' \) which refers to an atomic action performed by a statement in the body of \( E' \), \( \phi \theta \mu \) gives one of the possible statuses of signals causing this atomic action. Note that, by Prop. 3.2.14, it follows that if \( E \xrightarrow{l} \) and \( E \xrightarrow{l} \), then \( \mathcal{I}(E) \) can be computed indifferently by considering either \( l_1 \) or \( l_2 \).

Given a label \( l \) such that \( E \xrightarrow{l} \), we denote with \( \mathcal{I}(l) \) the set of ordered events \( \{ \phi | \phi \in \mathcal{I}(E) \} \) and \( |\phi| = S_l \).

**Example 3.2.9** Assume \( E = E_1 \parallel E_2 \), where:

\( E_1 \equiv \text{present } s_1 \text{ then exit } T \text{ else nothing begin}, \)

\( E_2 \equiv \text{present } s_2 \text{ then emit } s_3 \text{ else nothing end.} \)

We have \( E \xrightarrow{l} \), \( \leq i \leq 4 \), where:

\( l_1 = \langle \{s_1^+, s_2^+\}, \{s_1^+ T, s_2^+ s_3\}, \emptyset, \{T\} \rangle, l_2 = \langle \{s_1^+, s_2^+\}, \{s_1^+ T, s_2^+ s_3\}, \{T\} \rangle, l_3 = \langle \{s_1^+, s_2^+\}, \{s_2^+ s_3\}, \{T\}, 0 \rangle, l_4 = \langle \{s_1^+, s_2^+\}, \emptyset, \{s_1^+ T, s_2^+ s_3\}, 0 \rangle. \)

We have \( I(l_1) = \emptyset, I(l_2) = I(l_3) = \{s_1^+ s_2^- s_3^+, s_2^+ s_1^-\}, I(l_4) = \{s_1^+ s_2^- s_3^+, s_2^+ s_1^-\}, I(E) = I(l_1) \cup I(l_2) \cup I(l_3) \cup I(l_4) \).

In rules seq.1, seq.2 and seq.3 we assume a partial function \( \triangleright : \mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L} \) such that, given labels \( l \) and \( l' \) such that \( T_l = 0 \) implies \( S_l \uparrow S_{l'} \), we have:

\[
l \triangleright l' = \begin{cases} l \otimes \langle 0, 0, \bigcup_{\phi \in \mathcal{I}(E)} \mathcal{E}_l^\phi \cup N_l^\phi, 0 \rangle & \text{if } T_l \neq 0 \\
\bigcup_{\phi \in \mathcal{I}(E)} \mathcal{E}_l^\phi, 0 \rangle & \text{if } T_l = 0.
\end{cases}
\]

If \( T_l \neq 0 \) then \( S_{l \triangleright l'} = S_l \), since the reaction of \( E ; E' \) is caused by the input causing the reaction of \( E \). If \( T_l = 0 \) then \( S_{l \triangleright l'} = S_l \cup S_{l'} \), since the reaction of \( E ; E' \) is caused by inputs causing the reactions of \( E \) and \( E' \).

If \( E \xrightarrow{l} F, T_l \neq 0 \), and \( \phi \in \mathcal{I}(E) \), then \( |\phi| \) is not consistent with \( S_l \), since \( |\phi| = S_{l'} \) for some \( l' \) with \( T_{l'} = 0 \). So, for each \( \theta \mu \in \mathcal{E}_l \cup N_l \), \( \phi \theta \mu \) appears in \( \mathcal{N}_{l \triangleright l'} \).

If \( E \xrightarrow{l} F, T_l = 0 \), and \( \phi \in \mathcal{I}(E) \), then \( |\phi| \) is consistent with \( S_l \) if and only if \( |\phi| = S_l \). So, for \( \theta \mu \in \mathcal{E}_l \), \( \phi \theta \mu \) appears in \( \mathcal{E}_{l \triangleright l'} \) if \( |\phi| = S_l \) and \( \phi \theta \mu \) appears otherwise. Moreover, if \( \theta \mu \in N_l \), then \( \phi \theta \mu \) also appears in \( \mathcal{N}_{l \triangleright l'} \).

Since \( E ; E' \) pauses if \( E \) does, and since \( E ; E' \) exits a trap \( T \) if \( E \) does, if \( T_l \neq 0 \) then \( T_{l \triangleright l'} = T_l \). Since \( E ; E' \) has the same termination of \( E' \) when \( E \) terminates, if \( T_l = 0 \) then \( T_{l \triangleright l'} = T_l \).

**Example 3.2.10** Assume \( E \) as in Example 3.2.9. We have \( E ; \text{emit } s \xrightarrow{l} \text{nothing}, 1 \leq i \leq 4 \), where:

\( l_1 = \langle \{s_1^+, s_2^+\}, \{s_1^+ T, s_2^+ s_3\}, \{s_1^- s_2^- s_3^+, s_2^- s_3^+ s_1^+, s_3^+ s_1^+ s_2^+, s_1^+ s_2^+ s_3^-\}, \{T\} \rangle, l_2 = \langle \{s_1^+, s_2^+\}, \{s_1^+ T, s_2^+ s_3\}, \{s_1^- s_2^- s_3^+, s_2^- s_3^+ s_1^+, s_3^+ s_1^+ s_2^+, s_1^+ s_2^+ s_3^-\}, \{T\} \rangle, l_3 = \langle \{s_1^+, s_2^+\}, \{s_2^+ s_3\}, \{s_1^+ s_2^+ s_3^- s_1^- s_2^- s_3^+, s_2^- s_3^+ s_1^+, s_3^+ s_1^+ s_2^+, s_1^+ s_2^+ s_3^-\}, 0 \rangle, l_4 = \langle \{s_1^+, s_2^+\}, \{s_1^- s_2^- s_3^+, s_2^- s_3^+ s_1^+, s_3^+ s_1^+ s_2^+, s_1^+ s_2^+ s_3^-\}, \{T\} \rangle. \)
Causality terms $s_1\overline{s_2}$ and $s_2\overline{s_1}$ appear in labels of Example 3.2.10 to denote that the absence of both $s_1$ and $s_2$ cause the production of $s$. Note that in labels of transitions having the statement

$$E' \equiv \text{present } s_1 \text{ then nothing else (present } s_2 \text{ then nothing else emit } s \text{ end) end}$$

as source state, only $s_1\overline{s_2}$ appears. In fact, in this case, if $s_1$ is present then $s_2$ is not tested. So, we reject the choice of remove one of $s_1\overline{s_2}$ and $s_2\overline{s_1}$ in labels of Example 3.2.10. We maintain both causality terms since there is no logical order between the testing of $s_1$ and the testing of $s_2$.

Note that by Prop. 3.2.14 it follows that rules $seq_2$ and $seq_3$ do not depend on the choice of $l'$.

Consider now the statement $\text{signal } s \text{ in } E \text{ end}$, where the output signal $s$ of $E$ is fed back to the input signal $s$ of $E$. At circuit level, this corresponds to connecting the output pin $s$ of $C_E$ to the input pin $s$ of $C_E$.

Define a partial function $\text{loc} : \mathcal{S} \times \mathcal{L} \rightarrow \mathcal{L}$ for a signal $s$ and a label $l$ if and only if the following conditions are satisfied:

1. if $s^+ \in S_l$ then $s \in \text{Em}(l)$;
2. if $s \in \text{Em}(l)$ then $s^- \not\in S_l$;
3. if $s \in \text{Em}(l)$ then there exists $\vartheta s \in \mathcal{E}_l$ with $s^+, s^- \not\in |\vartheta|$;
4. if $s \not\in \text{Em}(l)$ then no $\vartheta s$ with $|\vartheta| \subseteq S_l \cup \{s^-, s^+\}$ is in $\mathcal{N}_l$.

By rule $\text{signal}$ it follows that if $E \xrightarrow{L} F$ represents a reaction of statement $E$, then $\text{loc}(s, l)$ is the label of a transition representing a reaction of $\text{signal } s \text{ in } E \text{ end}$. Condition 1 above expresses that if $s$ is present then it must be emitted by $E$, since $s$ is a local signal that cannot be supplied by the external environment. Condition 2 expresses that if $s$ is emitted by $E$ then no substatement of $E$ requiring the absence of $s$ is executed, since $s$ is fed back to $E$. Condition 3 expresses that if $s$ is emitted by $E$ then we can deduce that $s$ is present without making any assumption on the status of $s$. In fact, if $\vartheta s \in \mathcal{E}_l$ and neither $s^+ \in |\vartheta|$ nor $s^- \in |\vartheta|$, then $s$ is present since each input signal $z$ with $z^+ \in |\vartheta|$ (resp. $z^- \in |\vartheta|$) is present (resp. absent).

Condition 4 expresses that if $s$ is not emitted by $E$ then we can deduce that $s$ is absent without making assumptions on the status of $s$. In fact, if $\vartheta_1 s, \ldots, \vartheta_n s \in \mathcal{N}_l$ and $\gamma_i \in |\vartheta_i| \setminus (S_l \cup \{s^-, s^+\}), 1 \leq i \leq n$, then $s$ is absent since, for each $1 \leq i \leq n$, either an input signals $z_i$ with $z_i^+ = \gamma_i$ is absent or an input signal $z_i$ with $z_i^- = \gamma_i$ is present.

If we would not keep track of signal causality arising from discarded branches of statements $\text{present}$, then we would not be able to check condition 4.

**Example 3.2.11** Assume $E \equiv \text{present } s \text{ then emit } s \text{ else emit } s \text{ end}$. We have $E \xrightarrow{l_1} \text{nothing}$ and $E \xrightarrow{l_2} \text{nothing}$, where $l_1 = \langle \{s^+\}, \{s^+ s\}, \{s^- s\}, 0 \rangle$ and
\[ l_2 = \langle \{s^-\}, \{s^- s\}, \{s^+ s\}, 0 \rangle. \] The function \( \text{loc} \) is defined neither for the pair \((s, l_1)\) nor for the pair \((s, l_2)\). In fact, \((s, l_1)\) does not satisfy condition 3, and \((s, l_2)\) does not satisfy condition 2. So, no transition having \textbf{signal} \( s \) \textbf{in} \( E \) \textbf{end} as source state exists.

Let us assume that the conditions above are satisfied and that the causality terms in \( E_t \cup N_t \) having as action \( s \) are \( \vartheta_1 s, \ldots, \vartheta_v s \), where \( \vartheta_i = \gamma_{i,1} \cdots \gamma_{i,n_i}, 1 \leq i \leq v \). Let us consider the label \( l' \) such that:

1. If \( s \in \text{Em}(l) \), for each \( 1 \leq i \leq u \) we have that \( \vartheta_i s \in \mathcal{E}_t \), and for each \( u + 1 \leq i \leq v \) we have that \( \vartheta_i s \in \mathcal{N}_t \), then:

- \( S_{l'} = S_l \setminus \{s^+\} \);
- \( \mathcal{E}_{l'} = \{\vartheta[\vartheta_i/s^+] z[n/s] | \vartheta z \in \mathcal{E}_t, |\vartheta_i| \cap \{s^+\} = \emptyset, 1 \leq i \leq u\} \);
- \( \mathcal{N}_{l'} = \{\vartheta[\vartheta_i/s^+] z[n/s] | \vartheta z \in \mathcal{N}_t, |\vartheta_i| \cap \{s^+, s^-\} \neq \emptyset, 1 \leq j_i \leq n_i \} \cup \{\vartheta[\vartheta_i/s^+] z[n/s] | \vartheta z \in \mathcal{E}_t, |\vartheta_i| \cap \{s^+, s^-\} = \emptyset, u + 1 \leq i \leq v\} \);
- \( T_{l'} = T_t \).

2. If \( s \not\in \text{Em}(l) \) and \( \vartheta_i s \in \mathcal{N}_t \), \( 1 \leq i \leq v \), then:

- \( S_{l'} = S_l \setminus \{s^+\} \);
- \( \mathcal{E}_{l'} = \{\vartheta[\gamma_{i,1} \cdots \gamma_{i,j_i} / s^-] z[n/s] | \vartheta z \in \mathcal{E}_t, \{\gamma_{i,1}, \ldots, \gamma_{i,j_i}\} \subseteq S_t \) and \( \gamma_{i,j_i} \not\in \{s^+, s^-\} \) for every \( 1 \leq h \leq v \} \);
- \( \mathcal{N}_{l'} = \{\vartheta[\vartheta_i/s^+] z[n/s] | \vartheta z \in \mathcal{N}_t, 1 \leq i \leq v, \gamma_{i,j_i} \not\in \{s^+, s^-\}, 1 \leq j_i \leq n_i \} \cup \{\vartheta[\gamma_{i,1} \cdots \gamma_{i,j_i} / s^-] z[n/s] | \vartheta z \in \mathcal{E}_t, \{\gamma_{i,1}, \ldots, \gamma_{i,j_i}\} \not\subseteq S_t \) for some \( 1 \leq h \leq v \) and \( \gamma_{i,j_i} \not\in \{s^+, s^-\} \) for every \( 1 \leq h \leq v \} \);
- \( T_{l'} = T_t \).

Let us consider labels \( l \) and \( l' \) as above. Let consider the first of the two cases, namely let us assume that \( s \in \text{Em}(l) \).

We have \( S_{l'} = S_l \setminus \{s^+\} \) since \( s^+ \) refers to the \textbf{signal} \( s \) \textbf{local} to \textbf{signal} \( s \) \textbf{in} \( E \) \textbf{end}.

Given \( \vartheta s \in \mathcal{E}_t \cup \mathcal{N}_t \), we replace \( s \) by \( n \), since \( s \) cannot be viewed by the external environment. So, the action of producing \( s \) of a statement in the body of \( E \), corresponds to the action of termination of this statement in the body of \textbf{signal} \( s \) \textbf{in} \( E \) \textbf{end}.

Since the presence of \( s \) is caused by any event \( \vartheta_i \) such that \( |\vartheta_i| \cap \{s^+\} = \emptyset \), \( 1 \leq i \leq u \), and could be caused by any event \( \vartheta_i \) such that \( |\vartheta_i| \cap \{s^+, s^-\} = \emptyset \), \( u + 1 \leq i \leq v \), then, if \( \vartheta z \in \mathcal{E}_t \) and \( s^+ \) occurs in \( \vartheta \), we have that \( \{\vartheta[\vartheta_i/s^+] z[n/s] | |\vartheta_i| \cap \{s^+\} = \emptyset, 1 \leq i \leq u \} \subseteq \mathcal{E}_{l'} \) and \( \{\vartheta[\vartheta_i/s^+] z[n/s] | |\vartheta_i| \cap \{s^+, s^-\} = \emptyset, u + 1 \leq i \leq v \} \subseteq \mathcal{N}_{l'} \). Analogously, if \( \vartheta z \in \mathcal{N}_t \) and \( s^+ \) occurs in \( \vartheta \), then we have that \( \{\vartheta[\vartheta_i/s^+] z[n/s] | |\vartheta_i| \cap \{s^+, s^-\} = \emptyset, 1 \leq i \leq v \} \subseteq \mathcal{N}_{l'} \).

By condition 2 of the definition of \( \text{loc}(s, l) \), we cannot have a causality term \( \vartheta z \in \mathcal{E}_t \)
such that $s^-$ occurs in $\vartheta$. If $\vartheta z \in \mathcal{N}_l$ and $s^-$ occurs in $\vartheta$, then we have that 
\[
\{\vartheta[\gamma_{i_1}, \ldots, \gamma_{i_v}] / z^n[s~\{s^+, s^-, 1 \leq j_{i_h} \leq n_{i_h}\} \subseteq \mathcal{N}_p'. The reason is that if, for each $1 \leq i \leq v$, signal $s_{i,j}$ has not the status assumed by $\vartheta$, then $s$ is not emitted.
We have $T_l = T_p$ since signal $s$ in $E$ end terminates as $E$.

The case with $s \not\in \operatorname{Em}(l)$ is analogous.

\textbf{Example 3.2.12} Assume $E \equiv \text{signal } s_2 \text{ in } E_1 \parallel E_2 \text{ end}$, where
$E_1 \equiv \text{present } s_1 \text{ then emit } s_2 \text{ else nothing end}$,
$E_2 \equiv \text{present } s_2 \text{ then emit } s_3 \text{ else nothing end}$.

By rule $\text{signal}$ we have $E \xrightarrow{l_1} \text{signal } s_2 \text{ in nothing } \parallel \text{nothing end}$ and $E \xrightarrow{l_2} \text{signal } s_2 \text{ in nothing } \parallel \text{nothing end}$, where $l_1 = \langle \{s_1^+, \{s_1^+ s_3\}, 0, 0\}$ and $l_2 = \langle \{s_1^+\}, 0, \{s_1^+ s_3\}, 0\rangle$. Note that $E \approx \text{present } s_1 \text{ then emit } s_3 \text{ else nothing end}$.

Rules $\text{loop}_1$ and $\text{loop}_2$ are straightforward. We do not consider the case with $T_l = 0$ since the body of a $\text{loop}$ cannot terminate.

As in [16], let us denote with $\text{suspend imm } E \text{ when } s$ the statement

\begin{verbatim}
trap T in
  loop
    present s then pause else exit T end
  end
end;
\end{verbatim}

which differs from $\text{suspend } E \text{ when } s$ since $E$ can be suspended also at the first execution cycle.

Semantics of statement $\text{suspend } E \text{ when } s$ is given by rules $\text{suspend}_1$ and $\text{suspend}_2$.

Rule $\text{suspend}_1$ states that, if $E$ either terminates immediately or exits a trap, then so does $\text{suspend } E \text{ when } s$. Rule $\text{suspend}_2$ states that, if $E$ pauses, then so does $\text{suspend } E \text{ when } s$. In this case, at the next execution cycle, if $E$ will behave as $F$ then $\text{suspend } E \text{ when } s$ will behave as $\text{suspend imm } F \text{ when } s$. 


3.2. THE LABELED TRANSITION SYSTEM

Let us consider now the statement $\text{trap } T \text{ in } E \text{ end}$.

Rule $\text{trap}_{\text{1}}$ states that if $E$ either terminates or exits the trap $T$, then $\text{trap } T \text{ in } E \text{ end}$ terminates.

Rule $\text{trap}_{\text{2}}$ states that if $E$ pauses, then so does $\text{trap } T \text{ in } E \text{ end}$.

Rule $\text{trap}_{\text{3}}$ states that if $E$ exits a trap $T'$ with $\text{trap } T \text{ in } E \text{ end}$ in its body, then so does $\text{trap } T \text{ in } E \text{ end}$.

Given a transition $E \xrightarrow{t} F$ with $\vartheta_1 T, \ldots, \vartheta_n T$ the causality terms in $E_t \cup N_t$ having $T$ as action, $\vartheta_i = \gamma_{i,1} \cdots \gamma_{i,n_i}$, we denote with $T(l)$ the set of ordered events $\gamma_{i,1} \cdots \gamma_{j_i,k_i} \cdots \gamma_{n_i,1} \cdots \gamma_{n_i,j_{n_i}}$, where $1 \leq j_{k_i} \leq n_i$ for $1 \leq h \leq n$. Given an ordered event $\phi \in T(l)$, if each signal $s$ with $s^+ \in |\phi|$ is present and each signal $s$ with $s^- \in |\phi|$ is absent, then $E$ does not exit the trap $T$, since no statement exit $T$ is executed.

Now, given a label $l$ and a trap name $T$, let $l'$ be the label such that:

- $S_T = S_l$;
- $E_{l'} = E_l[\{\phi \vartheta p \mid \phi \in T(l), |\phi| \uparrow S_l, |\vartheta| \uparrow |\vartheta_i| \text{ for some } 1 \leq i \leq n \}/\vartheta p][\vartheta n/\vartheta T]$;
- $N_{l'} = N_l[\{\phi \vartheta p \mid \phi \in T(l), |\vartheta| \uparrow |\vartheta_i| \text{ for some } 1 \leq i \leq n \}/\vartheta p][\vartheta n/\vartheta T] \cup \{\phi \vartheta p \mid \phi \in T(l), |\phi| \not\vartheta S_l, |\vartheta| \uparrow |\vartheta_i| \text{ for some } 1 \leq i \leq n, \vartheta p \in E_l\}$;
- $T_{l'} = \begin{cases} 
1 & \text{if } T_l = 1 \\
0 & \text{if } T_l = 0 \text{ or } T_l = \{T\} \\
T_l \setminus \{T\} & \text{otherwise.}
\end{cases}$

If $T_l = 1$ then $T_{l'} = 1$, since $\text{trap } T \text{ in } E \text{ end}$ pauses whenever $E$ pauses. If $T_l = 0$ or $T_l = \{T\}$ then $T_{l'} = 0$, since $\text{trap } T \text{ in } E \text{ end}$ terminates whenever $E$ either terminates or exits the trap $T$. If $T_l \subseteq T$ and $T_l \neq \{T\}$ then $T_{l'} = T_l \setminus \{T\}$, since $\text{trap } T \text{ in } E \text{ end}$ exits a trap $T'$ when $E$ does it.

If there exists a causality term $\vartheta p \in E_l$ with $|\vartheta| \uparrow |\vartheta_i|$ for some $1 \leq i \leq n$, then we replace it by the set of causality terms of the form $\phi \vartheta p$, where $\phi \in T(l)$ and $\phi \uparrow S_l$, since the whole statement pauses only if $E$ does not exit the trap $T$. Causality terms of the form $\phi \vartheta p$, where $\phi \in T(l)$ and $|\phi| \not\vartheta S_l$, are added to $N_{l'}$. Analogously, we replace a causality term $\vartheta p$ in $N_l$ such that $|\vartheta| \uparrow |\vartheta_i|$ for some $1 \leq i \leq n$ by the set of causality terms of the form $\phi \vartheta p$, with $\phi \in T(l)$.

Causality terms of the form $\vartheta T$ in $E_l \cup N_l$ are replaced by $\vartheta n$, since the action of exiting trap $T$ of a statement in the body of $E$ corresponds to the action of termination of this statement in the body of $\text{trap } T \text{ in } E \text{ end}$.

Now, since causality terms of the form $\vartheta T$ have been replaced by causality terms of the form $\vartheta n$, it may happen that a pair of causality terms $\vartheta_1 \vartheta_2 n$ and $\vartheta_1 p$ are in $E_{l'} \cup N_{l'}$. To remove this redundant information, we consider the label $l'' = l' \otimes l'$. Now, if $\vartheta s^- n \in E_{l''}$ then we remove it and we add $\vartheta s^+ n$ to $N_{l''}$, since $\vartheta s^- n$ cannot appear in any label. Analogously, if $\vartheta s^- n \in N_{l''}$ then we remove it and we add $\vartheta s^+ n$ to either $E_{l''}$, if $|\vartheta| \cup \{s^+\} \subseteq S_{l''}$, or to $N_{l''}$, otherwise. Then, we consider the label so obtained as $tr(T, l)$.
Example 3.2.13  Let $E \equiv E_1 \parallel E_2$, where:
$E_1 \equiv \text{present } s_1 \text{ then pause else nothing end},$
$E_2 \equiv \text{present } s_2 \text{ then exit } T \text{ else nothing end}.$

By rules $\text{trap}_1$, $\text{trap}_2$ and $\text{trap}_3$, we have $\text{trap } T \text{ in } E \text{ end } \stackrel{\Delta}{\rightarrow}, 1 \leq i \leq 4$, where:
$$l_i = \langle \{s_1^+, s_2^+ \}, \emptyset, \{s_2^+ s_1^+ p \}, 0 \rangle, \quad l_2 = \langle \{s_1^+, s_2^+ \}, \{s_2^- s_1^+ p \}, \emptyset, 1 \rangle,$$
$$l_3 = \langle \{s_1^+, s_2^- \}, \emptyset, \{s_2^- s_1^+ p \}, 0 \rangle, \quad l_4 = \langle \{s_1^-, s_2^+ \}, \emptyset, \{s_2^- s_1^+ p \}, 0 \rangle.$$

The LTS defined by the transition rules in Table 3.1 satisfies the following property:

Proposition 3.2.14  Given a statement $E$ and transitions $E \xrightarrow{\Delta} F_1$, $E \xrightarrow{\Delta} F_2$, we have that $E_{l_1} \cup N_{l_1} = E_{l_2} \cup N_{l_2}$.

Proof  By structural induction over $E$.

In order to relate Esterel statements having the same input/output behavior, we consider the bisimulation on the states of the LTS of Table 3.1. Given statements $E_1$ and $E_2$ such that $E_1 \approx E_2$, the external environment is not able to distinguish between them. In fact, at each execution cycle, if $E_1$ and $E_2$ are stimulated with the same set of input signals by the environment, then they respond by communicating the same set of output signals. The following theorem states that no Esterel context is able to distinguish between $E_1$ and $E_2$, namely that Esterel constructs preserve bisimulation.

Theorem 3.2.15  The bisimulation on Esterel statements is a congruence.

Proof  The thesis follows by the fact that the transition system specification given in Table 3.1 consists of a countable set of rules in GSOS format.

Examples of bisimilar Esterel statements have been given in Examples 3.2.5, 3.2.6 and 3.2.12.

In the following, we will denote with $[E]$ the part of labeled transition system reachable from $E$, for every statement $E$.

We recall that an LTS interpretation for Esterel has been given in [16]. The semantics of [16] considers LTS transitions of the form $E \xrightarrow{\langle s_i, s_o, k \rangle} E'$, where $S_i$ is an event over the set of input signals $I$, $S_o$ is an event over the set of output signals $O$, and $k$ is an integer giving information on the termination of $E$. Transitions having a statement of the form $\text{signal } s \text{ in } E \text{ end}$ as source state are inferred from transitions having $E$ as source state and from sets $\text{Must}(E, S_i)$ and $\text{Cannot}(E, S_i)$. Now, $\text{Must}(E, S_i)$ contain local and output signals that must be emitted by $E$ when input signals take the value given by $S_i$, and $\text{Cannot}(E, S_i)$ contain local and output signals that cannot be emitted by $E$ when input signals take the value given by $S_i$. The sets $\text{Must}(E, S_i)$ and $\text{Cannot}(E, S_i)$ are constructed compositionally w.r.t. the structure of $E$. 

Our labels carry information on signal causality which does not appear in labels of [16]. This information is exploited in [16] to compute sets \( \text{Must}(E, S_i) \) and \( \text{Cannot}(E, S_i) \).

From the correspondence between our LTS interpretation and the circuit semantics (see Lemma 3.3.2 and Lemma 3.3.7), and from the correspondence between the LTS interpretation of [16] and the circuit semantics (see Theorem 2, Chapter 4 and Theorem 6, Chapter 13 of [16]) it follows that the two LTS interpretations agree. More precisely, we infer \( E \xrightarrow{l} E' \) from rules in Table 3.1 if and only if transition \( E \xrightarrow{(S_i, S_o, k)} E' \) is inferred in [16], where \( S_i = S_t \), \( Em(l) = S_o \), \( k = 0 \) iff \( T_{i} = 0 \), \( k = 1 \) iff \( T_{i} = 1 \) and \( k = i \) iff \( E \) exits \( T \), \( T \in T_{i} \) and \( T \) is the \( i \)th outermost trap.

Given statements \( E \) and \( F \), we consider \( E \) and \( F \) to be equivalent if they are bisimilar according to our LTS semantics. If one considers the LTS semantics of [16], it is reasonable to consider \( E \) and \( F \) to be equivalent iff they are bisimilar, \( \text{Must}(E, S_i) = \text{Must}(F, S_i) \), and \( \text{Cannot}(E, S_i) = \text{Cannot}(F, S_i) \), for every input event \( S_i \). It is easier to prove the property of congruence of our equivalence, because we exploit the format of SOS rules.

### 3.3 Correspondence between SOS and circuit semantics

In this section we show the correspondence between the SOS semantics given in the previous section and the circuit semantics of [16]. Namely, we show that our labeled transition system carries all the information which is needed both to establish whether circuits corresponding to statements are constructive and to recover their input/output behavior.

The following proposition states that two arbitrary transitions having a statement \( E \) as source state represent reactions of \( E \) to different inputs from the environment.

**Proposition 3.3.1** Given transitions \( E \xrightarrow{l_1} F_1 \) and \( E \xrightarrow{l_2} F_2 \), then \( S_{l_1} \not\equiv S_{l_2} \).

**Proof** By structural induction on \( E \).

*Base case.*

If \( E \) is a basic statement then exactly one transition has \( E \) as source state, so that the thesis is immediate.

*Induction step.*

We must consider the following cases.

- \( E \equiv \text{present } s \text{ then } E_1 \text{ else } E_2 \text{ end.} \)
  - If both \( E \xrightarrow{l_1} F_1 \) and \( E \xrightarrow{l_2} F_2 \) represent reactions of \( E_1 \) then the thesis follows by inductive hypothesis on \( E_1 \) and by the definition of \( s^+(l, l') \).
  - If both \( E \xrightarrow{l_1} F_1 \) and \( E \xrightarrow{l_2} F_2 \) represent reactions of \( E_2 \) then the thesis
follows by inductive hypothesis on $E_2$ and by the definition of $s^-(l', l)$.

If $E \xrightarrow{h_1} F_1$ represents a reaction of $E_1$ and $E \xrightarrow{t_2} F_2$ represents a reaction of $E_2$ then the thesis follows by the fact that $s^+ \in S_{t_1}$ and $s^- \in S_{t_2}$.

- $E \equiv E_1 \parallel E_2$.
  The thesis follows by inductive hypothesis on $E_1$ and $E_2$ and by the definition of function $\otimes$.

- $E \equiv E_1; E_2$.
  The thesis follows by inductive hypothesis on $E_1$ and $E_2$ and by the definition of function $\triangleright$.

- $E \equiv \text{signal } s \text{ in } E' \text{ end}$.
  The thesis follows by inductive hypothesis on $E'$ and by the definition of function $\text{loc}$.

- $E \equiv \text{loop } E' \text{ end}$.
  The thesis follows by inductive hypothesis on $E'$.

- $E \equiv \text{suspend } E' \text{ when } s$.
  The thesis follows by inductive hypothesis on $E'$.

- $E \equiv \text{trap } T \text{ in } E' \text{ end}$.
  The thesis follows by inductive hypothesis on $E'$.

This completes the proof.

We show now how the information carried by LTS labels permits to deduce how output wires of a circuit stabilize electrically when the electrical value at which input wires stabilize is known.

**Lemma 3.3.2** Given a transition $E \xrightarrow{I} F$, when circuit $C_E$ is activated (i.e. wire GO is set) the following facts follow:

1. if $\vartheta_0 \in E_t \cup N_t$, $o \in O$, each input wire $i$ such that $i^+ \in |\vartheta|$ is kept stable at 1 and each input wire $i$ such that $i^- \in |\vartheta|$ is kept stable at 0, then the output wire $o$ stabilizes at 1;

2. if $\vartheta_1 o, \ldots, \vartheta_v o$ are the causality terms in $E_t \cup N_t$ having $o$ as action, $o \in O$, and, for each $1 \leq j \leq v$, either an input wire $i_j$ such that $i_j^+ \in |\vartheta_j|$ is kept stable at 0 or an input wire $i_j$ such that $i_j^- \in |\vartheta_j|$ is kept stable at 1, then the output wire $o$ stabilizes at 0;

3. if each input wire $i$ such that $i^+ \in S_t$ is kept stable at 1 and each input wire $i$ such that $i^- \in S_t$ is kept stable at 0, then an output wire $o$, $o \in O$, stabilizes at 1 if $o \in Em(l)$, while it stabilizes at 0 if $o \notin Em(l)$;
3.3. CORRESPONDENCE BETWEEN SOS AND CIRCUIT SEMANTICS

4. If each input wire \( i \) such that \( i^+ \in S_i \) is kept stable at 1 and each input wire \( i \) such that \( i^- \in S_i \) is kept stable at 0, then we have the following facts:

(a) If \( T_i = 0 \) then the termination wire \( k_0 \) stabilizes at 1;
(b) If \( T_i = 1 \) then the termination wire \( k_1 \) stabilizes at 1;
(c) If \( T_i \subseteq T \) then the termination wire \( k_T \) stabilizes at 1, where \( T \) is the outermost trap in \( T_i \).

Proof

By structural induction on \( E \).

Base case.

Let us assume \( E \equiv \text{emit} \ o \). We have \( E \xrightarrow{L} \text{nothing} \), where \( l = (\emptyset, \{e\}, 0, 0) \). The thesis follows by the fact that both the output wire \( o \) and the termination wire \( k_0 \) of \( C_E \) stabilize at 1.

The proof for the other basic statements is analogous.

**Induction step.**

We must consider the following cases.

- **\( E \equiv \text{present} \ s \text{ then } E_1 \text{ else } E_2 \text{ end} \).**
  
  By rules \textit{present}L1 and \textit{present}L2, if \( E \xrightarrow{L} F \) then there exist transitions \( E_1 \xrightarrow{l_1} F_1 \) and \( E_2 \xrightarrow{l_2} F_2 \) such that either \( l = s^+(l_1, l_2) \) and \( F \equiv F_1 \), or \( l = s^-(l_2, l_1) \) and \( F \equiv F_2 \).

  Let us prove fact 1.

  By definition of \( s^+(l_1, l_2) \) and \( s^-(l_2, l_1) \), if \( \emptyset o \in \mathcal{E}_i \cup \mathcal{N}_i \) then either \( \emptyset = s^+\phi \) and \( \phi o \in \mathcal{E}_i \cup \mathcal{N}_i \), or \( \emptyset = s^-\phi \) and \( \phi o \in \mathcal{E}_i \cup \mathcal{N}_i \). We assume the first case. The other is analogous. By inductive hypothesis on \( E_1 \), if \( C_{E_1} \) is activated and each input wire \( i \) such that \( i^+ \in |\phi| \) (resp. \( i^- \in |\phi| \)) is kept stable at 1 (resp. 0) then the output wire \( o \) of \( C_{E_1} \) stabilizes at 1. Now, since \( s \) is kept stable at 1 and \( C_E \) is activated, then \( C_{E_1} \) is activated. It follows that the output wire \( o \) of \( C_{E_1} \) stabilizes at 1. This wire is connected, through an or-gate, to the output wire \( o \) of \( C_E \), so that fact 1 follows.

  Fact 2 follows by inductive hypothesis on \( E_1 \) and \( E_2 \) and by the fact that the output wire \( o \) of \( C_E \) is obtained as or-ring of the output wires \( o \) of \( C_{E_1} \) and \( C_{E_2} \).

  Fact 3 follows directly by facts 1 and 2.

  To prove fact 4, let us assume that \( l = s^+(l_1, l_2) \). The other case is analogous. The thesis follows by inductive hypothesis on \( E_1 \), by the fact that if \( s \) stabilizes at 1 and \( C_E \) is activated then \( C_{E_1} \) is activated, by the fact that wire \( k_i \) of \( C_{E_i} \) is connected, through an or-gate, to wire \( k_i \) of \( C_E \), \( i \geq 0 \), and by the fact that \( T_i = T_{i_1} \).

- **\( E \equiv E_1 \parallel E_2 \).**
  
  By rules \textit{parallel}L1 and \textit{parallel}L2, if \( E \xrightarrow{L} F \) then there exist transitions
\[ E_1 \xrightarrow{l_1} F_1 \text{ and } E_2 \xrightarrow{l_2} F_2 \text{ such that } l = l_1 \otimes l_2. \]

Facts 1 and 2 follow by inductive hypothesis on \( E_1 \) and \( E_2 \), by the fact that when \( C_E \) is activated then both \( C_{E_1} \) and \( C_{E_2} \) are activated, and by the fact that the output wire \( o \) of \( C_E \) is obtained as orring of the output wires \( o \) of \( C_{E_1} \) and \( C_{E_2} \).

Fact 3 follows directly by facts 1 and 2.

Fact 4 follows directly by inductive hypothesis on \( E_1 \) and \( E_2 \) and by the definition of \( \mathcal{T}_{l_1 \otimes l_2} \).

- \( E \equiv E_1; E_2 \).

By rules seq1, seq2 and seq3, if \( E \xrightarrow{l} F \) then there exist transitions \( E_1 \xrightarrow{l_1} F_1 \) and \( E_2 \xrightarrow{l_2} F_2 \) such that \( l = l_1 \uplus l_2 \) and either \( \mathcal{T}_{l_1} = 0 \) and \( F \equiv F_2 \), or \( \mathcal{T}_{l_1} = 1 \) and \( F \equiv F_1; E_2 \), or \( \mathcal{T}_{l_1} \subseteq \mathcal{T} \) and \( F \equiv \text{nothing} \).

Let us prove fact 1.

If \( \partial o \in \mathcal{E}_l \cup \mathcal{N}_l \) then either \( \partial o \in \mathcal{E}_l \cup \mathcal{N}_l \), or \( \partial = \phi \partial' \), \( \phi \in I(E_1) \), and \( \partial' o \in \mathcal{E}_{l_1} \cup \mathcal{N}_{l_1} \). In the first case, since the activation of \( C_E \) implies the activation of \( C_{E_1} \), the output wire \( o \) of \( C_{E_1} \) stabilizes at 1 by inductive hypothesis on \( E_1 \).

Since this wire is connected, through an or-gate, to the output wire \( o \) of \( C_E \), fact 1 follows. In the second case, since by inductive hypothesis on \( E_2 \) if each wire \( i \) with \( i^+ \in |\phi'| \) (resp. \( i^- \in |\phi'| \)) stabilizes at 1 (resp. 0) then the output wire \( o \) of \( C_{E_2} \) stabilizes at 1, and this wire is connected, through an or-gate, to the output wire \( o \) of \( C_E \), we must prove that \( C_{E_2} \) is activated. This happens since, by inductive hypothesis on \( E_1 \) (fact 4), if each input wire \( i \) with \( i^+ \in |\phi| \) (resp. \( i^- \in |\phi| \)) stabilizes at 1 (resp. 0), and \( \phi \in I(E_1) \), then the wire \( k_0 \) of \( C_{E_1} \) stabilizes at 1. Now, this wire is connected to the wire \( G_0 \) of \( C_{E_2} \).

Fact 2 follows by inductive hypothesis on \( E_1 \) and \( E_2 \), and by the fact that the output wire \( o \) of \( C_E \) is obtained as orring of the output wires \( o \) of \( C_{E_1} \) and \( C_{E_2} \).

Fact 3 follows directly by facts 1 and 2.

If \( \mathcal{T}_l \neq 0 \) then fact 4 follows by inductive hypothesis on \( E_1 \). If \( \mathcal{T}_l = 0 \) then fact 4 follows by inductive hypothesis on \( E_2 \).

- \( E \equiv \text{signal } s \text{ in } E' \text{ end} \).

By rule signal, if \( E \xrightarrow{l} F \) then there exists a transition \( E' \xrightarrow{l'} F' \) such that \( F' \equiv \text{signal } s \text{ in } E' \text{ end} \) and \( l = \text{loc}(s, l') \).

Let us prove fact 1. If \( \partial o \in \mathcal{E}_l \cup \mathcal{N}_l \) then, by definition of function \( \text{loc} \), a causality term \( \partial' o \) is in \( \mathcal{E}_F \cup \mathcal{N}_F \) and one of the following cases holds:

- \( \partial = \partial' \): fact 1 follows by inductive hypothesis.

- \( \partial = \partial'|\phi| \): fact 1 follows by inductive hypothesis if we prove that, when each wire \( i \) such that \( i^+ \in |\phi| \) stabilizes at 1 and each wire \( i \) such that \( i^- \in |\phi| \) stabilizes at 0, then \( s \) stabilizes at 1. Now, this property
follows by inductive hypothesis (fact 1), since, by definition of function \( loc \), there exists a causality term \( \phi s \in \mathcal{E}_\ell \cup \mathcal{N}_\ell \).

- \( \vartheta = \theta_0' [\phi/s^-] \): fact 1 follows by inductive hypothesis if we prove that, when each wire \( i \) such that \( i^+ \in \phi \) stabilizes at 1 and each wire \( i \) such that \( i^- \in \phi \) stabilizes at 0, then \( s \) stabilizes at 0. Now, this property follows by inductive hypothesis (fact 2), since, by definition of function \( loc \), given \( \phi_1 s, \ldots, \phi_m s \) the causality terms in \( \mathcal{E}_\ell \cup \mathcal{N}_\ell \) having \( s \) as action, 

\[
\phi_i = \gamma_{i,1} \cdots \gamma_{i,n_i},
\]

we have that \( \phi = \gamma_{i,1,1} \cdots \gamma_{i,1,j_i} \cdots \gamma_{i,m,1} \cdots \gamma_{i,m,j_m} \).

Let us prove fact 2. If \( \vartheta_1 o, \ldots, \vartheta_v o \in \mathcal{E}_\ell \cup \mathcal{N}_\ell \) then, by definition of function \( loc \), there exist causality terms \( \vartheta_1' o, \ldots, \vartheta_v' o \in \mathcal{E}_\ell \cup \mathcal{N}_\ell \) and we have that \( \vartheta_1, \ldots, \vartheta_v = \vartheta_1, \ldots, \vartheta_1(k_1), \ldots, \vartheta_v(k_m), \ldots, \vartheta_v(m, k_m) \). By inductive hypothesis, the thesis follows if we prove that, for each \( 1 \leq j \leq m \), there exists a wire \( i_j' \) which stabilizes at 0 (resp. 1) and \( i_j' \in [\vartheta_j'] \) (resp. \( i_j' \in [\vartheta_j] \)).

For each \( 1 \leq j \leq m \) we have one of the following cases.

- \( \vartheta((j, 1)) = \vartheta_j' \) and \( k_j = 1 \): in this case we can take \( i_j' = i_{(j, 1)} \).

- \( \vartheta((j, k_j)) = \vartheta_j' [\phi_h/s^+] \), where for \( \phi_h s \in \mathcal{E}_\ell \cup \mathcal{N}_\ell \), \( 1 \leq h \leq k_j \). Now, if \( i_{(j, h)} \not\in [\phi_h] \) for some \( h \) then we can take \( i_j' = i_{(j, h)} \). Otherwise, if \( i_{(j, h)} \in [\phi_h] \) for each \( h \) then \( s \) stabilizes at 0 by inductive hypothesis (fact 2) and we can take \( i_j' = s \).

- \( \vartheta((j, h)) = \vartheta_j' [\phi_h/s^-] \), where for \( \psi_1 s, \ldots, \psi_m s \) the causality terms in \( \mathcal{E}_\ell \cup \mathcal{N}_\ell \) having \( s \) as action, \( \psi_i = \gamma_{i,1} \cdots \gamma_{i,n_i} \), we have that \( \phi_h = \gamma_{i,1,1} \cdots \gamma_{i,1,j_i} \cdots \gamma_{i,m,1} \cdots \gamma_{i,m,j_m} \). If \( i_{(j, h)} \not\in [\phi_h] \) for some \( h \) then we can take \( i_j' = i_{(j, h)} \). Otherwise, if \( i_{(j, h)} \in [\phi_h] \) for each \( h \), then \( s \) stabilizes at 1 by inductive hypothesis (fact 1) and we can take \( i_j' = s \).

Fact 3 follows immediately by fact 1 and fact 2.

Since \( \mathcal{T}_l = \mathcal{T}_\ell \), fact 4 follows by inductive hypothesis if we prove that each input wire \( i \) such that \( i^+ \in S_\ell \) stabilizes at 1 and each input wire \( i \) such that \( i^- \in S_\ell \) stabilizes at 0. Now, by definition of function \( loc \), \( S_l = S_\ell \setminus \{s^-, s^+\} \).

So, if \( s^+ \in S_\ell \) then we must prove that \( s \) stabilizes at 1, while if \( s^- \in S_\ell \) then we must prove that \( s \) stabilizes at 0. If \( s^+ \in S_\ell \) then there exists a causality term \( \vartheta s \in \mathcal{E}_\ell \) with \( [\vartheta] \subseteq S_l \). So, by inductive hypothesis on \( \mathcal{E}' \) (fact 1), if each wire \( i \) such that \( i^+ \in S_l \) stabilizes at 1 and each wire \( i \) such that \( i^- \in S_l \) stabilizes at 0 then \( s \) stabilizes at 1. If \( s^- \in S_\ell \) then, given \( \psi_1 s, \ldots, \psi_m s \) the causality terms in \( \mathcal{N}_\ell \) having \( s \) as action, \( \psi_i = \gamma_{i,1} \cdots \gamma_{i,n_i} \), then \( [\gamma_{i,1} \cdots \gamma_{i,j_i} \cdots \gamma_{i,m,1} \cdots \gamma_{i,m,j_m}] \subseteq S_l \). So, by inductive hypothesis on \( \mathcal{E}' \) (fact 2), if each wire \( i \) such that \( i^+ \in S_l \) stabilizes at 1 and each wire \( i \) such that \( i^- \in S_l \) stabilizes at 0 then \( s \) stabilizes at 0.

- \( E \equiv \text{loop } E' \text{ end.} \)

By rules \text{loop} \_1 and \text{loop} \_2, if \( E \xrightarrow{I} F \) then there exists a transition \( E' \xrightarrow{I} F' \).
such that either $F \equiv F' \land E$ and $T' = 1$, or $F \equiv \text{nothing}$ and $T' \subseteq T$. Facts 1-3 follow by inductive hypothesis on $E'$. Fact 4 follows by inductive hypothesis on $E'$ and by the fact that each wire $k_i, i \geq 1$, of $C_{E'}$ is connected to the wire $k_i$ of $C_E$.

- $E \equiv \text{trap } T \text{ in } E' \text{ end}.$
  By rules $\text{trap}_1, \text{trap}_2$ and $\text{trap}_3$, if $E \xrightarrow{L} F$ then there exists a transition $E' \xrightarrow{L'} F'$ such that either $T' \in \{0, T\}$ and $T_i = 0$, or $T' = 1$ and $T_i = 1$, or $T' \subseteq T, T' \neq \{T\}$ and $T_i = T_i \setminus \{T\}$. Facts 1-3 follow by inductive hypothesis on $E'$. If $T' = 0$ or $T' = \{T\}$ then fact 4 follows by the fact that the wire $k_0$ of $C_E$ is obtained by orring the wires $k_0$ and $k_T$ of $C_{E'}$. If $T' = 1$, then fact 4 follows by the fact that the wire $k_1$ of $C_{E'}$ is connected to the wire $k_1$ of $C_E$. If $T' \subseteq T, T' \neq \{T\}$, then fact 4 follows by the fact that the wire $k_T$ of $C_{E'}$ is connected to the wire $k_{T'}$ of $C_E$, for every $T' \neq T$.

- $E \equiv \text{suspend } E' \text{ when } s.$
  The thesis follows immediately by inductive hypothesis.

This completes the proof.

We say that an event $S_f$ is an input event if and only if it is an assumption over all input signals, namely either $i^+ \in S_f$ or $i^- \in S_f$ for every $i \in I$.

We introduce now a notion of constructiveness of statements.

**Definition 3.3.3** A statement $E$ is constructive if and only if for each input event $S_f$ there exists a transition $E \xrightarrow{L} F$ such that $S_f \uparrow S_f$.

**Example 3.3.4** The statement signal $s$ in $E$ end, where

$E \equiv \text{present } s \text{ then emit } s \text{ else emit } s \text{ end}$

as in Example 3.2.11, is nonconstructive, since no transition has it as source state.

A statement constructive as in Def. 3.3.3 is reactive and deterministic. The reactivity follows by the fact that a reaction exists for every input event. The determinism follows by Prop. 3.3.1.

By Lemma 3.3.2 (facts 3 and 4) it follows that if a statement $E$ is constructive according to Def. 3.3.3, then the circuit $C_E$ is constructive in its initial state. In fact, given an arbitrary value at which input wires stabilize, all output wires stabilize. We prove now that if $E$ is nonconstructive then $C_E$ is nonconstructive in its initial state.

**Lemma 3.3.5** Given a statement $E$ and an input event $S_f$ such that $E \xrightarrow{L} F$ for any label $l$ such that $S_f \uparrow S_f$, when each input wire $i$ such that $i^+ \in S_f$ is kept stable at 1 and each input wire $i$ such that $i^- \in S_f$ is kept stable at 0, then some wire in $C_E$ does not stabilize electrically.
3.3. CORRESPONDENCE BETWEEN SOS AND CIRCUIT SEMANTICS

Proof By structural induction over $E$.

Base case.
If $E$ is a basic statement then, by rules in Table 3.1, we have $E \xrightarrow{l} \textit{nothing}$ and $S_I = \emptyset$. So, $S_I \uparrow S_F$ for any input event $S_I$, and the thesis follows immediately.

Induction step.
The only nontrivial case is that $E \equiv \textit{signal} \ s \textit{ in} \ E'$ \textit{end}. In this case, by the hypothesis one of the following cases follows:

- there does not exist any transition $E' \xrightarrow{l'} F'$ such that $S_I \uparrow S_F$. In this case the thesis follows by inductive hypothesis over $E'$.

- for each transition $E' \xrightarrow{l'} F'$ such that $S_I \uparrow S_F$, the function $\textit{loc}$ is not defined for $s$ and $l'$. We can prove that $s$ does not stabilize. In fact, either there exist causality terms in $E_F$ of the form $\vartheta s$ such that $\{s^-, s^+\} \cap |\vartheta| \neq \emptyset$, or there exist causality terms $\vartheta_1 s, \ldots, \vartheta_n s$ in $E_F \cup N_F$ with $|\vartheta_i| \subseteq S_F \cup \{s^+, s^-\}$. By Lemma 3.3.2, if we consider circuit $C_E$, then the electrical status at which the output wire $s$ stabilizes depends on the electrical status at which the input wire $s$ is kept stable. As a consequence, if the output wire $s$ is connected to the input wire $s$, then $s$ cannot stabilize.

This completes the proof.

Therefore, constructiveness of statements and constructiveness of circuits are related as stated by the following theorem.

Theorem 3.3.6 A statement $E$ is constructive as in Def. 3.3.3 if and only if the circuit $C_E$ is constructive in its initial state.

Proof "If": by Lemma 3.3.5. "Only if": by Lemma 3.3.2.

We show now that the LTS defined by rules in Table 3.1 carries information sufficient to infer how circuits evolve state-by-state.

Lemma 3.3.7 Given a constructive statement $E$ and a transition $E \xrightarrow{l} F$ such that $\tau_l = 1$, if circuit $C_E$ is activated and each input wire $i$ such that $i^+ \in S_I$ (resp. $i^- \in S_I$) is kept stable at 1 (resp. 0), then, when $C_E$ will be resumed, it will behave as $C_F$.

Proof By structural induction over $E$.

Base case.
If $E \equiv \textit{pause}$ then $F \equiv \textit{nothing}$. At the next execution cycle, since wire $\textit{RES}$ and the latch will be set, only the output wire $k_0$ will stabilize at 1. This means that the thesis follows, since when circuit implementing $\textit{nothing}$ is activated, only the output wire $k_0$ stabilizes at 1.

If $E$ is a basic statement and $E \neq \textit{pause}$, then $\tau_l \neq 1$ for every label $l$ such that
\( E \xrightarrow{l} \), so that the thesis follows immediately.

**Induction step.**

We must consider the following cases.

- \( E \equiv \text{present } s \text{ then } E_1 \text{ else } E_2 \text{ end.} \)
  
  By rules \( \text{present}_1 \) and \( \text{present}_2 \), if \( E \xrightarrow{l} F \) then there exist transitions \( E_1 \xrightarrow{l_1} F_1 \) and \( E_2 \xrightarrow{l_2} F_2 \) such that either \( l = s^+(l_1, l_2) \) and \( F \equiv F_1 \), or \( l = s^-(l_2, l_1) \) and \( F \equiv F_2 \). Let us assume the first case. The other is analogous. The thesis follows by inductive hypothesis on \( E_1 \) and by the fact that if \( s \) stabilizes at 1 then the activation of \( C_E \) implies that \( C_{E_1} \) is activated and that \( C_{E_2} \) is not activated. So, the wire \( \text{SEL} \) of \( C_{E_2} \) will not be set at the following cycle and \( C_{E_2} \) will not be resumed.

- \( E \equiv E_1 \parallel E_2. \)
  
  By rule \( \text{parallel}_1 \), if \( E \xrightarrow{l} F \) with \( T_i = 1 \), then there exist transitions \( E_1 \xrightarrow{l_1} F_1 \) and \( E_2 \xrightarrow{l_2} F_2 \) such that \( l = l_1 \otimes l_2 \) and \( F = F_1 \parallel F_2. \) So, the thesis follows by inductive hypothesis on \( E_1 \) and \( E_2 \) and by the fact that the activation of \( C_E \) implies that both \( C_{E_1} \) and \( C_{E_2} \) are activated.

- \( E \equiv E_1; E_2. \)
  
  By rules \( \text{seq}_1 \) and \( \text{seq}_2 \), if \( E \xrightarrow{l} F \) with \( T_i = 1 \), then there exist transitions \( E_1 \xrightarrow{l_1} F_1 \) and \( E_2 \xrightarrow{l_2} F_2 \) such that \( l = l_1 \triangleright l_2 \), and either \( T_{i_1} = 0, T_{i_2} = 1 \) and \( F \equiv F_2 \), or \( T_{i_1} = 1 \) and \( F \equiv F_1; E_2 \). In the first case the thesis follows by inductive hypothesis on \( E_2 \), by the fact that the activation of \( C_E \) implies that \( C_{E_1} \) is activated, and by the fact that Lemma 3.3.2 (fact 4) implies that the wire \( K_0 \) of \( C_{E_1} \) stabilizes at 1 so that \( C_{E_2} \) is activated. In the second case the thesis follows by inductive hypothesis on \( E_1 \), by the fact that the activation of \( C_E \) implies that \( C_{E_1} \) is activated, by the fact that Lemma 3.3.2 (fact 4) implies that the wire \( K_0 \) of \( C_{E_1} \) stabilizes at 0 so that \( C_{E_2} \) is not activated.

- \( E \equiv \text{signal } s \text{ in } E' \text{ end.} \)
  
  By rule \( \text{signal} \), if \( E \xrightarrow{l} F \) then there exists a transition \( E' \xrightarrow{l} F' \) such that \( l = loc(s, l') \) and \( F \equiv \text{signal } s \text{ in } F' \text{ end.} \) So, the thesis follows by inductive hypothesis on \( E' \).

- \( E \equiv \text{loop } E' \text{ end.} \)
  
  By rule \( \text{loop}_1 \), if \( E \xrightarrow{l} F \) with \( T_i = 1 \), then there exists a transition \( E' \xrightarrow{l} F' \) such that \( F \equiv F'E. \) The thesis follows by inductive hypothesis on \( E' \) and by the fact that \( C_E \) behaves as circuit \( C_{E' ; E} \).

- \( E \equiv \text{suspend } E' \text{ when } s. \)
  
  By rule \( \text{suspend}_2 \), if \( E \xrightarrow{l} F \) with \( T_i = 1 \), then \( E' \xrightarrow{l} F' \) and \( F \equiv \text{suspend imm } F' \text{ when } s. \) The thesis follows since, at the next cycle, if \( s \)
stabilizes at 1 then the wire $k_1$ of $C_{E'}$ stabilizes at 1, while if $s$ stabilizes at 0 then $C_{E'}$ is resumed and, by inductive hypothesis, it behaves as $C_{E'}$.

* $E \equiv \text{trap } T \text{ in } E'$ end.

By rule trap-2, if $E \xrightarrow{l} F$ with $T_l = 1$ then there exists a transition $E' \xrightarrow{l'} F'$ such that $F \equiv \text{trap } T \text{ in } F'$ end and $l = tr(T, l')$. So, the thesis follows by inductive hypothesis on $E'$.

This completes the proof.

We introduce now the definition of constructiveness of modules.

**Definition 3.3.8** A module $M$ is constructive if and only if, for each sequence of transitions $E_0 \xrightarrow{l_0} E_1, \ldots, E_{n-1} \xrightarrow{l_{n-1}} E_n$ such that $E_0 \equiv E_M$, statements $E_0, \ldots, E_n$ are constructive.

Constructiveness of modules as in Def. 3.3.8 and constructiveness of circuits are related as stated by the following theorem.

**Theorem 3.3.9** A module $M$ is constructive as in Def. 3.3.8 if and only if the circuit $C_M$ is constructive.

**Proof** Directly by Theorem 3.3.6 and Lemma 3.3.7.

Note that modules have finite states and their constructiveness is decidable.
Chapter 4

An axiomatization of Esterel

In this chapter we give an axiomatic semantics for Esterel. Our aim is to define an equality relation on Esterel statements so to characterize behavioral equivalent statements. Axiomatic semantics may be used for transformation of programs and for proof by rewriting. To the best of our knowledge, no axiomatic semantics has been given up to now, neither for Esterel, nor for any other synchronous language.

As a notion of behavioral equivalence over statements, we consider bisimulation. Our choice is justified by the fact that, by Theorem 3.2.15, Lemma 3.3.2 and Lemma 3.3.7, bisimilar statements are not distinguished by any Esterel context and are mapped to circuits which cannot be distinguished by the external environment.

We give a system of inference rules (conditional axioms) giving an axiomatization over Esterel. Then, we prove that this axiomatization is sound and complete modulo bisimulation on constructive Esterel statements, namely we prove that bisimilar constructive statements are exactly those equated by our axiomatization. This means that, given an arbitrary constructive Esterel statement $E$, we obtain exactly all statements bisimilar to $E$ by applying our axioms.

The chapter is organized as follows. In Section 4.1 we give our set of axioms and the proof of soundness. In Section 4.2 we give the proof of completeness.

4.1 The axiomatization

We begin with introducing the process algebra Esterel$^+$, which strictly contains the terms of the process algebra Esterel. Then we provide an axiomatization over Esterel$^+$ and we prove that this is sound and complete modulo bisimulation on constructive Esterel statements.

Techniques to give axiomatizations for a superset of a given language are well-established in literature. As an example, in [73] it is proved that process algebras offering operations of nondeterministic choice, prefixing and merge can be finitely axiomatized modulo bisimulation only by extending the original signature. In [11] it is shown how a finite axiomatization can be given by adding to the original signature
the “left merge” operation. In [2] an algorithm is given to construct from a GSOS language $L$ both a superset $L'$ of $L$ and a finite unconditional axiomatization $\mathcal{A}$ such that $\mathcal{A}$ and the Approximation Induction Principle (AIP) [12, 35] together are sound and complete modulo bisimulation on $L'$. Intuitively, AIP is an infinitary conditional axiom which states that two processes are equated provided that all their finite projections are. In general, one is interested in avoiding infinitary axioms like AIP. In [1] an algorithm is given to construct from a regular GSOS language $L$ both a superset $L'$ of $L$ and a finite unconditional axiomatization $\mathcal{A}$ such that $\mathcal{A}$, the Recursive Definition Principle (RDP) and the Recursion Specification Principle (RSP) [70, 13] together are sound and complete modulo bisimulation on $L'$.

We cannot exploit here results of [2, 1]. In those papers, process algebras offering the operation of summation “+” and the operation of prefixing “.” are considered. Prefixing permits to prefix a process $p$ by an action $a$, where $a$ is an action observable by the external world, namely an action that may be a label of the LTS. Summation denotes the nondeterministic choice. In [2, 1], like in [70], the idea is to have axioms such that, for a given process $p$, one can infer $p = \Sigma_{i\in I} a_i \cdot p_i$. Namely, the idea is to transform every process into a “head normal form”. We cannot have head normal forms for Esterel, because, in general, we cannot simulate concurrent programs by sequential ones.

So, we propose a countably infinite axiomatization over Esterel$^+$ which is sound and complete modulo bisimulation on Esterel, all axioms are finitary, all axioms except RSP are unconditional.

Let us assume a set of recursion variables $\text{Var}$ ranged over by $P$. The terms (statements) of the algebra Esterel$^+$ are those generated by the following BNF-like grammar:

$$E ::= \text{nothing} \mid \text{emit } s \mid \text{pause} \mid \text{present } s \text{ then } E \text{ else } E \text{ end } \mid E \ || \ E \mid E; E \mid \text{signal } s \text{ in } E \text{ end } \mid \text{loop } E \text{ end } \mid \text{suspend } E \text{ when } s \mid \text{trap } T \text{ in } E \text{ end } \mid \text{exit } T \mid \text{rec } P.E \mid P$$

where $s$, $T$ and $P$ range over $S$, $T$ and $\text{Var}$, respectively.

Construct $\text{rec}$ is analogous to recursion constructs of most process algebras. Its operational semantics is defined by the following transition rule:

$$\frac{E[\text{rec } P.E/P] \xrightarrow{I} F}{\text{rec } P.E \xrightarrow{I} F} \quad (\text{rec}).$$

Construct $\text{rec}$ permits to simulate behaviors that can be defined by combining construct $\text{loop}$ and mechanism $\text{trap}-\text{exit}$, as illustrated by the following example.

**Example 4.1.1** Let us assume the Esterel$^+$ statements $E_1$ and $E_2$ such that:

$E_1 \equiv \text{rec } P.\text{ present } s \text{ then } (\text{pause}; P) \text{ else } \text{nothing } \text{ end }$,

$E_2 \equiv \text{trap } T \text{ in } (\text{loop } (\text{present } s \text{ then } \text{pause } \text{ else } \text{exit } T \text{ end}) \text{ end}) \text{ end }$. 
4.1. THE AXIOMATIZATION

We have \( E_1 \approx E_2 \). In fact, \( E_1 \xrightarrow{\ell} \text{nothing}; E_1, E_1 \xrightarrow{\ell} \text{nothing}, E_2 \xrightarrow{\ell} \text{trap } T \) in \((\text{nothing}; \text{loop } (\text{present } s \text{ then pause else exit } T \text{ end}) \text{ end})\). \( E_2 \xrightarrow{\ell} \text{nothing} \), where \( \ell = (\{s^+\}, \{s^+ \cdot \}, \emptyset, 1) \) and \( \ell = (\{s^-\}, \emptyset, \{s^+ \cdot \}, 0) \), and, in general, \( E \approx \text{nothing}; E \) for any \( E \).

Before presenting the axiomatization, we introduce some notations.

Given a signal \( s \in S \) and a statement \( E \), we write if \( s^+ \text{ then } E \) for present \( s \) then \( E \) else nothing end, and we write if \( s^- \text{ then } E \) for present \( s \) then nothing else \( E \) end.

Moreover, given an ordered event \( \vartheta \in (S^+)^* \), we denote with if \( \vartheta \text{ then } E \) the statement if \( \vartheta \text{ then } E \equiv \begin{cases} E & \text{if } \vartheta = \epsilon \\ \text{if } \gamma \text{ then } \phi \text{ then } E & \text{if } \vartheta = \gamma \phi. \end{cases} \)

Given a statement \( E \), we say that an occurrence of a variable \( P \in V \) or is free in \( E \) if it does not appear in any statement \( \text{rec } P \cdot F \) in the body of \( E \).

A variable \( P \) is free in \( E \) if an occurrence of \( P \) is free in \( E \). The set of variables free in \( E \) is denoted with \( \text{Free}(E) \).

A variable \( P \) is guarded in \( E \) if and only if each free occurrence of \( P \) in \( E \) appears in a subexpression \( F;P \) such that \( F \) cannot terminate immediately.

Intuitively, if we consider a statement \( \text{rec } P \cdot E \) with \( P \) free in \( E \), then we are sure that if \( P \) is guarded in \( E \) then every reaction of \( \text{rec } P \cdot E \) is finite. As an example, let us consider statements \( E_1 \equiv \text{pause}; P \) and \( E_2 \equiv \text{emit } s; P \). Variable \( P \) is guarded in \( E_1 \) but not in \( E_2 \). Now, \( \text{rec } P \cdot E_1 \) performs a statement \( \text{pause} \) at each execution cycle, while \( \text{rec } P \cdot E_2 \) executes \( \text{emit } s \) infinitely times at the first execution cycle.

We consider the axiomatization over \( \text{Esterel}^+ \) given by axioms in Tables 4.1-4.7 below. We explain axioms and we prove their soundness modulo bisimulation.

\[
E \parallel F = F \parallel E \quad (\parallel_1) \quad E \parallel \text{nothing} = E \quad (\parallel_3)
\]
\[
E \parallel (F \parallel G) = (E \parallel F) \parallel G \quad (\parallel_2) \quad E \parallel E = E \quad (\parallel_4)
\]

Table 4.1: Axioms for “\( \parallel \)”.

Axioms \( \parallel_1, \parallel_2 \) and \( \parallel_3 \) in Table 4.1 state that construct \( \parallel \) is commutative and associative and has \text{nothing} as neutral element. Axiom \( \parallel_2 \) allows to denote with \( E \parallel F \parallel G \) both statements \( E \parallel (F \parallel G) \) and \((E \parallel F) \parallel G \). Axiom \( \parallel_4 \) follows by the fact that every statement \( E \) is deterministic and by the fact that statements running in parallel are perfectly synchronized.

Soundness modulo bisimulation of axioms in Table 4.1 follows by the fact that \( \otimes \) is commutative and associative (see Prop. 3.2.7-3.2.8), has \( \emptyset, \{en\}, \emptyset, 0 \) as neutral element, and is such that \( l \otimes l = l \) for every label \( l \) constructed by means of rules in Table 3.1.
present \( s \) then \( E \) else \( F \) end = if \( s^+ \) then \( E \) \| if \( s^- \) then \( F \)   \( (?1) \)
if \( s^+ \) then (\( E \parallel F \)) = if \( s^+ \) then \( E \) \| if \( s^+ \) then \( F \)   \( (?2) \)
if \( s^- \) then (\( E \parallel F \)) = if \( s^- \) then \( E \) \| if \( s^- \) then \( F \)   \( (?3) \)

Table 4.2: Axioms for present _then _else _end.

Axiom \(?1\) in Table 4.2 is justified by the fact that, in both statements, \( E \) is executed if \( s \) is present, while \( F \) is executed if \( s \) is absent.
Axiom \(?2\) is justified by the fact that both \( E \) and \( F \) are executed if and only if \( s \) is present.
Axiom \(?3\) is analogous.

To prove the soundness of axiom \(?1\), let us consider statements \( H \equiv \text{present } s \) then \( E \) else \( F \) end and \( H' \equiv \text{if } s^+ \text{ then } E \parallel \text{if } s^- \text{ then } F \), and let us assume that \( E \xrightarrow{t_1} E' \) and \( F \xrightarrow{t_2} F' \).
We have \( H \xrightarrow{t^+_{1,2}} E' \). Now, \( E \xrightarrow{t_1} E' \) if \( s^+ \) then \( E \xrightarrow{t} E' \), where \( l'_1 = s^+(l_1, \delta) \).
Moreover, \( F \xrightarrow{t_2} F' \) if \( s^- \) then \( F \xrightarrow{t'_{2}} \text{nothing} \), where \( l''_2 = s^+(\delta, l_2) \). So, we have \( H' \xrightarrow{t_{1,2}} E' \parallel \text{nothing} \), where \( l'_1 \otimes l''_2 = s^+(l_1, l_2) \) and, by axioms in Table 4.1, \( E' \approx E' \parallel \text{nothing} \).
Analogously, we have \( H \xrightarrow{t} F' \) iff \( H' \xrightarrow{t} \text{nothing} \parallel F' \), \( l = s^- (l_2, l_1) \), and, therefore, \( H \approx H' \).

Let us consider now axiom \(?2\) and statements \( H \equiv \text{if } s^+ \text{ then } (E \parallel F) \) and \( H' \equiv \text{if } s^+ \text{ then } E \parallel \text{if } s^+ \text{ then } F \).
We have \( H \xrightarrow{t} F' \parallel F' \) iff \( E \xrightarrow{t_1} E', F \xrightarrow{t_2} F' \) and \( l = s^+(l_1 \otimes l_2, \delta) \). Now, \( E \xrightarrow{t_1} E' \) and \( F \xrightarrow{t_2} F' \) iff \( H' \xrightarrow{t} E' \parallel F' \), where \( l' = s^+(l_1, \delta) \otimes s^+(l_2, \delta) \). Note that \( l = l' \).
Moreover, we have \( H \xrightarrow{t} \text{nothing} \) iff \( E \xrightarrow{t_1} E', F \xrightarrow{t_2} F', l = s^- (\delta, l_1 \otimes l_2) \). Now, \( E \xrightarrow{t_1} E', F \xrightarrow{t_2} F' \) iff \( H' \xrightarrow{t} \text{nothing} \parallel \text{nothing} \), where \( l' = s^- (\delta, l_1) \otimes s^- (\delta, l_2) \). Note that \( l = l' \), and, by axioms in Table 4.1, \( \text{nothing} \approx \text{nothing} \parallel \text{nothing} \).
So, it follows that \( H \approx H' \).

The soundness of axiom \(?3\) can be proved analogously.

Axiom \(\|_5\) in Table 4.3 is justified by the fact that the statement in the right side of “=” pauses, independently of the status of signal \( s \), if the environment prompts every signal \( s \) such that \( s^+ \in \{\varnothing\} \) and does not prompt any signal \( s \) such that \( s^- \in \{\varnothing\} \). Moreover, in this case, both statements will behave as \( E \) at the next execution cycle. Note that we could not have an analogous axiom with an arbitrary statement replacing pause; \( E \). As an example, we could not have \( H \equiv H' \), for \( H \equiv \text{emit } z \) and \( H' \equiv \text{emit } z \parallel \text{if } s^+ \text{ then } \text{emit } z \parallel \text{if } s^- \text{ then } \text{emit } z \). In fact, \( H' \) terminates only when the status of signal \( s \) is known. So, signal \( s, z \) in \( (H; \text{if } z \text{ then } \text{emit } s) \) end is constructive, while signal \( s, z \) in \( (H'; \text{if } z \text{ then } \text{emit } s) \) end is nonconstructive.

Axiom \(\|_5\) is justified by the fact that, if the environment prompts every signal
\[ s^-, s^+ \not\in [\vartheta] \Rightarrow \begin{array}{ll}
    \text{if } \vartheta \text{ then } \text{pause}; E
  & \text{if } \vartheta \text{ then } \text{pause} \quad (\|5) \\
  & \text{if } \vartheta s^+ \text{ then } \text{pause}; E \\
  & \text{if } \vartheta s^- \text{ then } \text{pause}; E
\end{array}
\]
\[ \gamma \not\in [\vartheta] \Rightarrow \begin{array}{ll}
    \text{if } \vartheta \text{ then } \text{pause} = \text{if } \vartheta \text{ then } \text{pause} \quad (\|6) \\
  & \text{if } \vartheta \gamma \text{ then } \text{pause}
\end{array}
\]
\[ \begin{array}{ll}
    [\vartheta] = [\vartheta'] \Rightarrow & \begin{array}{ll}
      \text{if } \vartheta \text{ then } \text{pause}; E
    & \text{if } \vartheta \text{ then } \text{pause}; (E \parallel F) \quad (\|7) \\
    & \text{if } \vartheta' \text{ then } \text{pause}; F
    & \text{if } \vartheta' \text{ then } \text{pause}
    
    \end{array}
\end{array}
\]
\[ \begin{array}{ll}
    [\vartheta] \subseteq [\vartheta'] \Rightarrow & \begin{array}{ll}
      \text{if } \vartheta \text{ then } \text{exit } T
    & \text{if } \vartheta' \text{ then } \text{pause}; E
    & \text{if } \vartheta' \text{ then } \text{pause}
    
    \end{array}
\end{array}
\]
\[ \gamma \not\in [\vartheta] \Rightarrow \begin{array}{ll}
    \text{if } \vartheta \text{ then } \text{pause} = \text{if } \vartheta \text{ then } \text{pause} \quad (\|8) \\
    & \text{if } \vartheta \gamma \text{ then } \text{nothing}
\end{array}
\]

Table 4.3: Other axioms for “\(\|\)".

\(s\) such that \(s^+ \in [\vartheta]\) and does not prompt any signal \(s\) such that \(s^- \in [\vartheta]\), then the statement in the right side of “=“ pauses independently of the status of the signal \(s\) such that \(\gamma \in \{s^+, s^-, \vartheta\}\).

Axiom \(\|7\) is justified by the fact that in both statements, \(E\) and \(F\) start in the same cycle and run concurrently.

Axiom \(\|8\) is justified by the fact that both statements cannot pause.

Axiom \(\|9\) is justified by the fact that, if the environment prompts every signal \(s\) such that \(s^+ \in [\vartheta]\) and does not prompt any signal \(s\) such that \(s^- \in [\vartheta]\), then the statement in the right side of “=“ pauses independently of the status of the signal \(s\) such that \(\gamma \in \{s^+, s^-, \vartheta\}\).

To prove the soundness of axiom \(\|5\), let us consider statements
\(H \equiv \text{if } \vartheta \text{ then } \text{pause}; E\) and
\(H' \equiv \text{if } \vartheta \text{ then } \text{pause} \parallel \text{if } \vartheta s^+ \text{ then } \text{pause}; E \parallel \text{if } \vartheta s^- \text{ then } \text{pause}; E\).

We have \(H \rightarrow nothing; E\) if either \(H' \rightarrow nothing \parallel nothing; E \parallel nothing\) or
\(H' \rightarrow nothing \parallel nothing \parallel nothing; E\), where \(l = \langle [\vartheta], \{\vartheta p\}, \emptyset, 1\rangle\),
\(l_1 = l \otimes \langle [\vartheta] \cup \{s^+, \vartheta s^+ p\}, \emptyset, 1 \rangle \otimes \langle [\vartheta] \cup \{s^+, \emptyset, \vartheta s^- p\}, 0 \rangle\),
\(l_2 = l \otimes \langle [\vartheta] \cup \{s^-, \emptyset, \vartheta s^+ p\}, 0 \rangle \otimes \langle [\vartheta] \cup \{s^-, \vartheta s^- p\}, \emptyset, 1 \rangle\).
Now, \( l = l_1 \). In fact, \( \partial s^+ p \not\in \mathcal{E}_t \) and \( \partial s^- p \not\in \mathcal{N}_t \), since \( \partial p \in \mathcal{E}_t \), and \( s^+ \not\in \mathcal{S}_t \). Analogously, \( l = l_2 \). Moreover, by axiom \( \|_3 \), nothing; \( E \approx \text{nothing} \| \text{nothing}; E \| \text{nothing and nothing}; E \approx \text{nothing} \| \text{nothing}; E \| \text{nothing}. \)

Let \( \partial = \gamma_1 \ldots \gamma_n \). We have \( H \xrightarrow{L} \text{nothing} \) with \( l = \langle \{\gamma_1, \ldots, \gamma_n\}, \emptyset, \{\partial p\}, 0 \rangle \) iff \( H' \xrightarrow{L'} \text{nothing} \| \text{nothing} \| \text{nothing} \), where \( l' = \langle \{\gamma_1, \ldots, \gamma_n\}, \emptyset, \{\partial p\}, 0 \rangle \otimes \langle \{\gamma_1, \ldots, \gamma_n\}, \emptyset, \{\partial s^- p\}, 0 \rangle \). Now, \( l = l' \) and nothing \( \approx \text{nothing} \| \text{nothing} \| \text{nothing} \).

It follows that \( H \approx H' \).

The soundness of axioms \( \|_6 \) and \( \|_9 \) could be proved analogously.

To prove the soundness of axiom \( \|_7 \), let us consider statements

\begin{align*}
H \equiv & \text{if } \partial \text{ then pause}; E \| \text{if } \partial' \text{ then pause}; F \text{ and} \\
H' \equiv & \text{if } \partial \text{ then pause}; (E \| F) \| \text{if } \partial' \text{ then pause}, \text{ with } |\partial| = |\partial'|.
\end{align*}

We have \( H \xrightarrow{L} \text{nothing} \| \text{nothing} \iff H' \xrightarrow{L} \text{nothing} \| \text{nothing} \), and \( H \xrightarrow{L} \text{nothing}; E \| \text{nothing}; F \iff H' \xrightarrow{L} (\text{nothing}; (E \| F)) \| \text{nothing} \), where nothing; \( E \| \text{nothing}; F \approx (\text{nothing}; (E \| F)) \| \text{nothing} \) by axiom \( \|_3 \) and by the fact that nothing; \( E \approx E \) for every statement \( E \). So, \( H \approx H' \).

To prove the soundness of axiom \( \|_8 \), let us consider statements

\begin{align*}
H \equiv & \text{if } \partial \text{ then exit } T \| \text{if } \partial' \text{ then pause}; E \text{ and} \\
H' \equiv & \text{if } \partial \text{ then exit } T \| \text{if } \partial' \text{ then pause}, \text{ with } |\partial| \subseteq |\partial'|.
\end{align*}

We have \( H \xrightarrow{L} \text{nothing} \| \text{nothing} \) with \( \tau_1 = 0 \) if \( H' \xrightarrow{L} \text{nothing} \| \text{nothing} \), and \( H \xrightarrow{L} \text{nothing} \) with \( \tau_1 \subseteq \tau \) if \( H' \xrightarrow{L} \text{nothing} \). So, \( H \approx H' \).

\[
P \text{ guarded in } E, F = E [F/P] \Rightarrow \quad \text{rec } P.E = E \left[ \text{rec } P.E / P \right] \quad (\text{rec}_1) \\
\text{loop } E \text{ end } = \text{rec } P.(E; P) \quad (\text{loop}_1)
\]

Table 4.4: Axioms for \text{rec} and \text{loop } \text{ end}.

Axioms \( \text{rec}_1 \) and \( \text{rec}_2 \) in Table 4.4 are standard. They correspond to the Recursive Definition Principle and to the Recursive Specification Principle, respectively. Axiom \( \text{loop}_1 \) states that construct \text{rec} embeds construct \text{loop}.

The soundness of axiom \( \text{rec}_1 \) follows directly by transition rule \text{rec}.

Since \( \text{rec}_1 \) is sound, to prove the soundness of \( \text{rec}_2 \) it is sufficient to prove that, for any pair of statements \( F \) and \( G \), \( F \approx E[F/P] \) and \( G \approx E[G/P] \) imply \( F \approx G \). To this purpose, let us assume the relation \( \mathcal{R} = \{(E'[F/P], E'[G/P])\} \) and let us prove that \( \mathcal{R} \) is a bisimulation.

Since \( F \approx E[F/P] \) and \( \approx \) is a congruence, \( E'[F/P] \xrightarrow{L} \text{iff } E'[E[F/P]/P] \xrightarrow{L} \).

Analogously, \( E'[G/P] \xrightarrow{L} \text{iff } E'[E[G/P]/P] \xrightarrow{L} \). Since \( P \) is guarded in \( E \), we have \( E'[E[F/P]/P] \xrightarrow{L} \text{iff } E'[E[G/P]/P] \xrightarrow{L} \). Therefore, \( E'[F/P] \xrightarrow{L} H_1 \text{ iff } E'[G/P] \xrightarrow{L} H_2 \), for some \( H_1 \) and \( H_2 \). Note that \( H_1 \equiv E''[F/P] \) and \( H_2 \equiv E''[G/P] \).
for some statement \( E'' \), namely \( (H_1, H_2) \in R \).

So, we have proved that \( R \subseteq \mathcal{F}(R) \), namely that \( R \) is a bisimulation. Since \( (E[F/P], E[G/P]) \in R, F \approx E[F/P], \) and \( G \approx E[G/P], \) we have that \( F \approx G. \)

The soundness of axiom \( loop_1 \) follows directly by transition rules \( rec, \) \( loop_1 \) and \( loop_2. \)

We introduce now the notion of normal form which will be used to give axioms for constructs ,, trap, suspend and signal.

**Definition 4.1.2** A statement \( E \) is a normal form if and only if there exist statements \( F_1, \ldots, F_n, F_i \neq F_j, i \neq j, \) such that:

1. \( E \equiv F_1 \parallel \cdots \parallel F_n, \) and, for each \( 1 \leq i \leq n, \) we have that \( F_i \equiv \text{if } \vartheta_i \text{ then } G_i, \)
   where either \( G_i \equiv P, \) or \( G_i \) is a basic statement, or \( |\vartheta_i| \cap \{s^-, s^+\} \neq \emptyset \) for every \( s \in S \) and \( G_i \equiv \text{pause}; E_f(i). \)

2. if \( G_i \equiv \text{pause}; E_f(i) \) and \( G_j \equiv \text{pause}; E_f(j), 1 \leq i, j \leq n, i \neq j, \) then \( |\vartheta_i| \nvdash |\vartheta_j|; \)

3. if \( G_i \equiv \text{pause}; E_f(i) \) then there does not exist any \( 1 \leq j \leq n \) such that \( G_j \equiv \text{exit } T \) and \( |\vartheta_j| \subseteq |\vartheta_i|; \)

4. if either \( G_i \equiv \text{pause} \) or \( G_i \equiv \text{nothing} \) then there does not exist any \( 1 \leq j \leq n \) such that \( \vartheta_i = \vartheta_j \phi \) for some \( \phi \in (S^+) \), and \( G_j \equiv \text{pause}; \)

5. if \( G_i \equiv \text{nothing} \) and \( \vartheta_i = \phi \gamma, \) then there does not exist any \( 1 \leq j \leq n \) such that either \( \vartheta_j = \phi \gamma \psi \) or \( \vartheta_j = \phi \gamma \psi, \) for any \( \psi \in (S^+) \).

6. if \( G_i \equiv \text{pause} \) then there does not exist any \( 1 \leq j \leq n \) such that \( G_j \equiv \text{pause}; E_f(j) \) and \( \vartheta_j = \vartheta_i; \)

7. if \( G_i \equiv \text{pause} \) then, for each \( \vartheta \) such that \( \{s^-, s^+\} \cap |\vartheta| \neq \emptyset \) for every \( s \in S \) and \( |\vartheta_i| \subseteq |\vartheta|, \) there exists \( 1 \leq j \leq n \) such that either \( |\vartheta| = |\vartheta_j| \) and \( G_j \equiv \text{pause}; E_f(j), \) or \( |\vartheta_j| \subseteq |\vartheta| \) and \( G_j \equiv \text{exit } T. \)

Let us consider a normal form \( E \equiv F_1 \parallel \cdots \parallel F_n \) as in Def. 4.1.2 above.

Let us assume that \( G_i \equiv \text{pause}; E_f(i) \). Conditions 2 and 3 of Def. 4.1.2 imply that if the environment prompts each signal \( s \) such that \( s^+ \in |\vartheta_i| \) and does not prompt any signal \( s \) such that \( s^- \in |\vartheta_i|, \) so that \( G_i \) is executed at this cycle, then the statement \text{nothing} \parallel \cdots \parallel \text{nothing} \parallel E_f(i) \parallel \cdots \parallel \text{nothing} \) will be executed at the next cycle. In fact, by condition 2, if a statement \( F_j \) pauses then \( F_j \equiv \text{pause}, j \neq i, \) and, by condition 3, no statement \( F_j \) exits any \text{trap}, \( j \neq i. \)

Condition 4 implies that redundant statements of the form \text{if } \vartheta \text{ then pause} \) or of the form \text{if } \vartheta \text{ then nothing} \) do not appear as parallel components of \( E. \) As an example, the statement \( E \) in Example 3.2.5 does not satisfy this condition.
Condition 5 implies that redundant statements of the form if \( \vartheta \) then nothing do not appear as parallel components of \( E \). As an example, the statement \( E \) in Example 3.2.6 does not satisfy this condition.

Condition 6 implies that redundant statements of the form if \( \vartheta \) then pause do not appear as parallel components of \( E \).

Note that conditions 4 and 5 imply that if \( F_i \equiv \text{if } \vartheta_i \text{ then nothing} \) then there exists a label \( l \) such that \( E \xrightarrow{l} \) and \( \vartheta_i \) is in \( \mathcal{E}_i \).

Condition 4 imply that if \( F_i \equiv \text{if } \vartheta_i \text{ then pause} \) then there exists a label \( l \) such that \( E \xrightarrow{l} \) and \( \vartheta_i \) is in \( \mathcal{E}_i \).

\[
E \text{ normal form and } Free(E) = \emptyset \Rightarrow \begin{align*}
F : (F \parallel G) &= (E : F) \parallel (E : G) \quad (seq_1) \\
F : H &= E^H \quad (seq_2) \\
\text{pause; nothing} &= \text{pause} \quad (seq_3)
\end{align*}
\]

Table 4.5: Axioms for "i:"

Axiom \( seq_1 \) in Table 4.5 is justified by the fact that, in both statements, both \( F \) and \( G \) start exactly when \( E \) terminates and, then, they run in parallel.

Let us assume \( H \equiv E : (F \parallel G) \) and \( H' \equiv (E ; F) \parallel (E ; G) \), where \( E \xrightarrow{l} E' \), \( F \xrightarrow{l_2} F' \) and \( G \xrightarrow{l_3} G' \).

If \( \mathcal{T}_{i_1} = 1 \) then we have \( H \xrightarrow{l} (E ; F) \parallel (G ; F) \) iff \( H' \xrightarrow{l'} (E' ; F) \parallel (E' ; G) \), where \( l = l_1 \triangleright (l_2 \otimes l_3) \) and \( l' = (l_1 \triangleright l_2) \otimes (l_1 \triangleright l_3) \).

If \( \mathcal{T}_{i_2} = 0, \mathcal{T}_{i_3} \in \{0, 1\} \), then we have \( H \xrightarrow{l} (E ; F) \parallel (G ; F) \) iff \( H' \xrightarrow{l'} (E' ; F) \parallel (E' ; F) \), where \( l = l_1 \triangleright (l_2 \otimes l_3) \) and \( l' = (l_1 \triangleright l_2) \otimes (l_1 \triangleright l_3) \).

If either \( \mathcal{T}_{i_1} \subseteq \mathcal{T} \) or \( \mathcal{T}_{i_2} = 0 \) and \( \mathcal{T}_{i_3} \subseteq \mathcal{T} \) or \( \mathcal{T}_{i_3} \subseteq \mathcal{T} \), then we have \( H \xrightarrow{l} \text{nothing} \) iff \( H' \xrightarrow{l'} \text{nothing} \), where \( l = l_1 \triangleright (l_2 \otimes l_3) \) and \( l' = (l_1 \triangleright l_2) \otimes (l_1 \triangleright l_3) \).

Since \( l_1 \triangleright (l_2 \otimes l_3) = (l_1 \triangleright l_2) \otimes (l_1 \triangleright l_3) \), it follows that \( H \approx H' \).

Note that we could not have the distribute law \((E \parallel F) ; G = (E ; G) \parallel (F ; G)\), since the occurrence of \( G \) in \((E \parallel F) ; G\) starts when both \( E \) and \( F \) have terminated, while an occurrence of \( G \) in \((E ; G) \parallel (F ; G)\) starts when either \( E \) or \( F \) terminates.

Let us consider axiom \( seq_2 \) in Table 4.5. Given a normal form \( E \equiv (F_1 \parallel \ldots \parallel F_n) \), such that \( F_i \equiv \text{if } \vartheta_i \text{ then } G_i \), we denote with \( E^H \) the statement \( F_1^H \parallel \ldots \parallel F_n^H \parallel F \), where:

- \( F_i^H = \begin{cases} \text{if } \vartheta_i \text{ then pause;} & (E_{f(i)} ; H) \quad \text{if } F_i \equiv \text{if } \vartheta_i \text{ then pause;} \quad (E_{f(i)}) \\
F_i & \text{otherwise} \end{cases} \)

- \( F \equiv \text{if } \varphi_1 \text{ then } H \parallel \ldots \parallel \text{if } \varphi_n \text{ then } H \), where \( \{\varphi_1, \ldots, \varphi_n\} \) is the set of ordered events of the form \( \psi_1, \ldots, \psi_n \) such that \( \psi_{ij} \in \{\vartheta_i \} \cup \vartheta_{ij} \quad \text{if } G_j \notin \{\text{pause, exit } T, \text{ pause}; E_{f(i)}\} \) otherwise.
4.1. THE AXIOMATIZATION

Since $E$ is a normal form, if $F_i \equiv \textbf{if } \vartheta_i \textbf{ then pause } E_{f(i)}$, then we are sure that if the environment prompts every signal $s$ such that $s^+ \in |\vartheta_i|$ and does not prompt any signal $s$ such that $s^- \in |\vartheta_i|$, then $E; H$ pauses and will behave as $E_{f(i)}; H$ at the next execution cycle. In this case, the occurrence of $H$ in the body of if $\varphi_j$ then $H$ cannot start at the current cycle, since $|\vartheta_i| \uparrow |\varphi_j|$, for any $1 \leq j \leq h$ (in fact, $\varphi_j$ contains a string in $\vartheta_i$). So, $E^H$ behaves as $E; H$ at the current cycle, and will behave as $E_{f(i)}; H$ at the next one.

If $E$ starts and terminates at the current cycle, so that the occurrence of $H$ in the body of $E; H$ starts, then there exists some $\varphi_j$ such that the environment prompts signals as assumed by $\varphi_j$. Therefore, at least one occurrence of $H$ in the body of $E^H$ starts.

**Example 4.1.3** As in Examples 3.2.9 and 3.2.10, let us assume $E \equiv E_1 \parallel E_2$, where:

$E_1 \equiv \textbf{present } s_1 \textbf{ then exit } T \textbf{ else nothing end,}$
$E_2 \equiv \textbf{present } s_2 \textbf{ then emit } s_3 \textbf{ else nothing end.}$

By axiom seq2, we have $E; \text{emit } s = E \parallel \text{if } s_1^- s_2^- \text{ then emit } s \parallel$
if $s_1^+ s_2^+$ then emit $s \parallel$ if $s_1^- s_1^- \text{ then emit } s \parallel$ if $s_1^+ s_1^+$ then emit $s$.

Let us prove formally the soundness of axiom seq2. First of all, we note that

$\{\varphi_1, \ldots, \varphi_h\} = I(E)$.

Let us assume that $E \xrightarrow{l} \text{nothing } \parallel \ldots \parallel \text{nothing}$ with $T_i = 0$ and $H \xrightarrow{\ell'} H'$. We have $E; H \xrightarrow{\ell} H'$, where

$l = l_i \times l' = l_i \otimes |S_T| \cup \bigcup_{\varphi \in I(T_i)} \mathcal{E}_T^\varphi, \bigcup_{\varphi \in I(F) \setminus I(T_i)} \mathcal{E}_F^\varphi \cup \bigcup_{\varphi \in I(E) \setminus I(T_i)} \mathcal{N}_E^\varphi, T_F).$

Moreover, we have $E^H \xrightarrow{\ell'} \text{nothing } \parallel \ldots \parallel \text{nothing } \parallel H' \parallel \ldots \parallel H' \parallel \text{nothing } \parallel \ldots \parallel \text{nothing}$, where:

$\ell'' = l_i \otimes \bigotimes_{\varphi \in I(T_i)} (|S_T| \cup S_T, \mathcal{E}_T^\varphi, \mathcal{N}_E^\varphi, T_F) \otimes \bigotimes_{\varphi \in I(F) \setminus I(T_i)} (S_F, \emptyset, \mathcal{E}_F^\varphi \cup \mathcal{N}_F^\varphi, 0)$.

The two derivatives are bisimilar by axioms in Table 4.1, and $l = l''$ follows by the fact that, for each $\varphi \in I(E) \setminus I(T_i)$, we have that $S_\varphi$ is an event such that $S_\varphi \subseteq S_{l_i}$, and, for each $\varphi \in I(T_i)$, we have that $|\varphi| = S_{l_i} \subseteq S_T$.

Analogously, one can prove that $E; H \xrightarrow{\ell} (\text{nothing } \parallel \ldots \parallel \text{nothing } ; E_{f(i)} \parallel \ldots \parallel \text{nothing}); H$ if $E \xrightarrow{l} \text{nothing } \parallel \ldots \parallel \text{nothing } ; (E_{f(i)} ; H) \parallel \ldots \parallel \text{nothing}$, where these two derivatives are bisimilar by the fact that nothing; $E \approx E$ for every statement $E$ and by axiom $\parallel 3$.

Finally, one can prove that $E; H \xrightarrow{\ell} \text{nothing}$ with $T_i \subseteq T$ if $E \xrightarrow{\ell} \text{nothing}$.

It follows that $E; H \approx E^H$.

Axion seq3 is immediate.

Let us consider axiom susp in Table 4.6. Given a signal $s$ and a normal form $E \equiv F_1 \parallel \ldots \parallel F_n$, we denote with $E^s$ the statement $E^s \equiv F_1^s \parallel \ldots \parallel F_n^s$, where:
\[ E \text{ normal form and } \text{Free}(E) = \emptyset \Rightarrow \text{suspend } E \text{ when } s = E^s \quad (\text{susp}) \]
\[ E \text{ normal form and } \text{Free}(E) = \emptyset \Rightarrow \text{trap } T \text{ in } E \text{ end } = E^T \quad (\text{trap}) \]

Table 4.6: Axioms for suspend \_ when \_ and trap \_ in \_ end.

\[
F_i^s \equiv \begin{cases} 
F_i[\text{suspend imm } E_f(i) \text{ when } s/E_f(i)] & \text{if } F_i \equiv \text{ if } \bar{\vartheta}_i \text{ then pause } ; E_f(i) \\
F_i & \text{otherwise.}
\end{cases}
\]

The soundness of axiom \text{susp} follows by the following facts:

- suspend \( E \) when \( s \xrightarrow{t} \text{nothing} \) with \( \mathcal{T}_i = 0 \) iff \( E^s \xrightarrow{t} \text{nothing} \parallel \ldots \parallel \text{nothing} \), where \( \text{nothing} \approx \text{nothing} \parallel \ldots \parallel \text{nothing} \) by axiom \( \parallel_3 \);
- suspend \( E \) when \( s \xrightarrow{t} \text{nothing} \) with \( \mathcal{T}_i \subseteq \mathcal{T} \) iff \( E^s \xrightarrow{t} \text{nothing} \);
- suspend \( E \) when \( s \xrightarrow{t} \text{suspend imm (nothing} \parallel \ldots \parallel \text{nothing} ; E_f(i) \parallel \ldots \parallel \text{nothing} \) when \( s \) and only if \( E^s \xrightarrow{t} \text{nothing} \parallel \ldots \parallel \text{nothing} ; \text{suspend imm } E_f(i) \text{ when } s \parallel \ldots \parallel \text{nothing} \), where these two derivatives are bisimilar by axiom \( \parallel_3 \) and by the fact that \( \text{nothing} ; E \approx E \) for any statement \( E \).

Let us consider axiom \text{trap} in Table 4.6, a trap name \( T \) and a normal form \( E \equiv F_1 \parallel \ldots \parallel F_n \) such that \( F_i \equiv \text{ if } \bar{\vartheta}_i \text{ then exit } T, 1 \leq i \leq m \), and \( F_i \equiv \text{ if } \vartheta_i \text{ then } G_i, \ G_i \neq \text{exit } T, m + 1 \leq i \leq n \). Let us assume that \( \bar{\vartheta}_i = \gamma_i, \ldots, \gamma_{i,n} \). We denote with \( T(E) \) the set of all ordered events \( \gamma_{i_1,1} \ldots \gamma_{i_1,j_1} \ldots \gamma_{i_m,1} \ldots \gamma_{i_m,j_m} \), and we assume that \( T(E) = \{ \vartheta_1, \ldots, \vartheta_k \} \). Note that if signals have the status as given by some \( \vartheta_i, 1 \leq i \leq k \), then no \text{exit } T is executed. We denote with \( E^T \) the statement \( E^T \equiv F_1^T \parallel \ldots \parallel F_n^T \), where:

\[
F_i^T \equiv \begin{cases} 
F_i[\text{trap } T \text{ in } E_f(i) \text{ end } E_f(i)] & \text{if } F_i \equiv \text{ if } \vartheta_i \text{ then pause } ; E_f(i) \\
F_i[\text{nothing/exit } T] & \text{if } F_i \equiv \text{ if } \vartheta_i \text{ then exit } T \\
\text{if } \varphi_i \vartheta_i \text{ then pause } \parallel \ldots \parallel & \text{if } F_i \equiv \text{ if } \vartheta_i \text{ then pause,} \\
\text{if } \varphi_k \vartheta_i \text{ then pause} & |\vartheta_i| \uparrow |\vartheta_j| \text{ for some } 1 \leq j \leq m \\
F_i & \text{otherwise.}
\end{cases}
\]

If \( F_i \equiv \text{ if } \vartheta_i \text{ then exit } T \), then, since \( E \) is a normal form, there exists no \( j \) such that \( |\vartheta_i| \cup |\vartheta_j| \) and \( F_j \equiv \text{ if } \vartheta_j \text{ then pause } ; E_f(j) \). Namely, there exists no \( j \) such that \( |\vartheta_i| \uparrow |\vartheta_j| \) and \( F_j \equiv \text{ if } \vartheta_j \text{ then pause } ; E_f(j) \). Moreover, if \( F_j \equiv \text{ if } \vartheta_j \text{ then pause } ; |\vartheta_i| \uparrow |\vartheta_h| \text{ for some } 1 \leq h \leq m \), then we replace \( F_j \) by \( \varphi_i \vartheta_j \text{ then pause } ; \varphi_k \vartheta_j \text{ then pause} \). Therefore, the occurrences of \text{exit } T that are replaced by \text{nothing} do not preempt any pausing. This justifies axiom \text{trap}. 

84

CHAPTER 4. AN AXIOMATIZATION OF ESTEREL
Example 4.1.4 Let us assume that $S = \{s_1, s_2\}$ and let us consider statements $E$ and $E'$ such that
\[ E \equiv \text{if } s_1^+ s_2^- \text{ then pause; } F \parallel \text{if } s_1^+ \text{ then pause } \parallel \text{if } s_2^+ \text{ then exit } T, \]
\[ E' \equiv \text{if } s_1^+ s_2^- \text{ then pause; } \text{trap } T \text{ in } F \text{ end } \parallel \text{if } s_2^- s_1^+ \text{ then pause } \parallel \text{if } s_2^+ \text{ then nothing.} \]
By axiom trap we have $\text{trap } T \text{ in } E \text{ end } = E'$.

Formally, we have $\text{trap } T \text{ in } E \text{ end } \xrightarrow{l} \text{nothing}$ with $T_l = 0$ if and only if either $E \xrightarrow{l} \text{nothing} \parallel \ldots \parallel \text{nothing with } T_l = 0$, or $E \xrightarrow{l} \text{nothing}$ with $T_l = \{T\}$. In both cases, $l = tr(T, l')$.

Now, $E \xrightarrow{l'} \text{nothing} \parallel \ldots \parallel \text{nothing}$ if and only if $E^T \xrightarrow{l''} \text{nothing} \parallel \ldots \parallel \text{nothing}$, and $E \xrightarrow{l} \text{nothing}$ with $T_l = \{T\}$ if and only if $E^T \xrightarrow{l'} \text{nothing} \parallel \ldots \parallel \text{nothing}$.

In both cases, we have $l'' = l$ by the definition of $tr(T, l')$, and $\text{nothing} \approx \text{nothing} \parallel \ldots \parallel \text{nothing}$ by axiom $\parallel_3$.

We have
\[ \text{trap } T \text{ in } E \text{ end } \xrightarrow{l} \text{trap } T \text{ in nothing} \parallel \ldots \parallel \text{nothing; } E_f(i) \parallel \ldots \parallel \text{nothing end} \]
if and only if
\[ E \xrightarrow{l} \text{nothing} \parallel \ldots \parallel \text{nothing; } E_f(i) \parallel \ldots \parallel \text{nothing} \]
if and only if
\[ E^T \xrightarrow{l'} \text{nothing} \parallel \ldots \parallel \text{nothing; } \text{trap } T \text{ in } E_f(i) \text{ end} \parallel \ldots \parallel \text{nothing}, \]
where $l'' = l$ by the definition of $tr(T, l')$.

and $\text{trap } T \text{ in nothing} \parallel \ldots \parallel \text{nothing; } E_f(i) \parallel \ldots \parallel \text{nothing end} \approx \text{nothing} \parallel \ldots \parallel \text{nothing; } \text{trap } T \text{ in } E_f(i) \text{ end} \parallel \ldots \parallel \text{nothing}$ by axiom $\parallel_3$ and by the fact that $\text{nothing; } E \approx E$ for any statement $E$.

Finally, we have $\text{trap } T \text{ in } E \xrightarrow{l} \text{nothing}$ with $T_l \subseteq T, T_l \neq \{T\}$, if and only if $E \xrightarrow{l} \text{nothing}$ with $T_l \subseteq T, T_l \neq \{T\}$, if and only if $E^T \xrightarrow{l'} \text{nothing}$ with $T_{l'} \subseteq T, T_{l'} \neq \{T\}$, and $l'' = l$ by the definition of $tr(T, l')$.

\[ \text{signal } s \text{ in (signal } s \text{ in } E \text{ end) end } = \text{signal } s \text{ in } E \text{ end} \quad (s_1) \]
\[ E \text{ normal form and } Free(E) = \emptyset \Rightarrow \]
\[ \text{signal } s \text{ in } E \text{ end } = \text{signal } s \text{ in } E_s \text{ end} \quad (s_2) \]
\[ E \text{ normal form, } E \text{ constructive and } Free(E) = \emptyset \Rightarrow \]
\[ \text{signal } s \text{ in } E_s \text{ end } = E \setminus \{s\} \quad (s_3) \]

Table 4.7: Axioms for signal _ in _ end.
Let us consider axiom \( s_1 \) in Table 4.7. This is justified by the fact that signal \( s \) is not in the input-output interface of statement \( \text{signal } s \text{ in } E \text{ end} \). Formally, the soundness of axiom \( s_1 \) follows by the fact that \( E \xrightarrow{l} F \) and \( \text{loc}(s, l) \) is defined if and only if \( \text{signal } s \text{ in } E \text{ end} \xrightarrow{\text{loc}(s, l)} \text{ signal } s \text{ in } F \text{ end} \) if and only if \( \text{signal } s \text{ in } (\text{signal } s \text{ in } E \text{ end}) \xrightarrow{\text{loc}(s, l)} \text{ signal } s \text{ in } (\text{signal } s \text{ in } F \text{ end}) \) end, and function \( \text{loc} \) is such that \( \text{loc}(s, l) = \text{loc}(s, \text{loc}(s, l)) \).

Let us consider axiom \( s_2 \). Given a normal form \( E \equiv F_1 \parallel \ldots \parallel F_n \) and a signal \( s \), we denote with \( E_s \) the statement \( E_s \equiv F_{s,1} \parallel \ldots \parallel F_{s,n} \), where:

\[
F_{s,i} \equiv \begin{cases} 
F_i | \text{signal } s \text{ in } E_i \text{ end} / E_i \text{end} \parallel E_{f(i)} \parallel \ldots \parallel E_{f(i)} \parallel \ldots 
\text{ if } F_i \equiv \text{if } \vartheta_i \text{ then } \text{pause } ; E_{f(i)} \\
F_i \text{ otherwise.}
\end{cases}
\]

Axiom \( s_2 \) is justified by two facts. The first is that the status of signals does not propagate between two execution cycles. The second is that, since \( E \) is a normal form, if it pauses then it will behave as \( \text{nothing} \parallel \ldots \parallel \text{nothing} ; E_{f(i)} \parallel \ldots \parallel \text{nothing} \) at the next execution cycle, for some \( 1 \leq i \leq n \). So, it does not matter whether \( s \) is local to \( E_{f(i)} \) or \( s \) is local to \( \text{nothing} \parallel \ldots \parallel \text{nothing} ; E_{f(i)} \parallel \ldots \parallel \text{nothing} \).

Note that it is essential that \( E \) is a normal form. In fact, as an example, we do not have, in general, \( H \approx K \), for \( H \) and \( K \) statements such that:

\[H \equiv \text{signal } s \text{ in } (\text{pause } ; E \parallel \text{pause } ; F) \text{ end},\]

\[K \equiv \text{signal } s \text{ in } (\text{pause } ; \text{signal } s \text{ in } E \text{ end} \parallel \text{pause } ; \text{signal } s \text{ in } F \text{ end}) \text{ end}.
\]

In fact, at the second execution cycle, statements \( E \) and \( F \) in the body of \( H \) view the same signal \( s \), while this cannot happen for \( E \) and \( F \) in the body of \( K \).

Formally, the soundness of axiom \( s_2 \) follows by the following facts:

- \( \text{signal } s \text{ in } E \text{ end} \xrightarrow{l} \text{ signal } s \text{ in } (\text{nothing} \parallel \ldots \parallel \text{nothing} ; E_{f(i)} \parallel \ldots \parallel \text{nothing}) \text{ end} \) with \( \mathcal{T}_l = 1 \)
  if and only if
  \( \text{signal } s \text{ in } E_s \text{ end} \xrightarrow{l} \text{ signal } s \text{ in } (\text{nothing} \parallel \ldots \parallel \text{nothing} ; \text{signal } s \text{ in } E_{f(i)} \text{ end}) \parallel \ldots \parallel \text{nothing} \) end,
  and the two derivatives are bisimilar by axioms \( s_1 \) and \( ||_3 \).

- \( \text{signal } s \text{ in } E \text{ end} \xrightarrow{l} \text{ signal } s \text{ in } (\text{nothing} \parallel \ldots \parallel \text{nothing}) \text{ end} \) with \( \mathcal{T}_l = 0 \) if \( \text{signal } s \text{ in } E_s \text{ end} \xrightarrow{l} \text{ signal } s \text{ in } (\text{nothing} \parallel \ldots \parallel \text{nothing}) \text{ end} \).

- \( \text{signal } s \text{ in } E \text{ end} \xrightarrow{l} \text{ signal } s \text{ in } \text{nothing} \text{ end} \) with \( \mathcal{T}_l \subseteq \mathcal{T} \) iff
  \( \text{signal } s \text{ in } E_s \text{ end} \xrightarrow{l} \text{ signal } s \text{ in } \text{nothing} \) end.

Let us consider axiom \( s_3 \), a constructive normal form \( E \equiv F_1 \parallel \ldots \parallel F_n \), where \( F_i \equiv \text{if } \vartheta_i \text{ then } G_i \), and statement \( E_s \equiv F_{s,1} \parallel \ldots \parallel F_{s,n} \). Let \( G_{s,i} \) be the statement such that \( F_{s,i} \equiv \text{if } \vartheta_i \text{ then } G_{s,i} \).
Let us denote with $\Theta_{s^+}$ and $\Theta_{s^-}$ the following sets of ordered events:

$\Theta_{s^+} = \{ \vartheta_i | 1 \leq i \leq n, s^-, s^+ \not\in [\vartheta_i], G_i \equiv \text{emit } s \};$

$\Theta_{s^-} = \{ \phi_i, \ldots, \phi_m | \text{ for every } 1 \leq j \leq m \text{ we have } G_{ij} \equiv \text{emit } s, \phi_{ij} \in \overline{\vartheta_i}, \phi_{ij} = \phi'_{ij} \gamma_{ij} \text{ and } \gamma_{ij} \not\in \{s^+, s^-\} \}.$

Note that if the environment prompts every signal $s$ such that $s^+ \in [\vartheta]$ and does not prompt any signal $s$ such that $s^- \in [\vartheta], \vartheta \in \Theta_{s^+},$ then $E$ produces $s.$ On the contrary, if the environment prompts every signal $s$ such that $s^+ \in [\vartheta]$ and does not prompt any signal $s$ such that $s^- \in [\vartheta], \vartheta \in \Theta_{s^-},$ then $E$ does not produce $s.$

Let us denote with $G_i \setminus \{s\}$ the statement

$G_i \setminus \{s\} \equiv \begin{cases} \text{nothing} & \text{if } G_{s,i} \equiv \text{emit } s \\ G_{s,i} & \text{otherwise.} \end{cases}$

Let us denote with $F_i \setminus \{s\}$ the following statement:

$F_i \setminus \{s\} \equiv \forall (\vartheta \in \Theta_{s^+}, \phi \in \Theta_{s^-}) \text{ if } \vartheta_i [\vartheta/s^+] [\phi/s^-] \text{ then } G_i \setminus \{s\}.$

Finally, let us denote with $E \setminus \{s\}$ the statement $E \equiv F_1 \setminus \{s\} \ldots \parallel F_n \setminus \{s\}.$

Intuitively, axiom $s_3$ replaces occurrences of emit $s$ by nothing, replaces every test for the presence of $s$ by a set of tests, each for the presence of an event causing $s,$ and replaces every test for the absence of $s$ by a set of tests, each for the absence of all events causing $s.$

**Example 4.1.5** Let $a, b, c, d$ and $s$ signals in $S.$ Let $E$ and $E'$ be the statements

$E \equiv \text{if } a^+ \text{ then } \text{emit } s \parallel \text{if } b^+ \text{ then } \text{emit } s \parallel \text{if } s^+ \text{ then } \text{emit } c \parallel \text{if } s^- \text{ then } \text{emit } d.$

$E' \equiv \text{if } a^+ \text{ then } \text{nothing} \parallel \text{if } b^+ \text{ then } \text{nothing} \parallel \text{if } a^+ \text{ then } \text{emit } c \parallel \text{if } b^- \text{ then } \text{emit } c \parallel \text{if } a^- b^- \text{ then } \text{emit } d \parallel \text{if } b^- a^- \text{ then } \text{emit } d.$

By axioms $s_3$ we have signal $s$ in $E \text{ end } = E'.$

To prove formally the soundness of axiom $s_3,$ we must prove that

signal $s$ in $E_s \text{ end } \overset{\text{l}}{\rightarrow} H$ iff $E \setminus \{s\} \overset{\text{l}'}{\rightarrow} H', \text{ where } l = l'$ and $H \approx H'.$

Now, signal $s$ in $E_s \text{ end } \overset{\text{l}}{\rightarrow} H$ iff $F_{s,i} \overset{\text{l}_i}{\rightarrow} H_i, l = \text{loc} (s, l_1 \otimes \ldots \otimes l_n),$ and $H \equiv \text{signal } s \text{ in } H_1 \parallel \ldots \parallel H_n \text{ end.}$

Analogously, $E \setminus \{s\} \overset{\text{l}'}{\rightarrow} H'$ iff $F_i \setminus \{s\} \overset{\text{l}'}{\rightarrow} H'_i, l' = l_1' \otimes \ldots \otimes l_n', \text{ and } H' \equiv H'_1 \parallel \ldots \parallel H'_n.$

Axioms $\parallel_1$ and $\parallel_2$ imply that we can assume $G_1, \ldots, G_v \equiv \text{emit } s$ and $G_{v+1}, \ldots, G_n \not\equiv \text{emit } s,$ for some $0 \leq v \leq n.$

We distinguish between the case with $s \in \text{Em} (l_1 \otimes \ldots \otimes l_n)$ and the case with $s \not\in \text{Em} (l_1 \otimes \ldots \otimes l_n).$

- $s \in \text{Em} (l_1 \otimes \ldots \otimes l_n).$

  For every $1 \leq i \leq v,$ we have $F_{s,i} \overset{\text{l}_i}{\rightarrow} \text{nothing}$ with either $E_i = \{\vartheta_i s\}$ or $N_i = \{\vartheta_i s\}.$ By axioms $\parallel_1$ and $\parallel_2$ we can assume that $\vartheta_i s \in E_i$ for $1 \leq i \leq u,$ and $\vartheta_i s \in N_i$ for $u + 1 \leq i \leq v.$

  We deduce $l = l'$ and $H \approx H'$ by the following facts:
– For an arbitrary \( 1 \leq j \leq n \), \( F_{s,j} \xrightarrow{\ell_j} H_j \) with \( \mathcal{E}_{l_j} = \{ \partial s^+ \psi \mu \} \) iff
\[
F_j \setminus \{ s \} \xrightarrow{\ell_j} H_j \parallel \ldots \parallel H_j \parallel \text{nothing} \parallel \ldots \parallel \text{nothing}, \quad \text{where} \quad \mathcal{E}_{l_j} = \{ \partial \psi \mu[n/s] \mid \psi \in \Theta_{s+}, 1 \leq i \leq u \}, \quad \mathcal{N}_{l_j} = \{ \partial \psi \mu[n/s] \mid \psi \in \Theta_{s+}, u+1 \leq i \leq v \}.
\]
So, \( \partial \psi \mu[n/s] \in \mathcal{E}_{l_i} \) iff \( \partial \psi \mu[n/s] \in \mathcal{E}_{l_j} \), \( 1 \leq i \leq u \), \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_i} \) iff \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_j} \), \( u+1 \leq i \leq v \).

– For an arbitrary \( 1 \leq j \leq n \), \( F_{s,j} \xrightarrow{\ell_j} \text{nothing} \) with \( \mathcal{N}_{l_j} = \{ \partial s^+ \psi \mu \} \) iff
\[
F_j \setminus \{ s \} \xrightarrow{\ell_j} \text{nothing} \parallel \ldots \parallel \text{nothing}, \quad \mathcal{N}_{l_j} = \{ \partial \psi \mu[n/s] \mid \psi \in \Theta_{s+} \}.
\]
So, \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_i} \) iff \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_j} \), \( 1 \leq i \leq v \).

– For an arbitrary \( 1 \leq j \leq n \), \( F_{s,j} \xrightarrow{\ell_j} \text{nothing} \) with \( \mathcal{N}_{l_j} = \{ \partial s^- \psi \mu \} \) iff
\[
F_j \setminus \{ s \} \xrightarrow{\ell_j} \text{nothing} \parallel \ldots \parallel \text{nothing}, \quad \mathcal{N}_{l_j} = \{ \partial \psi \mu[n/s] \mid \psi \in \Theta_{s-} \}.
\]
So, \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_i} \) iff \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_j} \), \( \phi \in \Theta_{s-} \).

– For an arbitrary \( 1 \leq j \leq n \), \( F_{s,j} \xrightarrow{\ell_j} H_j \), \( \partial \mu \in \mathcal{E}_{l_j} \) (resp. \( \partial \mu \in \mathcal{N}_{l_j} \)), \( \{ s^-, s^+ \} \cap \{ \partial \} = \emptyset \), \( \mu \neq s \), iff \( F_j \setminus \{ s \} \xrightarrow{\ell_j} H_j \), \( \partial \mu \in \mathcal{E}_{l_j} \) (resp. \( \partial \mu \in \mathcal{N}_{l_j} \)).

**s \notin \text{Em}(l_1 \otimes \ldots \otimes l_n).**

For every \( 1 \leq i \leq v \) we have \( F_{s,i} \xrightarrow{\ell_i} \text{nothing} \) with \( \mathcal{N}_{l_i} = \{ \partial_i s \} \).

We deduce \( l = l' \) and \( H \approx H' \) by the following facts:

– For an arbitrary \( 1 \leq j \leq n \), \( F_{s,j} \xrightarrow{\ell_j} H_j \) with \( \mathcal{E}_{l_j} = \{ \partial s^- \psi \mu \} \) iff \( F_j \setminus \{ s \} \xrightarrow{\ell_j} H_j \parallel \ldots \parallel H_j \parallel \text{nothing} \parallel \ldots \parallel \text{nothing}, \quad \mathcal{E}_{l_j} = \{ \partial \psi \mu[n/s] \mid \psi \in \Theta_{s-} \}, \quad \mathcal{N}_{l_j} = \{ \partial \psi \mu[n/s] \mid \psi \in \Theta_{s-} \} \}
\]
So, we have that \( \partial \psi \mu[n/s] \in \mathcal{E}_{l_i} \) iff \( \partial \psi \mu[n/s] \in \mathcal{E}_{l_j} \), and \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_i} \) iff \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_j} \), \( \phi \in \Theta_{s-} \).

– For an arbitrary \( 1 \leq j \leq n \), \( F_{s,j} \xrightarrow{\ell_j} \text{nothing} \) with \( \mathcal{N}_{l_j} = \{ \partial s^- \psi \mu \} \) iff
\[
F_j \setminus \{ s \} \xrightarrow{\ell_j} \text{nothing} \parallel \ldots \parallel \text{nothing}, \quad \mathcal{N}_{l_j} = \{ \partial \psi \mu[n/s] \mid \psi \in \Theta_{s-} \}.
\]
So, we have that \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_i} \) and only if \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_j} \), \( \phi \in \Theta_{s-} \).

– For an arbitrary \( 1 \leq j \leq n \), \( F_{s,j} \xrightarrow{\ell_j} \text{nothing} \) with \( \mathcal{N}_{l_j} = \{ \partial s^+ \psi \mu \} \) iff
\[
F_j \setminus \{ s \} \xrightarrow{\ell_j} \text{nothing} \parallel \ldots \parallel \text{nothing}, \quad \mathcal{N}_{l_j} = \{ \partial \psi \mu[n/s] \mid \psi \in \Theta_{s+} \}.
\]
So, \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_i} \) iff \( \partial \psi \mu[n/s] \in \mathcal{N}_{l_j} \), \( 1 \leq i \leq v \).

– For an arbitrary \( 1 \leq j \leq n \), \( F_{s,j} \xrightarrow{\ell_j} H_j \), \( \partial \mu \in \mathcal{E}_{l_j} \) (resp. \( \partial \mu \in \mathcal{N}_{l_j} \)), \( \{ s^-, s^+ \} \cap \{ \partial \} = \emptyset \), \( \mu \neq s \), iff \( F_j \setminus \{ s \} \xrightarrow{\ell_j} H_j \), \( \partial \mu \in \mathcal{E}_{l_j} \) (resp. \( \partial \mu \in \mathcal{N}_{l_j} \)).
Up to now we have proved the soundness modulo bisimulation of axioms in Tables 4.1-4.7. So, we can give the following result:

**Lemma 4.1.6** Given Esterel$^+$ statements $E$ and $E'$, $E = E'$ implies $E \approx E'$.

### 4.2 The proof of completeness

In this section we prove that axioms in Table 4.1-4.7 give an axiomatization complete modulo bisimulation on constructive Esterel statements.

First of all we will prove that, given an arbitrary constructive Esterel statement $E$, there exists a normal form $E'$ such that $E = E'$ and all derivatives of $E'$ are normal forms. Then, we will introduce the notion of guarded recursive specification and we prove that every guarded recursive specification has a solution which is unique modulo $\approx$. Finally, we will exploit these two results to prove that two arbitrary constructive Esterel statements $E$ and $E'$ such that $E \approx E'$ are equated by axioms in Tables 4.1-4.7. In fact, we prove that $E = F$ and $E' = F'$, where $F$ and $F'$ are normal forms and solutions of the same guarded recursive specification.

We begin with introducing some notions.

We say that a variable $P$ is *strongly guarded* in a statement $E$ if and only if $P$ is guarded in $E$, and no free occurrence of $P$ appears in the left side of a ";" or in the body of statements `suspend`, `signal`, `trap` and `loop`.

An Esterel$^+$ statement $E$ is *well formed* if and only if:

- for each statement `rec P.F` in the body of $E$, the variable $P$ is strongly guarded in $F$;
- every variable $P \in \text{Free}(E)$ is strongly guarded in $E$.

Note that axioms transform well formed statements in well formed statements. Note also that every Esterel statement $E$ is well formed, since no variable appears in $E$.

The following Lemma states that an arbitrary constructive Esterel statement $E$ can be transformed, by applying axioms, in a normal form that has only normal forms as derivatives.

**Lemma 4.2.1** Given a well-formed constructive Esterel$^+$ statement $E$, there exist normal forms $E_1, \ldots, E_m$ such that:

- $E_i = F_{i_1} \| \ldots \| F_{i_m}$, $F_{i,j} \equiv \text{if } \vartheta_{i,j} \text{ then } G_{i,j}$, and, if $G_{i,j} \equiv \text{pause}; E_{f(i,j)}$, then $f(i,j) \in \{1, \ldots, m\}$;
- $E = E_1$. 
Chapter 4. An Axiomatization of Estérel

Proof. By structural induction on $E$.

Base case.

If $E$ is a basic statement, $E \neq \text{pause}$, then the thesis is immediate, since $E$ is a normal form.

If $E \equiv \text{pause}$, then, by axiom $\text{seq}_3$ we infer $E = \text{pause; nothing}$, and, by axioms $||_5$ and $||_6$, we infer $\text{pause; nothing = pause $\|$ if } s^+ \text{ then pause; nothing $\|$ ... $\|$ if } s^- \text{ then pause; nothing}$, which is a normal form.

Induction step.

As inductive hypothesis, let us assume that, given statements $E'$ and $E''$, there exist normal forms $E_1',\ldots, E_{m'}, E_{1''},\ldots, E_{m''}$, such that $E_{i'} = F_{i',1} \|$ ... $\|$ $F_{i'_{n'}}$, $E_{i''} = F_{i'',1} \|$ ... $\|$ $F_{i''_{n''}}$, $F_{i',j} \equiv \text{if } \partial_{i',j} \text{ then } G_{i',j}$, $F_{i'',j} \equiv \text{if } \partial_{i'',j} \text{ then } G_{i'',j}$, $E' = E_1', E'' = E_{m''}$.

We consider the following cases:

- $E \equiv E' \|$ $E''$.
  
  Since $E' = E_1'$ and $E'' = E_{1''}$, we infer $E = E_{1'} \|$ $E_{1''}$ by the fact that $=$ is a congruence. Let us denote with $\mathcal{T} \subseteq \{1',\ldots, m'\} \times \{1'',\ldots, m''\}$ the least set such that:

  - $(1',1'') \in \mathcal{T}$;
  - if $(i',i'') \in \mathcal{T}$, $F_{i',j'} \equiv \text{if } \partial_{i',j'} \text{ then } G_{i',j'}$, $F_{i'',j''} \equiv \text{if } \partial_{i'',j''} \text{ then } G_{i'',j''}$, $E_{f(i',j')} \equiv \text{if } \partial_{i',j'} \uparrow \partial_{i'',j''} \text{ then } (f(i',j'), f(i'',j'')) \in \mathcal{T}$.

  The thesis follows if we infer $E_{i'} \|$ $E_{i''} = E_{i',j'}$, with $E_{i',j'}$ an arbitrary normal form having only normal forms as derivatives, for every $(i',i'') \in \mathcal{T}$.

  Statement $E_{i'} \|$ $E_{i''}$ satisfies Cond. 1 of Def. 4.1.2, provided that we remove every component $F_{i'',j''}$ such that $F_{i'',j''} \equiv F_{i',j'}$, for some $1 \leq j' \leq n_{i'}$, by means of axioms $||_1$, $||_2$ and $||_4$.

  By applying axiom $||_7$, we infer $E_{i'} \|$ $E_{i''} = E_{i',j'}$, for a statement $F_{i',j'}$ satisfying Cond. 1-2 of Def. 4.1.2.

  By applying axiom $||_8$, we infer $F_{i',j'} = G_{i',j'}$, for a statement $G_{i',j'}$ satisfying Cond. 1-3 of Def. 4.1.2.

  By applying axioms $||_6$ and $||_9$, we infer $G_{i',j'} = H_{i',j'}$, for a statement $H_{i',j'}$ satisfying Cond. 1-4 of Def. 4.1.2.

  By applying axioms $?_2$, $?_3$ and $||_3$, we infer $H_{i',j'} = K_{i',j'}$, for a statement $K_{i',j'}$ satisfying Cond. 1-5 of Def. 4.1.2. In fact, let us assume that $H_{i',j'}$ does not satisfy Cond. 5 of Def. 4.1.2, namely $H_{i',j'} \equiv H_1 \|$ ... $\|$ $H_n$, where there exist $1 \leq h, k \leq n$ such that $H_h \equiv \text{if } \partial_h \text{ then nothing}$, $H_k \equiv \text{if } \partial_k \text{ then nothing}$, $\partial_h = \phi_h \gamma$, and either $\partial_k = \phi_h \gamma \psi$, or $\partial_k = \phi_h \overline{\psi}$, for some $\psi \in (S^+)^*$. Since $\text{if } \phi_h \gamma \text{ then nothing}$ and $\text{if } \phi_h \overline{\psi} \text{ then nothing}$ denote the same statement, we can assume the first case. By axioms $||_1$ and $||_2$ we can assume $h = 1$ and $k = 2$. Now, we infer:
4.2. THE PROOF OF COMPLETENESS

if $\vartheta_1$ then nothing $\parallel$ if $\vartheta_1$ then (if $\psi$ then $G_2$) =
if $\vartheta_1$ then (nothing $\parallel$ if $\psi$ then $G_2$) =
if $\vartheta_1$ then (if $\psi$ then $G_2$) $\equiv H_2$
where the equalities are inferred by means of axioms $\tau_2$ and $\tau_3$ and by means of axiom $\parallel_3$, respectively.

Finally, by axioms $\text{seq}_1$, $\text{seq}_3$, $\tau_2$, $\tau_3$ and $\parallel_3$ we infer $K_{\vartheta,\vartheta'} = I_{\vartheta,\vartheta'}$ for a statement $I_{\vartheta,\vartheta'}$ satisfying Cond. 1-6 of Def. 4.1.2. In fact, let us assume that $K_{\vartheta,\vartheta'}$
does not satisfy Cond. 6 of Def. 4.1.2, namely $K_{\vartheta,\vartheta'} \equiv K_1 \parallel \ldots \parallel K_n$, where
$K_i \equiv \text{if } \vartheta \text{ then } \text{pause}; K_j \equiv \text{if } \vartheta \text{ then } \text{pause}; \tilde{K}$. By axioms $\parallel_1$ and $\parallel_2$ we
can assume $i = 1$ and $j = 2$. Now, we have:

if $\vartheta$ then pause; $K =$
if $\vartheta$ then pause; ($K \parallel$ nothing) =
if $\vartheta$ then ((pause; $K$) $\parallel$ (pause; nothing)) =
if $\vartheta$ then (pause; $K$) $\parallel$ if $\vartheta$ then (pause; nothing) =
if $\vartheta$ then (pause; $K$) $\parallel$ if $\vartheta$ then pause,
where the equalities are obtained by means of axiom $\parallel_3$, axiom $\text{seq}_1$, axioms
$\tau_2$ and $\tau_3$, and axiom $\text{seq}_3$, respectively.

Note that $I_{\vartheta,\vartheta'}$ satisfies also Cond. 7 of Def. 4.1.2, since $E_{\vartheta}$ and $E_{\vartheta'}$ do.
So, we can take $E_{\vartheta,\vartheta'} \equiv I_{\vartheta,\vartheta'}$.

$E \equiv \text{present } s \text{ then } E'$ else $E''$ end.

Since $E' = E_{\vartheta'}$ and $E'' = E_{\vartheta''}$, we infer $E = \text{present } s \text{ then } E_{\vartheta'}$ else $E_{\vartheta''}$
end by the fact that $=$ is a congruence.

By axiom $\tau_1$ we infer

present $s$ then $E_{\vartheta'}$ else $E_{\vartheta''}$ end $= \text{if } s^+ \text{ then } E_{\vartheta'} \parallel$ if $s^-$ then $E_{\vartheta''}$.

By axioms $\tau_2$ and $\tau_3$ we infer
if $s^+$ then $E_{\vartheta'}$ $\parallel$ if $s^-$ then $E_{\vartheta''}$ =
if $s^+$ then $F_{\vartheta',1}$ $\parallel$ \ldots $\parallel$ if $s^+$ then $F_{\vartheta',n_{\vartheta'}}$ $\parallel$ if $s^-$ then $F_{\vartheta',1}$ $\parallel$ \ldots $\parallel$
if $s^-$ then $F_{\vartheta',n_{\vartheta'}}$,
which satisfies all conditions of Def. 4.1.2.

$E = E' ; E''$.

Since $E' = E_{\vartheta'}$ and $E'' = E_{\vartheta''}$, we infer $E = E_{\vartheta'} ; E_{\vartheta''}$ by the fact that $=$ is a congruence. Then we infer $E_{\vartheta'} ; E_{\vartheta''} = E_{\vartheta';\vartheta''}$, and the thesis follows as in the case for $\parallel$.

$E \equiv \text{rec } P.E'$.

Let us consider statements $H_{\vartheta_1}, \ldots, H_{\vartheta_m}$ such that $H_{\vartheta_i} \equiv E_{\vartheta}[E/P], 1 \leq i \leq m'$.
We have $H_{\vartheta} \equiv E_{\vartheta}[E/P] \equiv (F_{\vartheta,1}[E/P] \parallel \ldots \parallel F_{\vartheta,n_{\vartheta}}[E/P]) \equiv$
$F_{\vartheta,1}[E/P] \parallel \ldots \parallel F_{\vartheta,n_{\vartheta}}[E/P] \equiv$
$F_{\vartheta,1}[H_{f(\vartheta,1)}, E_{f(\vartheta,1)}]/E_{\vartheta}[E/P] \parallel \ldots \parallel F_{\vartheta,n_{\vartheta}}[H_{f(\vartheta,n_{\vartheta})}, E_{f(\vartheta,n_{\vartheta})}]/E_{\vartheta}[E/P] =$
$F_{\vartheta,1}[H_{f(\vartheta,1)}, E_{f(\vartheta,1)}]/[E'[E/P]/P] \parallel \ldots$
$\ldots \parallel F_{\vartheta,n_{\vartheta}}[H_{f(\vartheta,n_{\vartheta})}, E_{f(\vartheta,n_{\vartheta})}]/[E'[E/P]/P] =$
$F_{\vartheta,1}[H_{f(\vartheta,1)}, E_{f(\vartheta,1)}]/E_{\vartheta}[E'[E/P]/P] \parallel \ldots$
\[ \cdots \parallel F_{n, n'}[H_{f, n, n'} / E_{f, n, n'}][E_Y / E / P] / P \equiv \\
F_{n, 1}[H_{f, n, 1} / E_{f, n, 1}][H_{Y, 1} / P] \parallel \cdots \parallel F_{n, n'}[H_{f, n', n'} / E_{f, n', n'}][H_{Y, n'} / P], \]

where the equalities are inferred by axiom \textit{rec}_1 and by the fact that \( = \) is a congruence. Now, \( H_{n'} \) has been rewritten as a parallel composition of normal forms having only normal forms as derivatives, where no free occurrence of \( P \) appears. Let us call \( H_{n'} \) such statement. Every \( H_{n'} \) can be transformed in a normal form as in the case for “\( \parallel \)”. Finally, the thesis follows since we have 
\[ E = E'[E / P] = E[Y, E / P] \equiv H_{n'}. \]

- \( E \equiv \text{loop} \ E' \text{ end}. \)

By axiom \textit{loop}, we infer \( E = \text{rec} \ P, (E'; P) \), so that the thesis follows as in the case for construct \textit{rec}.

- \( E \equiv \text{signal} \ s \ \text{in} \ E' \text{ end}. \)

Since \( E' = E_{n'} \), we infer \( E = \text{signal} \ s \ \text{in} \ E_{n'} \text{ end} \) by the fact that \( = \) is a congruence. Let us consider an arbitrary \( 1 \leq i' \leq n_{n'} \). By axiom \textit{s}_2 we have: \( \text{signal} \ s \ \text{in} \ E_{n'} \ \text{end} = \text{signal} \ s \ \text{in} \ E_{n'} \text{ end} \), where \( E_{n'} \) is a normal form.

By axiom \textit{s}_3 we have:
\[ \text{signal} \ s \ \text{in} \ E_{n'} \text{ end} = E_{n'} \setminus \{ s \}. \]

Now, \( E_{n'} \setminus \{ s \} \equiv K_{n, 1, 1} \parallel \cdots \parallel K_{n, 1, n'} \parallel \cdots \parallel K_{n', n, n'} \parallel \cdots \parallel K_{n', n, n', n'}, \)

where:
\[- K_{n, j, h} \equiv \text{if } \vartheta_{n, j, h} \text{ then pause}; \text{signal} \ s \ \text{in} \ E_{f, n, i} \ \text{end}, \]
\[- F_{n, j, h} \equiv \text{if } \vartheta_{n, j, h} \text{ then pause}; E_{f, n, i}; \]
\[- K_{n', j, h} \equiv \text{if } \vartheta_{n', j, h} \text{ then nothing} \]
\[- F_{n', j, h} \equiv \text{if } \vartheta_{n', j, h} \text{ then emit} \ s; \]
\[- K_{n', j, h} \equiv \text{if } \vartheta_{n', j, h} \text{ then} G_{n', j}, \text{ otherwise.} \]

Namely:
\[- K_{n', j, h} \equiv \text{if } \vartheta_{n', j, h} \text{ then pause}; E_{f, n, i} \setminus \{ s \}, \]
\[- F_{n', j, h} \equiv \text{if } \vartheta_{n', j, h} \text{ then pause}; E_{f, n, i}; \]
\[- K_{n', j, h} \equiv \text{if } \vartheta_{n', j, h} \text{ then nothing} \]
\[- F_{n', j, h} \equiv \text{if } \vartheta_{n', j, h} \text{ then emit} \ s; \]
\[- K_{n', j, h} \equiv \text{if } \vartheta_{n', j, h} \text{ then} G_{n', j}, \text{ otherwise.} \]

So, \( E_{n'} \setminus \{ s \} \) has been rewritten as a parallel composition of normal forms having only normal forms as derivatives. Let us call \( K_{n'} \) such statement. Every \( K_{n'} \) can be transformed in a normal form \( K'_{n'} \) as in the case of “\( \parallel \)”. So, \( K_{n'}, \ldots, K_{n'} \) are the statements we were looking for, and \( E = K_{n'} \).

- \( E \equiv \text{trap} \ T \ \text{in} \ E' \text{ end}. \)

Since \( E' = E_{n'} \), we infer \( E = \text{trap} \ T \ \text{in} \ E_{n'} \ \text{end} \) by the fact that \( = \) is a congruence.
4.2. THE PROOF OF COMPLETENESS

By axiom \( \text{trap} \), we have \( \text{trap } T \in E \) end = \( E^T \), and \( E^T \equiv F^T_{\nu \lambda} \| \ldots \| F^T_{\nu \rho} \), where:

\[
F^T_{\nu \lambda} = \begin{cases} 
F_{\nu \lambda}[E^T_{f(\nu \lambda)} / E_{f(\nu \lambda)}] & \text{if } F_{\nu \lambda} \equiv \text{if } \partial_{\nu \lambda} \text{ then pause } ; E_{f(\nu \lambda)} \\
F_{\nu \lambda}[\text{nothing} / \text{exit } T] & \text{if } F_{\nu \lambda} \equiv \text{if } \partial_{\nu \lambda} \text{ then exit } T \\
\text{if } \phi_{\lambda} \partial_{\nu \lambda} \text{ then pause } \| \ldots & \text{if } F_{\nu \lambda} \equiv \text{if } \partial_{\nu \lambda} \text{ then pause} \\
\ldots & \phi_{k \lambda} \partial_{\nu \lambda} \text{ then pause } \\
\partial_{\nu \lambda} \uparrow \partial_{\rho \lambda}, G_{\nu \rho} \equiv \text{exit } T & \text{otherwise.}
\end{cases}
\]

Now, \( F^T_{\nu \lambda} \) is a parallel composition of normal forms, with normal forms as derivatives, which can be transformed in a normal form as in the case of \( \| \)”. So, \( F^T_{\nu \lambda}, \ldots, F^T_{\nu \rho} \) are the statements we were looking for, and \( E = F^T_{\nu \lambda} \).

- \( E \equiv \text{suspend } E' \text{ when } s. \)

Since \( E' = E_{\nu} \), we infer \( E = \text{suspend } E_{\nu} \text{ when } s \) by the fact that = is a congruence.

By axiom \( \text{susp} \), we have \( \text{suspend } E_{\nu} \text{ when } s = E^s_{\nu} \), for \( E^s_{\nu} \) a normal form, and \( E^s_{\nu} \equiv F^s_{\nu \lambda} \| \ldots \| F^s_{\nu \rho} \), where:

\[
F^s_{\nu \lambda} = \begin{cases} 
F_{\nu \lambda}[K_{\nu \lambda} / E_{f(\nu \lambda)}] & \text{if } F_{\nu \lambda} \equiv \text{if } \partial_{\nu \lambda} \text{ then pause } ; E_{f(\nu \lambda)} \\
F_{\nu \lambda} & \text{otherwise,}
\end{cases}
\]

where \( K_{\nu \lambda} \equiv \text{if } s^+ \text{ then pause } ; K_{\nu \lambda} \| \text{if } s^- \text{ then } E^s_{f(\nu \lambda)} \).

Now, if \( s^- \text{ then } E^s_{f(\nu \lambda)} \) can be transformed in a normal form as in the case for \( \text{present } \text{ then } \text{ else } \text{ end} \), so that \( F^s_{\nu \lambda} \) is transformed in a parallel composition of normal forms, with normal forms as derivatives, which can be transformed in a normal form as in the case of \( \| \)”. So, \( F^s_{\nu \lambda}, \ldots, F^s_{\nu \rho} \) are the statements we were looking for, and \( E = F^s_{\nu \lambda} \).

This completes the proof.

We recall now the notion of guarded recursive specification and a result stating that every guarded recursive specification has a solution, that is unique modulo \( = \).

A recursive specification over variables \( \bar{P} = P_1, \ldots, P_m \) is a set of equations

\[
P_i = F_i \quad 1 \leq i \leq m
\]

where \( F_i \) is a statement, \( 1 \leq i \leq m. \)

A solution of a recursive specification

\[
P_i = F_i \quad 1 \leq i \leq m
\]

is a set of statements \( \bar{E} \equiv E_1, \ldots, E_m \) such that \( E_i = F_i[E / \bar{P}], 1 \leq i \leq m. \)

A recursive specification

\[
P_i = F_i \quad 1 \leq i \leq m
\]

is guarded if and only if \( P_1, \ldots, P_m \) are guarded in \( F_1, \ldots, F_m. \)
The following lemma states that every guarded recursive specification has a solution which is unique modulo ‘=’. We present the proof given in [70].

**Lemma 4.2.2** Every guarded recursive specification

\[ P_i = F_i \quad 1 \leq i \leq m \]

has a solution \( \vec{E} \equiv E_1, \ldots, E_m \). Moreover, given another solution \( \vec{E}' \equiv E'_1, \ldots, E'_m \), we have \( E'_i = E_i, 1 \leq i \leq m \).

**Proof** By induction over \( m \).

*Base case: \( m = 1 \).*

By axiom \( \text{rec}_1 \), it follows that \( \text{rec} P_1, F_1 \) is a solution of the guarded recursive specification \( P_1 = F_1 \). By axiom \( \text{rec}_2 \), it follows that, given another arbitrary solution \( E'_1 \), we have \( E'_1 = \text{rec} P_1, F_1 \).

*Induction step: \( m > 1 \).*

Let us assume a guarded recursive specification

\[ P_i = G_i \quad 1 \leq i \leq m+1 \]

We begin by giving a solution of this guarded recursive specification.

Let us consider statements \( G_{m+1} \equiv \text{rec} P_{m+1}, F_{m+1} \), and \( G_i \equiv F_i[G_{m+1}/P_{m+1}], 1 \leq i \leq m \). The recursive specification

\[ P_i = G_i \quad 1 \leq i \leq m \]

is guarded and, by inductive hypothesis, it has a solution \( \vec{E} \equiv (E_1, \ldots, E_m) \). So, for \( 1 \leq i \leq m \), we have \( E_i = G_i[\vec{E} / \vec{P}] \), namely \( E_i = F_i[G_{m+1}/P_{m+1}][\vec{E} / \vec{P}] \).

Let us assume statement \( E_{m+1} \equiv G_{m+1}[\vec{E} / \vec{P}] \). For each \( 1 \leq i \leq m \) we have \( E_i = F_i[\vec{E} / \vec{P}, E_{m+1}/P_{m+1}] \). Moreover, \( E_{m+1} \equiv G_{m+1}[\vec{E} / \vec{P}] \equiv (\text{rec} P_{m+1}, F_{m+1})[\vec{E} / \vec{P}] \)

\[ = F_{m+1}[\text{rec} P_{m+1}, F_{m+1}/P_{m+1}][\vec{E} / \vec{P}] \equiv F_{m+1}[G_{m+1}/P_{m+1}][\vec{E} / \vec{P}] \equiv F_{m+1}[\vec{E} / \vec{P}, E_{m+1}/P_{m+1}] \]. Therefore, \( (\vec{E}, E_{m+1}) \) is the solution we were looking for.

We must prove now that this solution is unique modulo ‘=.

If \( (\vec{E}', \vec{E}'_{m+1}) \) is an arbitrary solution, \( \vec{E}' \equiv (E'_1, \ldots, E'_m) \), then we have \( E'_{m+1} = F_{m+1}[\vec{E}' / \vec{P}] \equiv F_{m+1}[\vec{E}'_{m+1}/P_{m+1}] \), namely \( E'_{m+1} = \text{rec} P_{m+1}, (F_{m+1}[\vec{E}' / \vec{P}]) \).

Since \( P_{m+1} \) is not free in \( E'_1, \ldots, E'_m \), we have \( E'_{m+1} = G_{m+1}[\vec{E}' / \vec{P}] \), and, for \( 1 \leq i \leq m \), \( E'_i = F_i[\vec{E}' / \vec{P}, G_{m+1}[\vec{E}' / \vec{P}]/P_{m+1}] \), namely \( E'_i = G_i[\vec{E}' / \vec{P}] \).

Since \( E_i = G_i[\vec{E} / \vec{P}], 1 \leq i \leq m \), by inductive hypothesis, we have \( E'_i = E_i, 1 \leq i \leq m \). Now, \( E'_{m+1} = E_{m+1} \) follows by the fact that \( E'_i = E_i, 1 \leq i \leq m \), by the fact that \( E_{m+1} = G_{m+1}[\vec{E} / \vec{P}] \) and \( E'_{m+1} = G_{m+1}[\vec{E}' / \vec{P}] \), and by the fact that
relation \( = \) is a congruence.
This completes the proof.

We show now that two arbitrary bisimilar constructive Esterel statements are
equated by axioms in Tables 4.1-4.7.

**Lemma 4.2.3** Given constructive Esterel statements \( E \) and \( E' \), \( E \approx E' \) implies
\( E = E' \).

**Proof** By Lemma 4.2.1, we can assume normal forms \( E_1, \ldots, E_m, E_{i'}, \ldots, E_{m'} \) such that:

- \( E = E_1, E' = E_{i'} \)
- \( E_i = F_{i,1} \parallel \cdots \parallel F_{i,n_i}, 1 \leq i \leq m \)
- \( E_{i'} = F_{i',1} \parallel \cdots \parallel F_{i',n_{i'}}, 1 \leq i' \leq m' \)
- \( F_{i,j} \equiv \text{if } \vartheta_{i,j} \text{ then } G_{i,j}, 1 \leq i \leq m, 1 \leq j \leq n_i \)
- \( F_{i',j'} \equiv \text{if } \vartheta_{i',j'} \text{ then } G_{i',j'}, 1 \leq i' \leq m', 1 \leq j' \leq n_{i'} \).

So, to prove the thesis it is sufficient to prove that \( E_1 = E_{i'} \).

Let us denote by \( I \subseteq \{1, \ldots, m\} \times \{1', \ldots, m'\} \) the set of pairs \((i, i')\) such that
\( E_i \approx E_{i'} \). Since \( E \approx E', E = E_1 \) and \( E' = E_{i'} \), and \( = \) is sound modulo \( \approx \), we have
\( (1, 1') \in I \).

We prove now that, given a pair \((i, i') \in I \), and an index \( 1 \leq j \leq n_i \), there exists an
index \( 1 \leq j' \leq n_{i'} \) such that:

- \( \vartheta_{i',j'} = \vartheta_{i,j} \);
- either \( G_{i,j} \equiv G_{i',j'} \text{ or } G_{i,j} \equiv \text{pause}; \ E_{f(i,j)}, G_{i',j'} \equiv \text{pause}; \ E_{f(j',f(j'))} \) and
  \((f(i,j), f(i', j')) \in I \).

We consider the following cases:

- \( G_{i,j} \equiv \text{emit } s \).
  There exists a label \( l \) such that \( E_i \xrightarrow{l} \) and \( \vartheta_{i,j}s \in E_l \). Since \( E_i \approx E_{i'} \), we have
  \( E_{i'} \xrightarrow{l} \). Since \( \vartheta_{i,j}s \in E_l \), there exists \( 1 \leq j' \leq n_{i'} \) such that \( \vartheta_{i',j'} = \vartheta_{i,j} \) and
  \( G_{i',j'} \equiv \text{emit } s \).

- \( G_{i,j} \equiv \text{nothing} \).
  Since \( E_i \) is a normal form and, in particular, it satisfies Cond. 4 and 5 of Def.
  4.1.2, there exists a label \( l \) such that \( E_i \xrightarrow{l} \) and \( \vartheta_{i,j}n \in E_l \). Since \( E_i \approx E_{i'} \), we have
  \( E_{i'} \xrightarrow{l} \). Since \( \vartheta_{i,j}n \in E_l \), there exists \( 1 \leq j' \leq n_{i'} \) such that \( \vartheta_{i',j'} = \vartheta_{i,j} \) and
  \( G_{i',j'} \equiv \text{nothing} \).
\[ G_{i,j} \equiv \text{exit } T. \]

There exists a label \( l \) such that \( E_i \xrightarrow{L} \) and \( \vartheta_{i,j} T \in \mathcal{E}_i \). Since \( E_i \approx E_{i'} \), we have \( E_{i'} \xrightarrow{L} \). Since \( \vartheta_{i,j} T \in \mathcal{E}_i \), there exists \( 1 \leq j' \leq n_{i'} \) such that \( \vartheta_{i',j'} = \vartheta_{i,j} \) and \( G_{i',j'} \equiv \text{exit } T \).

- \( G_{i,j} \equiv \text{pause} \) and \( \vartheta_{i,j} \cap \{ s^+, s^- \} = \emptyset \) for some \( s \in \mathcal{S} \).

Since \( E_i \) is a normal form and, in particular, it satisfies Cond. 4 of Def. 4.1.2, there exists a label \( l \) such that \( E_i \xrightarrow{L} \) and \( \vartheta_{i,j} p \in \mathcal{E}_i \). Since \( E_i \approx E_{i'} \), we have \( E_{i'} \xrightarrow{L} \). Since \( \vartheta_{i,j} p \in \mathcal{E}_i \), there exists \( 1 \leq j' \leq n_{i'} \) such that \( \vartheta_{i',j'} = \vartheta_{i,j} \) and \( G_{i',j'} \equiv \text{pause} \).

- \( G_{i,j} \equiv \text{pause} ; E_{f(i,j)} \).

Since \( E_i \) is a normal form and, in particular, it satisfies Cond. 2 and 3 of Def. 4.1.2, there exists a label \( l \) such that \( E_i \xrightarrow{L} \) nothing \( \| \ldots \| \) nothing ; \( E_{f(i,j)} \) nothing ; \( E_{f(i,j)} \) nothing and \( E_{f(i,j)} \approx E_{f(i',j')} \), for some \( 1 \leq j' \leq n_{i'} \) such that \( G_{i',j'} \equiv \text{pause} \).

Let us assume that \( \vartheta_{i,j} \neq \vartheta_{i',j'} \). Now, if \( \vartheta_{i,j} p \in \mathcal{E}_i \) then there exists \( 1 \leq h' \leq n_{i'} \) such that \( F_{i',h'} \equiv \text{if } \vartheta_{i,j} \text{ then pause} \). By axioms seq and \( \| \) we infer \( \vartheta_{i',j'} \) then pause ; \( E_{f(i',j')} \) if \( \vartheta_{i',j'} \) then pause = if \( \vartheta_{i',j'} \) then pause if \( \vartheta_{i',h'} \) then pause ; \( E_{f(i',j')} \) and we can repeat our reasoning. If \( \vartheta_{i,j} p \notin \mathcal{E}_i \), then there exists \( 1 \leq k \leq n_i \) such that \( \vartheta_{i,j} = \vartheta_{i,k} \phi \), for some \( \phi \in (\mathcal{S}_+^-)^* \), and such that \( G_{i,k} \equiv \text{pause} \). Since \( \vartheta_{i,k} p \in \mathcal{E}_i \), there exists \( 1 \leq k' \leq n_{i'} \) such that \( G_{i',k'} \equiv \text{pause} \) and \( \vartheta_{i,k'} = \vartheta_{i,k} \). Now, \( G_{i',k'} \equiv G_{i',k'} \) if \( \vartheta_{i,j} \) then pause, by axiom \( \|\| \). So, we are in the case above.

- \( G_{i,j} \equiv \text{pause} \) and \( \vartheta_{i,j} \cap \{ s^+, s^- \} \neq \emptyset \) for every \( s \in \mathcal{S} \).

Since \( E_i \) is a normal form and, in particular, it satisfies Cond. 4 of Def. 4.1.2, there exists a label \( l \) such that \( E_i \xrightarrow{L} \) and \( \vartheta_{i,j} p \in \mathcal{E}_i \). Since \( E_i \approx E_{i'} \), we have \( E_{i'} \xrightarrow{L} \). Since \( \vartheta_{i,j} p \in \mathcal{E}_i \), there exists \( 1 \leq j' \leq n_{i'} \) such that \( \vartheta_{i',j'} = \vartheta_{i,j} \) and either \( G_{i',j'} \equiv \text{pause} \), or \( G_{i',j'} \equiv \text{pause} ; E_{f(i',j')} \). In the first case the thesis is proved. In the second case, we have already proved that there exists a parallel component if \( \vartheta_{i,k} \) then pause ; \( E_{f(i,k)} \), \( 1 \leq i \leq k \), with \( E_{f(i,k)} \approx E_{f(i',j')} \). But this is impossible, by Cond. 6 of Def. 4.1.2.

Let us consider now the recursive specification

\[ P_{i,j} = E_{i,j} \quad (i,j) \in \mathcal{I} \]

where \( E_{i,j} \) is the statement such that \( E_{i,j} \equiv H_{i,j,1} \| \ldots \| H_{i,j,n_i} \), with either \( H_{i,j,j'} \equiv \text{if } \vartheta_{i,j} \text{ then pause} ; P_{f(i,j),f(i',j')} \), if both \( F_{i,j} \equiv \text{if } \vartheta_{i,j} \text{ then pause} ; E_{f(i,j)} \) and \( F_{i',j'} \equiv \text{if } \vartheta_{i,j} \text{ then pause} ; E_{f(i',j')} \),
or \( H_{i',j} \equiv F_{i,j} \equiv F_{i',j'} \), otherwise.

This recursive specification is guarded and has both \( \vec{G} \) and \( \vec{G}' \) as solutions, where \( G_{i,j} \equiv E_{i} \) and \( G_{i',j'} \equiv E_{i'} \) for each \((i,j) \in \mathcal{I}\). Now, by Lemma 4.2.2 it follows that \( E_i = E_{i'} \), \((i,i') \in \mathcal{I}\). Since \((1,1') \in \mathcal{I}\), we have \( E_1 = E_{1'} \). This completes the proof.

The following theorem states that axioms in Tables 4.1-4.7 give an axiomatization sound and complete modulo bisimulation on constructive Esterel statements.

**Theorem 4.2.4** Given constructive Esterel statements \( E \) and \( E' \), \( E = E' \) if and only if \( E \approx E' \).

**Proof** “If”: by Lemma 4.2.3. “Only if”: by Lemma 4.1.6.
Chapter 5

A distributed interpretation of Esterel

We have outlined in the introduction some motivations for considering semantics for synchronous languages that describe the behavior of programs in terms of distributed reactions between distributed configurations. These semantics should not compete with the classical ones, but should offer more concrete views of programs, useful to optimize hardware implementation, to improve the verification phase when the observer monitoring verification technique is adopted, and to support design of model-based schedulers.

An interesting issue is to have an integrated environment where one can easily define different semantics for the same language. Techniques to develop such environments have been successfully proposed for asynchronous process calculi. As examples, Degano and Priami have developed a framework to derive both interleaving and truly concurrent semantic models for CCS, π-calculus and Facile in [28], [29] and [31]. In these papers, a proved transition system is taken as the most concrete model of the considered process calculus, and more abstract models are obtained by relabeling the PTS by means of observation functions [27], namely functions which relabel the PTS by maintaining only part of the information carried by labels.

In this chapter, following Degano and Priami, we develop a framework to give different semantics for the synchronous language Esterel, and we show how these semantics support the implementation and verification phases.

As the most concrete semantic model of Esterel, we give a PTS with states representing statement configurations and transitions representing statement reactions. From the PTS one can recover how statement configurations are encoded by latches in the circuit implementation. In fact, circuit latches are synthesized in correspondence with occurrences of the delay statement “\texttt{pause}” in the body of statements. Now, a label of a transition entering a PTS state carries sufficient information to individuate occurrences of statement \texttt{pause} that are executed in the reaction represented by the transition, namely to individuate circuit latches that are set.

All sequences of transitions (computations) from a state of a PTS can be un-
folded and then organized as a tree, called in [28] proved tree. We define a locality observation function and an enabling observation function relabeling proved trees into locality trees and enabling trees, respectively. A locality tree highlights causality, induced by the sequencing construct ‘;’ of Esterel, between actions performed in a sequential component (location) of a statement. Intuitively, actions in the right side of a ‘;’ are caused by actions in the left side of the same ‘;’. An enabling tree highlights causality induced by the construct ‘;’ and by communications among statements running in parallel. Intuitively, an action that needs the presence of a signal to be performed is caused by the action producing such signal.

We develop a method to remove redundant latches from circuits implementing Esterel programs by exploiting information carried by locality trees. To prove the correctness of the method, we show that a transformed circuit and the original one are equivalent w.r.t. the input/output behavior. Moreover, this equivalence is preserved when the two circuits are inserted in whatsoever context. This congruence property is essential, since it guarantees that circuit optimization can be carried on compositionally w.r.t. the structure of statements. Then we prove that our latch removal does not give rise to an increasing of the size of circuit logic. Since we are able to remove latches which are not removed by the existing optimization methods of [89, 90], and, conversely, we do not remove latches removed in [89, 90], we discuss possible integration of our method and those mentioned now.

Finally, we demonstrate by an example how information in enabling trees can be exploited to improve the verification phase when the observer monitoring verification technique is adopted. In particular, we show that we are able to isolate actions of statements which actually cause the violation of a given property and that this cannot be done by considering classical semantics.

This chapter is organized as follows. In Section 5.1 we define a proved transition system for Esterel, and in Section 5.2 we define the abstraction functions by which, from the proved transition system, we obtain more abstract models. In Sections 5.3 and Section 5.4 we show applications to hardware implementation and verification, respectively.

## 5.1 The proved transition system

In this section we propose a proved transition system as our most concrete semantic model for Esterel. PTS states correspond to Esterel statements, PTS transitions correspond to Esterel reactions, and PTS labels carry detailed information on reactions. In particular, labels carry information sufficient to establish which component of a statement emits a signal or executes a pause.

In this chapter we make some assumptions on the statements we deal with.

We do not consider statement `suspend when` which is irrelevant for our treatment and would require only a trivial extension.

Given a statement $E$, we assume that all signals local to substatements of $E$
have different names. This assumption is not restrictive, since all statements satisfy it, modulo renaming of local signals. We will denote with $L_E$ the set of signals local to substatements of $E$.

Finally, we assume to deal with statements that are constructive according to Def. 3.3.3. We note that, since our PTS will carry all information carried by the LTS defined in Table 3.1, the analysis of constructiveness could be carried on also by considering the PTS.

We begin with introducing the notion of proof term.

**Definition 5.1.1** Given a string $\vartheta \in (\{|0,\|1,\|0,\|1,0,1\} \cup S^{\pm})^*$ and a symbol $\mu \in \{n, p\} \cup S \cup T$, $\vartheta \mu$ is a proof term with $\vartheta$ as proof and $\mu$ as action.

Proof terms differ from causality terms of Def. 3.2.1 as symbols in $\{|0,\|1,\|0,\|1,0,1\}$ may appear in proofs. Given a proof term $\vartheta \mu$, the proof $\vartheta$ highlights the syntactic context in which the statement performing the action denoted by $\mu$ is plugged.

Symbol $|0$ (resp. $|1$) is in $\vartheta$ if $\mu$ is performed by a statement in the left (resp. right) branch of a "$|$".
Symbol $\|0$ (resp. $\|1$) is in $\vartheta$ if $\mu$ is performed by a statement in the left (resp. right) side of a "$\|".\n
Symbol $l_0$ (resp. $l_1$) is in $\vartheta$ if $\mu$ is performed by a statement in the terminated (resp. started) instance of the body of a `loop_end`.\n
Symbol $s^+$ (resp. $s^-$) is in $\vartheta$ if $\mu$ is performed by a statement in the `then` (resp. `else`) branch of a `present s then else_end`.\n
Given a proof $\vartheta \in (\{|0,\|1,\|0,\|1,0,1\} \cup S^{\pm})^*$, we denote with $|\vartheta|$ the event $(|\vartheta| \upharpoonright S^\pm)$.

**Definition 5.1.2** A label is a tuple $l = (S_l, E_l, \mathcal{N}_l, \mathcal{T}_l)$ such that:

- $S_l$ is an event over $S$;
- $E_l$ is a set of proof terms such that, for each $\vartheta \mu \in E_l$, $|\vartheta| \subseteq S_l$;
- $\mathcal{N}_l$ is a set of proof terms such that, for each $\vartheta \mu \in \mathcal{N}_l$, $|\vartheta| \not\subseteq S_l$;
- $\mathcal{T}_l \in \{0, 1\} \cup 2^T$.

We will denote with $\mathcal{L}$ the set of labels as in Def. 5.1.2.

As in Chapter 3, a PTS transition $E \xrightarrow{t} F$ will represent the reaction of $E$ to an environment that supplies every input signal $s$ such that $s^+ \in S_l$ and does not supply any input signal $s$ such that $s^- \in S_l$. Proof terms in $E_l$ refer to atomic actions that are performed during the reaction represented by $E \xrightarrow{t} F$, while proof terms in $\mathcal{N}_l$ refer to atomic actions that are not performed during the reaction represented by $E \xrightarrow{t} F$. The component $\mathcal{T}_l$ gives information on the termination of $E$.

The proved transition system giving the most concrete semantics for Esterel is defined by the transition system specification in Table 5.1.
The PTS for Esterel.

\[
\begin{align*}
\text{nothing} & \xrightarrow{(\emptyset, \{e_n\}, \emptyset, \emptyset)} \text{nothing} & \text{(nothing)} \\
\text{emit } s & \xrightarrow{(\emptyset, \{e_s\}, \emptyset, \emptyset)} \text{nothing} & \text{(emit)} \\
\text{pause } & \xrightarrow{(\emptyset, \{e_p\}, \emptyset, 1)} \text{nothing} & \text{(pause)} \\
\text{exit } T & \xrightarrow{(\emptyset, \{e_T\}, \emptyset, \{T\})} \text{nothing} & \text{(exit)} \\
\end{align*}
\]

\[
\begin{align*}
E & \xrightarrow{l} F & \mathcal{T}_i = 0 \lor \mathcal{T}_i = \{T\} & \text{(trap-1)} \\
\text{trap } T \text{ in } E & \xrightarrow{\text{tr}(T)} \text{ end} & \text{nothinng} \\
E & \xrightarrow{l} F & \mathcal{T}_i = 1 & \text{(trap-2)} \\
\text{trap } T \text{ in } E & \xrightarrow{\text{tr}(T)} \text{ trap } T \text{ in } F \text{ end} & \text{nothinng} \\
E & \xrightarrow{l} F & \mathcal{T}_i \subseteq \mathcal{T} \land \mathcal{T}_i \neq \{T\} & \text{(trap-3)} \\
\text{trap } T \text{ in } E & \xrightarrow{\text{tr}(T)} \text{ end} & \text{nothinng} \\
E & \xrightarrow{l} F & E' \xrightarrow{U} F' & s'' \notin S_{f} & \text{(present-1)} \\
\text{present } s \text{ then } E \text{ else } E' \text{ end} & \xrightarrow{\text{s''}(\mathcal{T}, \mathcal{T}_f)} F' \\
E & \xrightarrow{l} F & E' \xrightarrow{V} F' & s'' \notin S_{f} & \text{(present-2)} \\
\text{present } s \text{ then } E \text{ else } E' \text{ end} & \xrightarrow{\text{s''}(\mathcal{T}, \mathcal{T}_f)} F' \\
E \parallel E' & \xrightarrow{t_{e}} F \parallel F' & S_{t} \uparrow S_{f}, \mathcal{T}_i, \mathcal{T}_f \in \{0,1\} & \text{(parallel-1)} \\
E \parallel E' & \xrightarrow{t_{e}} F \parallel F' & \mathcal{T}_i \subseteq \mathcal{T} \lor \mathcal{T}_f \subseteq \mathcal{T} & \text{(parallel-2)} \\
E & \xrightarrow{l} F & E' \xrightarrow{V} F' & S_{t} \uparrow S_{f} & \text{(seq-1)} \\
E & \xrightarrow{l} F & E' \xrightarrow{V} F' & \mathcal{T}_i = 0 \land S_{t} \uparrow S_{f} & \text{(seq-2)} \\
\end{align*}
\]
5.1. THE PROVED TRANSITION SYSTEM

The PTS for Esterel.

\[
E \xrightarrow{l} F, F' \xrightarrow{\ell} F' \quad \mathcal{T}_i \subseteq \mathcal{T} \quad \text{(seq-3)}
\]
\[
E \xrightarrow{\text{signal } s \text{ in } E \text{ end}} F \quad \text{signal } s \in E \quad \text{end} \quad \text{signal } s \in F \quad \text{end} \quad \mathcal{T}_i = 0, F \neq E \quad \text{(loop-1)}
\]
\[
F \xrightarrow{l} F', E \xrightarrow{\text{seq}(l, \text{new}(l'))} F' \quad \text{loop}_E F \text{ end} \quad \text{seq}(l, \text{new}(l')) \quad \text{loop}_E F' \text{ end} \quad \mathcal{T}_i = 1 \quad \text{(loop-2)}
\]
\[
F \xrightarrow{l} F', \quad \text{loop}_E F \text{ end} \quad \text{loop}_E F' \text{ end} \quad \mathcal{T}_i \subseteq \mathcal{T} \quad \text{(loop-3)}
\]

Table 5.1: The proved transition system for Esterel.

Transition rules for basic statements are exactly as in Table 3.1.

Given a set of proof terms \( \Theta \) and a proof \( \phi \in (\{||_0, ||_1, ;_0, ;_1, l_0, l_1\} \cup S_\downarrow^+)^* \), we denote with \( \Theta_\phi \) the set of proof terms \( \{\phi \theta \mu | \theta \mu \in \Theta\} \).

In rules \( \text{present-1} \) and \( \text{present-2} \) we assume that, given labels \( l \) and \( l' \) and a symbol \( \gamma \in S_\downarrow^+ \) such that \( \gamma \notin S_i \), the label \( \gamma(l, l') \) is defined as follows:

\[ \gamma(l, l') = (S_i \cup \{\gamma\}, E^l_i, N^l_i \cup N^{l'}_i, N^{l'}_i, \mathcal{T}_i). \]

Namely, the label \( \gamma(l, l') \) differs from that defined in Chapter 3 because we do not forget proof terms.

In rules \( \text{parallel-1} \) and \( \text{parallel-2} \) we assume a partial function \( \oplus : \mathcal{L} \times \mathcal{L} \to \mathcal{L} \) such that, given labels \( l \) and \( l' \) such that \( S_i \uparrow S_{l'} \), we have:

- \( S_{l \oplus l'} = S_i \cup S_{l'} \);
- \( E_{l \oplus l'} = E^l_i \cup E^{l'}_i \);
- \( N_{l \oplus l'} = N^l_i \cup N^{l'}_i \);
- \( \mathcal{T}_{l \oplus l'} = \begin{cases} \max(\mathcal{T}_i, \mathcal{T}_{l'}) & \text{if } \mathcal{T}_i, \mathcal{T}_{l'} \in \{0, 1\} \\ (\mathcal{T}_i \cup \mathcal{T}_{l'}) \cap \mathcal{T} & \text{otherwise.} \end{cases} \)

Differently to function \( \otimes \) of Chapter 3, function \( \oplus \) does not remove information carried by \( l \) and \( l' \). The reason is that \( \otimes \) removes information that is useless for
capturing the input/output behavior of statements, but that is needed for keeping
track of all statements pause that are executed at each cycle, namely for keeping
track of the latches that are set.

If \( \partial \mu \) refers to an atomic action performed by a statement in the body of \( E \),
then \( _0^0 \partial \mu \) appears in \( l \oplus l' \) and reflects that this atomic action is performed by a
statement in the left branch of "\(". Analogously, if \( \partial \mu \) refers to an atomic action
performed by a statement in the body of \( E' \), then \( _1^1 \partial \mu \) appears in \( l \oplus l' \) and reflects
that this atomic action is performed by a statement in the right branch of "\(".

**Example 5.1.3** Let us assume \( E \equiv E_0 \parallel E_1 \), where
\( E_0 \equiv \text{present } s_1 \text{ then emit } s_2 \text{ else nothing end}, \)
\( E_1 \equiv \text{present } s_3 \text{ then emit } s_4 \text{ else nothing end}. \)
By transition rule parallel,1, we have \( E \xrightarrow{l} \text{nothing} \parallel \text{nothing}, \)
where \( l = \{ \{ s_1^+, s_3^+ \}, \{ ||0 s_1^+ s_2, ||1 s_3^+ s_4 \}, \{ ||0 s_1^- n, ||1 s_3^- n \}, 0 \}. \)

Given an event \( S = \{ \gamma_1, \ldots, \gamma_n \} \), we denote with \( \hat{S} \) the ordered event \( \gamma_1 \ldots \gamma_n \),
such that \( |\hat{S}| = S \), and \( h \leq k \) for \( h \) and \( k \) such that \( \gamma_i \in \{ s_h^-, s_h^+ \}, \gamma_i \in \{ s_k^-, s_k^+ \}, \)
\( i_t < i_m \).

Given labels \( l_1, \ldots, l_n \) such that \( E \xrightarrow{l_t} \) and \( T_t = 0 \), we denote with \( Im(E) \) the
set of ordered events \{ \( \hat{S}_1, \ldots, \hat{S}_n \) \}.

In rules seq,1, seq,2 and seq,3, we assume a partial function \( \triangleright : \mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L} \) such that,
given labels \( l \) and \( l' \) such that \( T_t = 0 \) implies \( S_t \uparrow S_{t'} \), we have:
\[
l \triangleright l' = \begin{cases}
(S_t \cup S_{t'}, \mathcal{E}^{\triangleright}_t \cup \mathcal{E}^{\triangleright}_t \cup \mathcal{N}^{\triangleright}_t \cup \bigcup_{\partial \in Im(E)} \mathcal{N}^{\triangleright}_{\partial t} \cup \bigcup_{\partial \in Im(E) \setminus \{ \hat{s}_t \}} \mathcal{E}^{\triangleright}_{\partial t}, T_{t'} \) & \text{if } T_t = 0 \\
(S_t, \mathcal{E}^{\triangleright}_t \cup \mathcal{N}^{\triangleright}_t \cup \bigcup_{\partial \in Im(E)} \mathcal{E}^{\triangleright}_{\partial t} \cup \mathcal{N}^{\triangleright}_{\partial t}, T_{t'}) & \text{if } T_t \neq 0.
\end{cases}
\]

If \( \partial \mu \) refers to an atomic action performed by a statement in the body of \( E \), then
\( _0^0 \partial \mu \) appears in \( l \triangleright l' \) and reflects that this atomic action is performed by a statement
in the left side of "\(". Analogously, if \( \partial \mu \) refers to an atomic action performed by a
statement in the body of \( E' \), then \( _1^1 \partial \mu \) appears in \( l \triangleright l', \phi \in Im(E) \), and reflects
that this atomic action is performed by a statement in the right side of "\(".

**Example 5.1.4** Let us assume the statement
\( E \equiv \text{(present } s_1 \text{ then pause else nothing end); emit } s_2. \)
By rule seq,1 we have \( E \xrightarrow{l_1} \text{nothing} \), with \( l_1 = \{ \{ s_1^- \}, \{ ||0 s_1^- n, ||1 s_1^- s_2 \}, \{ ||0 s_1^+ p \}, 0 \}. \)
By rule seq,2 we have \( E \xrightarrow{l_2} F \), with \( l_2 = \{ \{ s_1^+ \}, \{ ||0 s_1^+ p \}, \{ ||0 s_1^- n, ||1 s_1^- s_2 \}, 1 \} \) and
\( F \equiv \text{nothing; emit } s_2. \)

In rule signal we assume a function \( si \) such that \( si \) is defined for a signal \( s \in \mathcal{S} \)
and a label \( l \in \mathcal{L} \) if the following conditions are satisfied:
- if \( s^+ \in S_t \) then \( s \in Em(l); \)
- if \( s \in Em(l) \) then \( s^- \not\in S_t. \)
If these conditions are satisfied then we have $\text{sig}(s, l) = l$.

Rule $\text{signal}$ does not take into account that the feedback of $s$ may give rise to paradoxes of causality, since we have assumed to deal with statements that are constructive according to Def. 3.3.3.

Occurrences of $s^+$ and $s^-$ are not replaced in proof terms of label $\text{sig}(s, l)$ by the respective causes, contrarily to what is done by function $\text{loc}$ in Chapter 3. This will be needed to compute causality relations between actions of production of local signals and actions requiring such signals.

Let us consider now statement $\text{loop } E \text{ end}$. We cannot have a rule analogous to rule $\text{loop}_1$ of Chapter 3 which implies that statements $\text{loop } E \text{ end}$ and $E; \text{loop } E \text{ end}$ are bisimilar. In fact, it is reasonable to consider these statements as equivalent if we do not want to discriminate statements having the same input/output behavior. But, if we wish our semantics to give information on circuits corresponding to statements, so that optimizations of circuits can be carried on, $\text{loop } E \text{ end}$ and $E; \text{loop } E \text{ end}$ must be distinguished, since they originate different circuits. In fact, for each statement $\text{pause}$ in the body of $E$, the circuit implementing $\text{loop } E \text{ end}$ contains one latch, while the circuit implementing $E; \text{loop } E \text{ end}$ contains two latches.

So, let us introduce the construct $\text{loop}_E \text{ - end}$, where the index remembers that when the body terminates then $E$ restarts immediately. The semantics of $\text{loop } E \text{ end}$ coincides with that of $\text{loop}_E \text{ E end}$.

Let us denote with $S'$ the set $\{s' \mid s \in S\}$. Given a label $l' \in \mathcal{L}$, let $\text{new}_E(l')$ be the label obtained by replacing in $l'$ each occurrence of a local signal $s \in L_E$ by $s'$.

We assume a partial function $\text{seq} : \mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L}$ such that, given labels $l_0$ and $l_1$ such that $S_0 \uparrow S_1$, we have:

$$\text{seq}(l_0, l_1) = (S_0 \cup S_1, \mathcal{E}_{l_0} \cup \mathcal{E}_{l_1}; S_0, N_{l_0} \cup N_{l_1}, C_l)$$

Rule $\text{loop}_1$ states that if $F$ terminates then $\text{loop}_E \text{ F end}$ performs both actions of $F$ and $E$. In this rule we require that $F \neq E$ since the body of a $\text{loop}_E \text{ F end}$ cannot terminate immediately. The label $\text{seq}(l, \text{new}_E(l'))$ in rule $\text{loop}_1$ contains proof terms of the form $l_0 \partial \mu$ corresponding to actions performed by statements in the terminated instance of the body of $E$, and proof terms of the form $l_1 \partial \mu$ corresponding to actions performed by statements in the restarted instance of the body of $E$. A proof term $\partial \mu$ with $\mu = s' \in S'$ refers to the signal $s$ local to the started instance of $E$. So, we distinguish between signals local to the started instance of $E$ and signals local to the terminated instance of $E$.

Rules $\text{loop}_2$ and $\text{loop}_3$ state that if $F$ does not terminate then $\text{loop}_E \text{ F end}$ reacts as $F$.

Let us consider now statement $\text{trap } T \text{ in } E \text{ end}$. As in Chapter 3, given a transition $E \xrightarrow{I} F$ with $\partial_1 T, \ldots, \partial_n T$ the proof terms in $\mathcal{E}_{l} \cup \mathcal{N}_{l}$ having $T$
as action, $\vartheta_i \cdot S^- = \gamma_{i,1} \cdots \gamma_{i,n_i}$, we denote with $T(l)$ the set of ordered events $\gamma_{i,1} \cdots \gamma_{i,j_i} \cdots \gamma_{i,1} \cdots \gamma_{i,n_i}$, where $1 \leq j_i \leq n_i$ for $1 \leq i \leq n$. Given an ordered event $\phi \in T(l)$, if each signal $s$ with $s^+ \in |\phi|$ is present and each signal $s$ with $s^- \in |\phi|$ is absent, then $E$ does not exit the trap $T$, since no statement exit $T$ is executed.

Now, given a label $l$ and a trap name $T$, $tr(T,l)$ is the label $l'$ such that:

- $S_T = S_l$;
- $E_T = E_l[\{ \vartheta \cdot p \mid \phi \in T(l), |\phi| \cdot \vartheta \cdot | \vartheta | \text{ for some } 1 \leq i \leq n \} / \vartheta \cdot p / \vartheta T]$;
- $N_T = N_l[\{ \vartheta \cdot p \mid \phi \in T(l), |\phi| \cdot \vartheta \cdot | \vartheta | \text{ for some } 1 \leq i \leq n \} / \vartheta \cdot p / \vartheta T] \cup \{ \vartheta \cdot p \mid \phi \in T(l), |\phi| \cdot S_i, |\vartheta | \cdot | \vartheta | \text{ for some } 1 \leq i \leq n, \vartheta \cdot p \in E_i \}$;
- $T_T = \begin{cases} 1 & \text{if } T_T = 0 \text{ or } T_T = \{T\} \\ T_T \setminus \{T\} & \text{otherwise.} \end{cases}$

Function $tr$ does not remove proof terms, contrarily to function $tr$ of Chapter 3.

Let us denote with $[E]_{pts}$ the portion of PTS reachable from state $E$. We interpret $[E]_{pts}$ as the most concrete semantics of statement $E$. $[E]_{pts}$ gives information on how configurations of $E$ are encoded by latches in circuit $C_E$. In fact, the initial configuration of $E$ is encoded by the boot latch, and, for each transition $F \xrightarrow{L} F'$ in $[E]_{pts}$ such that $\vartheta_1 \cdot p, \ldots, \vartheta_n \cdot p$ are the proof terms in $E_l$ having $p$ as action, the configuration of $E$ corresponding to state $F'$ is encoded by latches synthesized from occurrences of pause corresponding to $\vartheta_1 \cdot p, \ldots, \vartheta_n \cdot p$.

### 5.2 Locality trees and enabling trees

Given an Esterel statement $E$, we denote with $[E]_{pt}$ the tree obtained by unfolding $[E]_{pts}$. Following [28], we call $[E]_{pt}$ a proved tree.

In this section, we define two observation functions, the locality observation function, $Loc$, and the enabling observation function, $En$, transforming proved trees into locality trees and enabling trees, respectively. Locality trees highlight causality between actions induced by the construct "$;\$", while enabling trees highlight causality between actions induced by the construct "?$\$" and by communications between statements running in parallel.

To define the function $Loc$, for each computation $E \xrightarrow{L_1} \ldots \xrightarrow{L_n} E'$ of a proved tree $[E]_{pt}$, we consider a locality relation $\rightarrow_{(i_1, \ldots, i_n)}$ on proof terms in $\bigcup_{i=1}^n E_i$. This relation reflects causality between actions induced by the sequencing construct "$;\$". Intuitively, $(\vartheta \mu, \vartheta' \mu') \in \rightarrow_{(i_1, \ldots, i_n)}$ if $\vartheta \mu$ and $\vartheta' \mu'$ refer to actions performed by statements in a sequential component of $E$, and the action denoted by $\mu'$ can be performed only if the action denoted by $\mu$ has been performed.
5.2. LOCALITY TREES AND ENABLING TREES

The relation $\rightarrow_{(t_i, t_{i+1})}$ is derived from instantaneous locality relations $\rightarrow_t$, $1 \leq i \leq n$, and from temporal locality relations $\rightarrow_{(t_i, t_{i+1})}$, $1 \leq i < n$. Relation $\rightarrow_t$ relates proof terms corresponding to actions performed in the $i^{th}$ reaction of $E$. Relation $\rightarrow_{(t_i, t_{i+1})}$ relates proof terms corresponding to actions performed in the $i^{th}$ reaction of $E$ and proof terms corresponding to actions performed in the $(i+1)^{th}$ reaction of $E$.

**Definition 5.2.1** Given a label $l$, the instantaneous locality relation $\rightarrow_t$ contains precisely all pairs $(\partial \mu, \partial' \mu') \in E_t \times E_t$ such that, for some $\phi, \phi_0, \phi_1$, one of the following cases holds:

- $\partial = \phi \cdot \phi_0$ and $\partial' = \phi \cdot \phi_1$
- $\partial = \phi \cdot \phi_0$ and $\partial' = \phi \cdot \phi_1$.

Given proof terms $\partial \mu$ and $\partial' \mu'$ such that $(\partial \mu, \partial' \mu') \in \rightarrow_t$, either $\mu$ denotes an action performed by a statement in the left side of a “;” and $\mu'$ denotes an action performed by a statement in the right side of that “;”, or $\mu$ denotes an action performed by a statement in the terminated instance of the body $E$ of a statement loop $E$ end and $\mu'$ denotes an action performed by a statement in the started instance of $E$.

**Example 5.2.2** Let us assume $E \equiv$

```
present s then (((emit s_1 || emit s_2); pause); emit s_3) else nothing end.
```

We have that $E \xrightarrow{l} E'$, where $E' \equiv$ nothing; emit s_3,

```
l = \{\{s^+\}, \{s^+; s_0||s_1, s^+; s_0||s_2, s^+; s_0||p\}, \{s^-; n\}, 1\},
```

and $\rightarrow_t = \{(s^+; s_0||s_1, s^+; s_0||s_2, s^+; s_0||p)\}$, which reflects that pause can start only if both emit s_1 and emit s_2 have terminated.

Given an instantaneous locality relation $\rightarrow_t$, a proof term $\partial \mu$ is initial in $\rightarrow_t$ if there exists no $\partial' \mu'$ such that $(\partial' \mu', \partial \mu) \in \rightarrow_t$, and a proof term $\partial \mu$ is final in $\rightarrow_t$ if there exists no $\partial' \mu'$ such that $(\partial \mu, \partial' \mu') \in \rightarrow_t$.

**Definition 5.2.3** Given transitions $E \xrightarrow{l} F$ and $F \xrightarrow{k} G$, the temporal locality relation $\rightarrow_{(\mu, \nu)}$ contains precisely all pairs $(\partial \mu, \partial' \nu) \in E_t \times E_t$ such that $\partial \mu$ is final in $\rightarrow_t$, $\partial' \nu$ is initial in $\rightarrow_
u$, and $\partial / \{\{0, 0\}, \{1, 0\}\} = \partial' / \{\{0, 0\}, \{1, 0\}\}$.

Given proof terms $\partial \mu$ and $\partial' \nu$ such that $(\partial \mu, \partial' \nu) \in \rightarrow_{(\mu, \nu)}$, the action $\nu$ corresponds to the termination of a statement nothing that is the derivative (according to transition rules in Table 5.1) of the statement that performs action $\mu$. In general, we may have $\partial \neq \partial'$. In fact, symbols $s^+$, $s^-$ and $\cdot 1$ may occur in $\partial \setminus \partial'$ since transition rules present_1, present_2 and seq_1 remove both occurrences of statement present and left sides of statements “;” from PTS states. Moreover, symbol $l_1$ may occur in $\partial$ but not in $\partial'$, since $\partial'n$ is initial in $\rightarrow_{\nu}$, while $l_0$ may occur in $\partial'$ but not in $\partial$, since $\partial \mu$ is final in $\rightarrow_t$.  

Example 5.2.4 Given $E$, $l$, $E'$ as in Example 5.2.2 and given the label $l' = (\emptyset, \{a_0, n\}, s_3, \emptyset, 0)$, we have $E' \xrightarrow{l'} \text{nothing}$. As $s^+_{a_0} \cdot p$ is final in $\rightarrow_p$ and $s^+_{a_0} \cdot 1 \models \{0, 1, a_0\} = \emptyset$, we have $\rightarrow_{(l', p)} = \{(s^+_{a_0} \cdot p, a_0)\}$. This reflects that statement nothing in the body of $E'$ is the derivative of pause in the body of $E$.

Definition 5.2.5 Given a computation $E_1 \xrightarrow{l_1} \ldots \xrightarrow{l_n} E_{n+1}$, the locality relation $\rightarrow_{(l_1, \ldots, l_n)} \subseteq \bigcup_{i=1}^n \mathcal{E}_{l_i} \times \bigcup_{i=1}^{n-1} \mathcal{E}_{l_i}$ is such that:

$$\rightarrow_{(l_1, \ldots, l_n)} = \bigcup_{i=1}^n \rightarrow_{l_i} \cup \bigcup_{i=1}^{n-1} \rightarrow_{(l_i, l_{i+1})}.$$

Given proof terms $\vartheta$ and $\vartheta'$ such that $(\vartheta, \vartheta' p') \in \rightarrow_{(l_1, \ldots, l_n)}$, we say that $\vartheta$ is a local cause of $\vartheta' p'$.

Locality relations have been extensively studied also for asynchronous process calculi, such as CCS and $\pi$-calculus (as examples, see [28, 29]). These relations are defined over transitions and not over proof terms, since each PTS transition is labeled by exactly one proof term. Our definition of locality relation is technically more complex w.r.t. those in [28, 29]. The reason is that CCS and $\pi$-calculus offer a prefixing operator “-” that permits to prefix a process with an action, but one cannot prefix a process with another one. So, while we have Esterel statements of the form $(E_1 || E_2); E_3$ and of the form (present $s$ then $E_1$ else $E_2$ end); $E_3$, we cannot write CCS processes of the form $(p_1 \mid p_2) \cdot p_3$ or of the form $(p_1 + p_2) \cdot p_3$.

So, in CCS it cannot happen that an actions is caused by actions in both branches of a “-” or by actions in both branches of a “+”.

We define now the observation function $\text{Loc}$ transforming a proved tree $[E]_{pl}$ into the locality tree $\text{Loc}([E]_{pl})$.

Definition 5.2.6 The locality observation function $\text{Loc}$ relabels any computation $E_1 \xrightarrow{l_1} \ldots \xrightarrow{l_n} E_{n+1}$ as $E_1 \xrightarrow{\text{Loc}(l_1)} \ldots \xrightarrow{\text{Loc}(l_n)} E_{n+1}$, where $\text{Loc}(l_i)$ is the set of pairs $(\vartheta, K)$ such that:

- $\vartheta \in \mathcal{E}_{l_i}$, and $\mu = p$ implies that $\vartheta$ is final in $\rightarrow_{l_i}$;
- $K = \{(k, \vartheta' p') \mid k \in \{0, \ldots, i-1\}, \vartheta' p' \in \mathcal{E}_{l_{i-k}}, (\vartheta' p', \vartheta) \in \rightarrow_{(l_1, \ldots, l_i)}\}$.

Locality trees do not keep track of actions of pausing that are preempted. For a pair $(\vartheta, K)$ in a label of $\text{Loc}([E]_{pl})$, the set $K$ contains pairs $(k, \vartheta' p')$ with $k$ a backward pointer in $\text{Loc}([E]_{pl})$ and $\vartheta' p'$ a local cause of $\vartheta$.

To define the function $\text{En}$, for each computation $E \xrightarrow{l_1} \ldots \xrightarrow{l_n} E'$ of a proved tree $[E]_{pl}$, we consider an enabling relation $<_{(l_1, \ldots, l_n)}$ on proof terms in $\bigcup_{i=1}^n \mathcal{E}_{l_i}$. This relation reflects causality between actions induced by the construct “;” and by communications between statements running in parallel.
5.2. LOCALITY TREES AND ENABLING TREES

**Procedure** ComRel (in: $E$, $l$, out: $\sim_\ell$):

$\mathcal{E} := \emptyset$, $\mathcal{E}' := \mathcal{E}_i$, $\mathcal{N}' := \mathcal{N}_i \setminus \{\vartheta \mu | \exists s \notin L_E, (s^+ \in |\vartheta| \land s \in S_{il}) \lor (s^- \in |\vartheta| \land s^+ \in S_{il})\}$;

while $\mathcal{E} \neq \mathcal{E}'$ do

1. remove from $\mathcal{E}'$ and add to $\mathcal{E}$ every $\vartheta \mu$ such that:
   - for each $s^+ \in |\vartheta|$ with $s \in L_E$, there exists $\phi s$ such that $\phi s \in \mathcal{E}$
   - for each $s^- \in |\vartheta|$ with $s \in L_E$, no $\phi s$ is in $\mathcal{N}' \cup \mathcal{E} \cup \mathcal{E}'$
   - no $\vartheta' \mu' \in \mathcal{E}'$ exists such that $(\vartheta' \mu', \vartheta \mu) \in \sim_\ell$

2. remove from $\mathcal{N}'$ every $\vartheta \mu$ such that either there exists $s^+ \in |\vartheta|$ with $s \in L_E$ and no $\phi s$ in $\mathcal{N}' \cup \mathcal{E} \cup \mathcal{E}'$, or there exists $s^- \in |\vartheta|$ with $s \in L_E$ and $\partial s \in \mathcal{E}$;

3. add to $\sim_\ell$ every pair $(\vartheta s, \vartheta' \mu') \in \mathcal{E} \times \mathcal{E}'$ with $\vartheta = \vartheta_1 s^+ s_2$ and $s \in L_E$;

4. for each $\vartheta s \in \mathcal{E}, \vartheta' z \in \mathcal{N}_i$ such that $s^- \in |\vartheta'|$ and $s, z \in L_E$, add $(\vartheta s, \vartheta'' \mu'')$ to $\sim_\ell$ for each $\vartheta'' \mu'' \in \mathcal{E}'$ such that $\vartheta'' = \vartheta_1 s^- s_2$.

**Figure 5.1:** The procedure ComRel.

The relation $\prec_{\{a_1, \ldots, a_n\}}$ is derived from the relation $\mapsto_{\{a_1, \ldots, a_n\}}$ and from communication relations $\sim_i, 1 \leq i \leq n$. Each relation $\sim_i$ reflects causality between actions induced by communications. Intuitively, if there exist proof terms $\vartheta_1 s, \ldots, \vartheta_m s$ such that $(\vartheta_j s, \vartheta' \mu') \in \sim_i, 1 \leq j \leq m$, then the action denoted by $\mu'$ requires the presence of signal $s$ in order to be performed. This signal $s$ may be emitted by actions corresponding to proof terms $\vartheta_1 s, \ldots, \vartheta_m s$.

**Definition 5.2.7** Given a transition $E \xrightarrow{l} F$, the communication relation $\sim_\ell \subseteq \mathcal{E}_i \times \mathcal{E}_i$ is computed by the procedure ComRel in Fig. 5.1.

We explain now the procedure ComRel. To add a pair $(\vartheta s, \vartheta' \mu')$ to $\sim_\ell$, we must be sure that the statement emit $s$ performing the action denoted by $s$ is executed, namely we must determine that it is executed without making self-justified assumptions over the status of signals. So, we assume initially only the status of input signals as given by $S_l$ and we make no assumption on the status of local and output signals. We can say that a signal $s$ is emitted only if we discover that an emit $s$ is executed, and we can say that $s$ cannot be emitted only if we discover that no emit $s$ can start.

The set $\mathcal{E}$ contains proof terms corresponding to actions that are performed. Initially, we have $\mathcal{E} = \emptyset$. The set $\mathcal{N}_i \setminus \mathcal{N}'$ contains proof terms corresponding to actions that are not performed. This set contains initially a proof term $\vartheta \mu$ if the action $\mu$ cannot be performed due to the status of an input signal. Finally, if we do not know whether an action is performed or not, the corresponding proof term is in $\mathcal{E}'$ or in $\mathcal{N}'$. 
Now, for a signal $s$, if no proof term $\vartheta s$ is in $\mathcal{E} \cup \mathcal{E}' \cup \mathcal{N}'$ then we can say that $s$ cannot be emitted. If some proof term $\vartheta s$ is in $\mathcal{E}$ then we can say that $s$ has been emitted. Finally, if no proof term $\vartheta s$ is in $\mathcal{E}$ and some proof terms of the form $\vartheta s$ are in $\mathcal{E}' \cup \mathcal{N}'$, then the status of $s$ is unknown.

Given $\vartheta \mu \in \mathcal{E}'$, we discover that action $\mu$ is performed if each signal $s$ with $s^+ \in |\vartheta|$ (resp. $s^- \in |\vartheta|$) has been emitted (resp. cannot be emitted) and all actions $\mu'$ such that $(\vartheta' \mu', \vartheta \mu) \in \rightarrow_\gamma$ are terminated. In this case $\vartheta \mu$ is moved from $\mathcal{E}'$ to $\mathcal{E}$ (see step 1).

Given $\vartheta \mu \in \mathcal{N}'$, we discover that action $\mu$ is not performed if there exists a signal $s$ with $s^+ \in |\vartheta|$ (resp. $s^- \in |\vartheta|$) which cannot be emitted (resp. has been emitted). In this case $\vartheta \mu$ is removed from $\mathcal{N}'$ (see step 2).

Given proof terms $\vartheta s, \vartheta' \mu'$ as in step 3, action $\mu'$ requires that signal $s$ is present, since it can be performed by a statement that is in the then branch of a present $s$. Now, action $\mu'$ depends on action $s$, since this consists in the production of $s$.

Given proof terms $\vartheta s, \vartheta' z, \vartheta'' \mu''$ as in step 4, action $\mu''$ needs that signal $z$ is absent, since it can be performed by a statement in the else branch of a present $z$. Now, $z$ cannot be emitted since emit $z$ is in the else branch of a present $s$ and signal $s$ is emitted by the action corresponding to $s$. So, action $\mu''$ depends on this action $s$.

Given proof terms $\vartheta s$ and $\vartheta' \mu'$ such that $(\vartheta s, \vartheta' \mu') \in \sim_\nu$, we say that $\vartheta s$ is a communication cause of $\vartheta' \mu'$.

**Example 5.2.8** Let us assume $E \equiv \text{signal } s_1, s_2 \text{ in (emit } s_1 \parallel (F_1 \parallel F_2)) \text{ end}$, where:

$F_1 = \text{present } s_1 \text{ then nothing else emit } s_2 \text{ end}$,

$F_2 = \text{present } s_2 \text{ then nothing else emit } s_3 \text{ end}$.

We have $E \xrightarrow{\nu} \text{signal } s_1, s_2 \text{ in (nothing } \parallel (\text{nothing } \parallel \text{nothing})) \text{ end}$, with $\nu = \langle \emptyset, \{s_1, s_2, s_3, s_1^n, s_2^n, s_2^n \rangle \rangle$.

Now, $\sim_\nu = \{(s_1, s_2, s_3, s_1^n, s_2^n, s_2^n) \rangle \rangle$ reflects that both nothing and emit $s_3$ depend on emit $s_1$.

Note that in Example 5.2.8 we can determine that emit $s_3$ depends on emit $s_1$ since component $N_i$ of $\nu$ keeps track that emit $s_2$ is not executed.

**Definition 5.2.9** Given a computation $E_1 \xrightarrow{\mu_1} \ldots \xrightarrow{\mu_n} E_{n+1}$, the enabling relation $\prec_{(\mu_1, \ldots, \mu_n)} \subseteq \bigcup_{i=1}^n \mathcal{E}_i \times \bigcup_{i=1}^n \mathcal{E}_i$, is such that:

$$\prec_{(\mu_1, \ldots, \mu_n)} = \bigcup_{i=1}^n \sim_i \cup \rightarrow_{(\mu_1, \ldots, \mu_n)}^+.$$

For each pair $(\vartheta s, \vartheta' \mu') \in \sim_\nu$, all local and communication causes of $\vartheta s$ are inherited by $\vartheta' \mu'$, but not conversely. This is reasonable, as action $\mu'$ requires the presence of $s$ in order to be performed, while action $s$ does not require that signal $s$ is sensed.

We define now the observation function $En$ transforming a proved tree $[E]_{pl}$ into the enabling tree $En([E]_{pl})$. 
**Definition 5.2.10** The enabling observation function $En$ relabels any computation $E_1 \xrightarrow{l_1} \ldots \xrightarrow{l_i} E_{n+1}$ as $E_1 \xrightarrow{En(l_1)} \ldots \xrightarrow{En(l_n)} E_{n+1}$, where $En(l_i)$ the set of pairs $(\vartheta, K)$ such that:

- $\vartheta \in \mathcal{E}_i$;
- $K = \{(k, \vartheta') \mid k \in \{0, \ldots, i-1\}, \vartheta' \in \mathcal{E}_{i-k}, (\vartheta', \vartheta) < (a_1, \ldots, a_i)\}$.

Enabling trees highlight both local and communication causes of proof terms.

### 5.3 An application to circuit optimization

As discussed in [89, 90], the Esterel compiler generates circuits having much more latches than required.

Another possible method to implement Esterel in hardware is to compile statements in Finite State Machines, and to synthesize circuits from these FSMs by means of classical techniques. In this case, the minimum latch encoding ($\lceil \log_2 n \rceil$ latches for a $n$-states FSM) gives rise to a prohibitively large logic, and the one-hot encoding (a latch for each FSM state) requires too many latches. In [89, 90] it has been shown that circuits synthesized by the Esterel compiler compare favorably w.r.t. circuits synthesized by means of classical techniques as regards tradeoff between number of latches and size of the logic.

In this section we propose two techniques to remove redundant latches from circuits generated by the Esterel compiler. The first optimizes circuits synthesized from sequential statements (namely statements where construct “||” does not appear), by exploiting information in the LTS defined by rules in Table 3.1. The second technique optimizes circuits synthesized from arbitrary statements, by exploiting information in finite subtrees of locality trees. These two optimization techniques do not give rise to an increasing of the size of the circuit logic.

We recall that techniques to remove redundant latches from circuits generated by the Esterel compiler have been already proposed in [89, 90]. Techniques considered in these papers are based on the computation of (possibly over approximated) reachable state sets of circuits. In [89] relations over latches computed on the reachable state set of a circuit are exploited to express the value of some redundant latches as functions of the value of other latches. Redundant latches are then replaced by the logic implementing such functions. In [90] a notion of pair of exclusive sets of latches is given. Two sets of latches $R_1$ and $R_2$ are said to be exclusive if all latches in $R_2$ are unset when at least a latch in $R_1$ is set, and conversely. In this case, sets $R_1$ and $R_2$ are multiplexed through a single set, so that latches in one of the two sets are removed from the circuit, and both a multiplexer latch and the multiplexing logic are added.

Circuit transformations in [89, 90] do not exploit information over transitions between circuit states. As we take into account this information, we remove latches
which are not removed in [89, 90], and, conversely, as we do not consider relations
over latches as in [89, 90], there are latches that we do not remove but that are
removed in [89, 90].

Before presenting our optimization techniques, we introduce the notions of LTS
associated with a FSM. Given a set of input signals \( I \) and a set of output signals \( O \),
an LTS associated with a FSM is a LTS with finite states and with labels of the form
\((i,o)\), where \( i \) is an event over \( I \) and \( o \) is a subset of \( O \). LTS states represent FSM
states and LTS transitions represent FSM reactions. A transition labeled with \((i,o)\)
represents the reaction of the FSM to an environment that prompts each signal
\( s \) such that \( s^+ \in i \) and does not prompt any signal \( s \) such that \( s^- \in i \). During
this reaction, the FSM communicates signals in \( o \) to the environment. So, an LTS
associated with a FSM is the graphical representation of this FSM.

We may interpret Esterel statements as FSMs. To this purpose, we define a
sequential observation function \( Seq \) that, given a statement \( E \), relabels \([E]_{pts} \) as
\( Seq([E]_{pts}) \).

**Definition 5.3.1** The sequential observation function \( Seq \) relabels every transition
\( F \xrightarrow{\tau} F' \) in \([E]_{pts} \) as \( F \xrightarrow{\text{Seq}(l)} F' \), where \( Seq(l) = (S_t, Em(l) \setminus L_E) \).

The transition system \( Seq([E]_{pts}) \) can be viewed as a LTS associated with a FSM.
Every label in \( Seq([E]_{pts}) \) carries information only about the status of inputs and the
status of outputs. No information about signal causality is carried by \( Seq([E]_{pts}) \).
For this reason, one cannot define \( Seq([E]_{pts}) \) by induction on the syntax of \( E \).
Moreover, given statements \( E \) and \( E' \) such that \( Seq([E]_{pts}) \) and \( Seq([E']_{pts}) \) are
bisimilar, \( E \) and \( E' \) cannot be distinguished by the external environment, but, in
general, they can be distinguished by some Esterel context.

We begin with optimizing circuits synthesized from sequential statements.

Let us assume a set of latch names \( R = \{r_1, r_2, \ldots\} \) and let us denote with
\( \{r_{i_1}, \ldots, r_{i_n}\} \) a circuit state encoded by latches \( \{r_{i_1}, \ldots, r_{i_n}\} \).
Given a latch \( r \), we denote with \( in(r) \) the wire entering \( r \), and we denote with
\( out(r) \) the wire exiting from \( r \).

Given an arbitrary sequential statement \( E \), different states of \([E]\) are encoded
by different latches. This implies that, if \([E]\) contains sets of bisimilar states, then
redundant states in \([E]\) correspond to redundant latches in \( C_E \).

Given a sequential statement \( E \), let \( seq(E) \) be the circuit obtained by replacing
every set of latches encoding bisimilar states in \([E]\) by one latch only. Namely, given
latches \( r_1, \ldots, r_n \) encoding bisimilar states, we replace them by a new latch \( r \) such
that the logical orring of all wires \( in(r_i) \) is connected to \( in(r) \), and wire \( out(r) \) is
connected to some wire \( out(r_i), 1 \leq i \leq n \). Moreover, we recursively remove from
the circuit so obtained every and-gate having as non negated input either a removed
latch or a removed gate.
5.3. AN APPLICATION TO CIRCUIT OPTIMIZATION

Circuits $C_E$ and $\text{seq}(E)$ are equivalent, in the sense established by the following proposition.

**Proposition 5.3.2** Given a sequential statement $E$, circuits $C_E$ and $\text{seq}(E)$ cannot be distinguished by the external environment and by any operation on circuits available in the Esterel compiler.

**Proof** Let $Eq$ be the relation between states of $C_E$ and states of $\text{seq}(E)$ containing every pair $(\{r\}, \{r\})$ such that $r$ is not removed from $C_E$, and every pair $(\{r\}, \{r'\})$ such that $r$ has been replaced by $r'$.

Given a pair $(\{r\}, \{r\}) \in Eq$, if $r$ is set in $C_E$ and $r$ is set in $\text{seq}(E)$ then internal and output wires of $\text{seq}(E)$ stabilize as the corresponding wires of $C_E$. Moreover, if latches $r_1, \ldots, r_n$ in $C_E$ have been replaced by a new latch $\hat{r}$, then wire $\text{in}(\hat{r})$ is set in $\text{seq}(E)$ if and only if a wire $\text{in}(r_i)$ is set in $C_E$, for some $1 \leq i \leq n$. It follows that $C_E$ and $\text{seq}(E)$ enter states related by $Eq$.

Given a pair $(\{r\}, \{r'\}) \in Eq$, with $r'$ replacing $r$, wire $\text{out}(r')$ is connected to $\text{out}(r''$, where $r''$ is a latch such that $r$ and $r''$ encode bisimilar states. Therefore, $\text{out}(r)$ is connected to some gate in $C_E$ if and only if $\text{out}(r')$ is connected to a corresponding gate in $\text{seq}(E)$. This implies that, if $r$ and $r'$ are set, then internal and output wires of $\text{seq}(E)$ stabilize as the corresponding wires of $C_E$. Moreover, if latches $r_1, \ldots, r_n$ in $C_E$ have been replaced by a new latch $\hat{r}$, then wire $\text{in}(\hat{r})$ is set in $\text{seq}(E)$ if and only if a wire $\text{in}(r_i)$ is set in $C_E$, for some $1 \leq i \leq n$. It follows that $C_E$ and $\text{seq}(E)$ enter states related by $Eq$.

We have proved that, given a pair of states $(\{r\}, \{r\}) \in Eq$ with either $r' = r$ or $r'$ replacing $r$, $(\{r\} \text{ and } \{r\})$ cannot be distinguished by the environment, since $C_E$ and $\text{seq}(E)$ react to a given input by computing the same output. Moreover, since the logic computing this output is the same, states $(\{r\} \text{ and } \{r\})$ cannot be distinguished by any operation offered by the Esterel compiler. Finally, the thesis follows since we have proved that $C_E$ and $\text{seq}(E)$ evolve from states related by $Eq$ to states related by $Eq$ (i.e. $Eq$ is a bisimulation).

This completes the proof.

Prop. 5.3.2 implies that if a sequential statement $E$ is in the body of a statement $F$, then we can construct $C_F$ by taking either $\text{seq}(E)$ or $C_E$.

It is immediate that the number of latches in $\text{seq}(E)$ is less or equal than the number of latches in $C_E$. For each set of latches $\{r_1, \ldots, r_n\}$ replaced by one only, we add $n - 1$ binary or-gates to implement $\text{in}(r_1) \lor \ldots \lor \text{in}(r_n)$, and we remove gates implementing the “next state” and “output” functions of $n - 1$ latches. So, we do not pay in terms of the size of the logic.

Let us consider now an optimization technique based on locality trees.

Given a statement $E$ having $n$ occurrences of statement \texttt{pause} in its body, we mark such occurrences with $1, \ldots, n$. Then we modify the transition rule \texttt{pause} of
\[
\begin{align*}
l_{1,2} & = (\{\rho_1^+, \rho_2^+\}, \emptyset) & l_{1,3} & = (\{\rho_1^+, \rho_2^+\}, \emptyset) & l_{1,2} & = (\{\rho_1^+, \rho_2^+\}, \emptyset) \\
l_{1',2} & = (\{\rho_1^+, \rho_2^+\}, \emptyset) & l_{1',3} & = (\{\rho_1^+, \rho_2^+\}, \emptyset) & l_{1',2} & = (\{\rho_1^+, \rho_2^+\}, \emptyset) \\
l_{2,4} & = (\{\rho_2^-, \{ok_1\}\}) & l_{2,5} & = (\{\rho_2^-, \{ok_1\}\}) & l_{3,5} & = (\emptyset, \{ok_1\}) \\
l_{4,5} & = (\{\rho_2^-, e_1^-\}, \emptyset) & l_{4,5'} & = (\{\rho_2^-, e_1^-\}, \emptyset) & l_{4,5'} & = (\{\rho_2^-, e_1^-\}, \emptyset) \\
l_{5,2'} & = (\{e_1^+, \emptyset\}) & l_{5',2'} & = (\{e_1^+, \emptyset\}) & l_{5,2'} & = (\{e_1^+, \emptyset\}) \\
l_{6,7} & = (\emptyset, \{ok_2\}) & l_{6',7} & = (\emptyset, \{ok_2\}) & l_{6',7} & = (\emptyset, \{ok_2\}) \\
l_{7,8} & = (\emptyset, \{e_2^-\}) & l_{7',8} & = (\emptyset, \{e_2^-\}) & l_{7,8} & = (\emptyset, \{e_2^-\}) \\
l_{a_i,b_i} & = (\{ok_1^+, \{\beta_i\}\}) & l_{c_i,a_i} & = (\{\gamma_i^+, \{e_i\}\}) & l_{a_i,b_i} & = (\{\alpha_i^+, \{\beta_i\}\}) \\
\end{align*}
\]

Figure 5.2: The arbiter and the interface units.

Table 5.1 so that \(\langle \emptyset, \{e_{1i}\}, \emptyset, 1 \rangle\) is the label of the transition representing the reaction of the occurrence of \texttt{pause} marked with \(i\).

Locality trees carry information on how configurations of statements are encoded by latches in the circuit implementation. In fact, the initial configuration of a statement \(E\) is encoded by the boot latch, and, if \(F \xrightarrow{\vartheta} F'\) is a transition in \(\text{Loc}(\lbrack E \rbrack_\varnothing)\), then the configuration of \(E\) corresponding to state \(F'\) is encoded by the set of latches \(\{r \in R \mid (\vartheta r, K) \in l\ \text{for some } \vartheta\ \text{and } K\}\).

Locality trees carry information on wire connections in circuits implementing statements. In fact, a proof term \(\vartheta r\) corresponding to a statement \texttt{pause} reflects that every wire \(s\) such that \(s^+ \in \rbrack \vartheta\lbrack\) and the negation of every wire \(s\) such that \(s^- \in \lbrack \vartheta\rbrack\) are connected to \(in(r)\). Analogously, a proof term \(\vartheta z\) corresponding to a statement \texttt{emit} \(z\) reflects that every wire \(s\) such that \(s^+ \in \rbrack \vartheta\lbrack\) and the negation of every wire \(s\) such that \(s^- \in \lbrack \vartheta\rbrack\) are connected to wire \(z\).

Before formalizing our optimization technique based on locality trees, we give first an intuition.

Let us consider the FSM \(\mathcal{A}\) with LTS \(L_\mathcal{A}\) in Figure 5.2. This FSM specifies an arbiter of a resource \(R\) shared by users \(U_1\) and \(U_2\). FSM \(Int_i\) with LTS \(L_{Int_i}\) specifies a unit which interfaces \(U_i\) with \(\mathcal{A}\). In Figure 5.2 we denote with \(l_{i,j}\) the label of transition \(t_{i,j}\) from state \(i\) to state \(j\).
Unit $\text{Int}_i$ sends signal $\rho_i$ to request $R$ and signal $\epsilon_i$ to release it. Arbiter $\mathcal{A}$ sends signal $ok_i$ to assign $R$ to user $U_i$, $1 \leq i \leq 2$.

The arbiter $\mathcal{A}$ is initially in state 1 and is ready to serve requests from users. When in state 1, $\mathcal{A}$ gives higher priority to $U_1$. If both $\text{Int}_1$ and $\text{Int}_2$ request $R$ then $\mathcal{A}$ enters state 3, if only $\text{Int}_1$ requests $R$ then $\mathcal{A}$ enters state 2, and if only $\text{Int}_2$ requests $R$ then $\mathcal{A}$ enters state $2'$.

When in state 2, $\mathcal{A}$ assigns $R$ to $U_1$ and enters either state 5, if $U_2$ requests $R$, or state 4, otherwise.

When in state 3, $\mathcal{A}$ assigns $R$ to $U_1$ and enters state 5.

When in state 4, $\mathcal{A}$ waits until $U_1$ releases $R$ or $U_2$ requests it. If $U_1$ releases $R$ and $U_2$ requests it, then $\mathcal{A}$ enters state $2'$, so that $R$ can be assigned to $U_2$. If $U_1$ releases $R$ and $U_2$ does not request it, then $\mathcal{A}$ enters state $1'$. If $U_1$ does not release $R$ and $U_2$ requests it, then $\mathcal{A}$ enters state 5.

When in state 5, $\mathcal{A}$ waits until $U_1$ releases $R$ and then it enters state $2'$.

State $1'$ differs from state 1 since $U_2$ has higher priority. State $i'$ is the symmetric of $i$, $2 \leq i \leq 5$.

Unit $\text{Int}_i$ is initially in state $a_i$, where it waits for a request (represented by $\alpha_i$) by user $U_i$ to obtain $R$. When receiving $\alpha_i$, $\text{Int}_i$ sends $\rho_i$ to $\mathcal{A}$ and enters state $b_i$, where it waits until $\mathcal{A}$ sends $ok_i$. When receiving $ok_i$, $\text{Int}_i$ sends $\beta_i$ to $U_i$ to allow this user to use the resource $R$. Then it enters state $c_i$, where it waits until $U_i$ sends $\gamma_i$ to communicate that $R$ is free. When receiving $\gamma_i$, $\text{Int}_i$ sends $\epsilon_i$ to $\mathcal{A}$ and returns to state $a_i$.

We denote with $|$ the operator of parallel composition over FSMs. Let us consider FSM $\mathcal{A} \mid \text{Int}_1 \mid \text{Int}_2$ and let us assume that $\mathcal{A}$ is either in state 2 or in state 3. The output of $\mathcal{A}$ does not depend on which of 2 and 3 is active, since $\mathcal{A}$ emits signal $ok_1$ in both cases. Moreover, $\mathcal{A}$ enters state 4 if and only if $\text{Int}_2$ enters $a_2$, and $\mathcal{A}$ enters state 5 if and only if $\text{Int}_2$ enters $b_2$. Namely, the next state entered by $\mathcal{A}$ does not depend on which of state 2 and state 3 is active, but it is determined by the evolution of $\text{Int}_2$.

Let $\mathcal{A}'$ be the FSM obtained from $\mathcal{A}$ by replacing both states 2 and 3 by a new state 6 and by replacing transitions $t_{2,4}, t_{2,5}, t_{3,5}$ by new transitions $t_{6,4}, t_{6,5}$ labeled $(\{in(a_2)\}^+, \{ok_1\})$ and $(\{in(b_2)\}^+, \{ok_1\})$, respectively, where $in(a_2)$ denotes condition “$a_2$ is entering”, and analogously for $in(b_2)$. FSM $\mathcal{A}' \mid \text{Int}_1 \mid \text{Int}_2$ and $\mathcal{A} \mid \text{Int}_1 \mid \text{Int}_2$ are behaviorally equivalent, in the sense that their behavior cannot be distinguished by any observer. Intuitively, the reason is that states 3 and 2 distinguish whether $\text{Int}_2$ has requested $R$ or not. This information is redundant, since it can be recovered also from the state of $\text{Int}_2$.

Similar situations may happen, in general, when developing programs in a modular way.

Now, let us consider a sequential Esterel statement $E_A$ such that $Seq([E_A])$ is bisimilar to $L_\mathcal{A}$ and sequential statements $E_1, E_2$ such that $Seq([E_i])$ is isomorphic to $L_{\text{Int}_i}$, $1 \leq i \leq 2$. The system formed by the arbiter and both interface units is specified by statement $E \equiv \text{signal } ok_1, ok_2, \rho_1, \rho_2, \epsilon_1, \epsilon_2 \text{ in } E_A \mid E_1 \mid E_2 \text{ end.}$
We conjecture that one cannot give an Esterel statement \( E_A \) such that \( Seq([E_A]) \) is isomorphic to \( L_A \), since Esterel does not offer any construct equivalent to “\texttt{goto}”.

We can assume that circuit \( C_E \) is synthesized from \( C_{E_1}, C_{E_2} \) and \( seq(E_A) \). Circuit \( seq(E_A) \) has a latch for each state in \( L_A \). So, redundant information in \( A \) corresponds to redundant latches in \( seq(E_A) \) and, therefore, in \( C_E \). Let us denote with \( r_i \) the latch in \( seq(E_A) \) corresponding to state \( i \) in \( L_A \). Our idea is to replace latches \( r_2 \) and \( r_3 \) (identified as unifiable, see below) by a new latch \( r_6 \), to realize at circuit level the transformation described above at FSM level.

Now, in order to deduce that \( r_2 \) and \( r_3 \) can be replaced by a single latch, we must look at the evolution of the circuit. In fact, it is not sufficient to consider the reachable state set of the circuit while forgetting the possible transitions between states. This is the reason for which this optimization is not done by any transformation in \([89, 90]\).

To express that two latches may be replaced by one only, we introduce the concept of unifiability of latches. We need first some notations.

Given an event \( S \subseteq S^+ \), let us denote with \( Cond(S) \) the boolean expression \( \bigwedge_{s \in S} s \land \bigwedge_{s \in S} \neg s \) with variables ranging over the set of signals \( S \). This expression evaluates to true iff all variables corresponding to signals assumed to be present by \( S \) evaluate to true, and all variables corresponding to signals assumed to be absent by \( S \) evaluate to false.

Given a locality tree \( Loc([E]_{pl}) \), we denote with \( \mathcal{F}(Loc([E]_{pl})) \) the finite subtree obtained from \( Loc([E]_{pl}) \) by cutting each path \( E_1 \to E_2 \to \ldots E_n \to \ldots \) after node \( E_{n+1} \), provided that \( E_i \neq E_j \), \( 1 \leq i < j \leq n \), and \( E_{n+1} \equiv E_i \), for some \( 1 \leq i \leq n \).

**Definition 5.3.3** Given a circuit \( C_E \) implementing a statement \( E \), a latch \( r' \) is reachable from a latch \( r \) if and only if there exist transitions \( F \xrightarrow{t} F' \xrightarrow{t'} F'' \) in \( \mathcal{F}(Loc([E]_{pl})) \) such that a pair \((\partial r, K)\) is in \( l \), a pair \((\partial r', K')\) is in \( l' \) and a pair \((1, \partial r)\) is in \( K' \).

Given \( \partial r \) and \( \partial r' \) as in Def. 5.3.3, \( \partial r \) is a local cause of \( \partial r' \). In this case, when \( C_E \) is in the state corresponding to \( F' \), latch \( r \) is set. Moreover, wire \( out(r) \), every wire \( s \) such that \( s^+ \in |\phi| \) and the negation of every wire \( s \) such that \( s^- \in |\phi| \) are connected to \( in(r') \).

We denote with \( Reach(r) \) the set of latches reachable from \( r \).

Let us consider latches \( r \) and \( r' \) such that \( r' \in Reach(r) \) and ordered events \( \partial_1, \ldots, \partial_n \). Let us assume that for each transition \( F \xrightarrow{t} F' \) in \( \mathcal{F}(Loc([E]_{pl})) \) with a pair \((\partial r, K)\) in \( l \), there exist transitions \( F' \xrightarrow{t_i} F_i \) in \( \mathcal{F}(Loc([E]_{pl})) \) with \((\partial_i r', K_i)\) in \( l_i \), \( \partial_i \uparrow S^+ = \partial_i \) and \((1, \partial r)\) in \( K_i \), \( 1 \leq i \leq n \), there exists no transition \( F' \xrightarrow{t} F'' \) in \( \mathcal{F}(Loc([E]_{pl})) \) with a pair \((\partial r', K')\) in \( l' \) and \( \phi \uparrow S^+ \neq \partial_i \) for any \( 1 \leq i \leq n \), and there exists no transition \( F' \xrightarrow{t'} F'' \) in \( \mathcal{F}(Loc([E]_{pl})) \) with no pair \((\partial_i r', K')\) in \( l' \).
and no pair \((\phi, \mu, K_i')\) in \(l'\) with \(|\phi| \cap |\mu|\). In this case, we denote with \(\text{Trigger}(r, r')\) the condition \(\bigvee_{1 \leq i \leq n} \text{Cond}(|\mu_i|)\).

The meaning of \(\text{Trigger}(r, r')\) is that if latch \(r\) is set then \(\text{Trigger}(r, r')\) evaluates to true if and only if \(r'\) is set at next cycle.

If latches \(r\) and \(r'\) are such that \(r' \notin \text{Reach}(r)\) then we define \(\text{Trigger}(r, r')\) as the constant “false”.

**Definition 5.3.4** Given sets of latches \(R = \{r_1, \ldots, r_n\}\) and \(Q = \{q_1, \ldots, q_n\}\) in a circuit \(C_E\) such that \(R \cap Q = \emptyset\), latches \(r\) and \(r'\) such that \(r, r' \notin R \cup Q\) are unifiable w.r.t. \(R\) and \(Q\) if and only if the following conditions are satisfied:

1. \(\text{Trigger}(r, r) \lor \text{Trigger}(r, r') \iff \text{Trigger}(r', r) \lor \text{Trigger}(r', r')\);

2. \(R\) is the least set such that \(\text{Reach}(r), \text{Reach}(r') \subseteq R \cup \{r, r'\}\);

3. for each \(1 \leq i \leq n\), if there exists a transition \(F \xrightarrow{l} F'\) in \(\mathcal{F}(\text{Loc}([E]_{pl}))\) with \((\phi_r, K_i) \in l\) and either \((1, \phi) \in K_i\) or \((1, \phi') \in K_i\), then there exists a pair \((\psi_q, K_i) \in l'\);

4. for each signal \(s\), if there exists a transition \(F \xrightarrow{l'} F'\) in \(\mathcal{F}(\text{Loc}([E]_{pl}))\) such that \((\phi_s, K) \in l\) and either \((1, \phi) \in K\) or \((1, \phi') \in K\) and for each transition \(G \xrightarrow{l''} G'\) in \(\mathcal{F}(\text{Loc}([E]_{pl}))\) with \((\psi_r, K') \in l''\) (resp. \((\psi' r, K') \in l''\)), there exists a transition \(G' \xrightarrow{l'''} G''\) in \(\mathcal{F}(\text{Loc}([E]_{pl}))\) with \((\phi_s, K'') \in l'''\) and \((1, \psi r) \in K''\) (resp. \((1, \psi' r) \in K''\)), and there exists no transition \(G' \xrightarrow{l'''} G''\) with no pair \((\phi, K''') \in l'''\) and no pair \((\phi, K) \in l'\) with \(|\phi| \cap |\psi|\).

Let us consider Def. 5.3.4. The underlying idea is that of replacing latches \(r\) and \(r'\) in \(C_E\) by a new latch. Assume that either \(r\) or \(r'\) is set.

The next state of \(C_E\) does not depend on which of \(r\) and \(r'\) is set, but it is determined solely by the latch in \(Q\) that is set at next cycle and by the value of \(\text{Trigger}(r, r) \lor \text{Trigger}(r, r')\). In fact, if \(\text{Trigger}(r, r) \lor \text{Trigger}(r, r')\) evaluates to true then either \(r\) or \(r'\) is set at next cycle (see Cond. 1). If \(\text{Trigger}(r, r) \lor \text{Trigger}(r, r')\) evaluates to false then a latch in \(R\) is set at next cycle (see Cond. 2). In this case, a latch \(r_i \in R\) is set at next cycle if and only if \(\psi q_i \in Q\) is (see Cond. 3).

Also signals produced by \(C_E\) do not depend on which of \(r\) and \(r'\) is set (see Cond. 4).

Let us return to statement \(E = E_A \parallel E_1 \parallel E_2\). According to Def. 5.3.4, latches \(r_2\) and \(r_3\) are unifiable w.r.t. \(\{r_4, r_5\}\) and \(\{r_2, r_3\}\). Cond. 1 of Def. 5.3.4 is satisfied since \(\text{Trigger}(r_i, r_j) \iff \text{false}\) for \(i, j \in \{2, 3\}\). Cond. 2 is satisfied since \(\text{Reach}(r_2) = \{r_4, r_5\}\) and \(\text{Reach}(r_3) = \{r_5\}\). Cond. 3 is satisfied since for each label \(l\) in \(\mathcal{F}(\text{Loc}([E]_{pl}))\) with \((\phi r_4, K_4) \in l\), label \(l\) contains a pair \((\psi r_a, K_a)\), and for each label \(l\) in \(\mathcal{F}(\text{Loc}([E]_{pl}))\) with \((\phi r_5, K_5) \in l\), label \(l\) contains a pair \((\psi r_b, K_b)\). Cond. 4 is satisfied since for each transition \(G \xrightarrow{l} G'\) in \(\mathcal{F}(\text{Loc}([E]_{pl}))\).
with \((\partial r_2, K_2) \in \mathcal{L}'\) (resp. \((\partial r_3, K_3) \in \mathcal{L}'\)), every transition \(G' \xrightarrow{\mathcal{L}'} G''\) having \(G'\) as source state is such that \((\epsilon, K_1)\) is in \(\mathcal{L}'_2\).

Moreover, latches \(r_2'\) and \(r_3'\) are unifiable w.r.t. \(\{r_2, r_3\}\) and \(\{r_1, r_2\}\), latches \(r_2\) and \(r_3\) are unifiable w.r.t. \(\{r_2', r_3\}\) and \(\{r_1, r_2\}\), latches \(r_2'\) and \(r_3'\) are unifiable w.r.t. \(\{r_1, r_2\}\) and \(\{r_1, r_2\}\).

Let us denote with \(\text{not, or and and}\) the boolean operations over gates.

Given latches \(r, r', r_1, \ldots, r_n, q_1, \ldots, q_h\) as in Def. 5.3.4, let \(\text{Transf}(C_E)\) be the circuit obtained by applying to \(C_E\) the transformation \(\text{transf}\) consisting in the following steps:

1. remove both latches \(r\) and \(r'\) and add a new latch \(q\);

2. connect to \(\text{in}(q)\) the oring of \((\text{out}(q) \text{ and } (\text{Trigger}(r, r) \text{ or } \text{Trigger}(r, r'))))\), \(\text{in}(r)\) and \(\text{in}(r')\);

3. for each \(1 \leq i \leq n\) connect to \(\text{in}(r_i)\) the anding of \(\text{out}(q)\), \(\text{in}(q_i)\), and \(\text{not}(\text{Trigger}(r, r) \text{ or } \text{Trigger}(r, r'))\);

4. connect \(\text{out}(q)\) to every gate having in input either \(\text{out}(r)\) or \(\text{out}(r')\), and cut all wires from this gate to latches in \(R \cup \{r, r'\}\);

5. while there exists an and-gate with either a removed latch or a removed gate among its non negated inputs, remove such and-gate.

Transformation \(\text{transf}\) preserves the behavior of circuit \(C_E\), in the sense established by the following theorem.

**Theorem 5.3.5** Given a statement \(E\), circuits \(C_E\) and \(\text{Transf}(C_E)\) cannot be distinguished by the external environment and by any operation on circuits available in the Estrel compiler.

**Proof** Let \(E_q\) be the relation between states of \(C_E\) and states of \(\text{Transf}(C_E)\) such that \((S, S) \in E_q\) if \(r, r' \notin S\) and \((S, S \setminus \{r, r'\} \cup \{q\}) \in E_q\) otherwise.

Let us assume that both \(C_E\) and \(\text{Transf}(C_E)\) are in a state \(S\), where \(S \cap \{r, r'\} = \emptyset\). Given a value at input wires, all internal and output wires of \(C_E\) stabilize exactly as the corresponding wires of \(\text{Transf}(C_E)\). Moreover, either \(\text{in}(r)\) or \(\text{in}(r')\) is set in \(C_E\) if and only if \(\text{in}(q)\) is set in \(\text{Transf}(C_E)\) (see step 2 of \(\text{transf}\)). So, \(C_E\) and \(\text{Transf}(C_E)\) enter states related by \(E_q\).

Let us suppose that \(C_E\) is in a state \(S\) such that \(S \cap \{r, r'\} \neq \emptyset\) and \(\text{Transf}(C_E)\) is in state \(S \setminus \{r, r'\} \cup \{q\}\). Given a value at input wires, all internal and output wires of \(C_E\) that are not connected to latches in \(R \cup \{r, r'\}\) stabilize exactly as the corresponding wires in \(\text{Transf}(C_E)\) (see step 4). A latch \(r'' \notin R \cup \{r, r'\}\) is set in \(C_E\) if and only if \(r''\) is set in \(\text{Transf}(C_E)\). Either \(\text{in}(r)\) or \(\text{in}(r')\) is set in \(C_E\) if and only if \(\text{in}(q)\) is set in \(\text{Transf}(C_E)\) (see step 2). Finally, \(\text{in}(r_i)\) is set in \(C_E\) if and only if \(\text{in}(r_i)\) is set in \(\text{Transf}(C_E)\) (see step 3). Therefore, \(C_E\) and \(\text{Transf}(C_E)\)
enter states related by $E$.  
We have proved that, given a pair of states $(S, S') \in E$, $S$ and $S'$ cannot be distinguished by the environment, because $C_E$ and $\text{Transf}(C_E)$ react to a given input by computing the same output. Moreover, since the logic computing this output is the same, states $S$ and $S'$ cannot be distinguished by any operation offered by the Esterel compiler. Finally, the thesis follows since we have proved that $C_E$ and $\text{Transf}(C_E)$ evolve from states related by $E$ to states related by $E$ (i.e. $E$ is a bisimulation).

This completes the proof.

Transformation $\text{transf}$ optimizes circuits by replacing two latches by one only. In most cases, we do not pay in terms of size of the logic. In fact, in step 2 we add a constant number of gates, and in step 3 we add an or-gate implementing $\text{Trigger}(r, r)$ or $\text{Trigger}(r, r')$, an and gate implementing $\neg(\text{Trigger}(r, r))$ or $\text{Trigger}(r, r')$ and $\text{out}(q)$, and a binary and-gate to anding this with $\text{in}(q_i)$, for every $1 \leq i \leq n$. (note that the logic implementing both $\text{Trigger}(r, r)$ and $\text{Trigger}(r, r')$ is already available on $C_E$). So, we require $n + c$ binary gates, where $c$ is a constant. Now, in step 5 we remove gates implementing the “next state” function of both $r$ and $r'$. All these gates connect $r$ and $r'$ to latches in $R$. So, their number is linear in $n$. Moreover, we remove the logic implementing $\text{Trigger}(r', r)$ and the logic implementing $\text{Trigger}(r', r')$.

To understand what happens if we apply iteratively transformation $\text{transf}$, we have the following propositions.

**Proposition 5.3.6** Let us assume a circuit $C_E$, latches $r, r', r''$ pairwise unifiable w.r.t. $R$ and $Q$, and $r''$ the latch replacing $r$ and $r'$ in $\text{transf}$. Let us replace every occurrence of $r$ and $r'$ in $\mathcal{F}(\text{Loc}(\lfloor E \rfloor))$ by $r''$. Latches $r''$ and $r'''$ are unifiable w.r.t. $R$ and $Q$.

**Proof** Since the pairs $(r, r')$ and $(r, r'')$ are unifiable w.r.t. $R$ and $Q$, by Cond. 2 of Def. 5.3.4 it follows that $\text{Reach}(r) \subseteq R \cup \{r, r'\}$ and $\text{Reach}(r) \subseteq R \cup \{r, r''\}$. This implies that $r', r'' \notin \text{Reach}(r)$ and, therefore, $\text{Trigger}(r, r') \iff \text{false} \iff \text{Trigger}(r, r'')$. Analogously, one can prove that $\text{Trigger}(r', r), \text{Trigger}(r', r'')$, $\text{Trigger}(r'', r)$, $\text{Trigger}(r'', r')$ are equivalent to false. As a consequence, by Cond. 1 of Def. 5.3.4, it follows that $\text{Trigger}(r, r) \iff \text{Trigger}(r', r') \iff \text{Trigger}(r'', r'')$.

Latches $r''$ and $r'''$ satisfy Cond. 1 of Def. 5.3.4. In fact:

$\text{Trigger}(r'', r'') \lor \text{Trigger}(r'', r''') \iff \text{Trigger}(r'', r'')$,

$\text{Trigger}(r'', r''') \lor \text{Trigger}(r', r'') \iff \text{Trigger}(r', r') \lor \text{Trigger}(r', r')$ and

$\text{Trigger}(r, r) \iff \text{Trigger}(r', r') \iff \text{Trigger}(r'', r''')$.

Latches $r''$ and $r'''$ satisfy Cond. 2 of Def. 5.3.4, since $\text{Reach}(r'') \subseteq \{r''\} \cup R$ and $\text{Reach}(r''') = \text{Reach}(r) \cup \text{Reach}(r') \setminus \{r, r'\} \cup \{r''\} \subseteq R \cup \{r''\}$ by the hypothesis and what we have already proved.

Latches $r''$ and $r'''$ satisfy Cond. 3 and 4 of Def. 5.3.4 immediately.
This completes the proof.

**Proposition 5.3.7** Let us assume a circuit $C_E$, latches $r$ and $r'$ unifiable w.r.t. $R$ and $Q$, and latches $r''$ and $r'''$ unifiable w.r.t. $R'$ and $Q'$. Let $\hat{r}$ be the latch replacing $r$ and $r'$ in $\text{transf}$. Let us replace all occurrences of $r$ and $r'$ in $F(\text{Loc}([E]_{r\hat{r}}))$ by $\hat{r}$. If $\{r, r'\} \not\subset R'$ and $r, r' \not\subset Q$ then $r''$ and $r'''$ remain unifiable after the replacement of $r$ and $r'$ by $\hat{r}$.

**Proof** It is immediate to prove that if $\{r, r'\} \cap R' = \emptyset$ then $r''$ and $r'''$ remain unifiable w.r.t. $R'$ and $Q'$.

Let us consider the case with $r \in R' = \{r_1, \ldots, r_n\}$, namely $r = r_h$, for some $1 \leq h \leq n$. We prove that $r''$ and $r'''$ are unifiable w.r.t. $R' \setminus \{r\} \cup \{\hat{r}\}$ and $Q'$.

Latches $r''$ and $r'''$ satisfy Cond. 1, 3 and 4 of Def. 5.3.4 immediately. Moreover, $r''$ and $r'''$ are such that $\text{Reach}(r'') \subseteq R \setminus \{r\} \cup \{\hat{r}\}$. So, latches $r''$ and $r'''$ satisfy also Cond. 2 of Def. 5.3.4 and the thesis follows.

This completes the proof.

Let us consider again the statement $E = E_A \parallel E_1 \parallel E_2$. By Prop. 5.3.7 it follows that we can replace pairs of unifiable latches $\langle r_2, r_3 \rangle$, $\langle r_2, r_3 \rangle$, $\langle r_4, r_5 \rangle$, $\langle r_4, r_5 \rangle$ by single latches, provided that the replacement of $\langle r_2, r_3 \rangle$ (resp. $\langle r_4, r_5 \rangle$) is done before the replacement of $\langle r_4, r_5 \rangle$ (resp. $\langle r_4, r_5 \rangle$).

Now, let us assume the complete system formed by the arbiter $A$, the interface units $\text{Int}_1$, $\text{Int}_2$, and the users $U_1$ and $U_2$. Let $E_A$, $E_1$, $E_2$, $E_{U_1}$ and $E_{U_2}$ the corresponding Esterel statements. It is reasonable that all latches in the circuit implementing $E_i$ can be replaced by functions of latches in $C_{E_{U_i}}$. The reason is that $\text{Int}_i$ is completely controlled by $U_i$. So, we must optimize $\text{seq}(E_A)$, $E_{U_1}$ and $E_{U_2}$. Circuit $\text{seq}(E_A)$, which has ten latches, is transformed by $\text{transf}$ into a circuit with six latches. Both $\text{seq}(E_A)$ and $\text{transf}(\text{seq}(E_A))$ can be optimized by iteratively multiplexing pairs of exclusive sets of latches through a single set, so that one obtains circuits with four and three latches, respectively. This shows that we can improve the optimization phase by means of $\text{transf}$.

Now, optimization strategies in [89, 90] apply iteratively transformations taking care that the size of the added logic remains within a chosen bound. Since our techniques do not give rise to an increasing of the size of the circuit logic, they seem well-suited to be integrated with strategies in [89, 90].

Algorithms in [89, 90] are very efficient as they do not require explicit constructions of LTKs representing FSMs, so that the state explosion problem is avoided. The transformation of $C_E$ into $\text{seq}(E)$ requires the construction of $[E]$, but $E$ is sequential and there is no state explosion. Transformation $\text{transf}$ requires the construction of finite subtrees of proved trees. In order to tackle the state explosion problem, we can optimize circuits by working compositionally w.r.t. the syntactic structure of statements.
5.4. AN APPLICATION TO VERIFICATION

![Diagram of a ring of 3 cells]

Figure 5.3: The ring of 3 cells.

5.4 An application to verification

In this section we show how information in enabling trees can be exploited to simplify debugging when the verification technique of observer monitoring is adopted.

As a running example, we consider an Esterel program specifying a bus arbiter. This program has been given originally in [21].

Let us consider the following informal specification. A bus is shared by users $U_1, \ldots, U_n$. User $U_i$ may request to access the bus by sending a request signal $r_i$. In this case, $U_i$ is allowed to access the bus only if it receives an acknowledgment signal $a_i$. At most one requested access must be granted at each execution cycle.

The arbiter is implemented as a ring of $n$ cells, $C_1, \ldots, C_n$.

Each user is connected to a cell which receives requests through input signal $\text{Req}_{in}$ and acknowledges requests through output signal $\text{Ack}_{out}$. A token ring is introduced to ensure weak fairness. The $i^{th}$ cell receives the input token through signal $T_{in}$ at $j^{th}$ cycle, for $j$ such that $j \mod n = i$, and sends the output token through signal $T_{out}$ at subsequent cycle. So, a latch is introduced in each cell to “store” the token for one cycle. When receiving the token $T_{in}$, a cell acknowledges a request $\text{Req}_{in}$, since it has higher priority w.r.t. other cells. When receiving the token $T_{in}$, if a cell does not receive the request $\text{Req}_{in}$ then it passes a grant around the ring through output signal $G_{out}$. A cell acknowledges a request or passes the grant also when it receives the grant, namely when it receives the input signal $G_{in}$. So, the grant allows cells to acknowledge requests without having the token. Introducing the grant is needed to avoid that, at some cycle, there are requests from users and none of them is satisfied.

The Esterel module $\text{Cell}$ in Figure 5.4 implements a cell, while the Esterel module $\text{ring}_3$ in Figure 5.5 implements a ring of three cells. In Figure 5.3 we represent three users and the ring of three cells.

Now, let us consider the following safety property $\phi$: “if $U_3$ requests the bus at two subsequent cycles, then $U_2$ cannot obtain the bus at both cycles”. In order to check whether $\text{ring}_3$ satisfies $\phi$, we must translate $\phi$ into an Esterel statement $\Omega_\phi$
module Cell:
input: Req_in, T_in, G_in;
output: Ack_out, T_out, G_out;
loop
  present T_in then
    pause;
    emit T_out
  else
    pause
  end
end

||

loop
  (present G_in then present Req_in then emit Ack_out
   else emit G_out
   end
    else nothing
  end
||

present T_in then present Req_in then emit Ack_out
  else emit G_out
  end
  else nothing
end
); pause
end
end module

Figure 5.4: The module Cell.
module ring_3:
input: $r_1, r_2, r_3$;
output: $a_1, a_2, a_3$;
signal $t_1, g_1, t_2, g_2, t_3, g_3$ in
emit $t_1$

\[
(Cell[r_1/\text{Req.in}, t_1/T.in, g_1/G.in, a_1/Ack.out, t_2/T.out, g_2/G.out]
\]

\[
(Cell[r_2/\text{Req.in}, t_2/T.in, g_2/G.in, a_2/Ack.out, t_3/T.out, g_3/G.out]
\]

\[
(Cell[r_3/\text{Req.in}, t_3/T.in, g_3/G.in, a_3/Ack.out, t_1/T.out, g_1/G.out])
\]
end
end module

Figure 5.5: The module ring_3.

which runs in parallel with ring_3, observes its behavior, and emits an alarm signal $\alpha$ when it detects that ring_3 has violated $\phi$. The statement $\Omega_\phi$ is in Figure 5.6.

Let us denote with $E$ the statement $ring_3 \parallel \Omega_\phi$. To see that ring_3 violates $\phi$ it is sufficient to consider the computation $E \xrightarrow{\lambda} b_3$ of $Seq([E])$, where $l_1 = ([r_1^-, r_2^+, r_3^+], \{a_2, \gamma\})$ and $l_2 = ([r_2^+, r_3^+], \{a_2, \gamma, \alpha\})$. From this computation, one deduces that $\alpha$ is produced (namely, $\phi$ is violated) if both $r_2$ and $r_3$ are communicated (namely, both $U_2$ and $U_3$ request the bus) at the first two cycles.

Now, let us consider the computation $E \xrightarrow{\lambda} l'$ of $[E]$ such that $Seq(l) = l_1$ and $Seq([E]) = l_2$, where $Seq$ is the sequential observation function defined in Def. 5.3.1. Let us denote by $c_1$ the string $\|0\|1\|0$, by $c_2$ the string $\|0\|1\|1\|0$, and by $c_3$ the string $\|0\|1\|1\|1$. Proof terms in $E_i$ and $E_r$ of the form $c_3\theta\mu$ refer to actions performed by statements in the body of the statement implementing cell $C_i$, $1 \leq i \leq 3$. Analogously, we denote by $\Omega$ the string $\|1\$, so that proof terms of the form $\Omega\theta\mu$ in $E_i$ and $E_r$ refer to actions performed by statements in the body of $\Omega_\phi$.

Component $E_i$ contains, among the others, the following proof terms:

- $\|0\|0t_1$ corresponding to the production of $t_1$ by statement emit $t_1$ in the body of ring_3;

- $c_1 \|1\|0 \|1\|1t_1^r g_2$ corresponding to the communication of the output grant by cell $C_1$;

- $c_2 \|1\|0 \|0\|1g_2^r a_2$ corresponding to the communication of the acknowledgment by cell $C_2$;

- $\Omega \|0\|a_3^+ a_2^+ \gamma$ corresponding to the production of $\gamma$ by $\Omega_\phi$. 
loop
  present \( r_3 \) then present \( a_2 \) then emit \( \gamma \)
    else nothing
    else nothing
  end;
  pause
end

||

loop
  present \( \gamma \) then
    pause;
    present \( \gamma \) then emit \( \alpha \); loop pause; emit \( \alpha \) end

  end
  else
    nothing
  end;
  pause
end

Figure 5.6: The statement \( \Omega_{\phi} \).
Component $\mathcal{E}_F$ contains, among the others, the following proof terms:

- $c_1||_{0;1} t_2$ corresponding to the production of the output token by cell $C_1$;
- $c_2 ||_{1;10} l_{1;0} ||_{1} t_2^+ r_2^+ a_2$ corresponding to the production of the acknowledgment by cell $C_2$;
- $\Omega ||_{0} l_{1;0} r_3^+ a_2^+ \gamma$ corresponding the production of $\gamma$ by $\Omega_\phi$;
- $\Omega ||_{1;1} \gamma^+ a_1$ corresponding to the production of the alarm by $\Omega_\phi$.

Let us consider the enabling tree $En([E]_{\phi\alpha})$ and the computation $E \xrightarrow{\operatorname{En}(l)} \xrightarrow{\operatorname{En}(l')}$.

The label $En(l')$ contains, among the others, the following pairs:

- $(\Omega ||_{1;1} \gamma^+ a_1, K_1)$, with $K_1$ containing both pairs $(0, \Omega ||_{0} l_{1;0} r_3^+ a_2^+ \gamma)$ and $(1, \Omega ||_{0} l_{1;0} r_3^+ a_2^+ \gamma)$ which reflect that both the production of $\gamma$ at the first cycle and the production of $\gamma$ at the second cycle are causes of the production of the alarm at the second cycle;
- $(\Omega ||_{0} l_{1;0} r_3^+ a_2^+ \gamma, K_2)$, with $K_2$ containing $(0, c_2 ||_{1} l_{1;0} ||_{1} t_2^+ r_2^+ a_2)$ which reflects that at the second cycle the production of the acknowledgment by cell $C_2$ is a cause of the production of $\gamma$ by $\Omega_\phi$;
- $(c_2 ||_{1} l_{1;0} ||_{1} t_2^+ r_2^+ a_2, K_3)$, with $K_3$ containing $(0, c_1 ||_{0;1} t_2)$ which reflects that at the second cycle the production of the output token by cell $C_1$ is a cause of the production of the acknowledgment by cell $C_2$.

The label $En(l)$ contains, among the others, the following pairs:

- $(\Omega ||_{0} r_3^+ a_2^+ \gamma, K_4)$, with $K_4$ containing $(0, c_2 ||_{1;0} ||_{0} g_2^+ r_2^+ a_2)$ which reflects that at the first cycle the production of the acknowledgment by cell $C_2$ is a cause of the production of $\gamma$ by $\Omega_\phi$;
- $(c_2 ||_{1;0} ||_{0} g_2^+ r_2^+ a_2, K_5)$ with $K_5$ containing pair $(0, c_1 ||_{1;0} ||_{1} t_2^+ r_1^+ g_2)$ which reflects that at the first cycle the production of the output grant by cell $C_1$ is a cause of the production of the acknowledgment by cell $C_2$.

Now, from the computation $E \xrightarrow{\operatorname{En}(l)} \xrightarrow{\operatorname{En}(l')}$, we can deduce the reason for which $\text{ring}_3$ violates $\phi$. In fact, we deduce that cell $C_2$ can acknowledge the request of user $U_2$ at the first cycle because it receives the grant by cell $C_1$, and that $C_2$ can acknowledge the request of user $U_2$ at the second cycle because it receives the token by cell $C_1$. This information cannot be inferred from computation $E \xrightarrow{l_1} \xrightarrow{l_2}$ of $Sec([E])$, since $E$ is perceived as a sequential statement and interactions among the various components of $\text{ring}_3$ are abstracted.

In general, given a program $E$ and a property $\phi$ such that $E$ violates $\phi$, existing tools generate a counterexample computation $E \parallel \Omega_\phi \xrightarrow{l_1} \ldots \xrightarrow{l_n}$ of $Sec([E \parallel \Omega_\phi])$, with $\alpha$ appearing in $l_n$. 

To supply a user with information about the reason for which $E$ violates $\phi$, it is sufficient to construct the portion of $\mathcal{F}(\text{Loc}(\mathcal{E} \parallel \Omega_{\phi}[\mu]))$ containing the computation $E \parallel \Omega_{\phi} \xrightarrow{\ell_{1}} \ldots \xrightarrow{\ell_{n}}$, where $\text{Seq}(\ell'_{i}) = l_{i}$, $1 \leq i \leq n$, and then to relabel such computation as $E \parallel \Omega_{\phi} \xrightarrow{E_{n}(\ell'_{i})} \ldots \xrightarrow{E_{n}(\ell'_{n})}$. So, the efficiency of the tool is not significantly degraded.
Chapter 6

SOS and causal semantics for tdccp

In this chapter we give both a structural operational semantics in terms of a labeled transition system and a causal semantics in terms of causal trees for the paradigm tdccp.

We show the correspondence between our LTS semantics and the operational semantics of tdccp given in [86]. We mean that we prove that information in our labeled transition system permits to establish whether tdccp programs are reactive and deterministic and to deduce their input/output behavior.

We prove that bisimulation on tdccp programs is a congruence. This proof is immediate, because we consider a transition system specification consisting of a set of rules in de Simone format and of a recursion rule, so tdccp is a de Simone language.

Our causal semantics highlights causes of outputs produced by programs during reactions. Outputs may be caused by inputs produced by the environment and by actions performed by the programs themselves. Technically, we give a semantics, in SOS style, in terms of a proved transition system carrying all information that is needed to infer causality, and then we map proved trees, obtained by unfolding parts of this PTS, to causal trees. We demonstrate with an example how causal trees can be exploited to improve the verification phase when the observer monitoring verification technique is adopted.

A causal semantics has been given in [42] for the asynchronous paradigm Concurrent Constraint Programming (ccp). We show the correspondence between this causal semantics and our causal semantics. Namely, we prove that the two causal semantics coincide for programs belonging to both ccp and tdccp.

This chapter is organized as follows. In Section 6.1 we recall tdccp and the original operational semantics of [86]. In Section 6.2 we present our SOS semantics. In Section 6.3 we show the correspondence between this SOS semantics and the semantics of [86]. In Section 6.4 we present our causal semantics and we prove the correspondence between this causal semantics and the causal semantics of [42].
Finally, in Section 6.5 we deal with local variables, which have been introduced in [87] to extend the language of [86].

6.1 An overview of tdccp

In this section we recall the paradigm tdccp. For a complete treatment, we refer to [84, 85, 86, 87].

The paradigm tdccp extends the paradigm Concurrent Constraint Programming (ccp) [83] with constructs to deal with defaults for negative information and with discrete time. ccp assumes a notion of computation as deduction over systems of partial information. Information accumulates monotonically in a distributed context: a multiset of agents cooperate to produce constraints on shared variables. A set of primitive constraints (tokens) are assumed to specify possibly partial information on the values of a set of variables. Tokens are equipped with an entailment relation \( \vdash \) and a conjunction operation \( \wedge \): \( a \vdash b \) holds exactly if the information given by \( b \) follows from the information given by \( a \); \( a \wedge b \) gives information given by both \( a \) and \( b \).

A ccp program consists of a multiset of agents which cooperate by posting tokens in a shared store and by querying the store about validity of tokens. So, the agent “tell \( a \)” posts the token \( a \) in the store and the agent “if \( a \) then \( A \)” behaves as the agent \( A \) when the store entails \( a \), namely when the store contains tokens entailing \( a \). CCP cooperating agents are not synchronized, in the sense that each agent proceeds at its own rate. Construct if _ then _ can be used to achieve synchronization. In fact, the agent if \( a \) then \( A \) waits for token \( a \) and then it behaves as \( A \).

On the contrary, tdccp cooperating agents are perfectly synchronized, in the sense that they share the same clock. tdccp assumes a discrete notion of time and offers a new construct, next, to sequentialize interaction between agents and the store: the agent “next \( A \)” will behave as \( A \) at the next instant. According to the synchronous hypothesis, agents post tokens in the store and query the store without consuming time, while next consumes exactly one unit of time. In practice, a tdccp program has a discrete behavior: at each instant it reacts to an input, namely to a set of tokens posted in the store by the external environment. The reaction implies an accumulation of tokens in the store, so that the resulting store is interpreted as the response of the program. The store is refreshed between any instant and the subsequent one. Finally, tdccp agents may query the store also about the nonvalidity of tokens: the agent “if \( a \) else \( A \)” behaves as \( A \) if the store does not entail \( a \) at the current instant.

A constraint system is a system of partial information, consisting of a set \( \mathcal{D} \) of tokens (first-order formulas over an infinite set of variables \( \text{Var} \)), closed under conjunction, and an inference relation (logical entailment) \( \vdash \) relating tokens to tokens. The set \( \mathcal{D} \) is ranged over by \( a, b, \ldots \). The entailment relation induces, through
symmetric closure, a logical equivalence relation which is denoted with $\sim$. Formally, $a \sim b$ if and only if $a \vdash b$ and $b \vdash a$.

**Definition 6.1.1** A constraint system is a structure $\langle \mathcal{D}, \wedge, \vdash, \text{Var} \rangle$ such that:

- $\mathcal{D}$ is a set of first order formulas over $\text{Var}$;
- $\wedge : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$ is a total function;
- $\vdash \subseteq \mathcal{D} \times \mathcal{D}$ is a decidable relation;
- $\wedge$ and $\vdash$ satisfy the following requirements:
  - $a \vdash a$;
  - $a \vdash a'$ and $a' \wedge a'' \vdash b$ implies that $a \wedge a'' \vdash b$;
  - $a \vdash b$ and $a \wedge b \vdash b$;
  - $a \vdash b_1$ and $a \vdash b_2$ implies $a \vdash b_1 \wedge b_2$;
  - there exist tokens $\text{true, false} \in \mathcal{D}$ such that $a \vdash \text{true}$ and $\text{false} \vdash a$ for every $a \in \mathcal{D}$.

Sets of tokens closed under $\vdash$ are called constraints.

Token $\text{true}$ can be interpreted as the conjunction of tokens $a_1, \ldots, a_n$, where $n = 0$. Token $\text{false}$ can be interpreted as the conjunction of all tokens in $\mathcal{D}$.

**Example 6.1.2** The constraint system Gentzen [85] provides the very simple level of functionality that is needed to represent binary signals. The primitive tokens are atomic proposition $X, Y, Z, \ldots$, where $X, Y, Z$ range over variables. Every variable $X$ is associated with a binary signal, and the token $X$ represents the presence of this signal. The entailment relation is such that $a_1 \wedge \ldots \wedge a_n \vdash a$ if and only if $a = a_i$ for some $1 \leq i \leq n$. The constraint system Gentzen has been used in [94] to show that tdccp embeds compositionally both the data flow synchronous language Lustre and the state oriented synchronous language Argos which deal with binary signals.

We will assume to work within a given constraint system $\langle \mathcal{D}, \wedge, \vdash, \text{Var} \rangle$. We assume the following syntax for $\text{tdccp}$:

$$A ::= \text{skip} \mid \text{tell } a \mid \text{if } a \text{ then } A \mid \text{if } a \text{ else } A \mid \text{next } A \mid [A,A] \mid \text{rec } P,A \mid P$$

where $a, A$ and $P$ range over tokens, agents and recursion variables, respectively. We will identify agents and programs.

Informally, the agent $\text{skip}$ does nothing at each instant. Agent $\text{tell } a$ posts the token $a$ in the store and then terminates, namely it will behave as $\text{skip}$. Both agents $\text{if } a \text{ then } A$ and $\text{if } a \text{ else } A$ query the store about the validity of token $a$. If the store entails $a$ then the first agent behaves as $A$, else it terminates. Symmetrically, if the store entails $a$ then the second agent terminates, else it behaves as $A$. The agent
next \( A \) will behave as \( A \) at the next instant. The agent \([A_1, A_2]\) is the synchronous parallel composition of \( A_1 \) and \( A_2 \). Construct \texttt{rec} is the classical recursion construct.

We say that an occurrence of a recursion variable \( P \) is \textit{free} in \( A \) if it does not appear in the body of a \texttt{rec} \( P \). A variable \( P \) is free in \( A \) if there exists an occurrence of \( P \) free in \( A \). A variable \( P \) is \textit{guarded} in \( A \) if every free occurrence of \( P \) in \( A \) appears in the body of a \texttt{next} \( P \). It is required that recursion is guarded in order to be sure that at each instant the computation is finite. This is analogous to requiring the body of the Esterel statement \texttt{loop \_end} not to terminate immediately.

While Esterel statements communicate with their environment through two distinct sets of signals, one for inputs and one for outputs, \texttt{tdccp} agents and their environment communicate by producing constraints on a shared set of variables, and there is no distinction between input variables and output variables. As a consequence, given an agent \([A_1, A_2]\), tokens produced by \( A_1 \) are immediately available to \( A_2 \), and conversely. We do not need local variables to have communication between \texttt{tdccp} agents running in parallel.

Instantaneous communication may give rise to paradoxes of causality which originate nondeterminism or nonreactivity.

As an example, the agent \texttt{if a else tell a} is nonreactive if the store does not entail \( a \). In fact, the token \( a \) is posted in the store if and only if it is nonentailed by the store at the same instant, so no reaction is admissible. The agent \texttt{[if a else tell b, if b else tell a]} is nondeterministic if the store entails neither \( a \) nor \( b \). In fact, either the left branch produces \( b \) and the right branch terminates without producing \( a \), or the right branch produces \( a \) and the left branch terminates without producing \( b \).

Agents where paradoxes of causality may appear are rejected. Only deterministic and reactive agents are admitted.

In \texttt{tdccp} no notion of constructiveness is considered. In fact, tokens are assumed to be nonentailed by default. So, while the Esterel statement \texttt{signal a in (if a then emit a) end} is nonconstructive, the \texttt{tdccp} agent \( A \equiv \texttt{if a then tell a} \) has a well defined behavior: if the store entails \( a \) then \( A \) posts \( a \) in the store, otherwise it terminates because \( a \) is assumed to be nonentailed by default.

In [86, 87, 85] different solutions to define cyclic behaviors have been proposed. Construct \texttt{rec} (denoted with \( \mu \)) is used in [86].

In [87] the construct \texttt{hence} has been introduced: the agent \texttt{hence A} executes \( A \) at each instant, starting from the next one. Intuitively, \texttt{hence A} is equivalent to \texttt{next (rec P, [A, next P])}.

In [85] one may have recursive definitions of the form:

\[
p_1 = A_1 \\
\vdots \\
p_m = A_m,
\]

where calls of procedures \( p_1, \ldots, p_m \) may appear in \( A_1, \ldots, A_m \). In this case, these calls of procedures must be in the scope of a \texttt{next}, namely recursion is guarded. To
\[\sigma(\Gamma) \vdash a\]

\[\Gamma, \text{if } a \text{ then } A, \Delta \xrightarrow{b} ((\Gamma, A), \Delta)\]

\[\Gamma, \text{if } a \text{ else } A, \Delta \xrightarrow{b} ((\Gamma, A), \Delta)\]

\[\Gamma, \text{rec } P. A, \Delta \xrightarrow{b} ((\Gamma, A[\text{rec } P. A/P]), \Delta)\]

\[\Gamma, \text{next } A, \Delta \xrightarrow{b} (\Gamma, (A, \Delta))\]

Table 6.1: The relation \(\xrightarrow{b}\).

Ensure that at run-time there are only boundedly many different procedure calls, it is required that procedures have no parameters. Another possible choice is that procedure calls take exactly the same parameters as the procedure definitions. The two proposals are equivalent.

We briefly recall now the operational semantics of [86].

We begin with describing the behavior of agents at an instant, in terms of transitions between configurations. A configuration is a pair of multisets of agents: the agents currently active and the agents that will be executed at the next instant. Multisets of agents are ranged over by \(\Gamma, \Delta, \ldots\). In practice, construct \([\_, 
_]\) is assumed to be commutative and associative, and a multiset \(\Gamma = A_1, \ldots, A_n\) denotes the parallel composition of \(A_1, \ldots, A_n\). Moreover, \textit{skip} denotes the parallel composition of agents \(A_1, \ldots, A_n\), where \(n = 0\).

For any multiset of agents \(\Gamma\), let \(\sigma(\Gamma)\) denote the token \(\bigwedge \{a | \text{tell } a \in \Gamma\}\).

We define binary transition relations \(\xrightarrow{b}\), indexed by final guess \(b\), on configurations. Relation \(\xrightarrow{b}\) is defined in Table 6.1.

Given configurations \((\Gamma, \Delta)\) and \((\Gamma', \Delta')\) such that \((\Gamma, \Delta) \xrightarrow{b} (\Gamma', \Delta')\), \((\Gamma, \Delta)\) and \((\Gamma', \Delta')\) are activated at the same instant, namely during a reaction. The rule for \textit{if } a \text{ then } A\ states that \(A\) is activated only if \(a\) is entailed by the store. The rule for \textit{if } a \text{ else } A\ states that \(A\) is activated only if we guess that \(a\) will not be entailed by the store at the end of the reaction. It is not sufficient to require that \(\sigma(\Gamma) \nvdash a\).

The output of executing agent \(A\) on input \(a\) is determined by function \(r_o(A)\) such that:

\[r_o(A)(a) = \{b \in D | \exists b'. ([A, \text{tell } a], \emptyset) \xrightarrow{b^*} (\Gamma, \Delta) \nvdash b', b' = \sigma(\Gamma), b \sim b'\}.\]

Namely, \(b \in r_o(A)(a)\) if we guess that \(b\) is the output of \(A\) on input \(a\) and a token equivalent to \(b\) is produced by \([A, \text{tell } a]\).

Now, we describe the temporal behavior of agents in terms of a transition relation \(\rightsquigarrow\) over agents:

\[\Gamma \rightsquigarrow [A_1, \ldots, A_n] \quad \text{if} \quad \exists b \in D | (A, \emptyset) \xrightarrow{b^*} (\Gamma', \Delta) \nvdash b, \sigma(\Gamma') = b, \Delta = A_1, \ldots, A_n.\]
The output of executing agent \( A \) on a sequence of inputs \( a_1, \ldots, a_n \) is determined by function \( rt_o(A) \) such that:

\[
rt_o(A)(a_1, \ldots, a_n) = \{ b_1, \ldots, b_n \mid A_0 = A, r_o(A_i)(a_i) = b_i, [A_i, \text{tell } a_i] \leadsto A_{i+1} \}.
\]

The operational semantics described in Table 6.1 can be used to compute the result of running an agent in a given store only if the final store is known beforehand. The nondeterminism can be made effective by backtracking.

An agent \( A \) is reactive and deterministic if and only if for every sequence of tokens \( a_1, \ldots, a_n \) there exists a sequence of tokens \( b_1, \ldots, b_n \) such that \( b_1, \ldots, b_n \in rt_o(A)(a_1, \ldots, a_n) \), and this sequence is unique modulo \( \sim \).

**Example 6.1.3** Let \( A \) be the agent \([\text{if } a \ \text{else } \text{tell } b, \text{if } b \ \text{else } \text{tell } a]\). Since \( b \in rt_o(A)(\text{true}) \) and \( a \in rt_o(A)(\text{true}) \), agent \( A \) is nondeterministic.

We conclude this section with an example of a program specified in tdccp.

**Example 6.1.4** Let us consider the specification of the bus arbiter given in Section 5.4. Let us assume the constraint system Gentzen. A cell may be specified by the following tdccp agent \textit{Cell}:

\[
\text{rec } P. ( [ [ [ [ \text{if } T.in \ \text{then if } \text{Req.in then tell Ack.out}, \\
\text{if } G.in \ \text{then if } \text{Req.in then tell Ack.out} ], \\
\text{if } T.in \ \text{then if } \text{Req.in else tell } G.out ], \\
\text{if } G.in \ \text{then if } \text{Req.in else tell } G.out ], \\
\text{if } T.in \ \text{then next tell } T.out ], \\
\text{next } P ] ).
\]

The ring of three cells may be specified by the agent \textit{ring.3} defined as follows:

\[
[ \text{tell } t_1, \\\n[\text{Cell}[r_1/\text{Req.in}, t_1/T.in, g_1/G.in, t_2/T.out, g_2/G.out, a_1/Ack.out], \\
[\text{Cell}[r_2/\text{Req.in}, t_2/T.in, g_2/G.in, t_3/T.out, g_3/G.out, a_2/Ack.out], \\
[\text{Cell}[r_3/\text{Req.in}, t_3/T.in, g_3/G.in, t_1/T.out, g_1/G.out, a_3/Ack.out] ] ] ]
\]

where the component \text{tell } t_1 \text{ gives the token to the first cell at the first instant.}

### 6.2 The labeled transition system

In this section we propose a labeled transition system as an operational semantic model for tdccp. LTS states correspond to agents, LTS transitions correspond to agent reactions, and LTS labels carry information on variables in \textit{Var} and on causality between tokens.

We begin with introducing some notations.
6.2. THE LABELED TRANSITION SYSTEM

Let $\mathcal{D}^+_\gamma$ be the set $\{a^+, a^- | a \in \mathcal{D}\}$. Given a token $a \in \mathcal{D}$, the symbol $a^+$ denotes the validity of $a$, while $a^-$ denotes the nonvalidity of $a$. In the following, $\gamma$ will range over $\mathcal{D}^+_\gamma$. We assume a function $\omega: \mathcal{D}^+_\gamma \rightarrow \mathcal{D}^+_\gamma$ such that $\overline{a^+} = a^-$ and $\overline{a^-} = a^+$, for every $a \in \mathcal{D}$.

An event $D$ (over $\mathcal{D}$) is a subset of $\mathcal{D}^+_\gamma$ such that there do not exist tokens $a_1, \ldots, a_n, a$ such that $a_1^+, \ldots, a_n^+, a^- \in D$ and $a_1 \wedge \ldots \wedge a_n \vdash a$. An event $D$ is interpreted as the assumption that the store entails every token $a$ such that $a^+ \in D$ and does not entail any token $a$ such that $a^- \in D$.

Note that if we consider the constraint system Gentzen, then this definition of event coincides with that in Chapter 3.

Given events $D$ and $D'$, $D$ and $D'$ are consistent, written $D \upharpoonright D'$, if and only if $D \cup D'$ is an event.

Definition 6.2.1 Given an event $D$ and a token $a$, the pair $(D, a)$ is a causality term with $D$ as cause and $a$ as action.

Given a causality term $(D, a)$, action $a$ refers to the action of posting the token $a$ in the store. This action is performed by an agent $\text{tell} a$ that is in the body of a statement $\text{if } b \text{ then } a$, for every $b^+ \in D$, and in the body of a statement $\text{if } b \text{ else } a$, for every $b^- \in D$. So, causality terms reflect relations of causality between tokens.

We note that in Chapter 3 causality terms are formed by an ordered event, instead of an event, and by an action. We explain our choice by an example. Let us consider the Esterel statement $E \equiv \text{if } a^+ \text{ then if } b^+ \text{ then emit } c$ and the $\text{tdccp}$ agent $A \equiv \text{if } a \text{ then if } b \text{ then } c$. The causality term $a^+ b^+ c$ in labels of LTS transitions having $E$ as source state reflects that signal $c$ is emitted if both signals $a$ and $b$ are present. Moreover, $a^+ b^+ c$ reflects that $E$ terminates immediately either if both $a$ and $b$ are present, or if $a$ is absent, or if $a$ is present and $b$ is absent. So, if we consider $E; \text{emit } s$, we are able to construct causality terms $a^+ b^+ s$, $a^+ b^- s$ and $a^- s$ reflecting causes of the production of $s$. The causality term $\{(a^+, b^+)\}$ in labels of LTS transitions having $A$ as source state will reflect only that if the store entails both tokens $a$ and $b$ then token $c$ is posted in the store. No information on the termination of $A$ is needed because $\text{tdccp}$ does not offer any construct equivalent to construct "\(;\)" of Esterel.

A set of causality terms $\mathcal{E}$ is complete if, given $(D_1, a_1), \ldots, (D_k, a_k), (D, a) \in \mathcal{E}$ such that $b^+ \in D$ and $\bigwedge_{1 \leq i \leq k} a_i \vdash b$, then we have that $(D \setminus \{b^+\} \cup \bigcup_{1 \leq i \leq k} D_i, a) \in \mathcal{E}$. Given a set of causality terms $\mathcal{E}$, we denote with $\mathcal{E}^+$ the set of causality terms such that $\mathcal{E} \subseteq \mathcal{E}^+$, $\mathcal{E}^+$ is complete, and there exists no complete set of causality terms $\mathcal{E}'$ such that $\mathcal{E} \subseteq \mathcal{E}' \subset \mathcal{E}^+$.

In practice, $\mathcal{E}^+$ is the transitive closure of the causality relation corresponding to $\mathcal{E}$.

Definition 6.2.2 A label is a pair $l = \langle D_l, \mathcal{E}_l \rangle$ such that:

- $D_l$ is an event over $\mathcal{D}$;
• $\mathcal{E}_t$ is a complete set of causality terms;

• $D_t = \bigcup_{(D, a) \in \mathcal{E}_t} D \cup \{a^+\}$.

We will denote with $\mathcal{L}$ the set of labels as in Def. 6.2.2.

Let us assume a transition $A \xrightarrow{t} A'$. Given $\mathcal{E}$ an arbitrary subset of $\mathcal{E}_t$ such that $\bigcup_{(D, a) \in \mathcal{E}} D \cup \{a^+\} = D_t \cap \{a^+ | a \in D\}$, if the store entails every token $b$ such that $b^+ \in \bigcup_{(D, a) \in \mathcal{E}} D$ and does not entail any token $b$ such that $b^- \in D_t$, then agent $A$ performs the reaction represented by $A \xrightarrow{t} A'$. When the reaction is completed, every token $b$ such that $b^+ \in D_t$ is entailed by the store.

We note that labels as in Def. 6.2.2 differ w.r.t. labels as in Def. 3.2.2. While labels as in Def. 3.2.2 give information on the value of input signals and on causality between input signals and output signals, labels as in Def. 6.2.2 give information on shared variables and on causality between tokens, without any distinction between input tokens and output tokens.

Labels as in Def. 3.2.2 keep track of nonexecuted statements in order to carry on the analysis of constructiveness, while labels as in Def. 6.2.2 do not keep track of nonexecuted agents because no notion of constructiveness is considered in tdccp. Finally, labels as in Def. 6.2.2 do not carry information on termination of agents, because agents can neither exit traps nor terminate immediately.

The LTS giving the operational semantics of tdccp is defined by the transition system specification of Table 6.2.

Rule $\text{skip}$ states that $\text{skip}$ does nothing at every instant.

Rule $\text{tell}$ states that $\text{tell} a$ posts the token $a$ in the store and will behave as $\text{skip}$. The causality term $(\emptyset, a)$ in label $\{(\{a^+\}, \{(\emptyset, a)\})\}$ expresses that the action of posting $a$ in the store is performed independently of the information in the store.

Before explaining rules for constructs $\text{if ... then ...}$ and $\text{if ... else ...}$, we introduce some more notations.

Given a label $l$ and a symbol $\gamma \in D^+_t$ such that $D_t \uparrow \{\gamma\}$, we denote with $\gamma(l)$ the label:

$$\gamma(l) = \langle D_l \cup \{\gamma\}, \{(D \cup \{\gamma\}, a) | (D, a) \in \mathcal{E}_t \}^+ \rangle.$$

Let us consider the agent if $a$ then $A$. Rule $\text{then}_a \emptyset$ states that if the store entails $a$ then if $a$ then $A$ reacts as $A$. The label $a^+(l)$ highlights that actions performed by agents in the body of $A$ require that the store entails $a$. Rule $\text{then}_a l$ states that if the store does not entail $a$ then if $a$ then $A$ terminates, namely it posts token $\text{true}$ in the store and it will behave as $\text{skip}$.

**Example 6.2.3** Let $A$ be the agent if $a$ then $\text{tell} b$. We have $A \xrightarrow{l_1} \text{skip}$, where $l_1 = \langle \{a^+, b^+\}, \{(\{a^+\}, b)\} \rangle$, and $A \xrightarrow{l_2} \text{skip}$, where $l_2 = \langle \{a^-\}, \{(\{a^-\}, \text{true})\} \rangle$. 
Rules \(\text{else}_0\) and \(\text{else}_1\) for if a else A are analogous to then_0 and then_1, respectively.

Rule next states that next A will react as A at the next instant. At the current instant only true is posted in the store.

Rule par states that \([A_0, A_1]\) performs both reactions of \(A_0\) and \(A_1\). We assume a partial function \(\otimes : \mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L}\) such that, given labels \(l_0\) and \(l_1\) such that \(D_{l_0} \uparrow D_{l_1}\), we have:

\[
l_0 \otimes l_1 = \langle D_{l_0} \cup D_{l_1}, (\mathcal{E}_{l_0} \cup \mathcal{E}_{l_1})^+ \rangle.
\]

Label \(l_0 \otimes l_1\) carries information carried by both \(l_0\) and \(l_1\).

**Example 6.2.4** Let us assume \(A \equiv [\text{if } a \text{ then } \text{tell } b, \text{if } b' \text{ then } \text{tell } c]\), where \(b \vdash b'\).

We have \(A \xrightarrow{b} [\text{skip, skip}], A \xrightarrow{b'} [\text{skip, skip}]\) and \(A \xrightarrow{l_1} [\text{skip, skip}], \) where

\[
l_1 = \langle \{a^+, b^+, b'^+, c^+\}, \{(a^+), b\}, \{(a^+), c\}, \{(b'^+), c\} \rangle,
\]

\[
l_2 = \langle \{a^-, b'^+, c^+\}, \{(b'^+), c\}, \{(a^-), \text{true}\} \rangle, \quad \text{and}
\]

\[
l_3 = \langle \{a^-, b'^-\}, \{(b'^-), \text{true}\} \rangle.
\]

The following proposition implies that \([A_0, \text{skip}] \approx [A_0, A_0], [A_0, A_1] \approx [A_1, A_0]\) and \([[A_0, A_1], A_2] \approx [A_0, [A_1, A_2]]\), for \(A_0, A_1\) and \(A_2\) arbitrary tdccp agents.

**Proposition 6.2.5** Given labels \(l_0, l_1, l_2 \in \mathcal{L}\), we have that:

- \(l_0 \otimes \langle \emptyset, \emptyset \rangle = l_0\);
- \(l_0 \otimes l_0 = l_0\);
- \(l_0 \otimes l_1 = l_1 \otimes l_0\);
- \((l_0 \otimes l_1) \otimes l_2 = (l_0 \otimes (l_1 \otimes l_2))\).

**Proof** The first three properties are immediate. We have \(D_{(l_0 \otimes l_1) \otimes l_2} = D_{l_0} \cup D_{l_1} \cup D_{l_2} = D_{l_0 \otimes (l_1 \otimes l_2)}\) and \(\mathcal{E}_{(l_0 \otimes l_1) \otimes l_2} = (\mathcal{E}_{l_0} \cup \mathcal{E}_{l_1} \cup \mathcal{E}_{l_2})^+ = \mathcal{E}_{l_0 \otimes (l_1 \otimes l_2)}\).

Prop. 6.2.5 allows to denote with \(l_0 \otimes l_1 \otimes l_2\) both labels \((l_0 \otimes l_1) \otimes l_2\) and \(l_0 \otimes (l_1 \otimes l_2)\), and to denote with \([A_0, A_1, A_2]\) both agents \([[A_0, A_1], A_2]\) and \([A_0, [A_1, A_2]]\). Moreover, we can denote with \text{skip} the parallel composition of agents \(A_1, \ldots, A_n\), where \(n = 0\).

The following proposition implies that constructs if \text{-} then \text{-} and if \text{-} else \text{-} distribute w.r.t. construct \([\_, \_\).**

**Proposition 6.2.6** Given labels \(l_0, l_1 \in \mathcal{L}\) and a token \(a \in \mathcal{D}\), we have that:

- \(a^+ (l_0 \otimes l_1) = a^+ (l_0) \otimes a^+ (l_1)\);
\[ a^-(l_0 \otimes l_1) = a^-(l_0) \otimes a^-(l_1). \]

**Proof** The first property holds because \( D_{a^+(l_0 \otimes l_1)} = D_{l_0} \cup D_{l_1} \cup \{a^+\} = D_{a^+(l_0) \otimes a^+(l_1)} \) and \( \mathcal{E}_{a^+(l_0 \otimes l_1)} = \{(D \cup \{a^+\}, b) \mid (D, b) \in \mathcal{E}_{l_0} \cup \mathcal{E}_{l_1}\}^+ = \mathcal{E}_{a^+(l_0) \otimes a^+(l_1)}. \)

The second property could be proved analogously.

Note that if \( a \text{ then } [A_0, A_1] \) and \( [a \text{ then } A_0, \text{if } a \text{ then } A_1] \) are not distinguished by the operational semantics of [86].

In fact, we have that \( ((\Gamma, \text{if } a \text{ then } [A_0, A_1]), \Delta) \rightarrow_b^* ((\Gamma, A_0, A_1), \Delta) \text{ iff } \sigma(\Gamma) \vdash a \text{ iff } ((\Gamma, [\text{if } a \text{ then } A_0, \text{if } a \text{ then } A_1]), \Delta) \rightarrow_b^* ((\Gamma, A_0, A_1), \Delta). \)

Analogously, if \( a \text{ else } [A_0, A_1] \) and \( [a \text{ else } A_0, \text{if } a \text{ else } A_1] \) are not distinguished by the operational semantics of [86].

Rule \texttt{rec} is a standard recursion rule.

We say that a token \( a \) triggers a transition \( A \xrightarrow{t} A' \) if there exists a subset \( \mathcal{E} \) of \( \mathcal{E}_s \) such that \( \bigcup_{(D, c) \in \mathcal{E}} D \cup \{c^+\} = D_t \cap \{c^+ \mid c \in D\} \), \( D_t \cup \{a^+\} \) is an event, and \( a \vdash b \) for every \( b^+ \in D_t, (D, c) \in \mathcal{E}. \)

If \( a \) triggers \( A \xrightarrow{t} A' \) then this transition represents a reaction of \( A \) to a store entailing \( a \). When the reaction has been completed, every token \( b \) such that \( b^+ \in D_t \) is entailed by the store.

In order to relate \texttt{tdcpp} agents having the same input/output behavior, we consider the bisimulation on the states of the LTS of Table 6.2. Given agents \( A_0 \) and \( A_1 \) such that \( A_0 \approx A_1 \), the external environment is not able to distinguish between them. In fact, at each execution cycle, if \( A_0 \) and \( A_1 \) are stimulated with the same information in the store, then they respond by posting in the store the same information. The following theorem states that, if \( A_0 \) and \( A_1 \) do not contain any free recursion variable, then no \texttt{tdcpp} context is able to discriminate them, namely \texttt{tdcpp} constructs preserve bisimulation.

**Theorem 6.2.7** The bisimulation on \texttt{tdcpp} agents is a congruence.

**Proof** The thesis follows by the fact that \texttt{tdcpp} is a de Simone language.

In the following, we will denote with \([A]\) the part of LTS reachable from \( A \), for every agent \( A \).

### 6.3 Correspondence with the original semantics

In this section we show the correspondence between the SOS semantics defined in the previous section and the operational semantics of [86]. Namely, we show that our LTS carries information sufficient to deduce whether agents are deterministic and reactive and to recover their input/output behavior.
We begin with introducing some notations.
Given a string $\vartheta \in (\mathcal{D}_-^\uparrow)^*$, we denote with $\text{if } \vartheta \text{ then } A$ the agent

$$
\text{if } \vartheta \text{ then } A = \begin{cases} 
A & \text{if } \vartheta = \epsilon \\
\text{if } a \text{ then } \phi \text{ then } A & \text{if } \vartheta = a^+ \phi \\
\text{if } a \text{ else if } \phi \text{ then } A & \text{if } \vartheta = a^- \phi.
\end{cases}
$$

**Definition 6.3.1** An agent $A$ is a normal form if there exist agents $A_1, \ldots, A_n$ such that $A \equiv [A_1, \ldots, A_n]$, $A_i \equiv \text{if } \vartheta_i \text{ then } B_i$, and either $B_i \equiv \text{tell } a_i$, or $B_i \equiv \text{next } A_i'$, or $B_i \equiv \text{skip}$, where $A_i'$ is an arbitrary agent, $1 \leq i \leq n$.

First of all we prove that for every agent $A$ there exists a normal form $B$ such that $A$ and $B$ are equivalent w.r.t. the operational semantics of [86] and w.r.t. our SOS semantics.

Let us consider axioms in Table 6.3. Prop. 6.2.5 and 6.2.6 and transition rule rec imply that the axiomatization induced by these axioms is sound over bisimulation. Moreover, it is immediate that, given agents $A$ and $B$ such that $A = B$, we have $A \xrightarrow{L} A'$ if and only if $B \xrightarrow{L} B'$ and $A' = B'$ can be inferred by means of axioms $\|_1, \|_2, \|_3$ and $\|_4$.

Note that, given tokens $a$ and $b$ and agents $A$ and $B$ such that $A = B$ can be inferred by axioms in Table 6.3, we have that $b \in rt_o(A, a)$ and $[A, \text{tell } a] \leadsto A'$ if and only if $b \in rt_o(B, a)$ and $[B, \text{tell } a] \leadsto A'$.

**Proposition 6.3.2** Given an agent $A$, there exists a normal form $B$ such that $A = B$ can be inferred by axioms in Table 6.3.

**Proof** By structural induction over $A$.
*Basic case*: if either $A \equiv \text{tell } a$ or $A \equiv \text{skip}$ then the thesis is immediate because $A$ is a normal form.

*Inductive step*: we must consider the following cases:

- $A \equiv \text{if } a \text{ then } A_0$: by inductive hypothesis, there exists a normal form $B_0 \equiv [B_1, \ldots, B_n]$ such that $A_0 = B_0$. Since $=$ is a congruence, we have that $A = \text{if } a \text{ then } B_0$. If $n = 1$ then $\text{if } a \text{ then } B_0$ is a normal form. If $n \geq 2$ then we apply axiom then and we infer $A = [\text{if } a \text{ then } B_1, \ldots, \text{if } a \text{ then } B_n]$, which is a normal form.

- $A \equiv \text{if } a \text{ else } A_0$: this case is analogous to that above.

- $A \equiv \text{next } A_0$: the thesis is immediate because $A$ is a normal form.

- $A \equiv [A_0, A_1]$: by inductive hypothesis, there exist normal forms $B_0$ and $B_1$ such that $A_0 = B_0$ and $A_1 = B_1$. Since $=$ is a congruence, we have that $A = [B_0, B_1]$, which is a normal form.
\( A \equiv \text{rec} P, A_0 \): by axiom \( \text{rec} \) we infer \( A = A_0[A/P] \). By inductive hypothesis, there exists a normal form \( B_0 \) such that \( A_0 = B_0 \). So, we have that \( A_0[A/P] = B_0[A/P] \), which is a normal form.

This completes the proof.

The following Theorem establishes the correspondence between our SOS semantics and the operational semantics of [86].

Theorem 6.3.3 Given an agent \( A \) and tokens \( a \) and \( b \), the following facts are equivalent:

- \([A, \text{tell } a] \sim A' \) and \( b \in rt_o(A)(a) \);
- \( a \) triggers a transition \( A \xrightarrow{\ell} A' \) and \( b \sim \bigwedge_{d^+ \in D_h} d \land a \).

Proof We can assume that \( A \) is a normal form. In fact, if \( A \) is not a normal form, by Prop. 6.3.2 it follows that there exists a normal form \( B \) such that \( A = B \) can be inferred by axioms is Table 6.3 and, therefore, \( A \xrightarrow{\ell} A' \) if and only if \( B \xrightarrow{\ell} B' \), where \( A' = B' \) can be inferred by means of axioms \( ||\cdot||_4 \). Moreover, \([A, \text{tell } a] \sim A' \) and \( b \in rt_o(A, a) \) if and only if \([B, \text{tell } a] \sim A' \) and \( b \in rt_o(B, a) \). So, let us assume that \( A \) is a normal form and let us reason by structural induction over \( A \).

Basic case: if \( A \equiv \text{tell } c \) then we have \([A, \text{tell } a] \sim A' \) and \( b \in rt_o(A)(a) \) if and only if \( A' \equiv \text{skip} \) and \( b \sim a \land c \). We have \( A \xrightarrow{\ell} A' \) if and only if \( \ell = \langle \{c^+\}, \emptyset, \{\emptyset, \{e\}\} \rangle \) and \( A' \equiv \text{skip} \). Since \( \bigwedge_{d^+ \in D_h} c \land a \) triggers \( A \xrightarrow{\ell} A' \), the thesis follows.

If \( A \equiv \text{skip} \) then \([A, \text{tell } a] \sim A' \) and \( b \in rt_o(A)(a) \) if and only if \( A' \equiv \text{skip} \) and \( b \sim a \). We have \( A \xrightarrow{\ell} A' \) if and only if \( l = \langle \emptyset, \emptyset \rangle \) and \( A' \equiv \text{skip} \). Since \( \bigwedge_{d^+ \in D_h} = \text{true} \) and \( a \) triggers \( A \xrightarrow{\ell} A' \) then the thesis follows.

Inductive step: we must consider the following cases:

- \( A \equiv \text{if } c \text{ then } B \): we distinguish two cases:
  - \( a \vdash c \): we have \([A, \text{tell } a] \sim A' \) and \( b \in rt_o(A)(a) \) if and only if \([B, \text{tell } a] \sim A' \) and \( b \in rt_o(B)(a) \). By inductive hypothesis, this is equivalent to having that \( a \) triggers \( B \xrightarrow{\ell'} A' \) and \( b \sim a \land \bigwedge_{d^+ \in D_h} a \land c^+ \land \ell' \). By transition rule then, \( A \xrightarrow{\ell} A' \) if and only if \( B \xrightarrow{\ell'} A', \{c^+\} \uparrow D_h \) and \( l = c^+ \). Now, \( a \) triggers \( B \xrightarrow{\ell'} A' \) and \( a \vdash c \) if and only if \( a \) triggers \( A \xrightarrow{\ell} A' \). Since \( a \land \bigwedge_{d^+ \in D_h} d \sim a \land \bigwedge_{d^+ \in D_h} d \land c \sim a \land \bigwedge_{d^+ \in D_h} d \), the thesis follows.
  - \( a \not\vdash c \): we have \([A, \text{tell } a] \sim A' \) and \( b \in rt_o(A)(a) \) if and only if \( A' \equiv \text{skip} \) and \( b \sim a \). We have that \( a \) triggers \( A \xrightarrow{\ell} A' \) if and only if \( A' \equiv \text{skip} \) and \( l = \langle \{c^-\}, \{c^-\}, \text{true} \rangle \). Since \( \bigwedge_{d^+ \in D_h} d \sim \text{true} \), the thesis follows.
6.3. CORRESPONDENCE WITH THE ORIGINAL SEMANTICS

- \( A \equiv \text{if} \ c \ \text{else} \ B \): this case is analogous to that above.

- \( A \equiv \text{next} \ B \): we have \([A, \text{tell} \ a] \leadsto A'\) and \( b \in rt_o(A)(a)\) if and only if \( A' \equiv B \) and \( b \sim a \). We have \( A \rightarrow^l A'\) if and only if \( A' \equiv B \) and \( l = \langle \{ \text{true}^+ \}, \{ (0, \text{true}) \} \rangle \). Since \( \bigwedge_{d^+ \in D_l} d = \text{true} \) and \( a \) triggers \( A \rightarrow^l A'\), the thesis follows.

- \( A \equiv [A_1, \ldots, A_n] \): since \( A \) is a normal form, \([A, \text{tell} \ a] \leadsto [A'_1, \ldots, A'_n]\) and \( b \in rt_o(A, a)\) if and only if there exist indexes \( \{ i_1, \ldots, i_n \} \) and tokens \( a_1, \ldots, a_n, b_1, \ldots, b_n \) such that:
  1. \( \{ i_1, \ldots, i_n \} = \{ 1, \ldots, n \} \);
  2. \( a_1 = a, a_{j+1} = b_j \) for \( 1 \leq j < n, b_n = b \);
  3. \( b_j \in rt_o(A_{i_j})(a_j) \) and \([A_{i_j}, \text{tell} \ a_j] \leadsto A'_{i_j}\);
  4. \([A_{i_j}, \text{tell} \ a_j] \rightarrow^b (\Gamma_{i_j}, A'_{i_j}) \not\in b \) and \( \sigma(\Gamma_{i_j}) \sim b_j \).

By inductive hypothesis, item 3 is equivalent to having that \( a_j \) triggers \( A_{i_j} \rightarrow^l A'_{i_j} \) and \( b_j \sim a_j \land \bigwedge_{d^+ \in D_{i_j}} d \). Item 4 is equivalent to having that \( \{ b^+ \} \uparrow D_{i_j} \) for every \( 1 \leq j \leq n \), namely that \( l_1 \otimes \ldots \otimes l_n \) is defined.

These two facts and items 1, 2 above are equivalent to having that \( a \) triggers \( A \rightarrow^l [A'_1, \ldots, A'_n] \) and \( b \sim \bigwedge_{d^+ \in D_l} d \land a \), where \( l = l_1 \otimes \ldots \otimes l_n \).

- \( A \equiv \text{rec} \ P.B \): we have \([A, \text{tell} \ a] \leadsto A'\) and \( b \in rt_o(A)(a)\) if and only if \([B[A/P], \text{tell} \ a] \leadsto A'\) and \( b \in rt_o(B[A/P])(a)\). By inductive hypothesis, this is equivalent to having that \( a \) triggers \( B[A/P] \rightarrow^l A'\) and \( b \sim a \land \bigwedge_{d^+ \in D_l} d \). Now, the thesis follows because transition rules imply that \( B[A/P] \rightarrow^l A'\) if and only if \( A \rightarrow^l A'\).

This completes the proof.

The following proposition states that \([A]\) carries sufficient information to determine whether \( A \) is reactive and deterministic.

Proposition 6.3.4 An agent \( A \) is reactive and deterministic if and only if each token \( a \in D \) triggers exactly one transition \( A \rightarrow^l A' \) in \([A]\).

Proof "If": directly from Theorem 6.3.3. "Only if": by Theorem 6.3.3, if \( b \in rt_o(A)(a)\) then \( a \) triggers a transition \( A \rightarrow^l \) and \( b \sim a \land \bigwedge_{d^+ \in D_l} d \). This transition is unique because, if \( A \rightarrow^l A_1 \) and \( A \rightarrow^l A_2 \), then \( D_{l_1} \uparrow D_{l_2} \), as it could be immediately proved.
Example 6.3.5 Let $A$ be the agent $[\text{if } a \text{ else tell } b, \text{if } b \text{ else tell } a]$ of Example 6.1.3. This agent is nondeterministic. From Table 6.2 we infer $A \xrightarrow{l_1} [\text{skip, skip}]$ and $A \xrightarrow{l_2} [\text{skip, skip}]$, with $l_1 = \{\{a^-, b^+\}, \{\{a^-, b\}, \{b^+, \text{true}\}, \{a^-, \text{true}\}\}\}$ and $l_2 = \{\{a^+, b^-\}, \{\{a^+, \text{true}\}, \{b^-, b\}, \{b^-, \text{true}\}\}\}$. The transition labeled with $l_1$ reflects that if the store does not entail $a$ then $A$ reacts by posting $b$ in the store, while the transition labeled with $l_2$ reflects that if the store does not entail $b$ then $A$ reacts by posting $a$ in the store. Now, token $\text{true}$ entails both transitions.

6.4 The causal semantics

In this section we give our causal semantics for $\text{tdccp}$.

According to the operational semantics of [86], an observer of an agent $A$ can observe sequences of pairs of tokens $(i_1, o_1), \ldots, (i_n, o_n), \ldots$, with $i_n$ the token supplied by the environment at the $n^{th}$ instant and $o_n$ the token produced by $A$ at the same instant. If $o_n = a_{n,1} \land \ldots \land a_{n,k_n}$ then the observer cannot associate to each token $a_{n,i}$ its causes, namely the subset of tokens supplied either by the environment or by $A$ itself up to the $n^{th}$ instant which are sufficient to have the token $a_{n,i}$ as response of $A$ at the $n^{th}$ instant.

Example 6.4.1 Let $A$ be the agent $[A_0, A_1]$, where $A_i \equiv \text{if } a_i \text{ then next tell } b_i$, $i \in \{0, 1\}$. If the environment supplies both $a_0$ and $a_1$ at the first instant, then the observer observes $b_0 \land b_1$ at the second instant, but it is unable to deduce that $b_0$ is caused by $a_0$ only and that $b_1$ is caused by $a_1$ only.

The SOS semantics of the previous section permits only to associate a token posted in the store by an agent with its instantaneous causes, namely with the event causing it at the current instant.

In this chapter we follow [42], where a causal semantics for $\text{ccp}$ has been proposed as a finer semantics w.r.t. the original operational semantics of [88]. According to the semantics of [88], the observer of a $\text{ccp}$ agent can observe the final store resulting from its execution, namely the set of tokens posted in the store by the agent. In the refined semantics of [42], the observable final store consists of a set of contextual tokens of the form $a^b$, where the context $b$ contains exactly the token causing $a$, namely the token causing the execution of an agent $\text{tell } a$.

Now, if we consider the construct $\text{if } \_ \text{ else } \_$ then tokens may be caused by the absence of other tokens. Moreover, if we consider the construct $\text{next } \_$ then tokens produced by an agent at the $n^{th}$ instant may be caused by the presence or by the absence of tokens at instants $i \leq n$. Therefore, for every token $a$ produced by an agent at the $n^{th}$ instant, we will consider contextual tokens of the form $a^{(D_0, D_1, \ldots, D_{n-1}, a_{n-1})}$, with $D_0, \ldots, D_{n-1}$ events, to represent that $a$ is caused by the fact that at the $(n-i)^{th}$ instant the store contains information as assumed by $D_i$. 
We recall now the causal semantics of constructs \texttt{tell}, \texttt{if . then .} and \texttt{[. .]} given in [42]. Note that the language considered in [42] offers also a construct of nondeterministic choice and a recursion construct. These constructs are not included in \texttt{tdccp} because \texttt{tdccp} rejects nondeterminism and allows only temporal guarded recursion.

The behavior of \texttt{ccp} agents is described in terms of transitions between configurations, where a configuration \( \Gamma \) is a multiset of \textit{contexted agents} of the form \( A^a \), satisfying the following requirement:

\[
\text{if } A^a \in \Gamma \text{ then } \sigma(\Gamma) \vdash a
\]

where we identify \( A \) and \( A^\text{true} \), and \( \sigma(\Gamma) \) is defined as follows:

\[
\sigma(\Gamma) = \begin{cases} 
\sigma(A_1^b), \ldots, \sigma(A_n^b) & \text{if } \Gamma = A_1^b, \ldots, A_n^b \\
a & \text{if } \Gamma = (\text{tell } a)^b \\
\text{true} & \text{if } \Gamma = [A_0, A_1]^b \text{ or } \Gamma = (\text{if } a \text{ then } A_0)^b.
\end{cases}
\]

The transition relation is defined by the following rules:

\[
\Gamma, [A_0, A_1]^a \rightarrow \Gamma, A_0^a, A_1^a \quad \frac{\sigma(\Gamma) \vdash b}{\Gamma, (\text{if } b \text{ then } A)^a \rightarrow \Gamma, A^{b \triangleright a}}
\]

In practice, a configuration \( \Gamma \) contains an agent \( A^a \) if agent \( A \) has been caused by token \( a \). So, we require that \( \sigma(\Gamma) \vdash a \), namely that \( a \) is entailed by the store.

We use \( o \) to range over contexted tokens of the form \( a^b \). Now, the output of executing agent \( A \) on input \( a \) is determined by function \( r_{c_0}(A) \) such that:

\[
rc_o(A)(a) = o \mid [A^\text{true}, (\text{tell } a)^a] \rightarrow^* \Gamma \not\vdash \text{ and } \rho(\Gamma) = o
\]

where \( \rho(\Gamma) \) is defined as follows:

\[
\rho(\Gamma) = \begin{cases} 
\rho(A_1^b), \ldots, \rho(A_n^b) & \text{if } \Gamma = A_1^b, \ldots, A_n^b \\
a^b & \text{if } \Gamma = (\text{tell } a)^b \\
\text{true} & \text{if } \Gamma = [A_0, A_1]^b \text{ or } \Gamma = (\text{if } a \text{ then } A_0)^b.
\end{cases}
\]

In [42] it is defined a notion of equivalence over contexted tokens, and function \( r_{c_0}(A) \) does not return a contexted token \( o \) as above, but it returns the class of equivalence of \( o \).

To give the causal semantics for \texttt{tdccp} we proceed as follows. First of all we provide a proved transition system as a very concrete semantic model for \texttt{tdccp}. PTS states correspond to agents, PTS transitions correspond to agent reactions, and PTS labels carry detailed information on reactions. In particular, labels carry information sufficient to establish which component of an agent posts a token in the store. Given an agent \( A \), we denote with \( [A]_{\text{pts}} \) the part of PTS reachable from \( A \).
and we denote with $[A]_{pt}$ the proved tree obtained by unfolding $[A]_{pt}$. We define a causal observation function relabeling proved trees into causal trees. Labels of causal trees contain contextualized tokens highlighting causes of the production of tokens. We consider causal trees as the causal model of tdccp.

We begin with introducing the notions of proof term and of enriched label.

**Definition 6.4.2** Given a string $\vartheta \in (D^+_2 \cup \{||_0, ||_1\})^*$ and a token $a \in D$, $\vartheta a$ is a proof term with $\vartheta$ as proof and $a$ as action.

Proof terms differ w.r.t. causality terms of Def. 6.2.1 as causes of causality terms are subsets of $D^+_2$, while proofs of proof terms are strings and may contain symbols in $\{||_0, ||_1\}$. The proof $\vartheta$ of a proof term $\vartheta a$ highlights the syntactic context in which the agent posting $a$ in the store is plugged. The symbol $||_0$ in $\vartheta$ means that this agent is in the left side of a $[\_ , \_ ]$, while $||_1$ in $\vartheta$ means that it is in the right side of a $[\_ , \_ ]$.

We note that symbols $;_0 , ;_1 , l_0 , l_1$, which may appear in proofs of Def. 5.1.1, cannot appear in proofs of Def. 6.4.2. Symbols $;_0$ and $;_1$ were required in Chapter 5 to recover causality, induced by construct ";", between actions performed at the same instant by a sequential Esterel statement. A sequential tdccp agent cannot perform more than one action at a given instant, so we do not need symbols $;_0$ and $;_1$. Symbols $l_0$ and $l_1$ were required in Chapter 5 because we rejected axiom $\texttt{1oep} E \texttt{end} = E ; 1\texttt{oep} E \texttt{end}$. This axiom was rejected because circuits implementing the two statements are completely different. In this chapter we accept axiom $\texttt{rec} P.A = A[\texttt{rec} P.A/P]$, so we do not need symbols $l_0$ and $l_1$.

Given a proof $\vartheta$, we denote with $|\vartheta|$ the subset of $D^+_2$ such that:

$$|\vartheta| = \begin{cases} 
\{\gamma\} \cup |\phi| & \text{if } \vartheta = \gamma \phi \\
|\phi| & \text{if } \vartheta = ||_0 \phi \text{ or } \vartheta = ||_1 \phi \\
\emptyset & \text{otherwise.}
\end{cases}$$

**Definition 6.4.3** An enriched label is a tuple $l = (D_l, E_l, N_l)$ such that:

- $D_l$ is an event over $D$;
- $E_l$ is a set of proof terms such that $\bigcup_{\vartheta \in E_l} |\vartheta| \cup \{a^+\} = D_l$;
- $N_l$ is a set of proof terms such that, for every $\vartheta a \in N_l$, $D_l \not\vdash |\vartheta|$.

Let us assume an enriched label $l$ of a transition $A \xrightarrow{l} A'$. Component $D_l$ plays the same rôle of component $D_l$ of Def. 6.2.2. Proof terms in $E_l$ refer to actions performed by $\texttt{tell}$ agents in the reaction corresponding to $A \xrightarrow{l} A'$. In fact, for each proof term $\vartheta a \in E_l$, the store entails every token $b$ such that $b^+ \in |\vartheta|$ and does not entail any token $b$ such that $b^- \in |\vartheta|$.

Proof terms in $N_l$ refer to actions that are not performed by $\texttt{tell}$ agents in the reaction corresponding to $A \xrightarrow{l} A'$. In fact, for each proof term $\vartheta a \in N_l$, either
the store entails $b$ for some $b^- \in \{\theta\}$ or the store does not entail $b$ for some $b^+ \in \{\theta\}$. Information given by $N_t$ can be used to determine the reason for which tokens are not produced by $A$.

The proved transition system is defined by the transition system specification in Table 6.4.

Rules $\text{skip}$, $\text{tell}$ and $\text{next}$ differ w.r.t. the corresponding rules of Table 6.2 because enriched labels keep track that there exists no $\text{tell}$ agent that is not executed. Rule $\text{rec}$ is straightforward.

In rules $\text{then}_0$ and $\text{else}_0$ we assume that, given an enriched label $l$ and a symbol $\gamma \in D^+_\gamma$ such that $D_t \uparrow \{\gamma\}$, $\gamma(l)$ is the enriched label:

$$\gamma(l) = \langle D_t \cup \{\gamma\}, \{\gamma \vartheta a \mid \vartheta a \in E_t\}, \{\gamma \vartheta a \mid \vartheta a \in N_t\} \rangle.$$  

In rules $\text{then}_1$ and $\text{else}_1$ we assume that, given a label $l$ and a symbol $\gamma \in D^-_{\gamma}$, $N(\gamma, l)$ is the enriched label:

$$N(\gamma, l) = \langle \{\gamma\}, \{\gamma \text{true}\}, \{\gamma \vartheta a \mid \vartheta a \in E_t \cup N_t \text{ and } a \neq \text{true}\} \rangle.$$  

Finally, in rule $\text{par}$ we assume a partial function $\oplus$ mapping pairs of enriched labels to enriched labels such that, given enriched labels $l$ and $l'$ such that $D_t \uparrow D_v$, $l \oplus l'$ is the enriched label:

$$\langle D_t \cup D_v, \{\|_0 \vartheta a \mid \vartheta a \in E_t\} \cup \{\|_1 \vartheta a \mid \vartheta a \in E_v\}, \{\|_0 \vartheta a \mid \vartheta a \in N_t\} \cup \{\|_1 \vartheta a \mid \vartheta a \in N_v\} \rangle.$$  

Symbols $\|_0$ and $\|_1$ in proofs permit to relate tokens produced at the $n^{th}$ instant with their causes at instants up to $n$. In fact, given transitions $A_1 \xrightarrow{l_1} \ldots A_n \xrightarrow{l_n} A_{n+1}$, if $\vartheta a \in E_{n-1}$, $\vartheta' \text{true} \in E_{n-k}$ and $\vartheta' \uparrow \{\|_0, \|_1\}$ is a prefix of $\vartheta \uparrow \{\|_0, \|_1\}$, then $\text{tell} a$ corresponding to $\vartheta a$ is executed at the $n^{th}$ instant because every $b$ with $b^+ \in \vartheta'$ (resp. $b^- \in \vartheta'$) was entailed (resp. nonentailed) by the store at the $(n-k)^{th}$ instant.

**Example 6.4.4** Let $A$ be as in Example 6.4.1. We have $A \xrightarrow{\iota} [\text{tell} b_0, \text{tell} b_1]$ $\xrightarrow{\iota'} [\text{skip}, \text{skip}]$, $l = \langle \{a_0^+, a_1^+, \text{true}^+\}, \{\|_0 a_0^+ \text{true}, \|_1 a_1^+ \text{true}\}, \emptyset\rangle$, $l' = \langle \{b_0^+, b_1^+\}, \{\|_0 b_0, \|_1 b_1\}, \emptyset\rangle$. We deduce that $b_0$ (resp. $b_1$) is produced at the second instant because $a_0$ (resp. $a_1$) was valid at the first one from the fact that $\|_0 a_0^+ \uparrow \{\|_0, \|_1\}$ is a prefix of $\|_1 \uparrow \{\|_0, \|_1\}$ and $\|_1 a_1^+ \uparrow \{\|_0, \|_1\}$ is a prefix of $\|_0 \uparrow \{\|_0, \|_1\}$.

The PTS defined by the transition rules in Table 6.4 satisfies the following property:

**Proposition 6.4.5** Given an agent $A$ and transitions $A \xrightarrow{l_1} A_1$ and $A \xrightarrow{l_2} A_2$, $A_1$ and $A_2$ arbitrary, we have that $\{\vartheta a \in E_{t_1} \cup N_{t_1} \mid a \neq \text{true}\} = \{\vartheta a \in E_{t_2} \cup N_{t_2} \mid a \neq \text{true}\}$.
**Proof** By structural induction over \( A \).

Prop. 6.4.5 implies that rules \textit{then-1} and \textit{else-1} do not depend on the choice of \( l \).

Note that we can recover \([A]\) from \([A]_{pl}\) by replacing every enriched label \( \langle D_t, \mathcal{E}_t, N_t \rangle \) by \( \langle D_t, \{(\vartheta, a) \mid \vartheta a \in \mathcal{E}_t \} \rangle \).

We define now the causal observation function \( C \) transforming a proved tree \([A]_{pl}\) into the causal tree \( C([A]_{pl}) \).

**Definition 6.4.6** The causal relabeling function \( C \) relabels any computation \( A_1 \xrightarrow{l_1} \ldots \xrightarrow{l_n} A_{n+1} \ldots \) as \( A_1 \xrightarrow{C(l_1)} \ldots \xrightarrow{C(l_n)} A_{n+1} \ldots \), where \( C(l_i) \) is the pair \( \langle C^+(l_i), C^-(l_i) \rangle \) such that:

- for each \( \vartheta a \in \mathcal{E}_{l_n} \), we have \( a^K \in C^+(l_n) \), where \( K \) is the set of pairs \( (D, k) \) such that either \( k = 0 \) and \( D = |\vartheta| \), or \( D = |\vartheta'| \) for some \( \vartheta \text{'true} \in \mathcal{E}_{l_{n-k}} \) such that \( \vartheta' \upharpoonright \{ ||_0, ||_1 \} \) is a prefix of \( \vartheta \upharpoonright \{ ||_0, ||_1 \} \).

- for each token \( a \) such that there exists a proof term \( \vartheta a \in N_{l_n} \) and \( \bigwedge_{b \in D_{l_n}} b \not\vdash a \), we have \( a^K \in C^-(l_n) \), where \( K \) is the set of pairs \( (D, 0) \) with \( D \) a minimal set \( D \subseteq D_{l_n} \) such that \( D \not\vdash || \) for every \( \vartheta a \in N_{l_n} \).

Given a computation \( A_1 \xrightarrow{l_1} \ldots \xrightarrow{l_n} A_{n+1} \) of \([A]_{pl}\), for each action of posting \( a \) in the store performed by a \texttt{tell} agent in the body of \( A_n \), a contexted token \( a^K \) is in \( C^+(l_n) \). Now, for each \( 0 \leq k \leq n - 1 \), \( K \) contains the pair \( (D, k) \) if among the causes of the execution of the considered agent there is the validity (resp. nonvalidity) of every token \( b \) such that \( b^+ \in D \) (resp. \( b^- \in D \)) at the \((n - k)\)th instant of time. So, \( k \) is a backward pointer in the causal tree.

Moreover, if a token \( a \) is not entailed by the store resulting at the \( n \)th instant and there exist agents of the form \texttt{tell} \( a \) in the body of \( A_n \), then \( a^K \in C^-(l_n) \), where \( K \) is a set of pairs \( (D, 0) \) such that \( D \) is an event justifying the nonexecution of agents of the form \texttt{tell} \( a \).

Note that causal trees highlight only direct causes of production/non production of tokens. Indirect causes should be computed by transitivity.

Causal trees highlight causality between tokens. We could not compute this causality by exploiting causality relations between actions performed by agents, as it is done in Chapter 5. Let us explain the reason. Let us consider the Esterel statement

\[
E \equiv \text{signal } b \text{ in } (\text{if } a^+ \text{ then emit } b \parallel \text{if } b^+ \text{ then emit } c) \text{ end}
\]

and the \texttt{tdccp} agent

\[
[\text{if } a \text{ then } b, \text{if } b \text{ then } c].
\]

The enabling tree \( En([E]_{pl}) \) contains transition \( E \xrightarrow{l} \text{ signal } b \text{ in nothing} \parallel \text{nothing end, with } l = \{(||_0 a^+ b, \emptyset), (||_1 b^+ c, \{(0, ||_0 a^+ b)\})\} \). This label emphasizes
that the action of producing \( c \) is caused by the action of producing \( b \). The causal tree \( \mathcal{C}([A]_{pt}) \) contains transition \( A \xrightarrow{l} [\text{skip}, \text{skip}] \), with \( l' = \langle \{b(\{(a^+).0\})\}, c(\{(b^+).0\})\}, \emptyset \rangle \). This label emphasizes that \( b \) is caused by \( a \) and \( c \) is caused by \( b \). We could not replace \( l' \) by a label analogous to \( l \), namely by a label stating that the production of \( c \) by the agent in the right branch of \( A \) is caused by the production of \( b \) by the agent in the left branch of \( A \). In fact, \( b \) could be posted in the store also by the environment.

We show now that our causal semantics and the causal semantics of [42] coincide for agents belonging to both \( \text{ccp} \) and \( \text{tdccp} \).

Given a label \( l \) of a causal tree, let us denote with \( l^+ \) and \( l^- \) its two components. Let us assume a \( \text{ccp} \) agent \( A \). Given a transition \( A \xrightarrow{l} A' \) in \( C([A]_{pt}) \), \( l^+ = \{b_{i_1}^{K_1}, \ldots, b_{i_m}^{M_m}\} \), \( K_i = (D_i, 0) \), we say that a token \( a \) triggers \( A \xrightarrow{l} A' \) if and only if there exists a sequence of indexes \( i_1, \ldots, i_m \) such that \( \{i_1, \ldots, i_m\} = \{1, \ldots, m\} \) and a sequence of tokens \( a_1, \ldots, a_{m+1} \) such that:

- \( a_1 = a \), \( a_{j+1} = a_j \wedge b_{i_j} \) for \( 1 \leq j \leq m \);
- \( a_j \vdash \bigwedge_{d^+ \in D_{i_j}} d \);
- \( a_{m+1} \not\vdash d \) for every \( d^- \in \bigcup_{1 \leq i \leq m} D_i \).

Intuitively, \( a \) triggers \( A \xrightarrow{l} A' \) if \( A \xrightarrow{l} A' \) represents a reaction of \( A \) to a store entailing \( a \).

The following theorem establishes the correspondence between our causal semantics and the causal semantics of [42].

**Theorem 6.4.7** Given a \( \text{ccp} \) agent \( A \) and a token \( a \), the following facts are equivalent:

- \( \text{rc}_o(A)(a) = b_{i_1}^{K_1} \wedge \ldots \wedge b_{i_m}^{M_m} \wedge a^a \);
- \( a \) triggers a transition \( A \xrightarrow{l} A' \), \( l^+ = \{b_{i_1}^{K_1}, \ldots, b_{i_m}^{M_m}\} \cup \{\text{true}^{K_i} | m + 1 \leq i \leq n\} \), \( K_i = (D_i, 0) \) and \( \bigwedge_{d^+ \in D_i} d \sim c_i \) for \( 1 \leq i \leq m \).

**Proof** We can assume that \( A \) is a normal form. In fact, it is immediate that if \( A = B \) can be inferred by axioms in Table 6.3, then \( A \) and \( B \) have the same causal semantics according to [42], and \( C([A]_{pt}) \) and \( C([B]_{pt}) \) are such that \( A \xrightarrow{l} A' \) if and only if \( B \xrightarrow{l} B' \), where \( A' = B' \) can be inferred by means of axioms \( \|_1\|_4 \). So, let us assume that \( A \) is a normal form and let us reason by structural induction over \( A \).

**Basic case:** \( A \equiv \text{tell} e \). We have \( \text{rc}_o(A)(a) = e^{\text{true}} \wedge a^a \) and \( A \xrightarrow{l} \text{skip} \), where \( l^+ = \{e(\{(0^+).0\})\} \). Now, \( a \) triggers \( A \xrightarrow{l} A' \) and the thesis follows.

**Inductive step:** we must consider the following cases:

- \( A \equiv \text{if } e \text{ then } B \): we must distinguish two cases:
a \vdash e$: we have that $rc_o(B)(a) = b_1^{K_1} \land \ldots \land b_m^{K_m} \land a^a$ if and only if $rc_o(A)(a) = b_1^{K_1} \land \ldots \land b_m^{K_m} \land a^a$. By inductive hypothesis, this is equivalent to having that $a$ triggers $B \xrightarrow{L} A', \; l^+ = \{b_1^{K_1}, \ldots, b_m^{K_m}\} \cup \{\text{true}^{K_i} | m + 1 \leq i \leq n\}$, $K_i = (D_i, 0) \land \bigwedge_{d \in D_i} d \sim c_i$. If $a \not\vdash e$ then this is equivalent to having that $a$ triggers $A \xrightarrow{L} A'$ where $l^+ = \{b_1^{K_1}, \ldots, b_m^{K_m}\} \cup \{\text{true}^{K_i} | m + 1 \leq i \leq n\}$, $K_i = (D_i \cup \{e\}, 0)$. Now, since $\bigwedge_{d \in D_i} d \sim c_i$, we have that $\bigwedge_{d \in D_i \cup \{e\}} d \sim c_i \land e$ and the thesis follows.

- $a \not\models e$: we have $rc_o(A)(a) = a^a$ and we have that $A \xrightarrow{L} \text{skip}$, where $l^+ = \{\text{true}^{((e^a)^0)}\}$. Now, a triggers $A \xrightarrow{L} A'$ and the thesis follows.

• $A \equiv [A_1, \ldots, A_n]$, $A_i \equiv \text{if } \vartheta_i \text{ then } \text{tell } b_i$: we have that $rc_o(A)(a) = b_1^{K_1} \land \ldots \land b_m^{K_m} \land a^a$ if and only if there exist tokens $a_1, \ldots, a_m$ such that $a_1 = a$, $a_{j+1} = a_j \land b_j$, $rc_o(A_i)(a_j) = b_j^{K_j} \land a_j^a$, $a_j \vdash c_j$, and $a \land b_i \land \ldots \land b_m \not\vdash \bigwedge_{d \in D_i} d$, $h \in \{1, \ldots, n\} \setminus \{i_1, \ldots, i_m\}$. By inductive hypothesis this is equivalent to having that $a_j$ triggers $A_i \xrightarrow{L} A_i'$, $l^+_j = \{b_j^{K_j}\}$, $K_j = (D_j, 0) \land \bigwedge_{d \in D_i} d \sim c_j$ and $a_j \vdash c_j$, and to having that $A_h \xrightarrow{L} l_h = \{\text{true}^{((D_h)^0)}\}$, where $a_{m+1} \not\vdash d_h$ for some $d_h \in D_h$, $h \in \{1, \ldots, n\} \setminus \{i_1, \ldots, i_m\}$. This is equivalent to having that $a$ triggers $A \xrightarrow{L} A'$, where $l^+ = l^+_1 \cup \ldots \cup l^+_n$ and, therefore, the thesis follows.

This completes the proof.

We show now how the causal semantics of \texttt{tdccp} could be exploited during the verification phase.

Let us consider the agent \textit{ring}_3 of Example 6.1.4. As in Section 5.4, let us consider the following safety property $\phi$: “\textit{if the third user requests the bus at two subsequent instants, then the second user cannot obtain the bus at both instants}”. The property $\phi$ can be translated into the following \texttt{tdccp} agent:

$$\Omega_\phi \equiv \text{rec } P. \; \text{[if } r_3 \text{ then if } a_2 \text{ then next (if } r_3 \text{ then if } a_2 \text{ then tell } \alpha) \text{, next } P \text{].}$$

Let us consider now the agent $[\text{ring}_3, \Omega_\phi]$, and let us assume that both the second and the third user request the bus at the first and second instant. The second user obtains the bus at both instants, so that $\phi$ is not verified. Now, according to the classical semantics of [86], the observer of $[\text{ring}_3, \Omega_\phi]$ can observe the pair of tokens $(r_2 \land r_3, a_2 \land t_1 \land g_2)$ at the first instant and the pair of tokens $(r_2 \land r_3, a_2 \land t_2 \land \alpha)$ at the second instant. We mean that $rt_o([\text{ring}_3, \Omega_\phi])(r_2 \land r_3, r_2 \land r_3) = r_2 \land r_3 \land a_2 \land t_1 \land g_2, r_2 \land r_3 \land a_2 \land t_2 \land \alpha$. So, the observer can only detect that $\phi$ is violated by \textit{ring}_3. The observer cannot deduce the reasons for which \textit{ring}_3 violates $\phi$.

Now, let us consider the causal tree $C([\text{ring}_3, \Omega_\phi])$. We observe the computation $[\text{ring}_3, \Omega_\phi] \xrightarrow{L \cup U} \ldots$, where the contexted token $\alpha^{\{\{a_2^+ r_3^+\}^0, \{a_2^+ r_3^+\}\}}$ in
6.5 Local Variables

The language of [86] has been extended in [87] with the hiding construct “\texttt{new \_ in \_}”. Given a variable \(X \in \text{Var}\) and an agent \(A\), the agent \texttt{new X in A} behaves as \(A\) with \(X\) as local variable. Namely, the environment cannot constrain \(X\) and cannot view constraints on \(X\) produced by \(A\). As we have already argued in Section 6.1, local variables are not needed for communications between cooperating agents.

For every variable \(X \in \text{Var}\), it is assumed a function \(\exists_X : \mathcal{D} \rightarrow \mathcal{D}\) such that:

- \(a \vdash \exists_X a\);
- \(\exists_X (a \land \exists_X b) \sim \exists_X a \land \exists_X b\);
- \(\exists_X \exists_Y a \sim \exists_Y \exists_X a\);
- \(a \vdash b\) implies \(\exists_X a \vdash \exists_X b\).

The instantaneous operational semantics is obtained by adding the following rule to rules in Table 6.1

\[ ((\Gamma, \texttt{new X in A}, \Delta) \rightarrow b, ((\Gamma, A[Y/X]), \Delta)) \] where \(Y \in \text{Var}\) and \(Y\) is fresh in \(A, \Gamma\).

Function \(r_o(A)\) giving the result of executing \(A\) on input \(a\) must take into account fresh variables \(\bar{Y}\) introduced in the derivation of \(\rightarrow b\):

\[ r_o(A)(a) = \{ b \in \mathcal{D} \mid \exists b'. ([A, \texttt{tell} a, \emptyset] \rightarrow^{b'} ((\Gamma, \Delta) \not\vdash b'), b' = \sigma(\Gamma), b \sim \exists_{\bar{Y}} b' \} . \]
Also relation \( \sim \) must take into account such variables:
\[
\exists b \in D \mid (A, \emptyset) \to_t^* (I', \Delta) \quad \text{not} \quad \sigma(I') = b, \quad \Delta = A_1, \ldots, A_n.
\]
\[
A \sim \text{new } Y \text{ in } [A_1, \ldots, A_n].
\]

In order to deal with local variables, we must modify the labeled transition system of Table 6.2. We must consider transitions \( A \xrightarrow{l} A' \) with \( l \) of the form \( \langle D_t, E_t, V_t \rangle \), where \( D_t \) and \( E_t \) play the same role of the corresponding components of Def. 6.2.2, and \( V_t \) is the set of variables local to \( A \).

Rules \textit{tell} and \textit{skip} of Table 6.2 are replaced by the following ones which highlight that no variable is local to \textit{skip} and \textit{tell} a.

\[
\text{(skip)}
\]
\[
\text{skip} \xrightarrow{\langle \emptyset, \emptyset, \emptyset \rangle} \text{skip}
\]

\[
\text{(tell)}.
\]

The semantics of construct \textit{new in } is given by the following rule:

\[
\text{new } X \text{ in } A \xrightarrow{\langle D_t, E_t, V_t \cup \{ Y \} \rangle} \text{new } Y \text{ in } A'
\]

where \( A[Y/X] \) is the agent obtained by replacing occurrences of \( X \) in \( A \) by \( Y \), provided that \( Y \) does not appear in \( A \).

Rules \textit{then} \textit{0}, \textit{then} \textit{1}, \textit{else} \textit{0} and \textit{else} \textit{1} for agents if \( a \) \textit{then} \( A \) and if \( a \) \textit{else} \( A \) remain as in Table 6.2. Given a label \( l \) and a symbol \( \gamma \in \mathcal{D}_- \), \( \gamma(l) \) is defined as follows:

\[
\gamma(l) = \langle D_t \cup \{ \gamma \}, \{(D \cup \{ \gamma \}, a) \mid (D, a) \in E_t \}^+, V_t \rangle,
\]

provided that \( \gamma \) does not constrain variables in \( V_t \). In fact, variables constrained by \( \gamma \) are not local to \( A \).

Rule \textit{par} for agent \([A_0, A_1] \) remains as in Table 6.2. Given labels \( l_0 \) and \( l_1 \), \( l_0 \otimes l_1 \) is defined as follows:

\[
l_0 \otimes l_1 = \langle D_{t_0} \cup D_{t_1}, (E_{t_0} \cup E_{t_1})^+, V_{t_0} \cup V_{t_1} \rangle,
\]

provided that variables in \( V_{t_0} \) do not appear in \( l_1 \) and variables in \( V_{t_1} \) do not appear in \( l_0 \).

Now, we say that a token \( a \) that does not constrain variables in \( V_t \) triggers a transition \( A \xrightarrow{l} A' \) if there exists a subset \( E \) of \( E_t \) such that \( \bigcup_{(D, c) \in E} D \cup \{ c^+ \} = D_t \cap \{ c^+ \mid c \in D \} \), \( D_t \cup \{ a^+ \} \) is an event, and \( a \vdash b \) for every \( b^+ \in D, (D, c) \in E \).

Note that we must require that \( a \) does not constrain variables in \( V_t \) because these are local to \( A \).

The definition of normal form (cf. Def. 6.3.1) is modified as follows.
6.5. LOCAL VARIABLES

Definition 6.5.1 An agent $A$ is a normal form if there exist agents $A_1, \ldots, A_n$ such that $A \equiv \text{new } \vec X$ in $[A_1, \ldots, A_n]$, $A_i \equiv \text{if } b_i \text{ then } B_i$, and either $B_i \equiv \text{tell } a_i$, or $B_i \equiv \text{next } A'_i$, or $B_i \equiv \text{skip}$, where $A'_i$ is an arbitrary agent, $1 \leq i \leq n$.

Note that if $A$ is a normal form then $A \equiv \text{new } \vec X$ in $B$, where $B$ is a normal form according to Def. 6.3.1.

Given an agent $A$, we denote with $\var(A)$ the variables appearing in $A$. Given a token $a$, we denote with $\var(a)$ the variables appearing in $a$.

Let us consider axioms in Table 6.5. It is immediate that, given $A$ and $B$ such that $A = B$ can be inferred by these axioms, $A \xrightarrow{t} A'$ if and only if $B \xrightarrow{t} B'$ and $A' = B'$. Moreover, $A$ and $B$ are not distinguished by the operational semantics of [87]. Namely, $b \in rt_o(A(a))$ and $[A, \text{tell } a] \sim A'$ if and only if $b \in rt_o(B(a))$ and $[B, \text{tell } a] \sim A'$.

The following proposition states that every agent can be transformed in a normal form, while preserving its behavior.

Proposition 6.5.2 Given an agent $A$, there exists a normal form $B$ such that $A = B$ can be inferred by axioms in Table 6.3 and Table 6.5.

Proof By structural induction over $A$.

Basic case: if either $A \equiv \text{tell } a$ or $A \equiv \text{skip}$ then the thesis is immediate because $A$ is a normal form.

Inductive step: we must consider the following cases:

- $A \equiv \text{if } a \text{ then } A_0$: by inductive hypothesis, there exists a normal form $B_0$ such that $A_0 = B_0$. Since = is a congruence, we have that $A = \text{if } a \text{ then } B_0$. If $B_0 \equiv \text{new } \vec X$ in $B'_0$ then we apply axioms new$_3$ and new$_2$ and we transform $A$ in an agent of the form $\text{new } \vec Y$ in $\text{if } a \text{ then } B'_0$. Now, we proceed as in the proof of Prop. 6.3.2.

- $A \equiv \text{if } a \text{ else } A_0$: this case is analogous to that above.

- $A \equiv \text{next } A_0$: the thesis is immediate because $A$ is a normal form.

- $A \equiv [A_0, A_1]$: by inductive hypothesis, there exist normal forms $B_0$ and $B_1$ such that $A_0 = B_0$ and $A_1 = B_1$. Since = is a congruence, we have that $A = [B_0, B_1]$. We transform $[B_0, B_1]$ into a normal form $\text{new } \vec Y$ in $[B_0, B_1]$ by applying axioms new$_3$ and new$_1$.

- $A \equiv \text{rec } P, A_0$: by axiom rec we infer $A = A_0[A/P]$. By inductive hypothesis, there exists a normal form $B_0$ such that $A_0 = B_0$. So, we have that $A_0[A/P] = B_0[A/P]$, which is a normal form.

- $A \equiv \text{new } X$ in $A_0$: by inductive hypothesis there exists a normal form $B_0$ such that $A_0 = B_0$. Since = is a congruence, we infer $A = \text{new } X$ in $B_0$, which is a normal form.
This completes the proof.

The following theorem states the correspondence between the SOS semantics given in this section and the operational semantics of [87]

**Theorem 6.5.3** Given an agent $A$ and tokens $a$ and $b$, the following facts are equivalent:

- $[A, \text{tell } a] \leadsto A'$ and $b \in rt_o(A)(a)$;
- $a$ triggers a transition $A \overset{t}{\rightarrow} A'$ with $V_t = \vec{Y}$ and $b \sim a \land \exists \vec{Y} \bigwedge_{d \in D_t} d$.

**Proof** By structural induction over $A$. The proof for the basic cases and for constructs $\text{if_then_if_else_next, rec } P.$ and $[\_\_\_\_\_]$ has been given in the proof of Theorem 6.3.3.

So, let us assume that $A \equiv \text{new } \vec{X}$ in $B$, with $B$ a normal form according to Def. 6.3.1. We have that $[A, \text{tell } a] \leadsto A'$ and $b \in rt_o(A)(a)$ if and only if $[B[\vec{Y} / \vec{X}], \text{tell } a] \leadsto B'$, $\vec{b} \in rt_o(B[\vec{Y} / \vec{X}](a)$, $A' \equiv \text{new } \vec{Y}$ in $B'$ and $b \sim \exists \vec{Y} \vec{b}'$, where $\vec{Y}$ are variables fresh in $B$ and $a$. By inductive hypothesis, $[B[\vec{Y} / \vec{X}], \text{tell } a] \leadsto B'$ and $\vec{b} \in rt_o(B[\vec{Y} / \vec{X}](a)$ if and only if $a$ triggers a transition $B[\vec{Y} / \vec{X}] \overset{t}{\rightarrow} B'$ and $\vec{b} \sim a \land \bigwedge_{d \in D_t} d$. Now, $B[\vec{Y} / \vec{X}] \overset{t}{\rightarrow} B'$ and $\vec{Y}$ are fresh in $B$ if and only if $A \overset{(D_t, \vec{Y})}{\rightarrow} \text{new } \vec{Y}$ in $B'$. Since $\vec{Y}$ are fresh in $a$, $a$ triggers $A \overset{(D_t, \vec{Y})}{\rightarrow} \text{new } \vec{Y}$ in $B'$. So, the thesis follows.
The LTS for tdccp.

\[
\begin{align*}
\text{skip} & \xrightarrow{\emptyset k} \text{skip} \quad \text{(skip)} \\
tell a & \xrightarrow{\{a^+, \{\emptyset, a\}\}} \text{skip} \quad \text{(tell)} \\
A & \xrightarrow{t} A' \quad \{a^+\} \uparrow D_t \quad \text{(then}_0) \\
\text{if } a \text{ then } A & \xrightarrow{a^+ t} A' \quad \text{(then}_1) \\
\text{if } a \text{ else } A & \xrightarrow{a^- t} A' \quad \text{(else}_0) \\
\text{if } a \text{ else } A & \xrightarrow{a^+ t} \text{skip} \quad \text{(else}_1) \\
\text{next } A & \xrightarrow{\{true^+, \{\emptyset, true\}\}} A \quad \text{(next)} \\
A_0 & \xrightarrow{k_0} A_0' \quad A_1 \xrightarrow{k_1} A_1' \quad D_{k_0} \uparrow D_{k_1} \quad \text{(par)} \\
[A_0, A_1] & \xrightarrow{k_0 \oplus k_1} [A_0', A_1'] \\
A[\text{rec } P. A/P] & \xrightarrow{t} A' \quad \text{(rec)} \\
\text{rec } P. A & \xrightarrow{t} A' 
\end{align*}
\]

Table 6.2: The labeled transition system for tdccp.
### Axioms for \( \text{tdccp} \)

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>([A_0, A_1] = [A_1, A_0])</td>
<td>(\parallel_1)</td>
</tr>
<tr>
<td>([A_0, [A_1, A_2]] = [[A_0, A_1], A_2])</td>
<td>(\parallel_2)</td>
</tr>
<tr>
<td>([A_0, \text{skip}] = A_0)</td>
<td>(\parallel_3)</td>
</tr>
<tr>
<td>([A_0, A_0] = A_0)</td>
<td>(\parallel_4)</td>
</tr>
<tr>
<td>(\text{if } a \text{ then } [A_0, A_1] = [\text{if } a \text{ then } A_0, \text{if } a \text{ then } A_1])</td>
<td>(\text{then})</td>
</tr>
<tr>
<td>(\text{if } a \text{ else } [A_0, A_1] = [\text{if } a \text{ else } A_0, \text{if } a \text{ else } A_1])</td>
<td>(\text{else})</td>
</tr>
<tr>
<td>(\text{rec } P.A = A[\text{rec } P.A/P])</td>
<td>(\text{rec})</td>
</tr>
</tbody>
</table>

Table 6.3: axioms for \( \text{tdccp} \).
6.5. LOCAL VARIABLES

The PTS for tdccp.

\[
\begin{align*}
\text{skip} & \xrightarrow{\emptyset, \emptyset, \emptyset} \text{skip} \\
\text{tell } a & \xrightarrow{\langle \{a^+\}, \{a\}, \emptyset \rangle} \text{skip} \\
A & \xrightarrow{t} A' & \text{[}a^+\text{]} \uparrow D_t & \text{(then\_0)} \\
\text{if } a \text{ then } A & \xrightarrow{a^+(t)} A' \\
A & \xrightarrow{t} A' & \text{[}a^-\text{]} \uparrow D_t & \text{(else\_0)} \\
\text{if } a \text{ else } A & \xrightarrow{a^-(t)} A' \\
A & \xrightarrow{t} A' & \text{[}a^-\text{]} \uparrow D_t & \text{(else\_1)} \\
\text{if } a \text{ else } A & \xrightarrow{a^- (t)} \text{skip} \\
\text{next } A & \xrightarrow{\langle \{\text{true}^+\}, \{\text{true}\}, \emptyset \rangle} A \\
A_0 & \xrightarrow{t_0} A'_0 & A_1 & \xrightarrow{t_1} A'_1 & D_{t_0} \uparrow D_{t_1} & \text{(par)} \\
[A_0, A_1] & \xrightarrow{t_0 \oplus t_1} [A'_0, A'_1] \\
A & \xrightarrow{\text{rec } P, A / P} & A' & \text{(rec)} \\
\text{rec } P, A & \xrightarrow{t} & A'
\end{align*}
\]

Table 6.4: The proved transition system for tdccp.

Other axioms for tdccp.

\[
\begin{align*}
X \notin \text{var}(A_1) & \Rightarrow [\text{new } X \text{ in } A_0, A_1] = \text{new } X \text{ in } [A_0, A_1] & \text{new}_1 \\
X \notin \text{var}(a) & \Rightarrow \text{if } a \text{ then } \text{new } X \text{ in } A = \text{new } X \text{ in } \text{if } a \text{ then } A & \text{new}_2 \\
Y \notin \text{var}(A) & \Rightarrow \text{new } X \text{ in } A = \text{new } Y \text{ in } A[Y/X] & \text{new}_3
\end{align*}
\]

Table 6.5: other axioms for tdccp.
Chapter 7

SOS for Statecharts

Statecharts extends the notation of Finite State Machines by allowing FSM states to be refined by other FSMs and by allowing FSMs to run in parallel and to communicate by exchanging binary signals. Contrarily to Esterel and tdccp, Statecharts admits nondeterminism.

In the literature many operational semantics have been defined for several dialects of Statecharts. These semantics describe the behavior of Statecharts programs (called statecharts) in terms of sequences of steps, namely sequences of sets of FSM transitions.

As observed in [67], the nondeterminism may be explicitly controlled by the programmer, or may arise when possibly deterministic programs are composed in parallel. In fact, it may happen that two transitions of FSMs running in parallel are both enabled to fire, but the firing of each of them prevents the firing of the other. So, despite the fact that the two FSMs might be deterministic, there is a nondeterministic choice between the two transitions.

In [80] a semantics for Statecharts has been proposed which admits nondeterminism and enforces synchronous hypothesis, global consistency of signals and causality, in the sense that Statecharts is interpreted as a synchronous language, every binary signal takes precisely one value during each reaction, and signals produced by statecharts are causally justified by signals produced by the environment. This semantics was presented as an improvement w.r.t to the original operational semantics of [51] which does not enforce global consistency of signals.

Unfortunately, there exist statecharts having no meaning according to the semantics of [80], in the sense that there exist cases for which a procedure given in [80] to compute steps does not terminate. In this chapter we propose a “step semantics” for Statecharts that slightly differs w.r.t to the traditional step semantics of [80]. More precisely, we do not allow our procedure computing steps to fail.

Then, we provide a structural operational semantics for Statecharts in terms of a labeled transition system. More precisely, we define a process algebra where processes (called statechart terms) are configurations of statecharts, and we provide a semantics in SOS style for such processes. We show the correspondence between
the LTS semantics and our step semantics. We mean that we prove that information in our LTS permits to deduce the input/output behavior of statecharts.

In order to relate statecharts having the same input/output behavior, we consider the bisimulation over statechart terms. Since Statecharts admits nondeterminism, it may happen that several LTS transitions having the same LTS state as source state have the same label. For this reason, we consider well-known notions of equivalence that have been introduced in the theory of process calculi as equivalences coarser than bisimulation, and we investigate their property of congruence.

We prove that bisimulation is a congruence and that simulation and ready trace preorder on statechart terms are precongruences, by exploiting the format of rules defining the LTS. Then, we prove explicitly that failure preorder and trace preorder are precongruences.

The chapter is organized as follows. In Section 7.1 we recall Statecharts. In Section 7.2 and Section 7.3 we present our step semantics and our LTS semantics, respectively, and in Section 7.4 we prove their correspondence. Finally, in Section 7.5 we consider equivalences on statechart terms and we prove their property of congruence.

## 7.1 An overview of Statecharts

Statecharts extends the notation of Finite State Machines with concepts of hierarchy, concurrency and broadcast communication.

FSM transitions are labeled by pairs, where the first component is referred to as trigger and consists of a set of positive and negated signals, and the second component is referred to as action and consists of a set of positive signals. Intuitively, if the source state of a transition is active and the environment offers the signals in the trigger, but not the negated ones, then the transition is triggered; it fires and produces signals in the action. In this case the source state is deactivated and the target state is activated.

Hierarchy is achieved by allowing FSM states to be refined by injecting other FSMs. A FSM refining a state starts running when the state is activated and is preempted when the state is deactivated. Concurrency is achieved by allowing FSMs to run in parallel. FSMs running in parallel communicate by broadcasting and sensing binary signals.

FSM states are either of type “basic”, called basic states, or of type “and”, called and-states, or of type “or”, called or-states. Basic states cannot have substates, namely they cannot be refined. Immediate substates of an or-state \( n \) are orthogonal, in the sense that when \( n \) is active then exactly one of them is active. On the contrary, when an and-state is active then all its immediate substates are active. Immediate substates of an or-state may be connected by transitions and give a classical FSM, while immediate substates of an and-state represent activities running in parallel. An or-state has a privileged immediate substate called default state. When an
or-state is activated, also its default state is.

A Statechart program is usually called a *statechart*. Its states are organized as a tree-like structure, where the root of this tree is called the *root state* of the statechart, children of a state are its immediate substates and leaves of the tree are basic states.

The graphical convention is that states are depicted as boxes and the box of the substate of another state is drawn inside the area of the box of that state; and-states are depicted as boxes whose area is partitioned by dashed lines and each element of the partition is a parallel component of the state. A default state is marked by a dangling arrow. The statechart of Figure 7.1 consists of an and-state, state 9, having two or-states, 5 and 8, as immediate substates. They represent FSMs running in parallel. The former FSM consists of states 3 and 4 and of transition $t_3$, while the latter FSM consists of states 6 and 7 and of transition $t_2$. States 3 and 6 are the default states of 5 and 8, respectively. State 3 is an or-state and it is refined by a FSM consisting of states 1 and 2 and of transition $t_1$. State 9 is the root state. States 1, 2, 4, 6 and 7 are basic states.

We assume a countable set of signal $\mathcal{S}$, and, following Statecharts convention, we use $a, b, \ldots$ to range over $\mathcal{S}$ and we denote with $\bar{a}$ the negation of signal $a$.

Here we consider a “step-semantics” for Statecharts which enforces *causality*, *synchrony hypothesis* and *global consistency*, and admits *nondeterminism*.

We assume that a statechart evolves from *configurations* to configurations, starting from the *default configuration*. A configuration $C$ is a maximal set of states fulfilling the requirement that if an and-state is in $C$ then all its immediate substates are in $C$, and if an or-state is in $C$ then exactly one of its immediate substates is in $C$. The default configuration $C_0$ is such that, for every or-state $n \in C_0$, the default state of $n$ is in $C_0$. 
At each instant of time the environment prompts the statechart by communicating a set of signals in $S$, and the statechart reacts from the current configuration $C$ by performing a step. A step is a maximal set of transitions that are consistent, compatible, relevant in $C$, and triggered by the communicated signals. Two transitions are consistent if they belong to components running in parallel, and are compatible if in the action of one of the transitions there is no signal appearing negated in the trigger of the other (in the opposite case the execution of the former prevents the execution of the latter). A transition is relevant in a configuration $C$ if its source state is in $C$. When fired, a transition communicates instantaneously signals which can (instantaneously) trigger new (relevant and compatible) transitions (an instantaneous chain reaction). So, the set of transitions in a step are triggered either by signals communicated by the environment or by signals communicated by transitions in the same step, provided that the triggering of each transition can be causally justified starting from signals communicated by the environment (causality). The maximality of the step enforces the synchrony hypothesis, while the global consistency consists of the fact that all the transitions of the step are triggered by communicated signals, namely no signal is assumed to be both present and absent at the same instant.

When a step is performed from a configuration $C$, a new configuration $C'$ is entered. Configuration $C'$ is obtained from $C$ by removing all states that are (substates of) source states of transitions in the step, and by adding all states that are target states of transitions in the step. Moreover, if an and-state $n$ is entered then all its substates are entered, and if an or-state is entered then its default state is entered.

Note that in literature many semantics for several dialects of Statecharts have been proposed. Most of them are described and compared in [9]. In next section we will present formally our semantics and we compare it w.r.t. to the step semantics of [80] and [76]. As examples of other semantics, we mention those in [51, 49, 75, 66]. The semantics of [51] does not enforce global consistency. In fact, steps are computed as sequences of microsteps, and signal consistency is required only for microsteps. The semantics of [49] does not enforce synchronous hypothesis, because signals produced by transitions of a step are sensed by a statechart only when the subsequent step is triggered, namely they are “stored” for an instant of time. The semantics of [75] releases the requirement that transitions in a step are consistent, so that an FSM refining an FSM state can react during the reaction in which this state is deactivated. The language Argos [66] can be viewed as a dialect of Statecharts that rejects nondeterminism.

### 7.2 Statechart terms

In [97] (see also [98]) statecharts are represented by terms of a process algebra. Here we represent statechart configurations by terms of a process algebra. In practice,
we follow the philosophy of process calculi where one does not distinguish between processes and process states.

We assume sets of names $\mathcal{N}$ for (FSM) states and $\mathcal{T}$ for (FSM) transitions. We denote with $\tilde{\mathcal{S}}$ the set $\{\bar{a} \mid a \in \mathcal{S}\}$, we use $\gamma$ to range over $\mathcal{S} \cup \tilde{\mathcal{S}}$, and, given a set $\mathcal{S} \subseteq \mathcal{S} \cup \tilde{\mathcal{S}}$, we denote with $\tilde{\gamma}$ the set $\{\bar{\gamma} \mid \gamma \in \mathcal{S}\}$.

Terms of the statechart process algebra are those generated by the following BNF-like grammar:

$$p ::= [n] \mid [n : p_1, p_2] \mid [n : (p_1, \ldots, p_k), p, \alpha, T]$$

where $n$ ranges over $\mathcal{N}$, $p, p_1, \ldots$ range over terms, $\alpha \in \{1, \ldots, k\}$ and $T \subseteq \mathcal{T} \times \{1, \ldots, k\} \times 2^{\mathcal{S} \cup \tilde{\mathcal{S}}} \times 2^{\mathcal{S}} \times \{1, \ldots, k\}$.

Term $[n]$ represents configuration $\{n\}$ of the statechart consisting of a basic state $n$.

Term $[n : p_1, p_2]$ represents a configuration of a statechart having the and-state $n$ as root state and two parallel components. Term $[n : p_1, p_2]$ represents the configuration containing $n$ and the union of the configurations represented by $p_1$ and $p_2$, where $p_k$ represents a configuration of one of the two parallel components.

Term $[n : (p_1, \ldots, p_k), p, \alpha, T]$ represents a configuration of a statechart having the or-state $n$ as root state, $k$ orthogonal components, and a transition $t$ with the root state of the $i^{th}$ component as source state, $A$ as trigger, $B$ as action and the root state of the $j^{th}$ component as target state, for every tuple $\langle t, i, A, B, j \rangle \in T$. The root of the first component is the default state of $n$. Term $p_i$ represents the default configuration of the $i^{th}$ component. If $\alpha = i$ then $[n : (p_1, \ldots, p_k), p, \alpha, T]$ represents the configuration consisting of $\{n\}$ and of the configuration represented by $p_i$ where this is a configuration of the statechart having the root of the $i^{th}$ component as root state.
For instance, the default configuration of the statechart $z_1$ of Figure 7.2 is represented by $p = [7 : p_3, p_6]$, with $p_3 = [3 : (p_1, p_2), p_1, 1, \{(t_1, 1, \{a\}, \{b\}, 2)\}]$, $p_6 = [6 : (p_4, p_5), p_4, 1, \{(b_2, 1, \{b\}, \{c\}, 2)\}]$, and $p_i = [i]$ for $i \in \{1, 2, 4, 5\}$.

Function $\text{root}$ defined below maps a term $p$ representing a configuration of a statechart to the root state of this statechart:

$$ \text{root}(p) = \begin{cases} 
\{n\} & \text{if } p = [n] \\
\{n\} & \text{if } p = [n : (p_1, \ldots, p_k), p, \alpha, T] \\
\{n\} & \text{if } p = [n : p_1, p_2]. 
\end{cases} $$

Function $\text{states}$ defined below maps a term $p$ representing a configuration of a statechart to the set of states of this statechart:

$$ \text{states}(p) = \begin{cases} 
\{n\} & \text{if } p = [n] \\
\{n\} \cup \bigcup_{1 \leq i \leq k} \text{states}(p_i) & \text{if } p = [n : (p_1, \ldots, p_k), p, \alpha, T] \\
\{n\} \cup \bigcup_{1 \leq i \leq 2} \text{states}(p_i) & \text{if } p = [n : p_1, p_2]. 
\end{cases} $$

Function $\text{active}$ defined below maps a term $p$ to the set of states giving the configuration represented by $p$:

$$ \text{active}(p) = \begin{cases} 
\{n\} & \text{if } p = [n] \\
\{n\} \cup \text{active}(p) & \text{if } p = [n : (p_1, \ldots, p_k), p, \alpha, T] \\
\{n\} \cup \text{active}(p_1) \cup \text{active}(p_2) & \text{if } p = [n : p_1, p_2]. 
\end{cases} $$

Function $\text{conf}$ defined below maps a term $p$ representing a configuration of a statechart to the set of configurations of this statechart:

$$ \text{conf}(p) = \begin{cases} 
\{\{n\}\} & \text{if } p = [n] \\
\{n\} \cup c_i | c_i \in \text{conf}(p_i), 1 \leq i \leq k & \text{if } p = [n : (p_1, \ldots, p_k), p, \alpha, T] \\
\{n\} \cup c_1 \cup c_2 | c_i \in \text{conf}(p_i) & \text{if } p = [n : p_1, p_2]. 
\end{cases} $$

We will consider only terms $p$ such that $\text{active}(p) \in \text{conf}(p)$.

Function $\text{def}$ defined below maps a term $p$ representing a configuration of a statechart to the default configuration of this statechart:

$$ \text{def}(p) = \begin{cases} 
\{n\} & \text{if } p = [n] \\
\{n\} \cup \text{def}(p_1) & \text{if } p = [n : (p_1, \ldots, p_k), p, \alpha, T] \\
\{n\} \cup \text{def}(p_1) \cup \text{def}(p_2) & \text{if } p = [n : p_1, p_2]. 
\end{cases} $$

Function $\text{trans}$ defined below maps a term $p$ representing a configuration of a statechart to the set of transitions of this statechart:

$$ \text{trans}(p) = \begin{cases} 
\emptyset & \text{if } p = [n] \\
\bigcup_{1 \leq i \leq k} \text{trans}(p_i) \cup \bigcup_{(t,i,A,B,j) \in T} \{t\} & \text{if } p = [n : (p_1, \ldots, p_k), p, \alpha, T] \\
\text{trans}(p_1) \cup \text{trans}(p_2) & \text{if } p = [n : p_1, p_2]. 
\end{cases} $$
7.2. STATECHART TERMS

Given a term \([n: (p_1, \ldots, p_k), p, \alpha, T]\) and a tuple \([t, i, A, B, j] \in T\), we denote with \(tr(t)\) and \(act(t)\) the trigger \(A\) and the action \(B\) of \(t\), respectively. We will assume that \(act(t) \cap tr(t) = \emptyset\). Moreover, we denote with \(out(t)\) and \(in(t)\) the source state \(root(p_i)\) and the target state \(root(p_j)\) of \(t\), respectively. Finally, given \(n, n' \in \mathcal{N}\) and a term \(p\), we write \(n' \prec n\) if \(n'\) is a substate of \(n\), namely if there exists either a subterm \([n: (p_1, \ldots, p_k), p', \alpha, T]\) of \(p\) with \(n' \in \bigcup_{1 \leq i \leq k} \text{states}(p_i)\), or a subterm \([n: p_1, p_2]\) of \(p\) with \(n' \in \bigcup_{1 \leq i < 2} \text{states}(p_i)\). The transitive reflexive closure of \(\prec\) will be denoted with \(\prec^*\).

In the following we will assume that subterms of a term \(p\) have different names for transitions and states. Namely, given a term \([n: (p_1, \ldots, p_k), p, \alpha, T]\), we assume that \(n \notin \bigcup_{1 \leq i \leq k} \text{states}(p_i)\), \(t \notin \bigcup_{1 \leq i \leq k} \text{trans}(p_i)\) for every \([t, i, A, B, j] \in T\), \(\text{states}(p_i) \cap \text{states}(p_j) = \emptyset\) and \(\text{trans}(p_i) \cap \text{trans}(p_j) = \emptyset\), for \(1 \leq i < j \leq n\). Analogously, given a term \([n: p_1, p_2]\), we assume that \(n \notin \bigcup_{1 \leq i \leq 2} \text{states}(p_i)\), \(\text{states}(p_1) \cap \text{states}(p_2) = \emptyset\) and \(\text{trans}(p_1) \cap \text{trans}(p_2) = \emptyset\). This requirement is not restrictive, because we can rename states and transitions of terms.

Now, we present formally our step semantics.

Given a term \(p\), two transitions \(t_1, t_2 \in T\) are consistent, written \(t_1 \perp t_2\), if there exists a subterm \([n: p_1, p_2]\) of \(p\) such that \(t_1 \in \text{trans}(p_1)\) and \(t_2 \in \text{trans}(p_2)\). Namely, \(t_1\) and \(t_2\) are in components of \(p\) running in parallel.

Given a term \(p\) and a set of transitions \(T \in \text{trans}(p)\), function \(Cons: 2^{\text{trans}(p)} \rightarrow 2^{\text{trans}(p)}\) such that

\[
Cons(T) = \{t \in \text{trans}(p) \mid \forall t' \in T: t \perp t'\}
\]

gives the set of transitions consistent with those in \(T\).

Given a term \(p\) and a set of transitions \(T \in \text{trans}(p)\), function \(Comp: 2^{\text{trans}(p)} \rightarrow 2^{\text{trans}(p)}\) such that

\[
Comp(T) = \{t \in \text{trans}(p) \mid \forall t' \in T: act(t) \cap \overline{tr(t')} = \emptyset\}
\]

gives the set of transitions compatible with those in \(T\).

Given a term \(p\), the set of transitions \(Rel\) such that

\[
Rel = \{t \in \text{trans}(p) \mid out(t) \in \text{active}(p)\}
\]

contains exactly all transitions relevant in the configuration represented by \(p\).

Given a term \(p\) and a set of signals \(S\), function \(Trig: 2^S \rightarrow 2^{\text{trans}(p)}\) such that

\[
Trig(S) = \{t \in \text{trans}(p) \mid tr(t) \cap S \subseteq S \land tr(t) \cap \overline{S} \cap S = \emptyset\}
\]

gives the set of transitions triggered by \(S\).

Now, given a term \(p\) and a set of signals \(S\), the set of all possible steps from configuration \(\text{active}(p)\) is computed by function \(Step: 2^S \rightarrow 2^{2^{\text{trans}(p)}}\) such that:

\[
Step(S) = \{T \subseteq \text{trans}(p) \mid T \text{ is inseparable for } S \text{ and } Enabled(S, T) = T\}
\]
where \( \text{Enabled} : 2^S \times 2^{\text{trans}(p)} \rightarrow 2^{\text{trans}(p)} \) is the function such that
\[
\text{Disabled}(S, T) = \text{Rel} \cap \text{Cons}(T) \cap \text{Trig}(S \cup \bigcup_{t \in T} \text{act}(t)) \cap \text{Comp}(T)
\]
and a set of transitions \( T \subseteq \text{trans}(p) \) is inseparable for \( S \subseteq S \) if, for any \( T' \subset T \), \( \text{Disabled}(S, T) \cap (T \setminus T') \neq \emptyset \).

Let us assume a term \( p \), a set of signals \( S \) and a set of transitions \( T \) such that \( T \in \text{Step}(S) \). Condition \( T \subseteq \text{Disabled}(S, T) \) guarantees global consistency. Condition \( \text{Disabled}(S, T) \subseteq T \) guarantees the maximality of the step. The fact that \( T \) is inseparable guarantees causality. As an example, let us relabel transitions \( t_1 \) and \( t_2 \) of the statechart \( z_1 \) in Figure 7.2 by \( a/b \) and \( b/a \), respectively, and let us consider the initial configuration of \( z_1 \). Given \( T = \emptyset \) and \( T' = \{ t_1, t_2 \} \), we have \( T = \text{Disabled}(\emptyset, T) \) and \( T' = \text{Disabled}(\emptyset, T') \), but only \( T \) is inseparable for the empty set of signals. So, \( T \) is a reaction to an environment prompting the empty set of signals. This is not the case for \( T' \). Note that \( T' \) is not a step because \( t_1 \) and \( t_2 \) justify each other, but they are not justified by the environment.

Given a term \( p \) and a set of signals \( S \), the configuration reached from \( \text{active}(p) \) by \( T \in \text{Step}(S) \) consists of the set of states
\[
(\text{active}(p) \setminus \{ s \mid s \prec^* \text{out}(t) \text{ for some } t \in T \}) \cup \bigcup_{t \in T, \text{root}(p) = \text{in}(t)} \text{def}(p).
\]

We have presented a semantics that slightly differs w.r.t. the classical step semantics of [80]. In [80] the function \( \text{Disabled} \) is defined as follows:
\[
\text{Disabled}(S, T) = \text{Rel} \cap \text{Cons}(T) \cap \text{Trig}(S \cup \bigcup_{t \in T} \text{act}(t))
\]

As an example, we see the consequences of our definition of set of enabled transitions in the case of the statechart \( z \) of Figure 7.3. Let us assume that in the default configuration \( z \) receives signal \( a \) and not signal \( b \). While in the semantics of [80] only \( \{ t_1 \} \) would be a step, in our semantics both \( \{ t_1 \} \) and \( \{ t_2 \} \) are steps. Since Statecharts admits nondeterminism, we believe that it is reasonable that the system may choose between performing \( t_1 \) and performing \( t_2 \) (performing both transitions would violate global consistency) without disadvantaging transitions triggered by a negated signal, as it is implied by [80].

Moreover, if in the initial configuration \( z \) receives neither signal \( a \) nor signal \( b \), following the semantics of [80] the statechart fails, because there is no solution of the equation \( T = \text{Disabled}(\emptyset, T) \). Moreover, no method to detect failures is suggested in [80] and failures are not considered at all. Instead we allow execution of step \( \{ t_2 \} \). For our choice we have an intuitive explanation. We can interpret names of signals as names of shared boolean variables. A transition having a signal in his trigger (resp. action) reads (resp. writes) the corresponding variable. So, in order to be enabled, transitions compete for these shared variables. Readings may be concurrent.
and the same holds for writings (this because only one value can be written). On
the contrary, whenever a variable has been read a writing that changes the value of
the variable is not allowed (in the same step). This guarantees that all transitions
in a step read the same value in a variable.

Another method to avoid failures has been proposed in [76]. The semantics of
[76] coincides with that of [80], but, when failures are detected, the step consisting
of the empty set of transitions is performed.

Analogously to [80], we give a procedure to compute a step:

**Procedure** Step-Construction\((\text{in } S : 2^S)\)

\[
T := \emptyset;
\]

while \(T \subset 
\]

\[
\begin{align*}
\text{begin} & \quad \text{choice } t \in Enabled(S, T) \setminus T; \\
& \quad T := T \cup \{t\}
\end{align*}
\]

end

The procedure Step-Construction computes steps, as stated by the following propo-sition.

**Proposition 7.2.1** Given a set of signals \(S\) and a set of transitions \(T\), the procedure
Step-Construction may return \(T\) if and only if \(T \in \text{Step}(S)\).

**Proof** First of all we note that if a transition \(t\) is such that \(t \in Enabled(S, T')\),
for some set of transitions \(T'\) with \(T' \subset 
\]

\[
\begin{align*}
\text{choice } t \in Enabled(S, T') \setminus T'; \\
T := T' \cup \{t\}
\end{align*}
\]

Now, if \(T\) is a step then it is inseparable. Therefore, given \(T' \subset T\),
\(Enabled(S, T') \cap (T \setminus T') \neq \emptyset\). This means that the procedure, starting with the

Figure 7.3: A statechart that fails according to [80].
empty set of transitions, at each cycle can choose a transition in \( T \). When \( T \) has been constructed, Step-Construction terminates and returns \( T \).
This completes the proof.

## 7.3 The labeled transition system

In this section we propose a labeled transition system as an operational semantic model for Statecharts. LTS states correspond to statechart terms, LTS transitions correspond to \textit{substeps} (namely, subsets of steps), and LTS labels carry information on the status of signals and on signal causality.

While LTS transitions in LTSs for Esterel and \texttt{tdccp} correspond to reactions, LTS transitions in the LTS for Statecharts correspond to substeps, namely subsets of reactions. The reason is that a step of a statechart consisting of two components running in parallel is not, in general, the union of steps of the two components. As an example, let us consider the statechart \( z_2 \) in Figure 7.2. The set of transitions \( \{t_1, t_2\} \) is a step of \( z_2 \), but \( \{t_1\} \) is not a step of \( z_1 \). In fact, the possible steps of \( z_1 \) are \( \emptyset \), \( \{t_2\} \) and \( \{t_1, t_2\} \).

We begin with introducing some notations.

An \textit{event} \( A \) (over \( \mathcal{S} \)) is a subset of \( \mathcal{S} \cup \overline{\mathcal{S}} \) such that, for no signal \( a \in \mathcal{S} \), both \( a \) and \( \overline{a} \) are in \( A \). An event \( A \) is interpreted as an assumption over signals in \( \mathcal{S} \), namely a signal \( a \) is assumed to be present if \( a \in A \), while it is assumed to be absent if \( \overline{a} \in A \). We do not introduce symbols \( a^+ \) and \( a^- \) to denote presence and absence of signals, contrarily to what is done in Chapter 3. The reason is that Statecharts, contrarily to Esterel, offers notation to denote absence of signals, and we adopt such a notation.

**Definition 7.3.1** Given an event \( A \) and a set of signals \( B \) such that either \( B = \{b\} \) for some \( b \in \mathcal{S} \) or \( B = \emptyset \), the pair \( (A, B) \) is a \textit{causality term} with \( A \) as \textit{cause} and \( B \) as \textit{action}.

Given a causality term \( (A, B) \), action \( B \) refers to the action of producing signals in \( B \). This action is performed by a statechart transition having \( A \) as trigger. So, causality terms reflect causality relations between signals.

A set of causality terms \( \mathcal{E} \) is \textit{complete} if, given \( (A, B), (A', B') \in \mathcal{E} \) such that \( b \in B \cap A' \), then \( (A' \setminus \{b\}) \cup A, B' \in \mathcal{E} \).

Given a set of causality terms \( \mathcal{E} \), we denote with \( \mathcal{E}^+ \) the set of causality terms such that \( \mathcal{E} \subseteq \mathcal{E}^+ \), \( \mathcal{E}^+ \) is complete, and there exists no complete set of causality terms \( \mathcal{E}' \) such that \( \mathcal{E} \subseteq \mathcal{E}' \subseteq \mathcal{E}^+ \).

In practice, \( \mathcal{E}^+ \) is the transitive closure of the causality relation corresponding to \( \mathcal{E} \).

Note that this notion of complete set of causality terms and of operation \( \mathcal{E}^+ \) over sets of causality terms coincide with those of Chapter 6, provided that the constraints system \textit{Gentzen} is considered.
Definition 7.3.2 A label is a tuple \( l = \langle E_l, Y_l, Z_l \rangle \) such that:

- \( E_l \) is a complete set of causality terms such that \( \bigcup_{(A, B) \in E_l} A \cup B \) is an event;
- \( Y_l \) is a subset of \( S \) such that \( Y_l \cap \bigcup_{(A, B) \in E_l} A \cup B = \emptyset \);
- \( Z_l \) is an event such that \( Z_l \cap \bigcup_{(A, B) \in E_l} A \cup B = \emptyset \) and \( Z_l \cap Y_l = \emptyset \).

We will denote with \( \mathcal{L} \) the set of labels as in Def. 7.3.2.

Let us assume terms \( p \) and \( p' \) and an LTS transition \( p \xrightarrow{t} p' \). Component \( E_l \) describes the causality relationships between signals triggering statechart transitions in the considered substep and signals belonging to their actions. Components \( Y_l \) and \( Z_l \) justify the nonexecution of other statechart transitions relevant in configuration \( \text{active}(p) \). Component \( Y_l \) contains \( b \) if a transition relevant in \( \text{active}(p) \) does not fire to avoid production of \( b \), so that the substep corresponding to \( p \xrightarrow{t} p' \) can be joined with a substep containing a statechart transition having \( b \) in its trigger. The set \( Z_l \) contains \( a \) (resp. \( \overline{a} \)) if a transition relevant in \( \text{active}(p) \) is not triggered because \( a \) is absent (resp. present).

The LTS giving the operational semantics for Statecharts if defined by the transition system specification in Table 7.1.

---

The LTS for Statecharts.

\[
\begin{array}{ll}
\hline
[n] \xrightarrow{\emptyset,B,\emptyset} [n] & \text{(bas)} \\
\hline
[n: (p_1, \ldots, p_k), p, i, T] \xrightarrow{\text{lab}(A,B)} [n: (p_1, \ldots, p_k), p, j, T] & \langle t, i, A, B, j \rangle \in T \text{ \quad (or1)} \\
\hline
[n: (p_1, \ldots, p_k), p, i, T] \xrightarrow{E_l \neq \emptyset} [n: (p_1, \ldots, p_k), p', i, T] & \text{ (or2)} \\
\hline
[n: (p_1, \ldots, p_k), p, i, T] \xrightarrow{E_l = \emptyset} [n: (p_1, \ldots, p_k), p', i, T] & \langle l', i \rangle \in N(i, T), l \uparrow l' \text{ \quad (or3)} \\
\hline
[n: p_1, p_2] \xrightarrow{l_1 \otimes l_2} [n: p'_1, p'_2] & l_1 \uparrow l_2 \quad \text{(and)} \\
\hline
\end{array}
\]

- **Table 7.1**: The labeled transition system for Statecharts.

Rule \textit{bas} states that a statechart consisting of a basic state does nothing at every instant.
CHAPTER 7. SOS FOR STATECHARTS

Given an event $A$ and a set of signals $B$, let us denote with $\text{lab}(A, B)$ the label:

$$
\text{lab}(A, B) = \begin{cases} 
\{\{A, \{b\}\} | b \in B, \emptyset, \emptyset\} & \text{if } B \neq \emptyset \\
\{\{A, \emptyset\}, \emptyset, \emptyset\} & \text{if } B = \emptyset.
\end{cases}
$$

Rule or.1 states that if the environment prompts every signal $a$ such that $a \in A$ and does not prompt any signal $a$ such that $\overline{a} \in A$, then term $[n; (p_1, \ldots, p_k), p, i, T]$ such that $\langle t, i, A, B, j \rangle \in T$ may evolve to term $[n; (p_1, \ldots, p_k), p_j, j, T]$. Transition $[n; (p_1, \ldots, p_k), p, i, T] \xrightarrow{\text{lab}(A, B)} [n; (p_1, \ldots, p_k), p_j, j, T]$ represents the step formed by the statechart transition $\{t\}$. This is a complete step because transitions relevant in $\text{active}(p_i)$ are not consistent with $t$. When step $\{t\}$ is executed, configuration $\{n\} \cup \text{active}(p)$ is left and configuration $\{n\} \cup \text{def}(p_j)$ is reached.

Let us assume rule or.2. If $\mathcal{E}_i \neq \emptyset$ then the LTS transition $p \xrightarrow{\tau} p'$ represents a non empty substep of the $i$th orthogonal component of state $n$. If this substep is a complete step, then it is also a complete step of the statechart having $n$ as root state, because it contains transitions that are not consistent with those having $\text{root}(p_i)$ as source state.

So, given an or-state $n$, rule or.1 deals with steps consisting of single upper level statechart transitions, while rule or.2 deals with non empty substeps internal to subtrees of $n$.

In rule and we assume that, given labels $l_1$ and $l_2$, $l_1 \otimes l_2$ denotes the tuple

$$
l_1 \otimes l_2 = \langle \mathcal{E}_{i_1} \cup \mathcal{E}_{i_2} \rangle^+, Y_{i_1} \cup Y_{i_2}, Z_{i_1} \cup Z_{i_2} \rangle
$$

and $l_1 \uparrow l_2$ denotes that $l_1 \otimes l_2$ is a label in $\mathcal{L}$. Rule and states that a substep of two components running in parallel is the union of two substeps of such components.

Let us consider now rule or.3. Given a statechart transition $t$ with label $(A, B)$, we denote with $N(t)$ the set of LTS labels $\{\emptyset, \{b\}, \emptyset\} | b \in B \setminus A\} \cup \{\emptyset, \emptyset, \gamma\} | \gamma \in A\}$. An LTS transition with label $\langle \emptyset, \{b\}, \emptyset\rangle$ represents that $t$ does not fire in order to avoid production of $b$. Label $\langle \emptyset, \{b\}, \emptyset\rangle$ can be composed (w.r.t $\otimes$) with an LTS transition label representing a substep containing a statechart transition having $b$ in the trigger. An LTS transition with label $\langle \emptyset, \emptyset, \{a\}\rangle$ (resp. $\langle \emptyset, \emptyset, \{\overline{a}\}\rangle$) represents that $t$ cannot fire because $a$ is absent (resp. present).

Given a set $T \subseteq T \times \{1, \ldots, k\} \times 2^{S \times S} \times 2^{S \times \{1, \ldots, k\}}$ and an index $1 \leq i \leq k$ such that $\{t_1, \ldots, t_h\} = \bigcup_{(i, i_1, A, B, i_2) \in T | i_1 = i} T | i_1 = i$, we denote with $N(i, T)$ the set of labels such that:

$$
N(i, T) = \begin{cases} 
\{l_1 \otimes \ldots \otimes l_h \in \mathcal{L} | l_i \in N(t_i)\} & \text{if } h > 0 \\
\{\emptyset, \emptyset, \emptyset\} & \text{otherwise.}
\end{cases}
$$

Given a term $[n; (p_1, \ldots, p_k), p, i, T]$, an LTS transition label $l \in N(i, T)$ justifies that no transition having $\text{root}(p_i)$ as source state fires. Note that $\mathcal{E}_i = \emptyset$.

Rule or.3 states that when the configuration represented by $[n; (p_1, \ldots, p_k), p, i, T]$ is active, the empty substep is performed if no upper level transition fires and no
transition in \( \text{trans}(p_i) \) fires.

In Figure 7.4.c we show the LTS for the statechart \( z_1 \) of Figure 7.2. In Figure 7.4.a and in Figure 7.4.b we show the LTS’s for the subterms rooted in the states 3 and 6, respectively. We have labeled every LTS state with the names of the FSM states giving the corresponding statechart configuration.

Let us consider configuration \( \{1, 3, 4, 6, 7\} \). LTS transition with label \( l_{12} \) represents the empty substep, justified by the absence of both \( a \) and \( b \). LTS transition with label \( l_{13} \) represents the empty substep, justified by the fact that \( b \) is absent, so that \( t_2 \) cannot fire, and \( b \) cannot be produced, so that \( t_1 \) cannot fire. LTS transition with label \( l_{14} \) represents the empty substep, justified by the fact that neither \( b \) nor \( c \) can be produced. LTS transition with label \( l_{15} \) represents the empty substep, justified by the fact that \( a \) is absent, so that \( t_1 \) cannot fire, and \( c \) cannot be produced, so that \( t_2 \) cannot fire. LTS transition with label \( l_9 \) represents the step \( \{t_1, t_2\} \). LTS transition with label \( l_5 \) represents the step \( \{t_2\} \) \( (t_1 \) does not fire because \( a \) is absent). Finally, LTS transition with label \( l_8 \) represents the step \( \{t_1\} \) \( (t_2 \) does not fire to avoid production of \( c \)). Consider now the statechart \( z_2 \) of Figure 7.2 obtained by adding another component to the statechart \( z_1 \). In the corresponding LTS there will be a transition representing the step \( \{t_1, t_3\} \) obtained by completing the substep \( \{t_1\} \). This compositional construction of step \( \{t_1, t_3\} \) could not be done if we would not consider the LTS transition with label \( l_8 \).
We note that semantics for Statecharts in terms of LTSs have been already proposed in \cite{97, 98, 64, 60, 58, 62}.

The LTSs of \cite{97, 98} were intended to reflect statechart behaviors according to the step semantics of \cite{80}. Unfortunately, the agreement between LTS and step semantics fails, as argued in \cite{64, 58}.

An LTS semantics correctly reflecting the step semantics of \cite{80} has been given in \cite{58, 60}, following \cite{64}. In \cite{60} a compositional proof system for Statecharts has been developed. The idea is to consider propositional \(\mu\)-calculus \cite{81, 57} formulas where actions have the same structure of LTS labels, and to take LTSs as models of the logic. The construction of the LTS is well-suited for the compositional reduction of formulas in form \(\langle \alpha \rangle \phi\), where \(\langle \neg \rangle\) is the “next time” modality.

In \cite{64} we proposed both the step semantics described in Section 7.2 and an LTS semantics, and we stated their correspondence. The LTS semantics of this section slightly differs w.r.t. to that of \cite{64}. In fact, in \cite{64}, no rule analogous to rule \textit{bas} of Table 7.1 exists. As a consequence, while in \cite{64} a configuration with no relevant statechart transition is represented by a term \(p\) such that \(p \not\rightarrow\), the same configuration is represented here by a term \(p\) such that \(p \rightarrow\). Therefore, while in \cite{64} we needed rules with negative premises, like the following one

\[
\frac{p_1 \xrightarrow{t_1} p'_1 \quad p_2 \xrightarrow{t_2} \quad p_3}{[n:p_1, p_2] \xrightarrow{t_1} [n:p'_1, p_2]}
\]

the transition system specification of Table 7.1 does not contain rules with negative premises. We will exploit this fact to prove the property of precongruence of some preorders that we will introduce in Section 7.5.

In \cite{62} a process algebra called \textit{SPL} (Statecharts Process Algebra) is presented, and statecharts are compositionally encoded by \textit{SPL} terms. The operational semantics of \textit{SPL} is expressed in terms of an LTS with two types of transitions: \textit{action} transitions and \textit{clock} transitions. Action transitions correspond to performing statechart substeps, while clock transitions correspond to the progress of time. Action and clock transitions are not orthogonal concepts, but are connected via the \textit{ maximal progress assumption} \cite{52, 100}, namely the assumption that time progresses only if all internal computations have been completed. All components of a \textit{SPL} term are forced to take part to a clock transition. A statechart step corresponds to a \textit{chain} of action transitions, corresponding to a chain of substeps of the step, enclosed by two clock transitions. In \cite{62} it is proved that \textit{SPL} embeds Statecharts endowed with the step semantics of Section 7.2, and it is hinted at how other variants of Statecharts can be embedded. In practice, an LTS transition of \cite{64} corresponds to a chain of LTS action transitions, enclosed by two clock transitions, of \cite{62}.
7.4 Correspondence between SOS and step semantics

In this section we show the correspondence between our SOS semantics and our step semantics. Namely, we prove that our LTS carries sufficient information to deduce the input/output behavior of statecharts.

Given a set of signals $S$ and a complete set of causality terms $\mathcal{E}$, we say that $S$ triggers $\mathcal{E}$ if there exists a subset $\mathcal{E}'$ of $\mathcal{E}$ such that $\bigcup_{(A,B) \in \mathcal{E}'} A \cap S \subseteq S$, $\bigcup_{(A,B) \in \mathcal{E}'} A \cap S \cap S = \emptyset$ and $S \cap \bigcup_{(A,B) \in \mathcal{E}'} A \cup B = S \cap \bigcup_{(A,B) \in \mathcal{E}'} A \cup B$.

Intuitively, if $S$ triggers $\mathcal{E}$, then $S$ causally justifies all signals in $\bigcup_{(A,B) \in \mathcal{E}} A \cup B$.

This idea is formalized in the following lemma.

Lemma 7.4.1 Given relevant, pairwise consistent and compatible statechart transitions $t_1, \ldots, t_n$ labeled $(A_1, B_1), \ldots, (A_n, B_n)$, respectively, $B_i = \{b_{i,1}, \ldots, b_{i,m_i}\}$, $m_i \geq 0$, $1 \leq i \leq n$, the following facts are equivalent:

- $\{t_1, \ldots, t_n\} \subseteq \text{Enabled}(S, \{t_1, \ldots, t_n\})$ and $\{t_1, \ldots, t_n\}$ is inseparable for $S$;
- $S$ triggers $(\mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n)^+$, $\mathcal{E}_i = \begin{cases} (A_i, \{b_{i,1}\}, \ldots, (A_i, \{b_{i,m_i}\}) & \text{if } m_i > 0 \\ (A_i, \emptyset) & \text{if } m_i = 0. \end{cases}$

Proof By induction over $n$.

Basic case: $n = 1$. Since $a_1 \not\in B_1$, if $a_1 \in A_1$, we have that $\{t_1\} \subseteq \text{Enabled}(S, \{t_1\})$ and $\{t_1\}$ is inseparable for $S$ if and only if $(A_1 \cap S) \subseteq S$ and $A_1 \cap \overline{S} \cap S = \emptyset$. This is equivalent to having that $\mathcal{E}_1$ is a complete set of causality terms and $S$ triggers it.

Induction step: $n > 1$. Let $T$ be the set $\{t_1, \ldots, t_{n+1}\}$. Prop. 7.2.1 implies that $T \subseteq \text{Enabled}(S, T)$ and $T$ is inseparable for $S$ if and only if procedure Step-Construction on input $S$ can construct $T$ and it will return a superset of $T$. This means that there exists $1 \leq i \leq n + 1$ such that $(A_i \cap S) \subseteq S$, $A_i \cap \overline{S} \cap S = \emptyset$, $T \setminus \{t_i\} \subseteq \text{Enabled}(S \cup B_i, T \setminus \{t_i\})$ and $T \setminus \{t_i\}$ is inseparable for $S \cup B_i$. By inductive hypothesis, $T \setminus \{t_i\} \subseteq \text{Enabled}(S \cup B_i, T \setminus \{t_i\})$ and $T \setminus \{t_i\}$ is inseparable for $S \cup B_i$ if and only if $S \cup B_i$ triggers $(\mathcal{E}_1 \cup \ldots \cup \mathcal{E}_{i-1} \cup \mathcal{E}_{i+1} \cup \ldots \cup \mathcal{E}_{n+1})^+$. If $(A_i \cap S) \subseteq S$ and $A_i \cap \overline{S} \cap S = \emptyset$, this is equivalent to having that $S$ triggers $(\mathcal{E}_1 \cup \ldots \cup \mathcal{E}_{n+1})^+$. This completes the proof.

Given a set of signals $S$ and an LTS transition $p \xrightarrow{L} p'$, we say that $S$ triggers $p \xrightarrow{L} p'$ if $S$ triggers $\mathcal{E}_i$, $S \cap Y_i = \emptyset$, $S \cap Z_i = \emptyset$ and $a \in S$ for every $\pi \in Z_i$ such that $a \not\in \bigcup_{(A,B) \in \mathcal{E}_i} A \cup B$.

Namely, $S$ triggers $p \xrightarrow{L} p'$ if $S$ triggers $\mathcal{E}_i$, $S$ does not contain signals in $Y_i$, whose production must be avoided, $S$ does not contain signals in $Z_i$, which are assumed to be absent, and $S$ contains signals that are assumed to be present and are neither in the trigger nor in the action of the transitions fired.
The following theorem establishes the correspondence between SOS and step semantics.

**Theorem 7.4.2** Given a term $p$, a set of signals $S$, and a set of relevant, pairwise consistent and compatible transitions $\{t_1, \ldots, t_n\}$ labeled $(A_1, B_1), \ldots, (A_m, B_m)$, respectively, with $B_i = \{b_{i,1}, \ldots, b_{i,m_i}\}$, $m_i \geq 0$, $1 \leq i \leq n$, the following facts are equivalent:

- $\{t_1, \ldots, t_n\} \in \text{Step}(S)$ and a configuration $C$ is reached from $\text{active}(p)$ by $\{t_1, \ldots, t_n\}$;
- $S$ triggers an LTS transition $p \xrightarrow{t} p'$ such that:
  
  \[ - \mathcal{E}_t = (\mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n)^+, \mathcal{E}_i = \begin{cases} 
  \{(A_i, \{b_{i,1}\}), \ldots, (A_i, \{b_{i,m_i}\})\} & \text{if } m_i > 0 \\
  \{(A_i, \emptyset)\} & \text{if } m_i = 0.
  \end{cases} \]

  \[- \text{active}(p') = C; \]
  
  - for each $b \in Y_i$ there exists some $1 \leq i \leq n$ such that $\overline{b} \in A_i$.

**Proof** Let $\{t_{n+1}, \ldots, t_m\}$ be the set of transitions relevant in $\text{active}(p)$ and consistent with $\{t_1, \ldots, t_n\}$ such that $\{t_1, \ldots, t_n\} \cap \{t_{n+1}, \ldots, t_m\} = \emptyset$. Let $(A_i, B_i)$ be the label of $t_i$, $n + 1 \leq i \leq m$.

Now, $\{t_1, \ldots, t_n\} \in \text{Step}(S)$ if and only if $\{t_1, \ldots, t_n\} \in \text{Enabled}(S, \{t_1, \ldots, t_n\})$ and $\{t_1, \ldots, t_n\}$ is inseparable for $S$. This implies that, for every $n + 1 \leq i \leq m$, either $t_i$ is not triggered by $S \cup \bigcup_{1 \leq j \leq n} \text{act}(t_j)$, or $t_i$ is not compatible with some $t_j$, $1 \leq j \leq n$.

Given $l_i = \text{lab}(A_i, B_i)$ and $l' = l_1 \otimes \ldots \otimes l_n$, from Lemma 7.4.1 we infer that $\{t_1, \ldots, t_n\} \subseteq \text{Enabled}(S, \{t_1, \ldots, t_n\})$ and $\{t_1, \ldots, t_n\}$ is inseparable for $S$ if and only if $S$ triggers $\mathcal{E}_t$. Moreover, for every $n + 1 \leq i \leq m$, if $t_i$ is not triggered by $S \cup \bigcup_{1 \leq j \leq n} \text{act}(t_j)$, then there exists a label $l_i = \langle \emptyset, \emptyset, \{\gamma_i\} \rangle \in N(t_i)$ such that either $\gamma_i = a$ and $a \notin S \cup \bigcup_{1 \leq j \leq n} \text{act}(t_j)$, or $\gamma_i = \overline{a}$ and $a \in S \cup \bigcup_{1 \leq j \leq n} \text{act}(t_j)$. Otherwise, if $t_i$ is triggered by $S \cup \bigcup_{1 \leq j \leq n} \text{act}(t_j)$ but is not compatible with some $t_j$, $1 \leq j \leq n$, there exists a LTS label $l_i = \langle \emptyset, \{b\}, \emptyset \rangle \in N(t_i)$ such that $\overline{b} \in A_j$.

These facts are equivalent to having that $S$ triggers $p \xrightarrow{t} p'$, with $l = l' \otimes l_{n+1} \otimes \ldots \otimes l_m$, and that for each $b \in Y_i$ there exists some $1 \leq i \leq n$ such that $\overline{b} \in A_i$.

The fact that configuration $C$ is reached via $\{t_1, \ldots, t_n\}$ if and only if $\text{active}(p') = C$ follows directly by rules in Table 7.1 and could be proved by induction over $p$.

This completes the proof.

In Theorem 7.4.2 the requirement that if $b \in Y_i$ then there exists some $1 \leq i \leq n$ such that $\overline{b} \in A_i$ ensures that $p \xrightarrow{t} p'$ represents a complete step and not a strict substep of a step. As an example, let us consider statechart $z_1$ of Figure 7.2 in configuration $\{1, 3, 4, 6, 7\}$ and $S = \{a\}$. We have that $\{t_1, t_2\} = \text{Step}(S)$. Now, if we consider the LTS in Figure 7.4.c, $S$ triggers $s \xrightarrow{t} t$ and $s \xrightarrow{t} v$. LTS transition
labeled by \( l_9 \) represents step \( \{t_1, t_2\} \) and \( Y_{l_9} = \emptyset \). LTS transition labeled by \( l_8 \) represents the incomplete substep \( \{t_1\} \), \( Y_{l_8} = \{c\} \) and \( \pi \not\in \bigcup_{\{A, B\} \in \mathcal{E}_{l_8}} A \).

7.5 Equivalences

Many different theories of equivalences have been proposed in the literature for models which are intended to be used to describe and reason about concurrent or nondeterministic systems. This is mainly due to the large number of properties which may be relevant in the analysis of such systems. Almost all the proposed equivalences are based on the idea that two systems are equivalent whenever no external observer can distinguish them. In fact, for any given system it is not its internal structure which is of interest but its behavior with respect to the outside world, namely its effect on the environment and its reactions to stimuli from the environment.

The various equivalences proposed assume different concepts of observation, namely they consider different aspects of the behavior of systems as being of interest.

As we have already argued in the previous chapters, the bisimulation equivalence relates two processes if they have the same branching structure. In the theory of process calculi many coarser notions of equivalence have been proposed to release this requirement at different degrees.

Let us recall the notions of simulation [74] and ready simulation [19] over LTS states. As in Chapter 2, we denote with \( s_0, s_1, \ldots \) LTS states, and with \( \text{Act} \) the set of actions ranged over by \( a_1, a_2, \ldots \).

**Definition 7.5.1** Let us assume an LTS. A binary relation \( \mathcal{R} \) on states is a *simulation* if, whenever \( s_1 \mathcal{R} s_2 \), if \( s_1 \xrightarrow{a_1} s'_1 \) then there exists a transition \( s_2 \xrightarrow{a} s'_2 \) such that \( s'_1 \mathcal{R} s'_2 \).

A binary relation \( \mathcal{R} \) on states is a *ready simulation* if it is a simulation and, whenever \( s_1 \mathcal{R} s_2 \), if \( s_1 \xrightarrow{a} s' \) then \( s_2 \xrightarrow{a} s' \).

Given LTS states \( s_1 \) and \( s_2 \), we write \( s_1 \preceq_S s_2 \) if there exists a simulation \( \mathcal{R} \) with \( s_1 \mathcal{R} s_2 \), and we write \( s_1 \preceq_{RS} s_2 \) if there exists a ready simulation \( \mathcal{R} \) with \( s_1 \mathcal{R} s_2 \).

Note that a simulation \( \mathcal{R} \) is a bisimulation if and only if it is symmetric. Relations \( \preceq_S \) and \( \preceq_{RS} \) are preorders.

Let us introduce now the notions of ready trace, trace and failure of an LTS state.

**Definition 7.5.2** Let us assume an LTS.

- A sequence \( X_0 a_1 X_1 \ldots a_n X_n \) with \( X_i \subseteq \text{Act} \) and \( a_i \in \text{Act} \) is a *ready trace* of a state \( s_0 \) if \( s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots s_{n-1} \xrightarrow{a_n} s_n \) and \( \text{initials}(s_i) = X_i \) for \( i = 0, \ldots, n \). We write \( s \preceq_{RT} s' \) if the set of ready traces of state \( s \) is included in that of state \( s' \).
• A pair \((\sigma, X)\) with \(\sigma \in \text{Act}^*\) and \(X \subseteq \text{Act}\) is a failure of a state \(s\) if \(s \xrightarrow{\sigma} s'\) for some state \(s'\) such that \(\text{initials}(s') \cap X = \emptyset\). We write \(s \sqsubseteq_F s'\) if the set of failures of state \(s\) is included in that of state \(s'\).

• A sequence \(a_1 \ldots a_n\) with \(a_i \in \text{Act}\) is a trace of a state \(s_0\) if \(s_0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n\) for some \(s_n\). We write \(s \sqsubseteq_T s'\) if the set of traces of state \(s\) is included in that of state \(s'\).

Relations \(\sqsubseteq_{RT}\), \(\sqsubseteq_F\) and \(\sqsubseteq_T\) are preorders, called ready trace preorder, failure preorder and trace preorder, respectively.

We will denote with \(\approx_{RT}\), \(\approx_F\) and \(\approx_T\) the kernels of preorders \(\sqsubseteq_{RT}\), \(\sqsubseteq_F\) and \(\sqsubseteq_T\), respectively.

As established in [36], the following relations hold:

\[
\approx \subseteq \sqsubseteq_{RS} \subseteq \sqsubseteq_S \\
\approx \subseteq \sqsubseteq_{RS} \subseteq \sqsubseteq_{RT} \subseteq \sqsubseteq_F \subseteq \sqsubseteq_T
\]

In chapters 3 and 6 we have not considered the preorders recalled in Def. 7.5.1 and Def. 7.5.2. The reason is that LTSs of Table 3.1 and Table 6.2 are such that, for every state \(s\) and for every pair of transitions \(s \xrightarrow{a_1} s'\), \(s \xrightarrow{a_2} s''\), we have \(a_1 \neq a_2\), and, therefore, kernels of such preorders coincide with the bisimulation. This follows from the following proposition.

**Proposition 7.5.3** Given an LTS such that for every state \(s\) and for every pair of transitions \(s \xrightarrow{a_1}\), \(s \xrightarrow{a_2}\) we have \(a_1 \neq a_2\), we have \(\approx_T = \approx\).

**Proof** Since \(\approx \subseteq \approx_T\), we must prove that \(\approx \supseteq \approx_T\). To do this, it is sufficient to show that \(\{(s_1, s_2) | s_1 \approx_T s_2\}\) is a bisimulation. Given \(s_1 \approx_T s_2\), \(s_1 \xrightarrow{a} s'_1\) if and only if there exists some \(s'_2\) such that \(s_2 \xrightarrow{a} s'_2\). Since \(s_1 \xrightarrow{a} s\) iff \(s = s'_1\) and \(s_2 \xrightarrow{a} s\) iff \(s = s'_2\), it follows that \(s_2 \approx_T s'_2\). This implies that \(\{(s_1, s_2) | s_1 \approx_T s_2\} \subseteq \mathcal{F}(\{(s_1, s_2) | s_1 \approx_T s_2\})\), namely that \(\{(s_1, s_2) | s_1 \approx_T s_2\}\) is a bisimulation. This completes the proof.

Since statecharts admit nondeterminism, and, as a consequence, it may happen that several LTS transitions having the same state as source state have the same label, we investigate the precongruence property of preorders of Def. 7.5.1 and Def. 7.5.2.

As an example of equivalent terms, terms corresponding to the default configurations of the statecharts in Figure 7.5 are equivalent w.r.t. \(\approx_T\).

From results in [38], it follows that positive GSOS is a precongruence format for both simulation preorder and ready simulation preorder.

Moreover, in [38] the ready trace format has been introduced and it has been proved that it is a precongruence format for ready trace preorder. Given a positive GSOS rule \(\rho\), the variable dependency graph of the premises of \(\rho\) is the graph having
variables appearing in the premises of $\rho$ as nodes, and an arc $x \to y$ for every premise $x \xrightarrow{a} y$ of $\rho$. A positive GSOS rule $\rho$ is in ready trace format if every pair of variables occurring in distinct positions of the target of $\rho$ are not connected in the symmetric closure of the variable dependency graph of the premises of $\rho$.

Theorem 7.5.4 Given the algebra of statechart terms, bisimulation is a congruence, and simulation, ready simulation and ready trace preorder are precongruences.

Proof The thesis follows by the fact that rules in Table 7.1 are in positive GSOS format and satisfy the requirement for the ready trace format.

Note that if a preorder is a precongruence then the kernel of the preorder is a congruence. So, the equivalence $\approx_{RT}$ over statechart terms is a congruence.

We note that in [38] it is required that a transition system specification is in path format [6] to ensure that simulation preorder is a precongruence. While positive GSOS format is more restrictive than path format, namely positive GSOS rules are path rules, this is not the case for the general GSOS format, because path rejects negative premises. Therefore, one cannot exploit the result of [38] to infer that simulation preorder is a precongruence if the transition system specification of [64] is considered.

An interesting issue is to establish whether rules with negative premises are needed to describe behavior of synchronous programs. We recall that there exist variants of Statecharts where priority relations over transitions can be defined. As examples, in [25] a transition having a state $n$ as source state has higher priority w.r.t. to all transitions internal to $n$, and in [63] one can define explicit priority relations over transitions. Explicit priority relations over transitions are analogous to the operator of priority "$\omega$" proposed in [5] for process algebras. Implicit priorities of [25] can be easily expressed by replacing rule $or_2$ of Table 7.1 by the following rule:

$$
\frac{p \xrightarrow{t} p'}{[n:(p_1,\ldots,p_k),p,i,T] \xrightarrow{t \otimes \mathcal{E}_l} [n:(p_1,\ldots,p_k),p',i,T]}
$$

$\mathcal{E}_l \neq \emptyset, l' \in N(i,T), l \uparrow l'$
We conjecture that rules with negative premises are needed to express explicit priorities of [63].

In [99] it is proved that de Simone format is a precongruence format for trace preorder and failure preorder. We cannot exploit this result because rule $or\_I$ in Table 7.1 is not in de Simone format. In fact, two occurrences of variable $p_j$ appear in the target of $or\_I$.

We must give the proof that the trace preorder is a precongruence.

**Theorem 7.5.5** The trace preorder on statechart terms is a precongruence.

**Proof** Let us assume that $p \sqsubseteq_T p'$. First of all we prove that, given an arbitrary term $p_2$, we have that $[n : p, p_2] \sqsubseteq_T [n : p', p_2]$. We have $[n : p, p_2] \xrightarrow{l_1} \ldots \xrightarrow{l_n}$ if and only if $p \xrightarrow{u_1} \ldots \xrightarrow{u_n}$, $p_2 \xrightarrow{v_1} \ldots \xrightarrow{v_n}$ and $l_i = u_i \otimes v_i$, $1 \leq i \leq n$. Since $p \sqsubseteq_T p'$, $p \xrightarrow{u_1} \ldots \xrightarrow{u_n}$ implies $p' \xrightarrow{u_1} \ldots \xrightarrow{u_n}$, and, therefore, $[n : p', p_2] \xrightarrow{l_1} \ldots \xrightarrow{l_n}$, as required.

Now we prove that, given arbitrary terms $p_1, \ldots, p_{k-1}, p_k, \ldots, p_k$, an arbitrary index $1 \leq \alpha \leq k$, an arbitrary set $T \subseteq T \times \{1, \ldots, k\} \times 2^S \times 2^S \times \{1, \ldots, k\}$, and an arbitrary term $\hat{p}$ such that $active(\hat{p}) \in conf(p_\alpha)$, we have that:

$[n : (p_1, \ldots, p_{k-1}, p, p_{k+1}, \ldots, p_m), \hat{p}, \alpha, T] \sqsubseteq_T [n : (p_1, \ldots, p_{k-1}, p', p_{k+1}, \ldots, p_m), \hat{p}, \alpha, T]$.

It is sufficient to prove that $[n : (p_1, \ldots, p_{k-1}, p, p_{k+1}, \ldots, p_m), p, i, T] \xrightarrow{l_1} q_1 \ldots \xrightarrow{l_n} q_n$, with $q_j = [n : (p_1, \ldots, p_{k-1}, p, p_{k+1}, \ldots, p_m), \hat{p}_j, i, T]$ for $1 \leq j \leq n$, implies that

$[n : (p_1, \ldots, p_{k-1}, p', p_{k+1}, \ldots, p_m), p', i, T] \xrightarrow{l_1} q'_1 \ldots \xrightarrow{l_n} q'_n$, with $q'_j = [n : (p_1, \ldots, p_{k-1}, p', p_{k+1}, \ldots, p_m), \hat{p}_j, i, T]$ for $1 \leq j \leq n$.

This holds, because $[n : (p_1, \ldots, p_{k-1}, p, p_{k+1}, \ldots, p_m), p, i, T] \xrightarrow{l_1} q_1 \ldots \xrightarrow{l_n} q_n$ if and only if $p \xrightarrow{u_1} \hat{p}_1 \ldots \xrightarrow{u_n} \hat{p}_n$ and either $l_i = u_i$ or $l_i = u_i \otimes v_i$ and $v_i \in N(i, T)$.

Since $p \sqsubseteq_T p'$, this implies that $p' \xrightarrow{u_1} \hat{p}'_1 \ldots \xrightarrow{u_n} \hat{p}'_n$. This is sufficient to infer that $[n : (p_1, \ldots, p_{k-1}, p', p_{k+1}, \ldots, p_m), p', i, T] \xrightarrow{l_1} q'_1 \ldots \xrightarrow{l_n} q'_n$ with $q'_j = [n : (p_1, \ldots, p_{k-1}, p', p_{k+1}, \ldots, p_m), \hat{p}'_j, i, T]$ for $1 \leq j \leq n$.

We prove now that the failure preorder is a precongruence.

**Theorem 7.5.6** The failure preorder on statechart terms is a precongruence.

**Proof** Let us assume that $p \sqsubseteq_F p'$. First of all we prove that, given an arbitrary term $p_2$, we have that $[n : p, p_2] \sqsubseteq_F [n : p', p_2]$. We have $[n : p, p_2] \xrightarrow{l_1} \ldots \xrightarrow{l_n} \hat{p}$ and $initials(\hat{p}) \cap L = \emptyset$ if and only if $p \xrightarrow{u_1} \ldots \xrightarrow{u_n} \hat{p}_1, p_2 \xrightarrow{v_1} \ldots \xrightarrow{v_n} \hat{p}_2$, $l_i = u_i \otimes v_i$, $1 \leq i \leq n$, and $initials(\hat{p}_1) \cap L_1 = \emptyset$, for $L_1 = \{ l \subset l' \in L \text{ and } \hat{p}_2 \xrightarrow{l'} \}$.

Since $p \sqsubseteq_F p'$, $p \xrightarrow{u_1} \ldots \xrightarrow{u_n} \hat{p}_1$ and $initials(\hat{p}_1) \cap L_1 = \emptyset$ implies $p' \xrightarrow{u_1} \ldots \xrightarrow{u_n} \hat{p}'_1$, and $initials(\hat{p}'_1) \cap L_1 = \emptyset$ for some $\hat{p}'_1$, and, therefore, $[n : p', p_2] \xrightarrow{l_1} \ldots \xrightarrow{l_n} \hat{p}'$ and $initials(\hat{p}') \cap L = \emptyset$, as required.
Now we prove that, given arbitrary terms \( p_1, \ldots, p_{i-1}, p, p_{i+1}, \ldots, p_k \), an arbitrary index \( 1 \leq \alpha \leq k \), an arbitrary set \( T \subseteq \mathcal{T} \times \{1, \ldots, k\} \times 2^{SU} \times 2^S \times \{1, \ldots, k\} \), and an arbitrary term \( \hat{p} \) such that \( \text{active}(\hat{p}) \in \text{conf}(p_\alpha) \), we have that:

\[
[n: (p_1, \ldots, p_{i-1}, p, p_{i+1}, \ldots, p_m), \hat{p}, \alpha, T] \subseteq_F [n: (p_1, \ldots, p_{i-1}, p', p_{i+1}, \ldots, p_m), \hat{p}, \alpha, T].
\]

Since \( p \subseteq_F p' \), we have that \( p \subseteq_T p' \). By Theorem 7.5.5 it follows that:

\[
[n: (p_1, \ldots, p_{i-1}, p, p_{i+1}, \ldots, p_m), \hat{p}, \alpha, T] \subseteq_T [n: (p_1, \ldots, p_{i-1}, p', p_{i+1}, \ldots, p_m), \hat{p}, \alpha, T].
\]

Let us assume that \( [n: (p_1, \ldots, p_{i-1}, p, p_{i+1}, \ldots, p_m), \hat{p}, \alpha, T] \xrightarrow{l_1} \ldots \xrightarrow{l_m} [n: (p_1, \ldots, p_{i-1}, p', p_{i+1}, \ldots, p_m), \hat{p}', \alpha', T] \) and \( \text{initials}(n: (p_1, \ldots, p_{i-1}, p, p_{i+1}, \ldots, p_m), \hat{p}', \alpha', T) \cap L = \emptyset \). If \( \alpha' \neq i \) then it is immediate that we have \( [n: (p_1, \ldots, p_{i-1}, p', p_{i+1}, \ldots, p_m), \hat{p}, \alpha, T] \) and \( \text{initials}(n: (p_1, \ldots, p_{i-1}, p, p_{i+1}, \ldots, p_m), \hat{p}', \alpha', T) \cap L = \emptyset \). Otherwise, if \( \alpha' = i \) then we have that \( [n: (p_1, \ldots, p_{i-1}, p', p_{i+1}, \ldots, p_m), \hat{p}, \alpha, T] \xrightarrow{l_1} \ldots \xrightarrow{l_m} [n: (p_1, \ldots, p_{i-1}, p', p_{i+1}, \ldots, p_m), \hat{p}' \alpha', T] \), where \( p \xrightarrow{u_1} \ldots \xrightarrow{u_m} p', \) \( \hat{p} \xrightarrow{u_1} \ldots \xrightarrow{u_m} \hat{p}' \) and either \( l_{n-m+1} = u_1 \) or \( l_{n-m+1} = u_1 \otimes v_1 \) and \( v_1 \in N(i, T) \), ..., either \( l_n = u_m \) or \( l_n = u_n \otimes v_n \) and \( v_n \in N(i, T) \). Now, \( \text{initials}(n: (p_1, \ldots, p_{i-1}, p, p_{i+1}, \ldots, p_m), \hat{p}', \alpha', T) \cap L = \emptyset \) if and only if \( \text{initials}(\hat{p}') \cap L_1 = \emptyset \), where \( L_1 = L \cup \{l | l \otimes v \in L \) and \( v \in N(i, T) \} \). Since \( p \subseteq_F p' \), we have that \( \text{initials}(\hat{p}') \cap L_1 = \emptyset \), and, therefore, \( \text{initials}(n: (p_1, \ldots, p_{i-1}, p', p_{i+1}, \ldots, p_m), \hat{p}', \alpha', T) \cap L = \emptyset \), which implies the thesis.

Note that from Theorem 7.5.5 and Theorem 7.5.6 it follows that the equivalences \( \approx_T \) and \( \approx_F \) are congruences.
Chapter 8

Conclusions

In this thesis we have demonstrated that Plotkin’s structural operational semantics applies well to the class of synchronous languages.

We have followed the SOS approach to provide both “classical” and “distributed” interpretations of some synchronous languages. Classical interpretations describe the behavior of programs in terms of reactions from global configurations to global configurations. On the contrary, distributed interpretations describe the behavior of programs in terms of reactions from distributed configurations to distributed configurations. We speak about distributed interpretations instead of “truly concurrent” ones because true concurrency is usually viewed as a counterpart of interleaving, and speaking about interleaving in the synchronous setting seems to be a nonsense.

We have interpreted the synchronous languages Esterel, tдccp and Statecharts as process algebras. LTSs describing the operational behavior of processes have nodes corresponding to program configurations, arcs corresponding to program evolution steps and labels describing interaction between programs and environment. Technically, this interaction is described by means of information depending on the particular language considered. As an example, the LTS for Esterel must carry information about actions that are not performed by programs, which is not needed in LTSs for tдccp and Statecharts. This difference is a consequence of the fact that Esterel, contrarily to tдccp and Statecharts, is a constructive language.

We have proved the agreement between our LTS interpretations and existing semantics of the considered languages. In particular, our LTS semantics for Esterel agrees with the so called circuit semantics of [16], in the sense that from our LTS we are able both to recover the input/output behavior of circuits corresponding to programs and to deduce whether circuits are constructive. Our LTS semantics for tдccp agrees with the operational semantics of [86], in the sense that from our LTS we can recover the input/output behavior of programs and their properties of determinism and reactivity. Finally, our LTS semantics for Statecharts agrees with a step semantics that we have proposed as a slight variant w.r.to that of [80], in the sense that from our LTS we can recover the input/output behavior of statecharts.

Our classical interpretations could support the development of proof systems for
the considered languages. In fact, given one of these languages, we could follow [59, 60] and provide a compositional proof system for verifying properties specified by propositional \( \mu \)-calculus formulas. We could consider \( \mu \)-calculus formulas where actions have the same structure of our LTS labels, and take the LTS as the model of the logic. The model checking problem could be always reduced to simpler tasks by exploiting both the structure of specifications and the structure of formulas. We believe that our construction of the LTS is well-suited for the compositional reduction of formulas of the form \( \langle \alpha \rangle \phi \), where \( \langle - \rangle \) is the “next time” modality.

By exploiting the format of SOS rules, we have proved that bisimulation on our LTSs is a congruence. The agreement between LTS semantics and existing operational semantics, and the property of congruence of the bisimulation, imply that bisimilar programs are distinguished neither by the external environment nor by any language context. It follows that bisimulation is a reasonable notion of behavioral equivalence on programs. Therefore, it is worth to provide axiomatizations sound and complete modulo bisimulation over the languages treated in this thesis.

We have provided an axiomatization over Esterel sound and complete modulo bisimulation. The major subtlety in giving such an axiomatization is that one cannot transform programs into “head normal forms”, as it is usually done for asynchronous process algebras (as an example, see [70]), because concurrent programs cannot be reduced to sequential ones. This is also the reason for which one cannot exploit algorithms given in [2, 1] to construct axiomatizations over languages belonging to suitable subclasses of the class of GSOS languages. In fact, these algorithms apply to process algebras where a process can be written as a summation of its immediate derivatives. We believe that it is interesting to investigate whether our approach applies to \( \text{tdccp} \) and Statecharts.

We have argued that it is interesting to study preorders and equivalence coarser than bisimulation on Statecharts programs. We have proved that some well known preorders are precongruences, by exploiting the format of the SOS rules when this was possible.

In giving our operational semantics for Esterel, \( \text{tdccp} \) and Statecharts, we have always used SOS rules with positive premises. We have also conjectured that negative premises are needed to express some features that are offered by dialects of Statecharts proposed in the literature. We believe that there exist dialects of Statecharts that, on one side, offer features making the language more expressive, and, on the other side, require SOS rules that do not satisfy formats guaranteeing interesting properties like precongruence of some preorders.

We have advocated distributed interpretations for synchronous languages, even though only classical semantics have been proposed in the literature until now.

We have demonstrated that distributed interpretations can be used to improve hardware implementation. In fact, we have proposed a technique, based on a distributed interpretation of Esterel, to remove redundant latches generated by the compiler. We have proved that optimized circuits and circuits synthesized by the compiler are distinguished neither by the external environment nor by any operation.
of composition of circuits offered by the compiler. So, we can freely optimize circuits by working compositionally on the structure of programs. We have also suggested how our optimization technique can be integrated with the existing ones.

We have shown that distributed interpretations can be exploited to improve debugging of programs, because they permit to isolate parts of a program that are responsible of the violation of a given property. This cannot be done by means of classical semantics. Moreover, we have proved that the distributed interpretation of \( \text{tdccp} \) and the causal semantics of \( \text{ccp} \) given in [42] coincide for programs belonging to both languages.

Distributed interpretations for Statecharts remain to be developed. Since a compositional implementation of Statecharts in circuits has been proposed in [32], a distributed interpretation of Statecharts could support optimization of circuits.
Bibliography


