UNIVERSITÁ DEGLI STUDI DI PISA
DIPARTIMENTO DI INFORMATICA

DOTTORATO DI RICERCA IN INFORMATICA
Universitá di Pisa-Genova-Udine

Ph.D. Thesis : TD-6/97

Verification of Temporal and Real-Time Properties of Statecharts

Francesca Levi

March 1997

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL:+39-50-887111. FAX: +39-50-887226
EMAIL: levifran@di.unipi.it

Abstract

This thesis concerns the formal verification of statecharts. Statecharts are a synchronous formalism for the specification of reactive systems that is obtained by introducing in classical state-transition diagrams notions of hierarchy, parallelism and communication. Propositional μ-calculus is a very convenient formalism for expressing the properties characterizing the correctness of statecharts. Compositional verification is essential for supporting stepwise development of correct specifications. A necessary step to realize it is a compositional semantics. We define a compositional labelled transition system semantics for statecharts, which agrees with the semantics of Pnueli and Shalev. Such a semantics is obtained via a translation of statecharts into a process language, called $\mathcal{SP}$, whose main feature is an operator of process refinement. We define a compositional proof system for deciding whether a closed $\mathcal{SP}$ process satisfies a μ-calculus formula, where techniques of local and compositional model checking are combined in the style of [12].

Statecharts as defined originally are not adequate for the specification of real-time systems, where also the quantitative aspect of time must be considered. We propose an extension of statecharts and of the language $\mathcal{SP}$ with real-time features by assuming a discrete time domain. We generalize the Pnueli and Shalev semantics to timed statecharts and we propose a semantics of timed $\mathcal{SP}$ processes as a discrete version of timed graphs. A real-time discrete extension of μ-calculus called RTL_μ is defined for expressing properties characterizing the correctness of timed processes, and a proof system for checking whether a set of timed states of a timed graph satisfies an RTL_μ formula is proposed. A generalization of the technique proposed by Winskel [92] is exploited for supporting local model checking. The proof system provides the basis for extending the compositional method for untimed processes to timed ones.

Acknowledgements

First of all I wish to thank my supervisor Andrea Maggiolo-Schettini for its help and encouragment during these two years. I also thank Andrea Masini who introduced me to the field of temporal and modal logics.

Thanks to my external referees Mads Dam and Colin Stirling for their careful reading of my thesis and for their comments and suggestions.

Contents

Introduction **1**

1 Introduction to Statecharts **7**
1.1 The syntax . . . 7
1.2 The semantics . . . 10
1.2.1 The original step semantics . . . 11
1.2.2 The semantics of Pnueli and Shalev . . . 14
1.3 A formalism similar to Statecharts: Argos . . . 16

2 Modal and Temporal logics **19**
2.1 Propositional μ-calculus . . . 19
2.2 Real-time Logics . . . 23

3 The Language $\mathcal{SP}$ **25**
3.1 The language $\mathcal{SP}$ . . . 26
3.2 The translation . . . 30
3.3 An example . . . 33
3.4 Related works . . . 35
3.5 Towards verification of statecharts . . . 38

4 A Compositional Proof system for $\mathcal{SP}$ **41**
4.1 The logic L_μ . . . 42
4.2 Introduction to the proof system . . . 42
4.3 The proof system . . . 45
4.3.1 Logical Rules . . . 45
4.3.2 Rules for *nil* . . . 45
4.3.3 Rules for prefixing . . . 46
4.3.4 Rules for choice . . . 46
4.3.5 Rules for process refinement . . . 47
4.3.6 Rules for recursion . . . 48
4.3.7 Rules for parallel composition . . . 49
4.4 Soundness and completeness of the proof system . . . 54
4.5 An example . . . 56

4.6 Related works 62

5 Timed Statecharts and the language $\mathcal{TSP}$ 65
5.1 Timed statecharts 67
5.1.1 The semantics of Timed Statecharts 67
5.2 The language $\mathcal{TSP}$ 72
5.2.1 The semantics of $\mathcal{TSP}$ processes 72
5.3 The Translation 78
5.4 Related works 79

6 A Proof System for Real-time μ-Calculus 81
6.1 The logic RTL_μ 81
6.2 The proof system 83
6.2.1 Logical rules 84
6.2.2 Structural rules 85
6.2.3 Rules for modal operators 85
6.2.4 Rules for fixpoints 86
6.3 Soundness and completeness of the proof system 86
6.4 An example 90
6.5 Conclusions and related works 96

7 A Compositional Proof System for $\mathcal{TSP}$ 99
7.1 Introduction to the proof system 99
7.2 The proof system 104
7.2.1 Rules for *nil* 105
7.2.2 Rules for prefixing 106
7.2.3 Rules for timed prefixing 106
7.2.4 Rules for choice 107
7.2.5 Rules for refinement 108
7.2.6 Rules for recursion 109
7.2.7 Rules for parallel composition 110
7.3 Soundness and completeness of the proof system 115
7.4 An example 116
7.5 Related works 122

Conclusions 123

Bibliography 125

A Proofs of chapter 3 133

B Proofs of chapter 4 147
B.1 Soundness of the proof system 147
B.2 Completeness of the proof system 154

C Proofs of chapter 6 **163**
C.1 Soundness of the proof system 163
C.2 Completeness of the proof system 165

D Proofs of chapter 7 **171**
D.1 Soundness of the proof system 171
D.2 Completeness of the proof system 178

Introduction

In a landmark paper [75] Pnueli identified a very important class of computing systems called *reactive systems*. Reactive systems are characterized in contrast to transformational ones by an ongoing interaction with the environment. Typical examples of reactive systems are operating systems and network communication protocols. Since reactive systems tend to be quite complex, formal tools of specification and verification are necessary. Among the various proposals, the synchronous languages [15, 69, 18, 35] have been successfully used both for programming and specifying reactive as well as real-time systems, whose behaviour depends in addition on timing constraints. A common feature of synchronous languages is to have broadcast as communication mechanism and input-output structured actions. Moreover they are based on the so called *synchrony hypothesis* [15]: the reaction of the system to environment prompts is instantaneous. This hypothesis is indeed an abstraction and relies on the assumption that the environment can be described as a discrete process and that the system is faster than the environment. This approach has several advantages:

- the reaction time is known and does not depend on the implementation;
- the behaviour is abstract and allows further refinement without having to redo the timing;
- the behaviour can be described by the sequence of reactions to inputs of the environment without reference to physical time.

However, the definition of a formal semantics for these formalisms turned out to be more difficult than originally expected. The main problem arises from the attempt to combine together the synchrony hypothesis with the criteria of modularity and causality that are useful for a specification language. The synchrony hypothesis implies system output to be produced simultaneously with the input that causes it. The principle of causality requires that for any event generated at a given time there exists a causal chain of events leading to it. On the other hand, modularity means that the composition of two systems can be defined on the basis of their observable behaviour. Unfortunately, as pointed out in [49], modularity and causality are orthogonal. Moreover, anomalous situations, well known as causal paradoxes, may arise from the interpretation of negative conditions such as "event a is absent". For

instance it may be the case that events disallow their own causes. However, negative conditions are essential for expressing priorities and interrupts due to the broadcast communication.

Formal verification of reactive and concurrent systems is a very active area of research. The properties that characterize the correctness of these systems are quite different from those of transformational programs. While aspects such as the input-output behaviour and termination are important for transformational systems, properties describing the generally infinite interaction with the environment are essential for reactive systems. Typical properties are deadlock-freedom, mutual exclusion, responsiveness that are usually classified into the wide classes of safety and liveness properties. Temporal and modal logics turned out to be suitable formalisms for expressing safety as well as liveness properties. A variety of temporal and modal logics have been studied, particularly branching and linear time [27], and model checking emerged as a very useful method for the verification of temporal properties of reactive systems. The underlying idea is to check whether the formula is valid over the model obtained by the computations of the system. Starting from the proposal of [20], both algorithmic and proof theoretic methods of model checking have been investigated.

The present thesis is concerned with the formal verification of statecharts [35]. Statecharts are a synchronous visual formalism for the specification of reactive systems that is based on the idea of enriching state-transition diagrams with notions of hierarchy, concurrency and broadcast communication. The formalism offers facilities of hierarchical structuring of states and of parallelism, which permit high level and succinct descriptions. Actually, statecharts have been used for the specification also of complex systems [26, 36]. The main weakness of the formalism is that, due to causality problems, there is not a widely accepted semantics and different approaches have been suggested [37, 50, 77, 72, 67]. The semantics of Pnueli and Shalev [77] is a very convincing proposal for solving the problems related to causal paradoxes, but it is not compositional and not modular.

In order to express correctness properties of statecharts we consider propositional modal μ-calculus, due to Pratt and Kozen [57, 79]. Propositional μ-calculus has a next time modality and, for expressing temporal properties, least and greatest fixpoints as opposed to classical modalities. This logic is very powerful, since it has been shown to include Propositional Dynamic Logic [32], Hennessy-Milner Logic [39] as well as the branching time temporal logics CTL and CTL^* and the linear time temporal logic TL [28, 24]. Moreover, this formalism has the advantage that models are labelled transition systems, which are commonly used to give the semantics of concurrent and reactive systems. In the last few years, the problem of model checking both algorithmically and deductively μ-calculus has been extensively studied. Traditional techniques [28] are based on computing fixpoints iteratively as suggested by the theorem of Tarski. However, due to the classical state explosion problem, these methods cannot cope with arbitrary systems whose state space is likely to be infinite. One very appealing way of reducing such a limitation is to

check whether a state satisfies a property without computing all the states of the model that satisfy the property. This approach is referred to as *local model checking* [60, 58, 89, 22, 92, 83, 84, 59, 85] and has the clear advantage that only the necessary part of the model is explored. Winskel [92] proposes a very elegant technique for avoiding the global computation of fixpoints by exploiting some interesting properties concerning the unfolding of fixpoints.

We propose a compositional labelled transition system semantics for statecharts by giving a translation into a process language called $\mathcal{SP}$. The main novelty of the language is an operator of process refinement for representing the statecharts hierarchical structure. Moreover, $\mathcal{SP}$ processes have locations as counterpart of statecharts boxes. The language is parametric in the set of basic actions and in some operations over actions and other process calculi such as SCCS [70] and CBS [78] can easily be expressed in $\mathcal{SP}$. We define an SOS semantics for $\mathcal{SP}$ processes and we show a translation of statecharts which agrees with the semantics of Pnueli and Shalev. The basic idea is that of taking a process where basic actions represent statechart transitions and the corresponding input-output label, and locations correspond to statechart boxes. However, the semantics of Pnueli and Shalev is not modular. Therefore, it is not possible to obtain the semantics of the whole statechart by considering the parts of the system as black boxes and by observing only their input-output behaviour. One way of achieving compositionality is to consider actions corresponding to transitions, where both the input-output and the causal ordering between input and output is represented. This is not surprising, since other authors have followed the same idea [87, 50, 48, 49]. It is important to stress that the language and the translation are not ad hoc for the Pnueli and Shalev semantics, since other semantics of statecharts, such as the ones of [37, 67], may be obtained by taking different actions describing transitions and different operations over actions.

Such a semantics provides a formal basis for verifying statecharts by using the classical techniques proposed for process algebras such as equivalences, preorders and model checking. However, model checking techniques are traditionally non compositional, since they rely on the global or local exploration of the model and are applicable only after the whole system has been developed. On the other hand, compositional verification would make it possible to decompose the design of a large system into the design of its subsystems, since the verification that a system meets a property is reduced to the verification of derived properties of its components. The first advantage is that, when a component of the system is modified, only the verification concerning that component must be redone. Secondly, it makes it possible to leave undefined parts of the system and still to reason about it. Moreover, the verification problem may be reduced to possibly simpler verification tasks. Due to the complexity of reactive systems, these features are essential. In recent years there has been a growing interest in this topic starting from the work of [14] and various approaches have emerged both for temporal logics and for μ-calculus. In particular, there have been many proposals about compositional reasoning for μ-calculus within the framework of CCS-like languages [13, 61, 90, 91, 10, 12]. The basic idea is to

exploit the algebraic structure of the process for reducing a satisfaction problem for a composed process into equivalent satisfaction problems for its components. The technique has been formalized as a set of reduction rules as well as a proof system, where both reductions and proof rules typically depend only on the top-level structure of the process. Hence they can be applied without inspecting the internal structure of components. However, the treatment of parallel composition meets some fundamental difficulties, and in this case it is usually necessary to consider the structure of one of the two components.

We propose a compositional proof system for verifying μ-calculus properties of $\mathcal{SP}$ processes. The proof system follows an approach that has been suggested for a CCS-like language in [12] that combines local and compositional reasoning. There are two types of rules:

1. logical rules, that concern the structure of the formula;

2. compositional rules working on the process structure for the *next* modalities.

A proof is constructed by exploiting only the structure of the formula and of the process, since both types of rules do not refer to the underlying labelled transition system. Fixpoints formulas are treated by the method proposed by Winskel [92] for local model checking. The proof system is proved to be sound in general and complete for finite-state processes. In particular, it is complete for statecharts.

Real-time systems are reactive systems that are required to meet timing constraints. Temporal and modal logics are not adequate for expressing properties concerning the quantitative aspect of time, that are essential for the correctness of real-time systems. In recent years there have been many attempts for extending temporal and modal logics to real-time and a variety of formalisms have been suggested [9]. Statecharts as defined originally can be used to specify real-time systems by considering time as an ordinary input event. This approach permits very elegant specifications, but it is not adequate for verifying the correctness of the specified system, since neither temporal logics nor real-time logics can be used for expressing the required properties. One way of solving this drawback consists of introducing into the language explicit real-time features and of giving a timed semantics with respect to a given time domain. An adequate real-time logic can be applied for expressing the correctness properties and model checking can be used for verification.

We propose an extension of statecharts with real-time features, where a minimal and a maximal delay is attached to transitions. Intuitively, a transition is enabled within its minimal and maximal delay and cannot be delayed, when the maximal delay is reached. The considered model of timed statecharts is based on the following assumptions:

- the environment is discrete;

- the reaction of the system is instantaneous (synchrony hypothesis).

In other words, it is possible to choose a priori a fixed quantum of time, which is an upperbound for the reaction time of the system and for the distance between two successive inputs of the environment. Under these assumptions the discrete time domain is sufficient for modeling the system and we can easily adapt the Pnueli and Shalev semantics to timed statecharts. This formalism permits to naturally express real-time constructs, such as timeouts and watchdogs.

We analogously extend also the $\mathcal{SP}$ language and we define its semantics as a discrete timed graph. Timed graphs [5] are obtained by extending classical automata with a finite set of clocks that proceed at a uniform rate on the time domain (either discrete or dense) and constrain the time at which transitions can be performed. The semantics of timed graphs is given as a generally infinite labelled transition system, where states correspond to a state of the graph together with a valuation for the clocks, and transitions are of two types: instantaneous transitions corresponding to actions of the system and timed transitions corresponding to the flow of time of some amount. In our framework, since the language is synchronous and the environment is discrete, we consider a discrete version of timed graphs, where exactly one action is performed at each instant. Such a semantics can be exploited for verifying real-time properties of timed statecharts via model checking, as shown by a variety of works within the framework of real-time logics also with respect to a dense time domain [3, 45, 82].

For expressing real-time properties we consider a discrete extension of μ-calculus, called RTL_μ, which has freeze quantifiers and timing constraints for relating the times of different states. Moreover it is based on the assumption that next state is equal to next time. We propose a proof system for checking whether a set of timed states of a discrete timed graph satisfies a formula. The method of Winskel is generalized to deal with sets of timed states for supporting local model checking. The proof system is proved to be sound in general and complete for timed graphs with a finite set of states. Therefore, it is complete for arbitrary timed statecharts. The completeness is proved by using a clock region technique. We prove that there exists a finite quotient of the labelled transition system corresponding to the timed graph which is sufficient for establishing whether the formula is satisfied. This observation suggests that we could also directly construct this finite quotient and reason on this model. However, our approach has the advantage that this quotient is constructed locally only if it is necessary. Moreover, the proposed proof system is the basis for extending the compositional method for $\mathcal{SP}$ processes to the timed case. The idea is that of replacing the rules for establishing both $< \alpha > A$ and $[\alpha]A$ by a set of compositional rules working on the structure of the process. These rules are very similar to the ones for the untimed case.

Organization of the thesis

The contents of the single chapters are:

- **CHAPTER** 1: The formalism of statecharts and the main proposals for its formal semantics are introduced.

- **CHAPTER** 2: This chapter presents a brief overview of propositional μ-calculus and of the existing real-time logics.

- **CHAPTER** 3: In this chapter the language $\mathcal{SP}$ and its SOS semantics are defined. The language is parametric in the set of actions and in some operations over actions. We show a translation of statecharts into closed $\mathcal{SP}$ processes according to the Pnueli and Shalev semantics.

- **CHAPTER** 4: This chapter presents the compositional proof system for checking whether a closed $\mathcal{SP}$ process satisfies a μ-calculus formula.

- **CHAPTER** 5: In this chapter timed statecharts and the language $\mathcal{TSP}$ (Timed $\mathcal{SP}$) are introduced. The semantics of timed statecharts is obtained by suitably extending the Pnueli and Shalev semantics, while that of closed $\mathcal{TSP}$ processes is based on a discrete version of timed graphs.

- **CHAPTER** 6: This chapter presents the discrete real-time extension of μ-calculus RTL_μ and a proof system for local model checking RTL_μ formulas. The proof system checks whether a set of timed states of a timed graph satisfies a formula and exploits the method proposed by Winskel for treating fixpoints formulas.

- **CHAPTER** 7: This chapter describes the generalization of the compositional technique of chapter 4 to $\mathcal{TSP}$ processes.

Chapter 1

Introduction to Statecharts

In this chapter we introduce the formalism of statecharts and we discuss the main features of its semantics. In particular, we show the drawbacks of the original semantics [37] due to the interpretation of negative conditions in the enabling of transitions and we discuss the very convincing proposal of Pnueli and Shalev [77].

1.1 The syntax

Statecharts have been proposed to overcome the limitations of state-transition diagrams that are flat and unstructured, while preserving their visual nature. The main idea is that of introducing notions of *parallelism*, *hierarchy* and *broadcast communication*. These features allow very succinct representations and naturally support stepwise development.

Figure 1.1 shows a basic statechart D with substates A, B and C. The idea is that the system can be either in substate A or B or C. The arrow without source specifies the default state of the basic statechart, namely the state that is

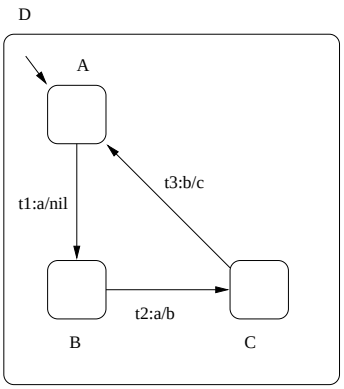

Figure 1.1: A basic statechart

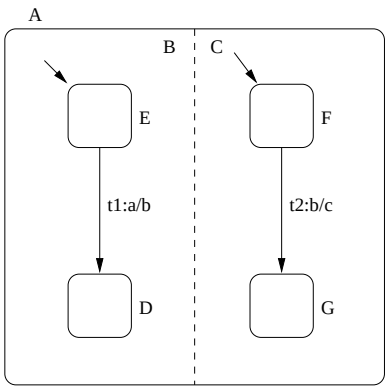

Figure 1.2: An AND statechart

entered in the initial configuration. Transitions are labelled by a pair t/a, where t is called the trigger and a the action. Intuitively, the trigger represents a condition on input events for the transition to be taken, while the action gives the set of events generated by the performance of the transition. For instance, transition t_2 can be performed iff the event a is communicated by the environment and produces event b. The event b is produced and is available anywhere in the system.

Figure 1.2 shows a statechart A that is given by the parallel composition of the two statecharts B and C. The graphical convention is that parallel components are separated by a dashed line. The intuitive interpretation is that the system is both in statechart B and in statechart C and that parallel components may communicate. Figure 1.3 shows an example of hierarchy, where substate B of statechart A is refined. The idea is that when the state B of statechart A is entered, then either C or D are entered.

A statechart is formally defined over a finite set Π of primitive events, that are used for the communication both with the environment and with its own subcomponents. We assume $nil \in \Pi$, where nil denotes the null event, and we denote by $\bar{\Pi} = \{\bar{e} \mid e \in \Pi\}$, where $\bar{e}$ is the negation of event e. For $\bar{e} \in \bar{\Pi}$, we denote by $\bar{\bar{e}}$ its negation, i.e. the event e.

In the following, we use the notation $i \in \{1, k\}$ for $i \in \{1, \ldots, k\}$.

Definition 1.1.1 *A* statechart *over* Π *is defined as follows:*

- $S : (\{S_1, \ldots, S_k\}, T, in, out, \chi, \delta)$ *is a* basic statechart, *where*
  - $\{S_1, \ldots, S_k\}$ *is a set of boxes, such that* $S \neq S_i$, *for each* $i \in \{1, k\}$. *Moreover, we define* $boxes(S) = \{S\} \cup \{S_1, \ldots, S_k\}$.
  - T *is a finite non-empty set of transitions.*
  - $in : T \to \{S_1, \ldots, S_k\}$, $out : T \to \{S_1, \ldots, S_k\}$ *give the target and source state of transitions, respectively.*

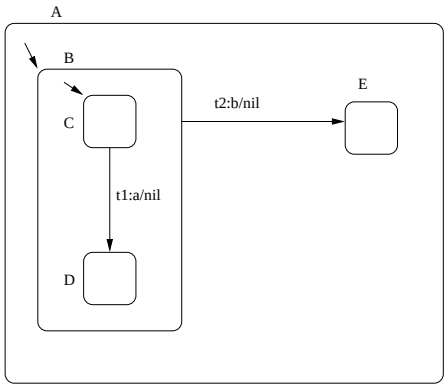

Figure 1.3: An example of hierarchy

- $\chi : T \to 2^{\Pi \cup \bar{\Pi}} \times 2^{\Pi}$ *is the labelling function of transitions. Let* trigger(t) *and* action(t) *denote the first and second component of* χ *respectively. We require that* $trigger(t), action(t) \neq \emptyset$ *and that* $e, \bar{e} \notin trigger(t) \cup action(t)$.
- $\delta : S \to \{S_1, \ldots S_k\}$ *is the default function giving the default substate.*

If $k = 0$ *then* S *is an* empty *statechart and* $boxes(S) = \{S\}$.

- $\bigtriangledown(S, \{S_1, \ldots, S_k\})$ *is a* refined statechart *iff*
  - S *is a basic non-empty statechart, where* $boxes(S) = \{S, S_1, \ldots, S_k\}$.
  - $\{S_1, \ldots, S_k\}$ *is a non-empty finite set of statecharts* S_i *over* Π*, such that* $\{S\} \cap boxes(S_i) = \emptyset$ *and* $boxes(S_i) \cap boxes(S_j) = \emptyset$*, for all* $i, j \in \{1, k\}$*, such that* $i \neq j$*. Moreover, we define* $boxes(\bigtriangledown(S, \{S_1, \ldots, S_k\})) = \{S\} \cup \bigcup_{i \in \{1,k\}} boxes(S_i)$.
- $S : AND(S_1, S_2)$ *is an* AND statechart *iff*
  - S_1, S_2 *are a non-empty statecharts over* Π*, and* $boxes(S) = \{S\} \cup \bigcup_{i \in \{1,2\}} boxes(S_i)$*, and* $\{S\} \cap boxes(S_i) = \emptyset$ *and* $boxes(S_i) \cap boxes(S_j) = \emptyset$*, for all* $i, j \in \{1, 2\}$*, such that* $i \neq j$.

The correspondence between the formal syntax of statecharts and the graphical description is quite obvious for basic statecharts. The statechart of figure 1.2 is formally defined as $A : \text{AND}(B, C)$, where B and C are basic statecharts. The statechart of figure 1.3 is represented by

$$\bigtriangledown(A : (\{B, E\}, \{t_2\}, \{in(t_2) = E\}, \{out(t_2) = B\}, \chi(t_2) = b/nil, \delta(A) = B),$$
$$\{B : (\{C, D\}, \{t_1\}, \{in(t_1) = D\}, \{out(t_1) = C\}, \chi(t_1) = a/nil, \delta(B) = C),$$
$$E : (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)\}).$$

The substate B of basic statechart A is refined by $B : (\{C, D\}, \{t_1\}, \{in(t_1) = D\}, \{out(t_1) = C\}, \chi(t_1) = b/nil, \delta(B) = C)$, while the substate E is refined by the empty statechart $E : (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$.

Note that the trigger of a transition may contain negated events, where $\bar{e}$, for $e \in \Pi$, is interpreted as "event e must be absent". It is important to stress that our definition differs from the original one [35], since we do not allow transitions to cross boundaries of boxes. This kind of syntax has been adopted by many other authors [50, 87, 88, 48] since it is more adequate for reasoning in a compositional way.

Configurations of a statechart S are maximal and consistent subsets of $boxes(S)$, where consistency means parallelism. Given $S_1, S_2 \in boxes(S)$, we define $LCA(S_1, S_2)$ as the statechart S', such that $S' \in boxes(S)$ and $S_1, S_2 \in boxes(S')$ and, for all $S'' \in boxes(S)$, such that $S_1, S_2 \in boxes(S'')$, $S' \in boxes(S'')$.

Definition 1.1.2 *Let S be a statechart. A set $c \in 2^{boxes(S)}$ is a* configuration *iff it is maximal and for each $S_1, S_2 \in c$, $LCA(S_1, S_2)$ is an AND statechart.*

A configuration $c \in C(S)$ is the *default configuration* iff, for each $S' \in boxes(S)$, such that S' is a basic statechart, $\delta(S') \in c$. We denote by $\Delta(S)$ the default configuration of S and by $C(S)$ the set of configurations of S.

1.2 The semantics

The semantics of statecharts is based on the *synchrony hypothesis* [15]: the reaction of the system to inputs is instantaneous. This is indeed an abstraction and it is justified by the following assumptions:

- The environment can be described as a discrete process, namely as an infinite sequence of inputs I_1, $I_2 \ldots$ occurring at successive instants of time;
- The system is infinitely faster than the environment, namely the reaction of the system to inputs I_i is completed before inputs I_{i+1} are produced.

The main advantage of this approach is that the semantics is abstract, since the behaviour can be described as the sequence of reactions to inputs I_i of the environment without reference to time and the reaction time does not depend on the implementation. Unfortunately, the synchrony hypothesis leads to well known causality problems so that the definition of the reaction to inputs of the environment, called a *step*, is very complex.

The basic idea is that a step consists of a maximal set of parallel transitionswhich are enabled in the configuration and which are *triggered* by the environment. By synchrony hypothesis, all the events generated by the transitions taken in a step are simultaneous with inputs. Hence, the performance of transitions may enable other transitions at the same time. Consider the AND statechart A of figure 1.2, where t_1 and t_2 are labelled by a/b, b/c, respectively. If the current configuration

is $\{A, B, C, E, F\}$ and if the input is $\{a\}$, transitions t_1 and t_2 must be taken in the same step, since t_1 is enabled with respect to $\{a\}$ and generates event b, which enables t_2.

However, this assumption leads to non-clear situations. Suppose that transitions t_1 and t_2 of statechart A are labelled by a/b and b/a, respectively. If the current configuration is $\{A, B, C, E, F\}$ and if the environment communicates neither a nor b, which reaction is possible?

It would be reasonable to assume both that transitions t_1 and t_2 are performed and that no transition is performed. If t_1 and t_2 are performed, events a and b are generated instantaneously and both t_1 and t_2 are enabled with respect to the set of current events $\{a, b\}$. On the other hand, if no transition is performed, both transition t_1 and transition t_2 is not enabled with respect to the input.

More complex situations arise in presence of negated events. Consider the statechart A of figure 1.2, where t_1 and t_2 are labelled by $\bar{a}/b$ and b/a, respectively. If the current configuration is $\{A, B, C, E, F\}$ and if the environment communicates neither a nor b, one could think that both t_1 and t_2 are performed. Actually, t_1 is enabled with respect to the input and its performance enables t_2, since it generates b. However, the performance of t_2 generates a instantaneously, that disables t_1.

These situations are common to all synchronous languages and can be interpreted in many ways. In the framework of statecharts a causal ordering between transitions taken in a step is required. Suppose that transitions t_1 and t_2 of statechart A are labelled by a/b and b/a, respectively. If the current configuration is $\{A, B, C, E, F\}$ and if the environment communicates neither a nor b, the only possible step is $\emptyset$, since each transition depends on the other.

However, the causality principle is not sufficient for achieving a precise semantics and a variety of interpretations for negated events have been given so that there is not a widely accepted semantics of statecharts [37, 50, 87, 88, 77, 65, 48, 72].

1.2.1 The original step semantics

The semantics of [37] is the first operational semantics of statecharts. The definition of step is based on the *synchrony hypothesis* and on the principle of *causality*.

These requirements are satisfied by defining a step as a maximal sequence of *microsteps*, which are causally and not temporally related. The first microstep consists of transitions that are enabled with respect to the input of the environment. Subsequent microsteps consist of transitions that are enabled with respect to the input augmented with the events generated in previous microsteps.

We denote by $T(S)$ the set of transitions of S, namely the union of sets T such that $S' : (\{S_1, \ldots, S_k\}, T, in, out, \chi, \delta) \in boxes(S)$.

Definition 1.2.1 *Let S be a statechart and $I \subseteq \Pi$.*

- *Assume $t_1, t_2 \in T(S)$. Transition t_1 is* consistent *with transition t_2 iff $t_1 \in$*

$T(S_1)$, $t_2 \in T(S_2)$, for $S_1, S_2 \in boxes(S)$, and $LCA(S_1, S_2)$ is an AND statechart.

- *Assume $T \subseteq T(S)$, $con(T) = \{t \in T(S) \mid t$ is consistent with t' for all $t' \in T\}$.*
- *Assume $c \in C(S)$, $rel(c) = \{t \in T(S) \mid out(t) \in c\}$.*
- *$trig(I) = \{t \in T(S) \mid \forall e \in trigger(t), e \in I$ and $\forall \bar{e} \in trigger(t), e \notin I\}$.*

Definition 1.2.2 *Given a configuration $c \in C(S)$ and a set of signals $I \subseteq \Pi$, a set of transitions $T \subseteq T(S)$ is a* microstep *with respect to I iff*

- *for each $t_1, t_2 \in T$, t_1 and t_2 are consistent;*
- *$T \subseteq rel(c)$;*
- *$T \subseteq trig(I)$.*

A step is defined as a maximal sequence of microsteps.

Definition 1.2.3 *Given a configuration $c \in C(S)$ and a set of input signals $I \subseteq \Pi$, a set of transitions $T \subseteq T(S)$ is a* step *iff*

- *there exists a sequence $T_1, T_2, \ldots, T_n$ of transition sets such that T_i is a microstep with respect to c and $I \bigcup_{j \prec i} gen(T_j)$, where $gen(T_j) = \bigcup_{t \in T_j} action(t)$;*
- *$T = \bigcup_{i \in \{1,n\}} T_i$ and for each $t_1, t_2 \in T$, t_1 and t_2 are consistent;*
- *T is maximal.*

The configuration $c' \in C(S)$ obtained from c via T is given by

$$c' = c - (\bigcup_{t \in T} boxes(out(t))) \cup (\bigcup_{t \in T} \Delta(in(t))).$$

Consider the statechart A of figure 1.2 and the configuration $\{A, B, C, E, F\}$, where transitions t_1 and t_2 labelled by a/b and b/c, respectively. If the environment produces a, then the sequence of microsteps $\{t_1\}, \{t_2\}$ gives a step. Transition t_1 is enabled with respect to the input and t_1 triggers t_2. The causality principle is ensured by the ordering of microsteps. Suppose that transitions t_1 and t_2 of statechart A are labelled by a/b and b/a, respectively. If the current configuration is $\{A, B, C, E, F\}$ and if the environment produces neither a nor b, the only step is the empty one, since no transition is enabled with respect to the input.

Note that transitions in a step are required to affect parallel components. Consider the statechart A of figure 1.3 where t_1 and t_2 are labelled by a/nil and b/nil, respectively. If the current configuration is $\{A, B, C\}$ and if environment produces both a and b, then either $\{t_1\}$ or $\{t_2\}$ is performed. Transitions t_1 and t_2 cannot be

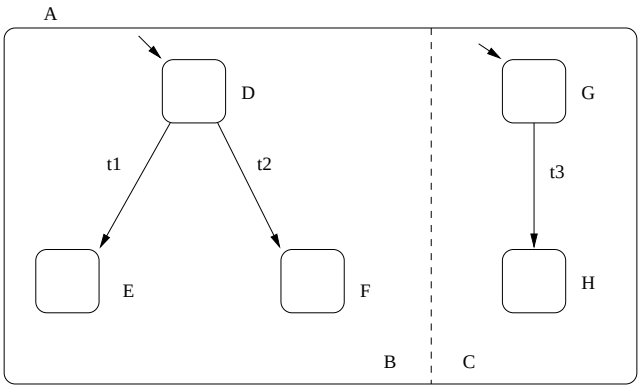

Figure 1.4: $t_1 : a/nil$, $t_2 : \bar{a}/nil$, $t_3 : nil/a$

taken in the same step, since they are conflicting. The configuration reached via t_1 is $\{A, B, D\}$, since state B is not left.

However, in this approach negated events in the trigger of transitions leads to some drawbacks. Negated events are fundamental for specifying priorities and interrupts [64]. Consider statechart of figure 1.3, where $trigger(t_1) = a$ and $trigger(t_2) = b$. Suppose that one wants transition t_2 to have priority over t_1. It is sufficient to add to the trigger of t_1 the condition $\bar{b}$. Therefore, when the environment produces both a and b, t_1 cannot be executed.

Consider the statechart of figure 1.4, where t_1, t_2 and t_3 are labelled by a/nil, $\bar{a}/nil$ and nil/a, respectively. If the current configuration is $\{A, B, C, D, G\}$ and if the environment does not produce event a, then there are two possible sequences of microsteps $\{t_2\}, \{t_3\}$ and $\{t_3\}, \{t_1\}$. Note that the sequence $\{t_3\}, \{t_2\}$ is not possible, since t_3 generates a that disables t_2. It is quite obvious that the interpretation of negated events is not the intuitive one, since one would expect that only $\{t_1, t_3\}$ is performed. Event a is indeed produced instantaneously. The problem is that t_1 requires the absence of signal a, while t_3 generates a, but this inconsistency is not detected. Another drawback is that steps depend on the ordering of microsteps, that are indeed performed at the same time.

Moreover, in this approach transitions may disallow their own causes. Consider the statechart A of figure 1.2, where t_1 and t_2 are labelled by $\bar{a}/b$ and b/a, respectively. If the current configuration is $\{A, B, C, E, F\}$ and if the environment produces neither a nor b, the sequence of microsteps $\{t_1\}$, $\{t_2\}$ is admissible. Actually t_1 is enabled with respect to the input and causes t_2. However, the performance of transition t_2 generates a, which disables t_1. This is a so called *causal paradox*.

As a conclusion, the semantics of [37] is not satisfactory, since causal paradoxes and anomalous situations occur because of negated events.

1.2.2 The semantics of Pnueli and Shalev

Pnueli and Shalev [77] propose a suitable approach for solving the problems of the semantics of [37]. The idea is that of requiring in addition a step to be globally consistent, namely that transitions in a step are enabled with respect to all the events generated in the step.

As a consequence the causes of a transition can never be disallowed. Suppose that transitions t_1 and t_2 of statechart A of figure 1.2, are labelled as $\bar{a}/b$ and b/a, respectively. If the current configuration is $\{A, B, C, E, F\}$ and if the environment communicates neither a nor b, the step $\{t_1, t_2\}$ cannot be taken, since it is not globally consistent. Transition t_1 triggers t_2, which generates a and disables t_1.

Consider the statechart A of figure 1.4, where t_1, t_2 and t_3 are labelled as a/nil, $\bar{a}/nil$ and nil/a, respectively. If the current configuration is $\{A, D, B, C, G\}$ and if the environment does not produce a, then the step $\{t_2, t_3\}$ is not allowed, since t_3 generates a, which disables t_2. Hence, the only possible step is $\{t_1, t_3\}$.

All these requirements are satisfied by defining a step as a subset of transitions which is the fixpoint of a relation $En_{c,I}$. The idea is that $t \in En_{c,I}(T)$ iff t is enabled in configuration c with respect to input I assuming that the transitions in T are taken.

Definition 1.2.4 *Let $c \in C(S)$ be a configuration, $I \subseteq \Pi$ and $T \subseteq T(S)$. We define $En_{c,I}(T) = rel(c) \cap con(T) \cap trig(I \cup gen(T))$, where $gen(T) = \bigcup_{t \in T} action(t)$.*

Definition 1.2.5 *Given a configuration $c \in C(S)$ and a set of input signals $I \subseteq \Pi$, the set of transitions $T \subseteq T(S)$ is an* admissible step *with respect to c and I iff $En_{c,I}(T) = T$ and if T is* inseparable*, i.e. for each $T' \subset T$, $En_{c,I}(T') \cap (T - T') \neq \emptyset$.*

The configuration $c' \in C(S)$ obtained from c via T is given by

$$c' = c - (\bigcup_{t \in T} boxes(out(t))) \cup (\bigcup_{t \in T} \Delta(in(t))).$$

We use the notation $c \stackrel{I,O}{\Longrightarrow} c'$ to denote that there exists an admissible step T, with respect to I, from c to c', such that $O = \bigcup_{t \in T} action(t)$.

Let us explain this definition. Maximality of a step is ensured by requiring $En_{c,I}(T) \subseteq T$. Consider the statechart A of figure 1.2 with t_1 and t_2 labelled by a/b and b/c, respectively, and the configuration $c = \{A, B, C, E, F\}$. We have $\{t_1, t_2\} = En_{c,\{a\}}(\{t_1\})$. Therefore, $\{t_1\}$ is not an admissible step, since it is not maximal. On the other hand, $\{t_1, t_2\} = En_{c,\{a\}}(\{t_1, t_2\})$. The condition of inseparability guarantees the existence of a causal order between transitions. For instance, $\{t_1, t_2\}$ is inseparable, since $En_{c,\{a\}}(\emptyset) = \{t_1\}$, $En_{c,\{a\}}(\{t_1\}) = \{t_1, t_2\}$ and $En_{c,\{a\}}(\{t_2\}) = \{t_1\}$.

On the other hand, if t_1 and t_2 are labelled by a/b and b/a, respectively, and the environment produces neither a nor b, then $En_{c,\emptyset}(\{t_1, t_2\}) = \{t_1, t_2\}$, but $\{t_1, t_2\}$

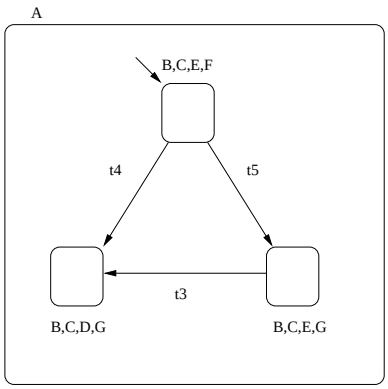

Figure 1.5: $t_4 : a/b$, $t_5 : \bar{a}/b$, $t_3 : a/nil$

is separable, since $En_{c,\emptyset}(\emptyset) \cap \{t_1, t_2\} = \emptyset$. Actually, no transition is caused by the input.

Global consistency is obtained by requiring $T \subseteq En_{c,I}(T)$. Suppose that t_1 and t_2 are labelled by $\bar{a}/b$ and b/a, respectively. We have $En_{c,\emptyset}(\{t_1, t_2\}) = \{t_2\}$ since t_1 is disabled by the generation of a. In this case, the reaction of the system to the empty input is not defined, since each set of transitions does not fulfill the requirements. This situation can be interpreted as a run-time error.

A different interpretation to causal paradoxes is proposed by [67]. In this approach a step is required to be globally consistent, but transitions with negative premises may prevent the performance of other transitions. For instance, consider the statechart A of figure 1.2 with t_1 and t_2 labelled by $\bar{a}/b$ and b/a, respectively, and the configuration $\{A, B, C, E, F\}$. With respect to the empty input, $\{t_1\}$ is an admissible step. The idea is that, even if t_2 is enabled, since t_1 produces b, t_2 is not performed, since it disables t_1. This interpretation solves causal paradoxes, but it may lead to unexpected behaviours.

Consider the statechart A of figure 1.4 and the configuration $\{A, B, C, D, G\}$. If the environment is empty, then there are two possible steps $\{t_1, t_3\}$ and $\{t_2\}$. In the second step transition t_3 is enabled, but it is not performed, since it disables t_2. However, the intuitive interpretation of the label nil/a is that this transition does not depend on the environment and must be performed, unless it is disabled by the performance of a conflicting transition. On the other hand, the label $\bar{a}/nil$ express a requirement on the set of communicated events, namely that a is absent, and does not say "a cannot be produced".

We believe that the semantics of Pnueli and Shalev gives the most appealing solution for dealing with negated events. Note that in this approach it may be the case that either a null reaction is defined or that no reaction is defined because of a causal paradox. In the first case the system is well-defined, while in the second one an error occurs.

It is important to stress that all the semantics for statecharts satisfy the principle

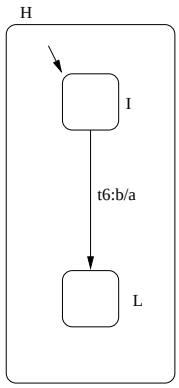

Figure 1.6: The basic statechart H

of causality. As pointed out in [49], this implies that they cannot be *modular*. In other words, the observable behaviour of the components, i.e their input-output, is not sufficient to obtain the semantics of the whole system.

Consider the statecharts in figures 1.2 and 1.5, where transitions t_1,t_2, t_4,t_5 and t_3 are labelled by a/nil, nil/b, a/b, $\bar{a}/b$ and a/nil, respectively. It is obvious that they are equivalent with respect to the Pnueli and Shalev semantics. However, the parallel compositions of the statechart of figure 1.2 and of the one of figure 1.5 with the one of figure 1.6 behave differently. In the first case, we have $\{A, B, C, E, F, H, I\} \stackrel{\emptyset,\{a,b\}}{\Longrightarrow} \{A, B, C, D, G, H, L\}$. Transition t_2 is enabled with respect to the input and causes t_6, that causes t_1. On the other hand, in the second case, there are no admissible steps from $\{A, B, C, E, F, H, I\}$ with respect to the empty input, because of the causal paradox. The problem is that in the statechart of figure 1.5 the production of b depends on the receiving of a, while in that of figure 1.2 the production of b is independent from the receiving of a.

In order to obtain a compositional semantics it is necessary to consider more information than just the input-output relation.

1.3 A formalism similar to Statecharts: Argos

Argos [69] is a visual synchronous formalism whose syntax is very similar to that of statecharts. However, in Argos there is a formal distinction between input and output events and local events are used to have inputs and outputs that are separated. The semantics differs in some aspects from that of statecharts. A system is required to be *reactive* and *deterministic*: for each configuration and for each input there exists exactly one reaction. If a system is not reactive and deterministic, it is considered incorrect. In a practical program environment this system must be rejected. These assumptions give rise to a different interpretation to causality problems. Consider the system of figure 1.2 and the configuration $\{A, B, C, E, F\}$ where

t_1 and t_2 are labelled as a/b and b/a, respectively. If events a and b are locals, there are two possible reactions, either $\emptyset$ or $\{t_1, t_2\}$ corresponding to the solutions to the equation $\{a = b\}$. The system is rejected, since it is not deterministic. On the other hand, suppose that t_1 and t_2 are labelled as $\bar{a}/b$ and b/a respectively. In this case, no reaction is defined, because the set of equations $\{\neg a = b \wedge a = b\}$ has no solutions. Also in this case the system is rejected.

Another difference regards the operator of refinement. Consider the system of figure 1.3 and the configuration $\{A, B, C\}$. Suppose that t_1 and t_2 are labelled by a/b and b/c, respectively. If the environment produces event a, then the step $\{t_1, t_2\}$ is performed and the reached configuration is $\{A, E\}$. Therefore, two non parallel transitions may be taken in the same step and may communicate. The choices regarding non determinism and causality of Argos are similar to the one of the language Esterel [15].

Chapter 2

Modal and Temporal logics

Modal logic was originally developed by philosophers to study different modes of truth. Temporal logic is a special type of modal logic that provides the possibility of reasoning about how the truth value of assertions change over time. Typical modalities as originally introduced by Prior [80] are *next time* ($<>$) P, which is true if there exists a successive world, where P holds, *sometimes* ($\Diamond$) P, which is true if there exists a future world, where P holds, and *always* ($\Box$) P, which is true if P is true and for all future worlds P is true.

Pnueli [75] argued that temporal logic is a useful formalism for reasoning about reactive systems, since it permits to express in a very elegant way the properties of safety and liveness characterizing the behaviour of those systems. The idea is that of assuming the modalities to quantify on the computations of a program. After this work temporal logic become the logic for reactive systems and both its theoretical and practical aspects have been extensively studied. In particular model checking as pioneered by the work of Clarke, Emerson and Sistla [20] has received a great success as verification method. Model checking consists of establishing either algorithmically or deductively whether a formula is valid over the model representing the behaviour of the system.

We assume the reader to be familiar with classical linear time and branching time temporal logics [27].

2.1 Propositional μ-calculus

Propositional modal μ-calculus [57, 79] is a very elegant and expressive logic, that has received a great interest in recent years for the verification of concurrent and reactive systems. This logic is very popular for many reasons. Firstly, it has been shown to include Propositional Dynamic Logic [32], Hennessy-Milner Logic [39] as well as the linear time logic TL and the branching time temporal logics CTL and CTL^* [28, 24]. Moreover, models of propositional μ-calculus are labelled transition systems, which are commonly used in giving the semantics to concurrent and reactive systems.

The main idea is to have a labelled next operator ($< \alpha >$) ranging over actions of the system, and least (μ) and greatest (ν) fixpoint operators, which permit to express temporal properties. Fixpoints are obtained by introducing logical variables in formulas and by interpreting open formulas as functions on states. Consider for instance the formula $< - > Z$, where Z is a logical variable ranging over states of the model and $< - >$ is the existential next operator with $-$ denoting any action. We can see $< - > Z$ as a function from sets of states to sets of states, namely the function that gives, for a set of states E, the set of states F from which a state of E is reachable in one step. Therefore, $\mu Z.A(Z)$ denotes the least fixpoint of the function corresponding to $A(Z)$, while $\nu Z.A(Z)$ denotes its greatest fixpoint.

Assume a set of logical variables VAR, a set of propositional symbols $PROP$ and a set $\mathcal{A}$ of actions. Formulas of propositional μ-calculus have the following syntax:

$$P \mid X \mid A \wedge A \mid < \alpha > A \mid \neg A \mid \nu X.A$$

where $X \in VAR$ is a logical variable, $P \in PROP$ is a propositional symbol, $\alpha \in \mathcal{A}$ is an action. A formula $< \alpha > A$ is interpreted as " there exists a next state reachable by executing an action α where A holds". Formula $\nu X.A$ stands for the greatest fixpoint of the function represented by A. Derived formulas are $[\alpha]A \equiv \neg < \alpha > \neg A$ and $\mu X.A \equiv \neg \nu X.\neg A[\neg X/X]$. A formula is *closed* iff, for each logical variable X, each occurrence of X is under the scope of νX. Moreover, for $K = \{\alpha_1, \ldots, \alpha_n\}$, where $\alpha_i \in \mathcal{A}$, formulas $[K]A$ and $< K > A$ are commonly used as shorthand for $\bigwedge_{i \in \{1,n\}} [\alpha_i]A$ and $\bigvee_{i \in \{1,n\}} < \alpha_i > A$, respectively.

A restriction on $\nu X.A$ is that each free occurrence of X in A should lie within the scope of an even number of negations. This condition is needed to ensure that the function corresponding to A is monotonic so that the existence of fixpoints is guaranteed by the theorem of Tarski.

Theorem 2.1.1 *(Theorem of Tarski) Let E be a set and $\psi : \mathcal{P}(E) \rightarrow \mathcal{P}(E)$ a monotonic function. Then ψ has a least fixpoint $\mu S.\psi(S)$ and a greatest fixpoint $\nu S.\psi(S)$ given by*

$$\mu S.\psi(S) = \bigcap \{S' \subseteq E \mid \psi(S') \subseteq S'\}$$
$$\nu S.\psi(S) = \bigcup \{S' \subseteq E \mid S' \subseteq \psi(S')\}.$$

Formulas are interpreted over labelled transition systems.

Definition 2.1.2 *A labelled transition system is $(Q, q_0, \Sigma, \mapsto)$, where Q is a set of states, q_0 is the initial state, Σ is a set of labels and $\mapsto \subseteq Q \times \Sigma \times Q$ is the labelled transition relation.*

In the following, we use the notation $q \stackrel{\sigma}{\mapsto} q'$ for representing the transition (q, σ, q').

Let $(\mathcal{S}, s_0, \mathcal{A}, \mapsto)$ be a labelled transition system, $\gamma : PROP \rightarrow 2^{\mathcal{S}}$ be a valuation function assigning subsets of $\mathcal{S}$ to propositional symbols $PROP$, and $\rho : VAR \rightarrow 2^{\mathcal{S}}$ be a valuation function assigning subsets of $\mathcal{S}$ to logical variables VAR.

The semantics of a μ-calculus formula A over the model $\mathcal{M} = ((\mathcal{S}, s_0, \mathcal{A}, \mapsto), \gamma)$ and the valuation ρ is inductively defined by $\| A \|_{\mathcal{M},\rho}$:

$$\begin{array}{ll}
\| X \|_{\mathcal{M},\rho} = & \rho(X) \\
\| P \|_{\mathcal{M},\rho} = & \gamma(P) \\
\| \neg A \|_{\mathcal{M},\rho} = & \mathcal{S} \setminus \| A \|_{\mathcal{M},\rho} \\
\| A_0 \wedge A_1 \|_{\mathcal{M},\rho} = & \| A_0 \|_{\mathcal{M},\rho} \cap \| A_1 \|_{\mathcal{M},\rho} \\
\| \nu X.A \|_{\mathcal{M},\rho} = & \bigcup \{ S' \subseteq \mathcal{S} \mid S' \subseteq \| A \|_{\mathcal{M},\rho[S'/X]} \} \\
\| < \alpha > A \|_{\mathcal{M},\rho} = & \{ s \in \mathcal{S} \mid \exists s' \in \mathcal{S}, s \overset{\alpha}{\mapsto} s', s' \in \| A \|_{\mathcal{M},\rho} \}.
\end{array}$$

We have used the notation $\rho[S'/X]$ for denoting the valuation ρ', which agrees with ρ except that $\rho'(X) = S'$. The semantics of fixpoint formulas is obviously based on theorem 2.1.1.

Given a model $\mathcal{M} = ((\mathcal{S}, \mathcal{A}, \mapsto), \gamma)$, $s \in \mathcal{S}$ and a formula A, we will say that $s \models_{\mathcal{M}} A$ iff $s \in \| A \|_{\mathcal{M},\rho}$ for each valuation ρ. Note that for closed formulas the semantics do not depend on the valuation of logical variables.

Classical liveness and safety properties can be expressed in this logic, since CTL is subsumed. For instance the CTL property $A\Box\phi$ is expressed by the formula

$$\nu Z.\phi \wedge [-]Z.$$

On the other hand, the formula

$$\nu Z.\phi \wedge < - > Z$$

represents the property "there exists a computation such that ϕ holds for all states" and is equivalent to the CTL property $E\Box\phi$. Liveness properties require least fixpoints. The CTL formula $E\Diamond\phi$ can be expressed by the formula

$$\mu Z.\phi \vee < - > Z.$$

In a similar way, $A\Diamond\phi$ is expressible as $\mu Z.\phi \vee [-]Z$. Note that also fairness properties such as "ϕ holds infinitely often along some computation" are expressible in propositional μ-calculus by using nesting of fixpoints, i.e. $\nu Z.\mu Y. < - > ((\phi \wedge Z) \vee Y)$.

Recent years have shown a great interest in the problem of model checking μ-calculus. The traditional approach relies on computing all the states of the model that satisfy the formula by traversing the global state space of the system. A complementary approach consists of providing a method for checking whether a given state satisfies the formula without computing all the states that satisfy the formula. A clear advantage of this local approach is that only the necessary part of the model is explored, so that the classical problems of state explosion are limited. The basic ideas of local model checking can be traced to the decision procedure of Street and Emerson [86].

Emerson and Lei [28] propose a global algorithm for model checking, where fixpoints are computed by standard iterative techniques as suggested by the theorem

of Tarski. The algorithm has complexity $O((nm)^{ad+1}))$, where n is the size of the formula, m the size of the model and ad is the alternation depth of the formula. Note that CTL is subsumed by μ-calculus of alternation depth one. The performance can be improved for some fragments of μ-calculus [30]. Andersen [11] suggests the use of a technique based on graphs for defining a global algorithm of complexity $O(n^{ad}k^{ad-1}m)$, where k is the number of states of the model. The main idea is that of constructing a boolean graph representing the product of the formula and of the model, where vertices are logical variables and edges represent dependencies between variables. The model checking problem is reduced to the problem of finding a marking for the boolean graph. In this paper, also a *local* algorithm based on similar techniques is proposed. A global algorithm for alternation free μ-calculus similar to the one in [11] is presented by [23]. Similar techniques are adopted by [81] for giving an incremental version of this algorithm, where the goal is of avoiding repeating all the proof, when the system is possibly modified. The main limitation of this algorithmic approach is due to the state explosion problem, since it cannot cope with arbitrary systems whose state space is infinite or large.

In contrast [60, 58, 89, 22, 92, 83, 84, 59, 85] advocate the use of either a proof system or a set of reduction rules for performing local model checking, where some interesting properties of fixpoints are exploited. One advantage of this *proof theoretic* approach is that it may be possible to prove properties also for infinite state systems. On the other hand, it is clear that only very simple proofs can be done by hand and some form of computer assistance is essential in practice. One could think to realize a proof assistant by exploiting for instance PVS [71] or HOL [33].

The main idea underlying this approach is of applying fixpoint induction for establishing whether a state satisfies a fixpoint formula, rather than computing the fixpoints with classical iterative techniques. The technique relies the following reduction lemma [57].

Lemma 2.1.3 *Let E be a set and $\psi : \mathcal{P}(E) \to \mathcal{P}(E)$ be a monotonic function. For $p \in E$,*

$$p \in \mu X.\psi(X) \Longleftrightarrow p \in \psi(\mu X.(\psi(X) \setminus \{p\}))$$
$$p \in \nu X.\psi(X) \Longleftrightarrow p \in \psi(\nu X.(\psi(X) \cup \{p\})).$$

This property suggests that $p \in\| \nu Z.A \|_{\mathcal{M},\rho}$ iff the assumption $p \in\| \nu Z.A \|_{\mathcal{M},\rho}$ implies $p \in\| A[\nu Z.A/Z] \|_{\mathcal{M},\rho}$. On the other hand, for the case of least fixpoints, $p \in\| \mu Z.A \|_{\mathcal{M},\rho}$ iff the assumption $p \notin\| \mu Z.A \|_{\mathcal{M},\rho}$ implies $p \in\| A[\mu Z.A/Z] \|_{\mathcal{M},\rho}$. Therefore, it is possible to check if a state satisfies a recursive formula by checking if the state satisfies a certain kind of unfolding of the formula. The main problem concerns the formalization of $p \in \psi(\mu X.\psi(X) \setminus p)$ and $p \in \psi(\mu X.\psi(X) \cup p)$.

In [89, 92], Winskel propose a very elegant solution by extending the syntax of recursive formulas with sets, such as $\nu X\{U\}.A$, where U is a finite set of states $U \subseteq \mathcal{S}$ and is called the set of tags.

The semantics of formulas with tagged fixpoints is naturally obtained by the reduction lemma:

$$\| \mu X\{U\}.A \|_{\mathcal{M},\rho} = \mu V.(\| A \|_{\mathcal{M},\rho[V/X]} \setminus U)$$
$$\| \nu X\{U\}.A \|_{\mathcal{M},\rho} = \nu V.(\| A \|_{\mathcal{M},\rho[V/X]} \cup U).$$

By lemma 2.1.3, $p \in \| \mu X\{U\}.A \|_{\mathcal{M},\rho}$ iff $p \in \| A[\mu X\{U, p\}.A/X] \setminus U \|_{\mathcal{M},\rho}$. Hence, if $p \notin U$, $p \models_{\mathcal{M}} \mu X\{U\}.A$ iff $p \models_{\mathcal{M}} A[\mu X\{U, p\}.A/X]$. On the other hand, for greatest fixpoints, $p \models_{\mathcal{M}} \nu X\{U\}.A$ iff either $p \in U$ or $p \models_{\mathcal{M}} A[\nu X\{U, p\}.A/X]$. This reasoning justifies the following set of reduction rules for correctness assertions proposed by Winskel. For a correctness assertion $p \vdash \nu X\{U\}.A$, the following reductions are used

$$\| p \vdash \nu X\{U\}.A \| = true \qquad p \in U$$
$$\| p \vdash \nu X\{U\}.A \| = \| p \vdash A[\nu X\{U, p\}.A/X] \| \quad p \notin U.$$

For $p \vdash \mu X\{U\}.A$,

$$\| p \vdash \mu X\{U\}.A \| = \| p \vdash A[\mu X\{U, p\}.A/X] \| \quad p \notin U.$$

Intuitively, the tags permit to record the set of states that have already been checked. A similar idea is adopted by Larsen [58] for a logic where each formula contains only either least or greatest fixpoints. On the other hand, Stirling and Walker [85] propose a tableau based proof system for full μ-calculus where definition lists are used as alternative to fixpoint formulas with tag sets. An equivalent version of this tableau system has been implemented in [22] and the method is extended to deal with infinite processes in [17, 16].

It is important to stress that this approach permits also the introduction of techniques specific to the program under consideration, such those offered by the algebraic theory of processes. For instance compositional proof systems guided by the process syntax rather than by the structure of the formula have been proposed for CCS-like processes [13, 61, 90, 91, 12].

2.2 Real-time Logics

Temporal and modal logics provide a natural way of expressing *qualitative* temporal requirements of reactive systems, such as invariance, responsiveness and so on. However, these logics are not adequate for expressing *quantitative* temporal requirements, which put timing constraints on the behaviour. A typical example of a real-time property is the *bounded response* property 'every event p is followed by a response q within 3 units of time'.

Since these quantitative properties are fundamental for the correctness of real-time systems, a number of real-time extensions of temporal and modal logics have been proposed over the past few years. There are three main approaches for extending both linear-time and branching time temporal logics.

- **Bounded operators** [31, 3, 8, 6, 63, 76]
  The idea is of replacing the classical temporal operators by time bounded operators. The operator $\Diamond_{[2,4]}$ is interpreted as eventually within 2 and 4 units of time. For instance, the bounded response property is expressed by the formula $A\Box(p \supset A\Diamond_{[0,3]}q)$ in a branching time framework.

- **Explicit clock** [76, 38]
  In this approach no new temporal operators are introduced but one state variable T representing the current time is used. With this understanding, the requirement of bounded response is expressed as

  $$A\Box([p \wedge x = T \supset A\Diamond[q \wedge T \leq x + 3]).$$

- **Freeze quantification** [7, 40, 2, 41]
  A quantifier $x.$ is used that binds variable x to the time of the temporal context, namely a formula $x.\phi(x)$ holds at time t iff $\phi(t)$ does. The bounded response property is expressed as $A\Box x.(p \supset A\Diamond y.(q \wedge y \leq x + 3)$.

The expressive power and the decidability of real-time logics depend on the chosen domain of time and on the operations allowed in time constraints [9, 8]. It is important to stress that the choice of the time domain depends on the nature of the system, that one wants to verify. A discrete time domain is mainly adequate for modeling synchronous systems, where all components are driven by a common global clock. Discrete time can be adopted also for asynchronous systems by assuming some fixed quantum a priori as an approximation. However, it is obvious that the accuracy with which the system is modeled is limited. In this case the time should be more realistically modeled by a dense domain.

In the last few years there has been a growing interest in studying the model checking for real-time logics. The most popular approach relies on a finite representation of the system as timed automaton (timed graph) [5] or as a timed transition system [44]. These models are obtained by introducing in classical transition systems clocks and clock constraint. However the model checking is not decidable for arbitrary real-time logics. In the case of linear time logics the model checking is decidable over discrete time and also over dense time, if the logic cannot express punctuality properties [7, 8, 6]. On the other hand, for branching time temporal logics model checking of punctuality properties is possible even over a dense time domain [3, 2, 45]. However, in this case it is necessary to consider constraints over time that do not contain addition of variables. Proof rules for checking bounded response and bounded invariance properties for linear time logics are presented in [43, 46].

Chapter 3

The Language $\mathcal{SP}$

In this chapter we propose a compositional labelled transition system semantics for statecharts by giving a translation into a process language called $\mathcal{SP}$. The main novelty of the language is an operator of process refinement for representing the statecharts hierarchical structure. Moreover, $\mathcal{SP}$ processes have locations as counterpart of statecharts boxes. The language is general, namely it is parametric in the set of basic actions and in some operations over actions and other process calculi such as SCCS [70] and the Calculus of Broadcasting Systems (CBS) [78] can easily be expressed in $\mathcal{SP}$.

We define the semantics of closed $\mathcal{SP}$ processes in the SOS style and we show a translation of statecharts into closed processes, which is correct with respect to the semantics of Pnueli and Shalev. The idea is to take a process where basic actions represent transitions and the corresponding communication and locations correspond to statechart boxes. The main problem is that the semantics of Pnueli and Shalev is not modular as we have discussed in chapter 1. Therefore, it is not possible to obtain the semantics of the whole statechart by considering the parts of the system as black boxes and by observing only their input-output behaviour. One way of achieving compositionality is to consider structured actions corresponding to transitions, where both the input-output and the causal ordering between input and output is represented. Similar techniques have been adopted by other authors [87, 50, 48, 49]. It is important to stress that the language and the translation are not ad hoc for the Pnueli and Shalev semantics and other semantics of statecharts, such as the ones of [37, 67], might be obtained by taking different actions describing transitions and different operations over actions.

Such a semantics is important for many reasons. It provides a formal basis for verifying statecharts by using the classical techniques proposed for the verification of process algebras such as equivalences, preorders and model checking. Secondly, this compositional semantics is a first step towards a compositional verification method, where the structure of the process is exploited for reducing a verification problem of a composed process to verification problems for its components.

The contents of this chapter have been already presented in [62].

3.1 The language $\mathcal{SP}$

In this section we present the process language $\mathcal{SP}$. We assume a set Act of basic actions, a set Loc of locations and a set Var of variables. On Act we assume an operation $\times : Act \times Act \to 2^{Act}$ of parallel composition and an operation $\# : Act \cup \{*\} \to 2^{Act}$. The idea is that actions of $\#(a)$ are in conflict with action a, i.e. the environment cannot enable both a and a', if $a' \in \#(a)$. For instance, the action " signal a is absent" is obviously in conflict with the action " signal a is received". The action $\{*\}$ is used to give the semantics of the empty process.

Moreover, we assume $\times$ to satisfy the following properties:

1. for each $a_1, a_2, a_3 \in Act$, $a_1 \times (a_2 \times a_3) = (a_1 \times a_2) \times a_3$ (associativity), where $\times$ is the straightforward extension to sets;

2. for each $a_1, a_2, a_3, a_4 \in Act$, such that $a_2 \in \#(a_5)$ and $a_4 \in \#(a_6)$ for some $a_5, a_6 \in Act$, if $a \in a_1 \times a_2$ and $a \in a_3 \times a_4$, then $a_2 = a_4$.

Definition 3.1.1 *We define processes $\mathcal{SP}$ as follows:*

$$\begin{array}{lll} b & ::= & nil \mid a.t \mid b_1 + b_2 \\ t & ::= & \ell :: b \mid rec\ x.t \mid t \bigtriangledown p \mid x \\ p & ::= & \ell :: t \mid \ell :: p_1 \times p_2 \end{array}$$

where $a \in Act$, $x \in Var$, $\ell \in Loc$.

The stratified grammar is motivated by the aim to have a hierarchical structure for locations that is analogous to the one of statechart boxes. Process $\ell :: t \in \mathcal{SP}$ denotes the process t at location ℓ and process $\ell :: p_1 \times p_2$ denotes the parallel composition of processes p_1 and p_2. In these cases ℓ corresponds to the outer box. As usual nil denotes the empty process, $b_1 + b_2$ denotes the choice between b_1 and b_2, while $a.t$ denotes process t prefixed by action a. Process $rec\ x.t$ is the classical recursive process and process $t \bigtriangledown p$ denotes process t refined by p.

We define $variables(p) = \{x \in Var \mid x \text{ is a subprocess of } p\}$ and $free(p) = \{x \in variables(p) \mid x \text{ does not occur inside the scope of } rec\ x\}$. We say that $p \in \mathcal{SP}$ is closed iff $free(p) = \emptyset$ and we denote by $\mathcal{CSP}$ the set of closed $\mathcal{SP}$ processes.

Definition 3.1.2 *Assume $p \in \mathcal{CSP}$. We define the sets of* current locations $loc(p)$ *and of* locations $Loc(p)$ *as*

- $loc(\ell :: t) = \{\ell\} \cup loc(t)$ *and* $Loc(\ell :: t) = \{\ell\} \cup Loc(t)$;
- $loc(\ell :: p_1 \times p_2) = \{\ell\} \cup loc(p_1) \cup loc(p_2)$ *and* $Loc(\ell :: p_1 \times p_2) = \{\ell\} \cup Loc(p_1) \cup Loc(p_2)$;
- $loc(\ell :: b) = \{\ell\}$ *and* $Loc(\ell :: nil) = \{\ell\}$, $Loc(\ell :: a.t) = Loc(\ell :: t)$, $Loc(\ell :: b_1 + b_2) = Loc(\ell :: b_1) \cup Loc(\ell :: b_2)$;

- $loc(rec\ x.t) = loc(t)$ *and* $Loc(rec\ x.t) = Loc(t)$;
- $loc(t \bigtriangledown p) = loc(t) \cup loc(p)$ *and* $Loc(t \bigtriangledown p) = Loc(t) \cup Loc(p)$.

Intuitively, for a process p describing a statechart, $loc(p)$ gives the configuration represented by the process, while $Loc(p)$ gives the boxes of the whole statechart.

We need some additional definitions for restricting to well-formed processes, such that locations of subprocesses are distinct. For a closed term t, we denote by $main(t)$ the location of $loc(t)$, such that

- if $t = \ell :: b$, then $main(t) = \ell$;
- if $t = rec\ x.t$, then $main(t) = main(t[rec\ x.t/x])$;
- if $t = t' \bigtriangledown p$, then $main(t) = main(t')$.

Definition 3.1.3 *Assume $p \in \mathcal{CSP}$. We say that p is* well-formed *iff*

- $p = \ell :: t$ *and* $\ell \notin Loc(t)$ *and t is well-formed;*
- $p = \ell :: p_1 \times p_2$ *and* $\ell \notin Loc(p_i)$, $Loc(p_1) \cap Loc(p_2) = \emptyset$, *and p_i is well-formed, for each* $i \in \{1, 2\}$;
- $t = \ell :: b$ *and* $\ell \notin Loc(b)$ *and b is well-formed;*
- $t = rec\ x.t'$ *and* $t'[rec\ x.t'/x]$ *is well-formed;*
- $t = t' \bigtriangledown p$ *and* $Loc(t') \cap Loc(p) = \{\ell\}$, *where either* $p = \ell :: t''$ *or* $p = \ell :: p_1 \times p_2$ *and* $\ell = main(t)$, *and t', p are well-formed;*
- $b = nil$ *is well-formed;*
- $b = a.t$ *iff t is well-formed;*
- $b = b_1 + b_2$ *iff* $Loc(b_1) \cap Loc(b_2) = \emptyset$ *and* b_i *is well-formed for each* $i \in \{1, 2\}$.

In the following we consider only well-formed processes.

The operational semantics of a closed process p is given as a labelled transition system $LTS(p) = (\mathcal{CSP}, p, \mathcal{S}(Act), \mapsto)$, where $\mathcal{CSP}$ is the subset of closed $\mathcal{SP}$ processes and $\mathcal{S}(Act)$ is the set of *semantic* actions, defined as follows.

Definition 3.1.4 *Let Act be a set of basic actions. We define $\mathcal{S}(Act)$ as the set of* semantic actions *such that, for each* $a \in Act$, $\tau \in \{\epsilon, \bar{\epsilon}\}$, $a : \tau \in \mathcal{S}(Act)$.

There are two types of actions: *non-idle* (ϵ) and idle ($\bar{\epsilon}$). A non-idle action represents a real action of the process, while an idle action represents the inability to do anything. This distinction is essential for ensuring maximal parallelism. The idea is that idle actions are taken iff no non-idle action is possible because of the behaviour of the environment.

We extend the operation $\times$ to semantic actions $\mathcal{S}(Act)$ as follows.

Definition 3.1.5 *Assume $\alpha_i = a_i : \tau_i \in \mathcal{S}(Act)$, for $i \in \{1,2\}$. We define $\alpha_1 \times \alpha_2$ as the set $\{a : \tau \in \mathcal{S}(Act)\}$, such that*

1. *$a \in a_1 \times a_2$;*
2. *$\tau = \epsilon$ if $\tau_i = \epsilon$, for some $i \in \{1,2\}$, and $\tau = \bar{\epsilon}$, otherwise.*

It is obvious that an idle action of the parallel composition must be an idle action of both components.

For $\alpha = a : \tau \in \mathcal{S}(Act)$, we use the notation $type(\alpha) = \tau$.

The transition relation is obtained by the following rules:

$$\frac{t \stackrel{\alpha}{\mapsto} t'}{\ell :: t \stackrel{\alpha}{\mapsto} \ell :: t'}$$

$$\frac{\{p_i \stackrel{\alpha_i}{\mapsto} p_i'\}_{i\in\{1,2\}} \quad \alpha \in \alpha_1 \times \alpha_2}{\ell :: p_1 \times p_2 \stackrel{\alpha}{\mapsto} \ell :: p_1' \times p_2'}\ par$$

$$\ell :: a.t \stackrel{a:\epsilon}{\mapsto} t \qquad\qquad \ell :: a.t \stackrel{b:\bar{\epsilon}}{\mapsto} \ell :: a.t \quad b \in \#(a)$$

$$\ell :: nil \stackrel{b:\bar{\epsilon}}{\mapsto} \ell :: nil \quad b \in \#(*)$$

$$\frac{\ell :: b_i \stackrel{\alpha}{\mapsto} t \quad type(\alpha) = \epsilon}{\ell :: b_1 + b_2 \stackrel{\alpha}{\mapsto} t}\ choice_1 \qquad \frac{\{\ell :: b_i \stackrel{\alpha}{\mapsto} \ell :: b_i\}_{i\in\{1,2\}} \quad type(\alpha) = \bar{\epsilon}}{\ell :: b_1 + b_2 \stackrel{\alpha}{\mapsto} \ell :: b_1 + b_2}\ choice_2$$

$$\frac{t \stackrel{\alpha}{\mapsto} t' \quad type(\alpha) = \epsilon}{t \bigtriangledown p \stackrel{\alpha}{\mapsto} t'}\ \bigtriangledown_1 \qquad \frac{p \stackrel{\alpha}{\mapsto} p' \quad type(\alpha) = \epsilon}{t \bigtriangledown p \stackrel{\alpha}{\mapsto} t \bigtriangledown p'}\ \bigtriangledown_2$$

$$\frac{t \stackrel{\alpha}{\mapsto} t \quad p \stackrel{\alpha}{\mapsto} p \quad type(\alpha) = \bar{\epsilon}}{t \bigtriangledown p \stackrel{\alpha}{\mapsto} t \bigtriangledown p}\ \bigtriangledown_3$$

$$\frac{t[rec\ x.t/x] \stackrel{\alpha}{\mapsto} t' \quad type(\alpha) = \epsilon}{rec\ x.t \stackrel{\alpha}{\mapsto} t'}\ rec_1$$

$$\frac{t[rec\ x.t/x] \stackrel{\alpha}{\mapsto} t[rec\ x.t/x] \quad type(\alpha) = \bar{\epsilon}}{rec\ x.t \stackrel{\alpha}{\mapsto} rec\ x.t}\ rec_2$$

Let us explain the rules. For processes $\ell :: t$ and $\ell :: p_1 \times p_2$, the location ℓ is the global location that is never left. The rule of parallel composition is the standard

rule for synchronous languages and is based on the operation of parallel composition of actions. Consider a process $\ell_1 :: \ell :: a.t$. If the environment enables action a then $\ell_1 :: \ell :: a.t$ evolves into $\ell_1 :: t$ by performing an action $a : \epsilon$ (non-idle) and location ℓ is left. On the other hand, if an action a' in conflict with a is enabled by the environment, then the process cannot perform any action. Therefore, the reached process is $\ell_1 :: \ell :: a.t$ and the action labelling the transition is $a' : \bar{\epsilon}$ (idle). Actions of type ϵ and type $\bar{\epsilon}$ give a different behaviour to the operators of refinement and of choice. Consider $\ell_1 :: \ell :: b_1 + b_2$. Rule $choice_1$ correspond to the classical rule for choice and is applicable iff the action is non-idle. On the other hand, b_1 and b_2 must agree on idle actions, as required by rule $choice_2$. In a similar way the semantics of $\ell_1 :: t \bigtriangledown p$ is defined. Idle actions of $\ell_1 :: t \bigtriangledown p$ must be idle actions of both $\ell_1 :: t$ and p. Note that when a non-idle action of p is performed, the process $\ell_1 :: t$ is not left. This is the basic idea of process refinement. Rule rec_1 is standard, while rule rec_2 reflects the idea that, when an idle action is performed, the process has not done anything. Note that rules $\bigtriangledown_3$ and $choice_2$ are motivated by the assumption that the reaction is maximal, namely an idle action is performed iff no non-idle action is possible.

Definition 3.1.6 *Assume $p \in \mathcal{CSP}$, we say that p is* regular *iff it has guarded recursion and for each variable x, x does not occur free in a subprocess $\ell :: p_1 \times p_2$ and in process p_1, such that $\ell :: t \bigtriangledown p_1$ is a subprocess of p.*

Definition 3.1.7 *Assume $p \in \mathcal{CSP}$. We define $\mathcal{R}_p$ as the subset of $\mathcal{CSP}$ such that*

- $p \in \mathcal{R}_p$;
- *if $p' \in \mathcal{R}_p$ and, if $p' \stackrel{\alpha}{\mapsto} p''$, then $p'' \in \mathcal{R}_p$.*

For a regular process $p \in \mathcal{CSP}$, $\mathcal{R}_p$ is finite. In other words, a regular process is finite-state.

It is important to point out the relationship of $\mathcal{SP}$ with other process languages such as SCCS and CBS. These languages without restriction can be seen as instances of $\mathcal{SP}$ by properly instantiating Act and $\times, \#$. Consider a simple version of $\mathcal{SP}$ without location declarations

$$x \mid a.p \mid p_1 + p_2 \mid p_1 \bigtriangledown p_2 \mid rec\, x.p \mid p_1 \times p_2.$$

For SCCS over Act, it is sufficient to consider $a \times b = ab$ and $\#(a) = \emptyset$, for each $a \in Act \cup \{*\}$. The language CBS has broadcast as the communication mechanism. Therefore, sending is autonomous, while receiving is controlled by the environment. Moreover, it is assumed that each time only one message is communicated. Given a set of names $\Delta = \{a, b, c, \ldots\}$, the set of basic actions is given by $\mathcal{A}? \cup \mathcal{A}!$, where $\mathcal{A}? = \{a? \mid a \in \Delta\}$ and $\mathcal{A}! = \{a! \mid a \in \Delta\}$. Intuitively $a?$ and $a!$ represent the action of receiving a and broadcasting a, respectively. The operational semantics

of CBS processes is given in the SOS style by considering transitions labelled by $\mathcal{A} = \mathcal{A}? \cup \mathcal{A}! \cup \mathcal{A}:$, where $\mathcal{A}: := \{a: \mid a \in \Delta\}$ and $a:$ is the action of *discarding* a. Discarding actions $\mathcal{A}:$ are used for achieving the following behaviour: a process cannot discard a message, if it is waiting for it. This is obtained by the following rules for receiving actions $a?.p \stackrel{a?}{\mapsto} p$ and $a?.p \stackrel{b:}{\mapsto} a?.p$, where $a \neq b$. On the other hand, for sending actions the transitions are $a!.p \stackrel{a!}{\mapsto} p$ and $a!.p \stackrel{b!}{\mapsto} a!.p$. These transitions are justified by the fact that sending is autonomous and that only one message can be produced at each time. An operation of parallel composition over $\mathcal{A}$ is defined for performing the commmunication. For instance the parallel composition of $a!$ and $a?$ gives $a!$, while that of $a?$ and $b!$ is not defined for $a \neq b$. It is quite obvious the similarity between idle actions and *discard* actions of CBS.

Therefore, a CBS process can be seen as an $\mathcal{SP}$ process by taking $Act = \mathcal{A}? \cup \mathcal{A}! \cup \mathcal{A}:$, $\times$ as the parallel product defined in [78], and $\#(a!)$ (resp. $\#(a?)$)$= \{b: \mid b \in \mathcal{A}, b \neq a\}$, and $\#(*) = \{a: \mid a \in \mathcal{A}\}$.

3.2 The translation

In this section we define a mapping α from statecharts into closed $\mathcal{SP}$ processes, such that the labelled transition system semantics of $\alpha(S)$ is equivalent to Pnueli and Shalev semantics of S.

Given a statechart S over Π, we construct a process with basic actions Act corresponding to transitions and describing their input-output, and locations Loc corresponding to the boxes of S. However, since the input-output is not sufficient for compositionality, also the causal ordering between events is recorded in basic actions.

For sets I, I_1, I_2, where $I_1 \subseteq I$, we use the notation $I[I_2/I_1]$ for denoting the replacement of subset I_1 by I_2.

Hence, we consider actions with the following syntax.

Definition 3.2.1 *Let Π be a finite set of primitive events. We define $\Sigma(\Pi)$ as the set of pairs $(\ell, \prec)$ where*

1. *$\ell \in 2^{\Pi \cup \bar{\Pi}}$ and ℓ is* consistent*, i.e. $e, \bar{e} \notin \ell$, for each $e \in \Pi$;*
2. *$\prec$ is an ordering over $2^\ell \times 2^\ell$ such that:*
   (a) *$\prec$ is an* irreflexive *ordering relation;*
   (b) *for each $C, C' \in 2^\ell$ such that $C' \subseteq C$, if $C'' \prec C'$ then $C[C''/C'] \prec C$.*

The ordering reflects the causality principle, so that $\{e, e''\} \prec \{e'\}$ means that both e and e'' are necessary for event e' to be produced. On the other hand, $\{e\} \prec \{e'\}$ and $\{e''\} \prec \{e'\}$ means that both e and e'' are sufficient for event e' to be produced. Suppose for instance $\{e\} \prec \{e'\} \prec \{e''\}$. Since the relation is transitive, e is sufficient

for event e'' to be produced. Moreover, we require labels to be *consistent* for global consistency.

For $\sigma = (\ell, \prec) \in \Sigma(\Pi)$, we define $trigger(\sigma) = \{e \in \Pi \cup \bar{\Pi} \mid \nexists C \in 2^{\Pi \cup \bar{\Pi}}$ such that $C \prec \{e\}\}$ and $action(\sigma) = \ell \setminus trigger(\sigma)$. The trigger of σ is the set of events required either to be present or absent in the input for enabling the step, while its action is the set of events produced by the step.

The operation $\times$ of parallel composition over actions $\Sigma(\Pi)$ is quite complex due to the communication between parallel components.

Given two orderings $\prec_1, \prec_2$, we denote by $(\prec_1 \cup \prec_2)^+$, the ordering $\prec$, such that $\prec_1 \cup \prec_2 \subseteq \prec$ and such that, if $C \prec C'$ and $C' \prec C''$, then $C \prec C''$ and such that, for each C, C', if $C' \subseteq C$ and $C'' \prec C'$, then $C[C''/C'] \prec C$.

Definition 3.2.2 *Given $\sigma_1 = (\ell_1, \prec_1) \in \Sigma(\Pi)$ and $\sigma_2 = (\ell_2, \prec_2) \in \Sigma(\Pi)$, we define $\sigma_1 \times \sigma_2 = \{\sigma = (\ell, \prec)\}$ such that*

- *$\ell = \ell_1 \cup \ell_2$ iff $\ell_1 \cup \ell_2$ is consistent;*
- *$\prec \subseteq (\prec_1 \cup \prec_2)^+$ such that:*
  - *$(\ell, \prec) \in \Sigma(\Pi)$;*
  - *$\prec$ is maximal.*

The idea is that the parallel composition of actions $(\ell_1, \prec_1)$ and $(\ell_2, \prec_2)$ is given by the union of events $\ell_1 \cup \ell_2$ and by the ordering, that is the maximal subset of $(\prec_1 \cup \prec_2)^+$, which is irreflexive. If $\ell_1 \cup \ell_2$ is not consistent then $\sigma_1 \times \sigma_2$ is empty.

Consider $\sigma_1 = (\{a, b\}, \{a\} \prec_1 \{b\})$ and $\sigma_2 = (\{b, c\}, \{b\} \prec_2 \{c\})$. We have $\sigma_1 \times \sigma_2 = \{\sigma\}$ with $\sigma = (\{a, b, c\}, \{a\} \prec \{b\}, \{b\} \prec \{c\}, \{a\} \prec \{c\})$. The communication is realized by the transitive closure of the two orderings. Actually, $trigger(\sigma) = \{a\}$ and $action(\sigma) = \{b, c\}$, while $trigger(\sigma_1) = \{a\}$, $trigger(\sigma_2) = \{b\}$. Event b is not required from the environment, since it is produced by one component.

Note that $(\prec_1 \cup \prec_2)^+$ may be reflexive. Consider $\ell_1 = (\{a, b\}, \{a\} \prec_1 \{b\})$ and $\ell_2 = (\{a, b\}, \{b\} \prec_2 \{a\})$. The parallel composition of σ_1 and σ_2 cannot be $\sigma = (\{a, b\}, \{a\} \prec \{b\}, \{b\} \prec \{a\})$, since $trigger(\sigma) = \emptyset$, and this contradicts the causality principle. We have $\sigma_1 \times \sigma_2 = \{\sigma_1, \sigma_2\}$ so that either event a or event b is required from the environment.

Definition 3.2.3 *Let $\sigma \in \Sigma(\Pi)$. We define $\#(\sigma) = \{(E, \emptyset) \mid E \subseteq \Pi \cup \bar{\Pi}$, such that E is consistent and $\forall e \in \Pi$, either $e \in E$ or $\bar{e} \in E$ and there exists $e \in E$, such that $\bar{e} \in trigger(\sigma)\}$. Moreover, we define $\#(*) = \{(E, \emptyset) \mid E \subseteq \Pi \cup \bar{\Pi}$, such that E is consistent and $\forall e \in \Pi$, either $e \in E$ or $\bar{e} \in E\}$.*

The idea is that $\sigma' \in \#(\sigma)$ iff for each input $I \subseteq \Pi$, if I enables σ', then I does not enable σ. Consider an action $\sigma = (\{a, b, c\}, \{a, b\} \prec \{c\})$. We have $\#(\sigma) = \{(\{\bar{a}, b, c\}, \emptyset), (\{\bar{b}, a, c\}, \emptyset), (\{\bar{a}, b, \bar{c}\}, \emptyset), (\{\bar{b}, a, \bar{c}\}, \emptyset), (\{\bar{a}, \bar{b}, \bar{c}\}, \emptyset), (\{\bar{a}, \bar{b}, c\}, \emptyset)\}$. If the environment does not produce both a and b the action σ cannot be performed, since $trigger(\sigma) = \{a, b\}$.

Proposition 3.2.4 *Let $\Sigma(\Pi)$ and $\times$, $\#$ as defined in definitions 3.2.1, 3.2.2 and 3.2.3, respectively. The following properties hold:*

1. *for each $\sigma_1, \sigma_2, \sigma_3 \in \Sigma(\Pi)$, $\sigma_1 \times (\sigma_2 \times \sigma_3) = (\sigma_1 \times \sigma_2) \times \sigma_3$ (associativity);*
2. *for each $\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6 \in \Sigma(\Pi)$, such that $\sigma_2 \in \#(\sigma_5)$ and $\sigma_4 \in \#(\sigma_6)$, if $\sigma \in \sigma_1 \times \sigma_2$ and $\sigma \in \sigma_3 \times \sigma_4$, then $\sigma_2 = \sigma_4$.*

The translation $\alpha(S)$ is inductively defined over the structure of statechart S. We need the following definition for refinement.

Definition 3.2.5 *Let $t \in \mathcal{T}$, $\{p_i\}_{i\in\{1,n\}} \in \mathcal{CSP}$ be closed processes and let $l_i \in Loc(t)$. We define $ref(t, \{(l_1, p_1), \ldots, (l_n, p_n)\})$ as the term obtained by substituting in t each occurrence of a subterm $l_i :: t'$ with $l_i :: t' \bigtriangledown p_i$.*

Moreover, for a transition $t \in T(S)$, we define $lab(t) = (trigger(t) \cup action(t), \{trigger(t) \prec \{e'\} \mid e' \in action(t) - \{nil, trigger(t)\}\})$. Intuitively $lab(t)$ gives the action $\sigma \in \Sigma(\Pi)$ with $trigger(\sigma)$ and $action(\sigma)$ corresponding to the ones of t.

Definition 3.2.6 *We define the translation α as follows.*

- *Let S be a basic empty statechart. Then $\alpha(S) = S :: S :: nil$.*
- *Let $S : (\{S_1, \ldots, S_n\}, T, in, out, \chi, \delta)$ be a basic non-empty statechart.*

  For each $i \in \{1, n\}$, let $T_i = \{t \in T \mid out(t) = S_i\}$.
  Let $t_{S_i} = rec\ s_i.S_i :: (\sum_{t\in T_i} lab(t).in(t))$, if $T_i \neq \emptyset$ and $t_{S_i} = S_i :: nil$, otherwise.

  Process $\alpha(S)$ is obtained by repeating the following steps, where initially $\alpha(S) = S :: t_{\delta(S)}$:

  - *If $\alpha(S)$ is closed then stop;*
  - *If $\alpha(S)$ is open, then for each $s_j \in free(\alpha(S))$, $\alpha(S) = \alpha(S)[t_{S_j}/s_j]$.*
- *Let S be a refined statechart $\bigtriangledown(S, \{S_1, \ldots, S_n\})$ and $\alpha(S) = S :: t_S$.*
  Then $\alpha(\bigtriangledown(S, \{S_1, \ldots, S_n\}))$ is defined as

  $$S :: ref((t_S, \{(S_1, \alpha(S_1)), \ldots, (S_n, \alpha(S_n))\}).$$

- *Let $S : AND(S_1, S_2)$. Then $\alpha(S) = S :: \alpha(S_1) \times \alpha(S_2)$.*

It is obvious that processes corresponding to statecharts are well-formed and regular.

The definitions of $lab(t)$ and of $\times$, $\#$ are given so that the semantics of $\alpha(S)$ agrees with the Pnueli and Shalev semantics of S as stated by the following theorem.

Assume $I \subseteq \Pi$ and $\ell \subseteq \Pi \cup \bar{\Pi}$. We say that I *triggers* ℓ ($I \rightsquigarrow \ell$) iff, for each $e \in \ell \cap \Pi$, $e \in I$, and for each $\bar{e} \in \ell \cap \bar{\Pi}$, $e \notin I$. Moreover, for $\alpha = \sigma : \tau \in \mathcal{S}(\Sigma(\Pi))$, we denote by $trigger(\alpha)$ and $action(\alpha)$ the sets $trigger(\sigma)$ and $action(\sigma)$, respectively.

Theorem 3.2.7 *Let S be a statechart over Π.*

- *For each $c \in C(S)$, such that c is reachable from $\Delta(S)$, there exists $p \in \mathcal{R}_{\alpha(S)}$, such that $loc(p) = c$. Moreover, for each $I \subseteq \Pi$, if $c \stackrel{I,O}{\Longrightarrow} c'$, then $p \stackrel{\alpha}{\mapsto} p'$, where $loc(p') = c'$ and $I \rightsquigarrow trigger(\alpha)$, $O - trigger(\alpha) = action(\alpha)$;*
- *For each $p \in \mathcal{R}_{\alpha(S)}$, if $p \stackrel{\alpha}{\mapsto} p'$, then there exist $c, c' \in C(S)$, where $loc(p) = c$, $loc(p') = c'$ and $c \stackrel{I,O}{\Longrightarrow} c'$, for each $I, O \subseteq \Pi$, such that $I \rightsquigarrow trigger(\alpha)$ and $O - trigger(\alpha) = action(\alpha)$.*

The proof of this theorem is by induction on the structure of the statechart and can be found in appendix A. Consistency of steps is guaranteed by the definition of actions $\Sigma(\Pi)$, while maximality is guaranteed by the definition of $\#$ and by the semantics of idle actions (rules $choice_2$ and $\bigtriangledown_3$). The most complex case is that of parallel composition. We show that the condition of inseparability is captured by the definition of $\times$ over $\Sigma(\Pi)$.

Note that by taking a different set of actions corresponding to transitions and different operations $\times$ and $\#$, we could have a process $\alpha(S)$, which has a different semantics, for instance the ones of [37, 67].

3.3 An example

We explain the translation for the statechart of figure 3.1, where transitions are labelled as

$t_1 : a/b, t_2 : a/b$
$t_3 : b/nil, t_4 : b/nil$
$t_5 : d/nil, t_6 : d/nil$

For basic statecharts E and F,

$\alpha(E) = E :: rec\ x.A :: lab(t_1).rec\ y.C :: lab(t_2).x$
$\alpha(F) = F :: rec\ x.B :: lab(t_3).rec\ y.D :: lab(t_4).x.$

In the following, let $\alpha(E) = E :: t_1$ and $\alpha(F) = F :: t_2$. Note that $loc(\alpha(E)) = \{E, A\}$ and $loc(\alpha(F)) = \{F, B\}$, which are the initial configurations of E and F, respectively. It is quite obvious that the semantics of $\alpha(E)$ and of $\alpha(F)$ correspond to the semantics of E and F, respectively. By rule rec_1,

$$\alpha(E) \stackrel{lab(t_1):\epsilon}{\mapsto} E :: rec\ y.C :: lab(t_2).t_1.$$

Since $trigger(lab(t_1)) = \{a\}$ and $action(lab(t_1)) = \{b\}$, this transition corresponds to the step $\{t_1\}$ from $\{E, A\}$ to $\{E, C\}$ with respect to inputs that contain a. On the other hand, by rule rec_2,

$$\alpha(E) \stackrel{\sigma:\bar{\epsilon}}{\mapsto} \alpha(E)$$

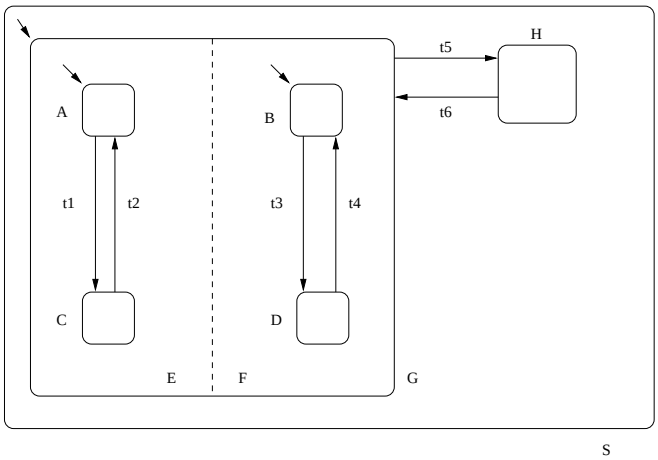

Figure 3.1: A statechart

for $\sigma \in \#(lab(t_1))$, i.e. $\sigma \in \{(\{\bar{a}, b, d\}, \emptyset), (\{\bar{a}, b, \bar{d}\}.\emptyset), (\{\bar{a}, \bar{b}, d\}, \emptyset), (\{\bar{a}, \bar{b}, \bar{d}\}, \emptyset)\}$. This transition corresponds to the empty step from $\{E, A\}$ with respect to inputs that do not contain a.

Since $G : \mathrm{AND}(E, F)$, then $\alpha(G) = G :: \alpha(E) \times \alpha(F)$. Maximality and consistency of steps are ensured by the definition of $\times$ over actions. By rule *par*,

$$\alpha(G) \overset{(\{a,b,nil\},\{a\} \prec \{b\}):\epsilon}{\mapsto} G :: p_1 \times p_2$$

where $p_1 = E :: rec\, y.C :: lab(t_2).t_1$ and $p_2 = F :: rec\, y.D :: lab(t_4).t_2$. This transition corresponds to the step $\{t_1, t_3\}$ from the configuration $\{G, E, F, A, B\}$ to the configuration $\{G, E, F, C, D\}$, when the environment produces a. Actually, $trigger((\{a, b, nil\}, \{nil, a\} \prec \{b\})) = \{a\}$. Steps such that one component takes a transition and the other one not, are obtained by combining a non-idle action of one component with an idle action of the other. For instance, from $\alpha(E) \overset{(\{\bar{a},b,\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} \alpha(E)$ and $\alpha(F) \overset{(\{b,nil\},\emptyset):\epsilon}{\mapsto} p_2$, we have

$$G :: \alpha(E) \times \alpha(F) \overset{\{b,nil,\bar{a},\bar{d}\},\emptyset):\epsilon}{\mapsto} G :: \alpha(E) \times p_2.$$

This transition corresponds to the non-empty step $\{t_3\}$, when the environment produces b and does not produce a and d. Note that there are no steps, such that E takes a transition and F does not take a transition, since t_1 triggers transition t_3. Finally, we have the following empty steps

$$\alpha(G) \overset{\sigma:\bar{\epsilon}}{\mapsto} \alpha(G)$$

for $\sigma \in \#(lab(t_1)) \cap \#(lab(t_3))$, i.e. $\sigma \in \{(\{\bar{a}, \bar{b}, d\}, \emptyset), (\{\bar{a}, \bar{b}, \bar{d}\}, \emptyset)\}$.

For the basic statechart S, we have

$$\alpha(S) = S :: rec\, x.G :: lab(t_5).rec\, y.H :: lab(t_6).x.$$

Hence, $\alpha(\bigtriangledown(S,\{G,H\})) = S :: t_6$ where

$$t_6 = rec\ x.(G :: lab(t_5).rec\ y.(H :: lab(t_6).x \bigtriangledown \alpha(H)) \bigtriangledown \alpha(G))$$

where $\alpha(H) = H :: H :: nil$.

Since $\alpha(G) \overset{(\{a,b,nil\},\{a\}\prec\{b\}):\epsilon}{\mapsto} G :: p_1 \times p_2$, by rule rec_1 and by rule $\bigtriangledown_2$,

$$\alpha(\bigtriangledown(S,\{G,H\})) \overset{(\{a,b,nil\},\{a\}\prec\{b\}):\epsilon}{\mapsto} S :: t_4 \bigtriangledown G :: p_1 \times p_2$$

where $t_4 = G :: lab(t_5).rec\ y.(H :: lab(t_6).t_6 \bigtriangledown \alpha(H))$.

Since $loc(S :: t_4 \bigtriangledown G :: p_1 \times p_2) = \{S, G, E, F, D, C\}$, this transition corresponds to the step $\{t_1, t_3\}$ with respect to inputs that contain event a. Since

$$S :: t_4 \overset{lab(t_5):\epsilon}{\mapsto} S :: rec\ y.(H :: lab(t_6).t_6 \bigtriangledown \alpha(H))$$

by rule $\bigtriangledown_1$,

$$\alpha(\bigtriangledown(S,\{G,H\})) \overset{lab(t_5):\epsilon}{\mapsto} S :: rec\ y.(H :: lab(t_6).t_6 \bigtriangledown \alpha(H)).$$

Since $loc(S :: rec\ y.(H :: lab(t_6).t_6 \bigtriangledown \alpha(H))) = \{S, H\}$, this transition corresponds to the step $\{t_5\}$ from configuration $\{S, G, E, F, A, B\}$ to configuration $\{S, H\}$, when the environment produces d.

Note that rules $\bigtriangledown_1$ and $\bigtriangledown_2$ can be applied, since the type of the action of $S :: t_4$ and $\alpha(G)$ is non-idle. On the other hand, empty steps from $\{S, G, E, F, A, B\}$ must be empty steps both of statechart G and of basic statechart S. We have an empty step iff the environment produce neither a, b nor d. This is reflected by rule $\bigtriangledown_3$, where we require the two parts to agree on idle actions.

Since $\alpha(G) \overset{(\{\bar{a},\bar{b},\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} \alpha(G)$ and $S :: t_4 \overset{(\{\bar{a},\bar{b},\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} S :: t_4$, then

$$S :: t_4 \bigtriangledown \alpha(G) \overset{(\{\bar{a},\bar{b},\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} S :: t_4 \bigtriangledown \alpha(G)$$

and by rule rec_2,

$$\alpha(\bigtriangledown(S,\{G,H\}) \overset{(\{\bar{a},\bar{b},\bar{d}\},\emptyset):\bar{\epsilon}}{\mapsto} \alpha(\bigtriangledown(S,\{G,H\})$$

Note that $\#(lab(t_1)) \cap \#(lab(t_3)) \cap \#(lab(t_5)) = (\{\bar{a},\bar{b},\bar{d}\},\emptyset)$.

3.4 Related works

A process algebra for statecharts has been presented in [88]. The proposed language has an operator of refinement similar to ours and the translation differs only in some details regarding the actions corresponding to transitions. However, the labelled transition system semantics of the language differs in many aspects.

The first difference arises in the definition of basic actions corresponding to transitions. In the definition of [88] transitions are represented by actions $(\ell, \prec)$,

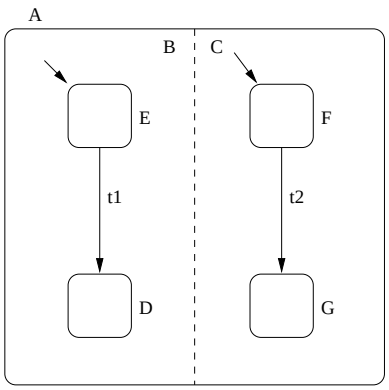

Figure 3.2: The statechart $A : \mathrm{AND}(B, C)$

where ℓ is a set of events and $\prec$ is an irreflexive and transitive ordering on ℓ. On the other hand, in our definition $\prec$ is an ordering over $2^{\Pi \cup \bar{\Pi}}$. This definition leads to some problems when a transition has a set of events as trigger.

Consider two parallel transitions t_1 and t_2 that are labelled by ac/b and b/c, respectively. The action corresponding to transition t_1 is $a_1 = (\{a, b, c\}, a \prec b, c \prec b)$, while that corresponding to t_2 is $a_2 = (\{b, c\}, b \prec c)$. The parallel composition of actions is defined as in our approach. However, if we combine a_1 and a_2 we obtain the set $\{(\{a, b, c\}, a \prec b, c \prec b), (\{a, b, c\}, a \prec b, b \prec c, a \prec c)\}$. The first action corresponds to take both transitions on input $\{a, c\}$. On the other hand, the action $(\{a, b, c\}, a \prec b, b \prec c, a \prec c)$ corresponds to take both transitions on input $\{a\}$. This is not correct, since event a is not sufficient for enabling both transitions.

In our definition, we have adopted a causal ordering between sets of events instead of single events. Hence, the action corresponding to t_1 is $a_1 = (\{a, b, c\}, \{a, c\} \prec \{b\})$, while the one corresponding to t_2 is $a_2 = (\{b, c\}, \{b\} \prec \{c\})$ and the parallel composition of a_1 and a_2 is given by $\{a = (\{a, b, c\}, \{a, c\} \prec \{b\}), a' = (\{a, b, c\}, \{b\} \prec \{c\})\}$, where $trigger(a) = \{a, c\}$ and $trigger(a') = \{a, b\}$. Therefore, event a is not sufficient to trigger the step $\{t_1, t_2\}$.

Another drawback of Uselton and Smolka's approach regards the labelled transition system semantics of the process language. Actually, they do not adopt semantic actions and represent explicitly only non-idle actions. This leads to a semantics that is not equivalent to the one of Pnueli and Shalev.

Consider the statechart of figure 3.2 where t_1 and t_2 are labelled by $\bar{a}/b$ and b/a, respectively. The $\mathcal{SP}$ process corresponding to this statechart is $p = A :: p_1 \times p_2$, where $p_1 = B :: rec\, x.E :: lab(t_1).D :: nil$ and $p_2 = C :: rec\, y.F :: lab(t_2).G :: nil$ with $lab(t_1) = (\{\bar{a}, b\}, \{\bar{a}\} \prec \{b\})$ and $lab(t_2) = (\{b, a\}, \{b\} \prec \{a\})$. The process obtained by Uselton and Smolka is similar to this one and differs only for the notation.

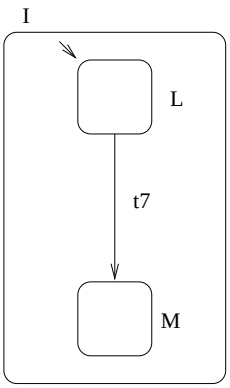

Figure 3.3: The basic statechart I

In their semantics the only transition for the process p is

$$p \overset{(\{b,a\},b\prec a)}{\mapsto} A :: p_1 \times C :: G :: nil$$

which corresponds to the step $\{t_2\}$ with respect to inputs containing event b. Actually, this is the only non-empty step from the configuration $\{A, B, C, E, F\}$, since with respect to inputs that contain neither b nor a, a causal paradox occurs. On the other hand, for inputs that contain a and do not contain b the empty set of transitions is an admissible reaction. However, the difference between the empty step on input a and the "run-time error" on empty input is not reflected by this approach. This gives rise to non-empty steps, that are not admissible. The problem arises, when a non-empty step of the parallel composition is given by an empty step of one component and by a non-empty step of the other component. Since empty steps are not explicitly represented, Uselton and Smolka use an additional rule for parallel composition with negative premises. However, non-admissible steps and empty steps cannot be distinguished by this rule.

Consider for instance the statechart $H : \mathrm{AND}(A, I)$, where I is shown in figure 3.3 and t_7 is labelled by nil/c. The corresponding process is

$$p' = H :: A :: (p_1 \times p_2) \times p_3$$

where $p_3 = I :: rec\ z.L :: lab(t_7).M :: nil = \alpha(I)$. By applying the rule with negative premises, we would obtain the following transition

$$p' \overset{(\{\bar{b},c,nil\},nil\prec c)}{\mapsto} H :: A :: (p_1 \times p_2) \times I :: M :: nil$$

where $(\{\bar{b}, c, nil\}, nil \prec c) \in (\{c, nil\}, nil \prec c) \times (\{\bar{b}\}, \emptyset)$. The action $(\{c, nil\}, nil \prec c)$ is the one corresponding to the non-empty step of I, $p_3 \overset{(\{c,nil\},nil\prec c)}{\mapsto} I :: M :: nil$, while the action $(\{\bar{b}\}, \emptyset)$ represents an empty step of A. The idea is that action

$(\{\bar{b}\},\emptyset)$ is in conflict with the action $(\{b,a\}, b \prec a)$ corresponding to the only non-empty step of A, since if the environment does not produce b then the only non-empty step of A is not enabled. However, since $trigger((\{\bar{b},c,nil\}, nil \prec c)) = \{nil,\bar{b}\}$, this transition corresponds to the step $\{t_7\}$ with respect to inputs that do not contain b. In particular, it is a step with respect to the input that contains neither a nor b. This is not an admissible step in the Pnueli and Shalev semantics, because of the causal paradox.

Therefore, empty steps cannot be obtained from the non-empty ones and must be explicitly represented. In the previous example, the transition $A :: p_1 \times p_2 \overset{(\{\bar{b},a\},\emptyset)}{\mapsto} A :: p_1 \times p_2$ gives the required empty step. Moreover, our approach allows us to define a single rule for parallel composition rather than the complex rules of [88]. Note that also the labelled transition system semantics presented in [87] suffer of the same drawbacks.

3.5 Towards verification of statecharts

In this chapter we have defined the process language $\mathcal{SP}$ and we have shown a translation of statecharts into closed $\mathcal{SP}$ processes according to the semantics of Pnueli and Shalev. This translation is very useful, since it allows us to construct in a compositional way the labelled transition system representing the behaviour of a statechart. Since statecharts are finite-state, properties of a statechart S expressed as μ-calculus formulas can be verified via model checking over the model $\mathcal{M}(\alpha(S)) = (LTS(\alpha(S)), \gamma)$, where $\gamma : boxes(S) \to \mathcal{CSP}$ is the valuation for propositional symbols, such that $p \in \gamma(P)$ iff $P \in loc(p)$.

It is important to stress that propositional μ-calculus is particularly adequate for expressing properties of statecharts. Actually, the next modalities range over labels of the model and allow us to express in a natural way assumptions on the behaviour of the environment.

Consider for instance the system obtained by the parallel composition of the statechart depicted in figure 3.4 with two copies DEV_1 and DEV_2 of the statechart depicted in figure 3.5, where transitions are labelled as follows

$$
\begin{array}{ll}
t_i : req_i/av?, a_i - ok & i \in \{1,3\} \\
t_4 : dev_1/nil & t_5 : nil/dev_1 - st \\
t_6 : dev_2/nil & t_7 : nil/dev_2 - st \\
t_8 : av?/nil & t_9 : nil/dev_i \\
t_{10} : dev_i - st/nil & t_{11} : dev_i - end/dev_i
\end{array}
$$

This statechart describes an arbiter which assigns two equal devices to three concurrent agents. When the arbiter is free and receives a request req_i from the i-th agent, it communicates that the request has been accepted by producing $a_i - ok$. At the same time the arbiter looks for an available device by producing $av?$ and waits for an answer. If the $i-th$ device is free, it produces the signal dev_i so that the

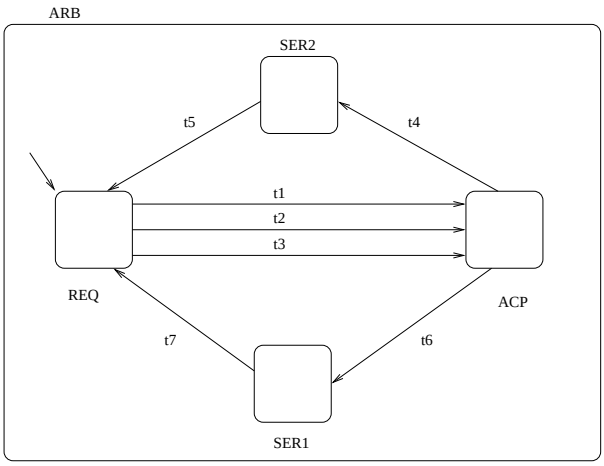

Figure 3.4: The arbiter

arbiter can assign it by producing $dev_i - st$. The same signal $dev_i - st$ is used for the communication with the agent and with the device. Agents release the devices by producing a signal $dev_i - st$ and the i-th device communicates to the arbiter that it is available via signal dev_i. Note that the choice between devices and requests is non deterministic.

In order to ensure the correctness of the system, we would like to verify the following properties:

- always, for each $i \in \{1, 2\}$, if the i-th device is not free, it cannot be assigned.
- always, if a request is accepted, then it is eventually served, if we assume that devices are eventually released by agents.

The first one is obviously a safety property, while the other one is a liveness property. The safety property can be expressed by the μ-calculus formula

$$\nu X.((\bigwedge_{i\in\{1,2\}} (\neg O_i \vee (O_i \wedge [K]false)) \wedge [-]X)$$

where $K = \{\alpha \in \Sigma(\Pi) \mid dev_i - st \in action(\alpha)\}$. The formula represents indeed the property that if the i-th device is not free, then the arbiter cannot produce signal $dev_i - st$ assigning this device.

On the other hand, the other property can be expressed by the formula

$$\nu X.(\bigwedge_{i\in\{1,3\}} ([K](A_i))) \wedge [-]X)$$

where $K = \{\alpha \in \Sigma(\Pi) \mid a_i - ok \in action(\alpha)\}$ and, for each $i \in \{1, 3\}$,

$$A_i \equiv \mu Y. \bigvee_{j\in\{1,2\}} ([K_1]false \vee [K_3]([K_2]false)) \vee [-]Y$$

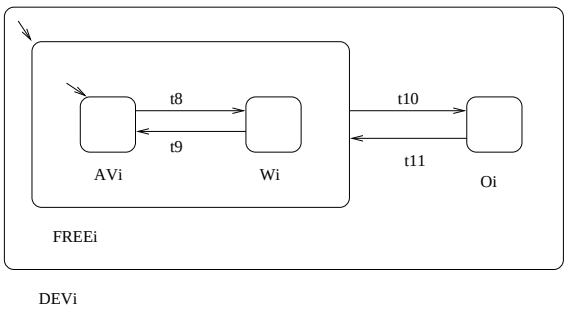

Figure 3.5: The i-th device

for $K_1 = \{\alpha \in \Sigma(\Pi) \mid dev_1 - st, dev_2 - st \notin action(\alpha)\}$ and $K_2 = \{\alpha \in \Sigma(\Pi) \mid dev_j - st \notin action(\alpha)\}$, $K_3 = \{\alpha \in \Sigma(\Pi) \mid dev_j - end \in trigger(\alpha)\}$.

Intuitively, the property states that, whenever a signal $a_i - ok$ is produced (the request is accepted), then eventually either signal $dev_j - st$(a device is assigned) is produced or, if a signal $dev_j - end$ is received (the device is released), then the signal $dev_j - st$ is produced.

Chapter 4

A Compositional Proof system for $\mathcal{SP}$

Since reactive systems tend to be very complex, stepwise and modular development is essential. For this purpose, it is necessary to perform the verification during the process of development, as opposed to verification after the system has been completely specified. The classical global and local model checking methods are not compositional, since rely on the exploration of the model representing the behaviour of the complete system. In contrast, in a compositional approach the verification that a system meets a property is reduced to the verification of certain derived properties of its components. The first advantage is that, when changing a part of the system, only the verification concerning that component must be redone. Secondly, it makes it possible to leave part of the system undefined and still to reason about it. Moreover, the verification task may be reduced into potentially simpler tasks.

Much recent research in the area has focussed on CCS-like languages. The basic idea is to reduce a satisfaction problem for a composed process into equivalent satisfaction problems for its components by exploiting the algebraic structure of the process. The technique has been formalized as a set of reduction rules as well as a proof system, where both the reductions and the proof rules typically depend only on the top-level structure of the process. Therefore, they can be applied without inspecting the internal structure of components. However, the treatment of parallel composition meets some fundamental difficulties and usually requires more informations concerning the structure of processes.

In this chapter we propose a compositional proof system for proving μ-calculus assertions of $\mathcal{SP}$ processes. The proof system is defined in the style of the one in [12], that combines ideas of early works [13, 61, 90, 91, 10]. There are two types of rules:

- logical rules that work only on the structure of the formula;
- compositional rules working on the structure of the process for the *next* modal-

ities, i.e. for formulas $< \alpha > A$ or $[\alpha]A$.

A proof for the statement $p \vdash < \alpha > A$ is obtained by proofs for $p_0 \vdash A_0$ and $p_1 \vdash A_1$, where p_0 and p_1 are subprocesses of p and A_0 and A_1 are derived formulas, and does not depend on the labelled transition system. The construction of the proof is guided both by the syntax of the process and by the formula in a top down approach. Fixpoints formulas are treated by the method of Winskel [92] of tagging fixpoints with processes that we have explained in chapter 2. However, due to the operator of process refinement and to the different operational semantics, our proof system differs significantly from the one of [12]. In particular, the distinction between idle and non-idle actions leads to some difficulties. The proof system is proved to be sound in general and complete for the class of regular closed processes that are finite-state.

This method has a direct application to statecharts due to the compositional translation of chapter 3. Since processes corresponding to statecharts are regular, the proof system is complete for arbitrary statecharts.

4.1 The logic L_μ

We consider formulas of propositional μ-calculus given in a negation free syntax and with tagged fixpoints [92]. Assume a set of logical variables VAR, a set of propositional symbols $PROP$, a set $\mathcal{A}$ of actions and a set of states $\mathcal{S}$. Formulas of L_μ are defined as

$$P \mid \neg P \mid X \mid A \wedge A \mid A \vee A \mid < \alpha > A \mid [\alpha]A \mid \nu X\{U\}.A \mid \mu X\{U\}.A$$

where $X \in VAR$ is a logical variable, $P \in PROP$ is a propositional symbol, $\alpha \in \mathcal{A}$ is an action and $U \subseteq \mathcal{S}$ is a subset of states (tag).

For $U = \emptyset$ we denote by $\nu X.A$ and $\mu X.A$ the formulas $\nu X\{U\}.A$ and $\mu X\{U\}.A$, respectively. Moreover, we use the constants $true$ and $false$ for $\nu X.X$ and $\mu X.X$, respectively.

For a set of basic actions Act and for a set of locations Loc, we say that A is a L_μ formula over Act and Loc iff it is defined over actions $\mathcal{S}(Act)$, propositional symbols Loc and has tags in $\mathcal{CSP}$, namely in the set of closed processes over Act and Loc.

Let $p \in \mathcal{CSP}$ and A be a closed formula, we say that $p \models A$ iff $p \models_{\mathcal{M}(p)} A$, where $\mathcal{M}(p) = (LTS(p), \gamma)$, with $\gamma : Loc \to 2^{\mathcal{CSP}}$, such that $P \in \gamma(p)$, iff $P \in loc(p)$.

4.2 Introduction to the proof system

We give an informal explanation of the basic ideas underlying the proof system for checking whether a closed process p satisfies a closed formula A. The proof of $p \vdash A$ is constructed by exploiting both the structure of process p and the structure of

formula A. We have rules of two types: logical rules that do not involve process operators, and compositional rules for formulas $< \alpha > A$ and $[\alpha]A$. The rules for $< \alpha > A$ and $[\alpha]A$ depend only on the syntax of the process and do not depend on A. Proofs are contructed upside down in a tableau-like fashion.

Consider a basic process $p = \ell_1 :: \ell_2 :: b.\ell_3 :: nil$ with $c \in \#(b)$. We have that $p \models< b : \epsilon > \ell_3$ and $p \models< c : \bar{\epsilon} > \ell_2$, since $p \stackrel{b:\epsilon}{\mapsto} \ell_1 :: \ell_3 :: nil$ and $p \stackrel{c:\bar{\epsilon}}{\mapsto} p$. Since p is a basic process not further decomposable, we can directly obtain its transitions from the syntax. It is obvious that we can deduce $p \vdash< b : \epsilon > \ell_3$ from $\ell_1 :: \ell_3 :: nil \vdash \ell_3$ and $p \vdash< c : \bar{\epsilon} > \ell_2$ from $p \vdash \ell_2$. Moreover, $\ell_1 :: \ell_3 :: nil \vdash \ell_3$ and $p \vdash \ell_2$ trivially hold, since $\ell_3 \in loc(\ell_1 :: \ell_3 :: nil)$ and $\ell_2 \in loc(p)$, respectively.

Consider the process

$$p = \ell_1 :: \ell_2 :: b.\ell_3 :: nil \bigtriangledown \ell_2 :: \ell_4 :: a.\ell_5 :: nil = \ell_1 :: t \bigtriangledown p'$$

and suppose $c \in \#(a) \cap \#(b)$. We have $p \models< a : \epsilon > \ell_1 \wedge \ell_5$, $p \models< b : \epsilon > \ell_3 \wedge \neg\ell_2$ and $p \models< c : \bar{\epsilon} > \ell_2 \wedge \ell_4$, since in the labelled transition system of p, $p \stackrel{a:\epsilon}{\mapsto} \ell_1 :: t \bigtriangledown \ell_2 :: \ell_5 :: nil$, $p \stackrel{b:\epsilon}{\mapsto} \ell_1 :: \ell_3 :: nil$ and $p \stackrel{c:\bar{\epsilon}}{\mapsto} p$.

We would like to prove these properties in a compositional way without constructing the labelled transition system. Consider the formula $< b : \epsilon > \ell_3 \wedge \neg\ell_2$. We have $p \models< b : \epsilon > \ell_3 \wedge \neg\ell_2$, since $p \stackrel{b:\epsilon}{\mapsto} \ell_1 :: \ell_3 :: nil$. Since $type(b : \epsilon) = \epsilon$, this transition is obtained by rule $\bigtriangledown_1$, from $\ell_1 :: t \stackrel{b:\epsilon}{\mapsto} \ell_1 :: \ell_3 :: nil$. Therefore, $\ell_1 :: t \models< b : \epsilon > \ell_3 \wedge \neg\ell_2$. This reasoning justifies a proof as the following.

$$\dfrac{\dfrac{\dfrac{p \vdash< b : \epsilon > \ell_3 \wedge \neg\ell_2}{\ell_1 :: t \vdash< b : \epsilon > \ell_3 \wedge \neg\ell_2}}{\ell_1 :: t \vdash \ell_3 \wedge \neg\ell_2}}{\underline{\ell_1 :: \ell_3 :: nil \vdash \ell_3} \qquad \underline{\ell_1 :: \ell_3 :: nil \vdash \neg\ell_2}}$$

On the other hand, consider the formula $< a : \epsilon > \ell_1 \wedge \ell_5$. We have $p \models< a : \epsilon > \ell_1 \wedge \ell_5$, since $p' \stackrel{a:\epsilon}{\mapsto} \ell_2 :: \ell_5 :: nil$ and by rule $\bigtriangledown_2$, $p \stackrel{a:\epsilon}{\mapsto} \ell_1 : t \bigtriangledown \ell_2 :: \ell_5 :: nil$. However, we cannot prove this property by repeating the same reasoning as before, since rules $\bigtriangledown_1$ and $\bigtriangledown_2$ are not symmetric. Therefore, it would not be correct to deduce $p \vdash< a : \epsilon > \ell_1 \wedge \ell_5$ from $p' \vdash< a : \epsilon > \ell_1 \wedge \ell_5$. These problems can be solved by adopting special formulas, called extended formulas, which have been introduced by [12]. We use a formula $(\ell_1 \wedge \ell_5)_{\ell_1 :: t \bigtriangledown}$ with the following meaning: $p'' \models (\ell_1 \wedge \ell_5)_{\ell_1 :: t \bigtriangledown}$ iff $\ell_1 :: t \bigtriangledown p'' \models \ell_1 \wedge \ell_5$. It is obvious that we can deduce $p \vdash< a : \epsilon > \ell_1 \wedge \ell_5$ from $p' \vdash< a : \epsilon > (\ell_1 \wedge \ell_5)_{\ell_1 :: t \bigtriangledown}$. The problem of checking $< a : \epsilon > \ell_1 \wedge \ell_5$ for process p has been reduced to the problem of checking a derived formula for its component p'. Extended formulas are treated by special rules, called shift rules, that build up the

structure of the process as is recorded in the extended assertion. The rules of the proof system are defined according to this idea and we have a proof as the following.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p \vdash < a : \epsilon > \ell_1 \wedge \ell_5}{p' \vdash < a : \epsilon > (\ell_1 \wedge \ell_5)_{\ell_1 :: t \bigtriangledown}}}{\ell_2 :: \ell_5 :: nil \vdash (\ell_1 \wedge \ell_5)_{\ell_1 :: t \bigtriangledown}}}{\ell_1 :: t \bigtriangledown \ell_2 :: \ell_5 :: nil \vdash \ell_1 \wedge \ell_5}\, shift}{\cdots}$$

Consider the formula $< c : \bar{\epsilon} > \ell_2 \wedge \ell_4$. We have $p \models\!< c : \bar{\epsilon} > \ell_2 \wedge \ell_4$, since $\ell_1 :: t \stackrel{c:\bar{\epsilon}}{\mapsto} \ell_1 :: t$, $p' \stackrel{c:\bar{\epsilon}}{\mapsto} p'$ and $p \models \ell_2 \wedge \ell_4$. Actually, since $type(c : \bar{\epsilon}) = \bar{\epsilon}$, $\ell_1 :: t$ and p' must agree on the idle step as required by rule $\bigtriangledown_3$. This property must be reflected by the rules of the proof system. A compositional proof of $< c : \bar{\epsilon} > \ell_2 \wedge \ell_4$ can be obtained as follows by exploiting the fact that $p \stackrel{\alpha}{\mapsto} p''$ with $type(\alpha) = \bar{\epsilon}$ iff $p = p''$.

$$\dfrac{p \vdash < c : \bar{\epsilon} > \ell_2 \wedge \ell_4}{\dfrac{\ell_1 :: t \vdash < c : \bar{\epsilon} > true}{\ell_1 :: t \vdash true}\, c \in \#(b) \qquad \dfrac{p' \vdash < c : \bar{\epsilon} > true}{p' \vdash true}\, c \in \#(a) \qquad \dfrac{p \vdash \ell_2 \wedge \ell_4}{\cdots}}$$

The proof system is obtained by defining for each process constructor rules similar to the previous ones. The most difficult case is that of parallel composition.

Consider a process $p = \ell_1 :: p_1 \times p_2$, where $p_1 = \ell_2 :: \ell_3 :: a.\ell_4 :: nil$ and $p_2 = \ell_5 :: \ell_6 :: b.\ell_7 :: nil$, and $c \in a \times b$. We have $p \models\!< c : \epsilon > \ell_4 \wedge \ell_7$, since $p \stackrel{c:\epsilon}{\mapsto} \ell_1 :: (\ell_2 :: \ell_4 :: nil) \times (\ell_5 :: \ell_7 :: nil)$. By definition of the semantics this transition is obtained by rule *par* from $p_1 \stackrel{a:\epsilon}{\mapsto} p_1'$ and $p_2 \stackrel{b:\epsilon}{\mapsto} p_2'$, where $p_1' = \ell_2 :: \ell_4 :: nil$ and $p_2' = \ell_5 :: \ell_7 :: nil$.

Our goal is that of avoiding the construction of $p \stackrel{c:\epsilon}{\mapsto} \ell_1 :: p_1' \times p_2'$. We exploit the fact that p_2 is a basic process not further decomposable so that we know by its syntax that there exists a transition $p_2 \stackrel{b:\epsilon}{\mapsto} p_2'$. This reasoning suggests the following compositional proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\ell_1 :: p_1 \times p_2 \vdash < c : \epsilon > \ell_4 \wedge \ell_7}{p_1 \vdash < a : \epsilon > (\ell_4 \wedge \ell_7)_{\ell_1 \times p_2'}}\, c : \epsilon \in (a : \epsilon) \times (b : \epsilon)}{p_1' \vdash (\ell_4 \wedge \ell_7)_{\ell_1 \times p_2'}}}{\ell_1 p_1' \times p_2' \vdash \ell_4 \wedge \ell_7}}{\cdots}$$

The problem of verifying whether $\ell_1 :: p_1 \times p_2$ satisfies $< c : \epsilon > \ell_4 \wedge \ell_7$ has been reduced to the problem of verifying whether p_1 satisfies $< a : \epsilon > (\ell_4 \wedge \ell_7)_{\ell_1 \times p_2'}$. Note that we have used an extended formula $(\ell_4 \wedge \ell_7)_{\ell_1 \times p_2'}$ with the obvious semantics $p'' \models (\ell_4 \wedge \ell_7)_{\ell_1 \times p_2'}$ iff $\ell_1 :: p'' \times p_2' \models \ell_4 \wedge \ell_7$.

Consider a process $p = \ell_1 :: p_1 \times p_2$, where p_2 is not a basic process. The rules are based on the idea of breaking the structure of p_2 in order to reduce the satisfaction problem for p to a satisfaction problem for some of its components.

4.3 The proof system

In this section we present the proof system.

4.3.1 Logical Rules

Logical rules do not involve process operators.

$$\frac{}{p \vdash P}\, prop_1 \quad P \in loc(p) \qquad \frac{}{p \vdash \neg P}\, prop_2 \quad P \notin loc(P)$$

$$\frac{p \vdash A_0 \wedge A_1}{p \vdash A_0 \quad p \vdash A_1}\, \wedge$$

$$\frac{p \vdash A_0 \vee A_1}{p \vdash A_0}\, \vee_1 \qquad \frac{p \vdash A_0 \vee A_1}{p \vdash A_1}\, \vee_2$$

$$\frac{p \vdash \mu X\{U\}.A}{p \vdash A[\mu X\{U, p\}.A/X]}\, \mu \quad p \notin U$$

$$\frac{p \vdash \nu X\{U\}.A}{p \vdash A[\nu X\{U, p\}.A/X]}\, \nu_1 \quad p \notin U$$

$$\frac{p \vdash \nu X\{U, p\}.A}{}\, \nu_2$$

These rules are quite obvious. The soundness of fixed points rules relies on the reduction lemma 2.1.3.

4.3.2 Rules for *nil*

$$\frac{\ell :: \ell_1 :: nil \vdash < a : \bar{\epsilon} > A}{\ell :: \ell_1 :: nil \vdash A}\, <> nil \quad a \in \#(*)$$

$$\frac{\ell :: \ell_1 :: nil \vdash [a : \epsilon]A}{} \, []nil_1$$

$$\frac{\ell :: \ell_1 :: nil \vdash [a : \bar{\epsilon}]A}{} \, []nil_2 \quad a \notin \#(*)$$

$$\frac{\ell :: \ell_1 :: nil \vdash [a : \bar{\epsilon}]A}{\ell :: \ell_1 :: nil \vdash A} \, []nil_3 \quad a \in \#(*)$$

These rules are quite obvious, since we have $\ell :: \ell_1 :: nil \stackrel{\alpha}{\mapsto} p'$ iff $\alpha = a : \bar{\epsilon}$, for $a \in \#(*)$, and $p' = \ell :: \ell_1 :: nil$.

4.3.3 Rules for prefixing

$$\frac{\ell :: \ell_1 :: a.t \vdash < a : \epsilon > A}{\ell :: t \vdash A} <>_1$$

$$\frac{\ell :: \ell_1 :: a.t \vdash [a : \epsilon]A}{\ell :: t \vdash A} \, []_1$$

$$\frac{\ell :: \ell_1 :: a_1.t \vdash < a_2 : \bar{\epsilon} > A}{\ell :: \ell_1 :: a_1.t \vdash A} <>_2 \quad a_2 \in \#(a_1)$$

$$\frac{\ell :: \ell_1 :: a_1.t \vdash [a_2 : \bar{\epsilon}]A}{\ell :: \ell_1 :: a_1.t \vdash A} \, []_2 \qquad a_2 \in \#(a_1)$$

$$\frac{\ell :: \ell_1 :: a_1.t \vdash [a_2 : \epsilon]A}{} \, []_3 \quad a_1 \neq a_2$$

$$\frac{\ell :: \ell_1 :: a_1.t \vdash [a_2 : \bar{\epsilon}]A}{} \, []_4 \quad a_2 \notin \#(a_1)$$

These rules are quite obvious, since we have $\ell :: \ell_1 :: a.t \stackrel{\alpha}{\mapsto} p'$ iff one of the following cases holds:

1. $\alpha = a : \epsilon$ and $p' = \ell :: t$;
2. $\alpha = b : \bar{\epsilon}$, for $b \in \#(a)$, and $p' = \ell :: \ell_1 :: a.t$.

4.3.4 Rules for choice

Rules are of two types: rules applicable when $type(\alpha) = \epsilon$, and rules applicable when $type(\alpha) = \bar{\epsilon}$. Actually, it is necessary to distinguish if either rule $choice_1$ or rule $choice_2$ is applicable.

1. Rules applicable iff $type(\alpha) = \epsilon$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash < \alpha > A}{\ell :: \ell_1 :: b_i \vdash < \alpha > A} + <> \epsilon$$

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash [\alpha]A}{\{\ell :: \ell_1 :: b_i \vdash [\alpha]A\}_{i \in \{1,2\}}} + [] \epsilon$$

2. Rules applicable iff $type(\alpha) = \bar{\epsilon}$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash < \alpha > A}{\{\ell :: \ell_1 :: b_i \vdash < \alpha > true\}_{i \in \{1,2\}} \quad \ell :: \ell_1 :: b_1 + b_2 \vdash A} + <> \bar{\epsilon}$$

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash [\alpha]A}{\ell :: \ell_1 :: b_1 + b_2 \vdash A} + [] \bar{\epsilon}_1$$

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash [\alpha]A}{\ell :: \ell_1 :: b_i \vdash [\alpha] false} + [] \bar{\epsilon}_2$$

These rules are based on the observation that $\ell :: \ell_1 :: b_1 + b_2 \stackrel{\alpha}{\mapsto} p'$ iff one of the following cases holds:

1. $\ell :: \ell_1 :: b_i \stackrel{\alpha}{\mapsto} p'$, for $i \in \{1, 2\}$, and $type(\alpha) = \epsilon$ (rule $choice_1$);
2. $\ell :: \ell_1 :: b_i \stackrel{\alpha}{\mapsto} \ell :: \ell_1 :: b_i$, for each $i \in \{1, 2\}$, and $type(\alpha) = \bar{\epsilon}$ and $p = p'$ (rule $choice_2$).

4.3.5 Rules for process refinement

Rules for process refinement are similar to the rules for choice. However, in order to reflect rule $\bigtriangledown_2$ of the operational semantics, *extended formulas* are used. We define the extended formula $(A)_{\ell::t\bigtriangledown}$ with the semantics

$$\| (A)_{\ell::t\bigtriangledown} \| = \{p \in \mathcal{CSP} \mid \ell :: t \bigtriangledown p \in \| A \|\}.$$

The corresponding shift rule is:

$$\frac{p \vdash (A)_{\ell::t\bigtriangledown}}{\ell :: t \bigtriangledown p \vdash A}$$

1. Rules applicable iff $type(\alpha) = \epsilon$.

$$\frac{\ell :: t \bigtriangledown p \vdash < \alpha > A}{\ell :: t \vdash < \alpha > A} \bigtriangledown <> \epsilon_1$$

$$\frac{\ell :: t \bigtriangledown p \vdash < \alpha > A}{p \vdash < \alpha > (A)_{\ell :: t \bigtriangledown}} \bigtriangledown <> \epsilon_2$$

$$\frac{\ell :: t \bigtriangledown p \vdash [\alpha]A}{p \vdash [\alpha](A)_{\ell :: t \bigtriangledown} \quad \ell :: t \vdash [\alpha]A} \bigtriangledown [] \epsilon$$

2. Rules applicable iff $type(\alpha) = \bar{\epsilon}$.

$$\frac{\ell :: t \bigtriangledown p \vdash < \alpha > A}{\ell :: t \vdash < \alpha > true \quad p \vdash < \alpha > true \quad \ell :: t \bigtriangledown p \vdash A} \bigtriangledown <> \bar{\epsilon}$$

$$\frac{\ell :: t \bigtriangledown p \vdash [\alpha]A}{\ell :: t \bigtriangledown p \vdash A} \bigtriangledown [] \bar{\epsilon}_1$$

$$\frac{\ell :: t \bigtriangledown p \vdash [\alpha]A}{\ell :: t \vdash [\alpha] false} \bigtriangledown [] \bar{\epsilon}_2$$

$$\frac{\ell :: t \bigtriangledown p \vdash [\alpha]A}{p \vdash [\alpha] false} \bigtriangledown [] \bar{\epsilon}_3$$

These rules are based on the observation that $\ell :: t \bigtriangledown p \stackrel{\alpha}{\mapsto} p'$ iff one of the following cases holds:

1. $\ell :: t \stackrel{\alpha}{\mapsto} p'$ and $type(\alpha) = \epsilon$ (rule $\bigtriangledown_1$);
2. $p \stackrel{\alpha}{\mapsto} p''$ and $type(\alpha) = \epsilon$, $p' = \ell :: t \bigtriangledown p''$ (rule $\bigtriangledown_2$);
3. $\ell :: t \stackrel{\alpha}{\mapsto} \ell :: t$ and $p \stackrel{\alpha}{\mapsto} p$ and $type(\alpha) = \bar{\epsilon}$, $p' = \ell : t \bigtriangledown p$ (rule $\bigtriangledown_3$).

4.3.6 Rules for recursion

$$\frac{\ell :: rec\ x.t \vdash [\alpha]A}{\ell :: t[rec\ x.t/x] \vdash [\alpha]A} rec[] \qquad \alpha \in \mathcal{S}(Act)$$

$$\frac{\ell :: rec\ x.t \vdash < \alpha > A}{\ell :: t[rec\ x.t/x] \vdash < \alpha > A} rec <> \quad \alpha \in \mathcal{S}(Act)$$

4.3.7 Rules for parallel composition

The case of parallel composition is the most complex. We would like to give a set of rules for proving in a compositional way $\ell :: p_1 \times p_2 \vdash [\alpha]A$ (*resp.* $< \alpha > A$) by exploiting the following property:

$$\ell :: p_1 \times p_2 \stackrel{\alpha}{\mapsto} p' \iff p_1 \stackrel{\alpha_1}{\mapsto} p_1' \quad p_2 \stackrel{\alpha_2}{\mapsto} p_2'$$

where $\alpha \in \alpha_1 \times \alpha_2$ and $p' = \ell :: p_1' \times p_2'$.

The idea is that of reducing a judgment $p \vdash\!< \alpha > A$ to judgments $\{\ell :: p_1 \times p_{2,j} \vdash\!< \alpha_j > A\}_{j\in\{1,n\}}$, where $p_{2,j}$ are components of p_2, depending on the structure of p_2. Unfortunately, the informations recorded in semantic actions are not sufficient.

Suppose that $p_2 = \ell_1 :: \ell_2 :: b_1 + b_2$ and that one wants to prove $\ell :: p_1 \times p_2 \vdash\!< \alpha > A$, where $type(\alpha) = \epsilon$. We would like to break the process $\ell_1 :: \ell_2 :: b_1 + b_2$.

One could think of obtaining a proof of $\ell :: p_1 \times p_2 \vdash\!< \alpha > A$ either from one of $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \vdash\!< \alpha > A$ or from one of $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 \vdash\!< \alpha > A$. However, this kind of deduction is not correct in general.

Suppose that $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \models\!< \alpha > A$. By definition of the semantics, there exists $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \stackrel{\alpha}{\mapsto} \ell :: p_1' \times p_2'$, such that $\ell :: p_1' \times p_2' \models A$. Moreover, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \stackrel{\alpha}{\mapsto} \ell :: p_1' \times p_2'$ has been obtained by rule *par* from $p_1 \stackrel{\alpha_1}{\mapsto} p_1'$ and $\ell_1 :: \ell_2 :: b_2 \stackrel{\alpha_2}{\mapsto} p_2'$ if $\alpha \in \alpha_1 \times \alpha_2$. However, it may be the case that $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \not\models\!< \alpha > A$. If $type(\alpha_2) = \epsilon$ this is correct, since by rule $choice_1$, $\ell_1 :: \ell_2 :: b_1 + b_2 \stackrel{\alpha_2}{\mapsto} p_2'$. On the other hand, if $type(\alpha_2) = \bar{\epsilon}$, rule $choice_1$ is not applicable.

Therefore, it is necessary to distinguish two cases, namely when p_2 performs an idle action and when p_2 performs a non-idle action. Since $type(\alpha_2)$ cannot be deduced from $type(\alpha)$, we adopt formulas with modal operators over *extended actions*, which allow us to specify the type of the two components.

Definition 4.3.1 *Let Act be a set of basic actions. We define* $\mathcal{E}(Act)$ *as the set of* extended actions $\{a : \tau\}$*, where* $a \in Act$ *and* τ *is an* extended *type such that:*

- $\epsilon, \bar{\epsilon}$ *are basic extended types;*
- *if* τ_1 *is an extended type and* τ_2 *is a basic extended type, then* $\tau_1 \times \tau_2$ *is an extended type.*

We define the function $type^* : \mathcal{E}(Act) \to \{\epsilon, \bar{\epsilon}\}$ as follows. For $a : \tau \in \mathcal{S}(Act)$, then $type^*(a : \tau) = \tau$. For $a : \tau \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$, where $\tau = \tau_1 \times \tau_2$, then $type^*(a : \tau) = \epsilon$ if $type^*(a : \tau_i) = \epsilon$, for some $i \in \{1, 2\}$, and $type^*(a : \tau) = \bar{\epsilon}$, otherwise.

The semantics of $< \alpha > A$ and $[\alpha]A$ with $\alpha \in \mathcal{E}(Act)$ is defined as follows. Assume $\alpha \in \mathcal{E}(Act)$. We say that $p \stackrel{\alpha'}{\mapsto} p'$ is consistent with α iff

1. $\alpha = a : \tau$ and $\alpha' = a : \tau'$, where $type(\alpha') = type^*(\alpha)$;

2. If $\alpha = a : \tau_1 \times \tau_2 \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$, then $p \stackrel{\alpha'}{\mapsto} p'$ has been been obtained by rule *par* from $p_i \stackrel{\alpha'_i}{\mapsto} p'_i$, such that $p = \ell :: p_1 \times p_2$, $p' = \ell :: p'_1 \times p'_2$, $\alpha' \in \alpha'_1 \times \alpha'_2$, and $p_i \stackrel{\alpha'_i}{\mapsto} p'_i$ is consistent with α_i, where $\alpha_i = a_i : \tau_i$ and $\alpha'_i = a_i : \tau'_i$.

For formulas $< \alpha > A$ and $[\alpha]A$ with $\alpha \in \mathcal{E}(Act)$, we define

$$\| < \alpha > A \| = \left\{ \begin{array}{l} p \in \mathcal{CSP} \mid \exists p' \in \mathcal{CSP}, p \stackrel{\alpha'}{\mapsto} p', p' \in \| A \| \\ p \stackrel{\alpha'}{\mapsto} p' \text{is consistent with } \alpha \end{array} \right\}$$

$$\| [\alpha]A \| = \left\{ \begin{array}{l} \{p \in \mathcal{CSP} \mid \forall p' \in \mathcal{CSP}, p \stackrel{\alpha}{\mapsto} p' \text{and is consistent with } \alpha \\ \Longrightarrow p' \in \| A \| \end{array} \right\}$$

Intuitively, $p \models < \alpha > A$ with $\alpha \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$ iff there exists $p \stackrel{\alpha'}{\mapsto} p'$, where $p' \models A$, such that $\alpha = a : \tau$, $\alpha' = a : \tau'$ and the transition is constructed consistently with τ. For instance, if $\tau = \epsilon \times \bar{\epsilon}$, then $p \stackrel{\alpha'}{\mapsto} p'$ must be obtained by rule *par* applied to premises $\{p_i \stackrel{\alpha'_i}{\mapsto} p'_i\}_{i \in \{1,2\}}$, where $type(\alpha'_1) = \epsilon$ and $type(\alpha'_2) = \bar{\epsilon}$.

In order to give the rules for the parallel composition, we extend the operation $\times$ to extended actions. Assume $a_1 : \tau_1 \in \mathcal{E}(Act)$, $a_2 : \tau_2 \in \mathcal{S}(Act)$. We define $(a : \tau_1) \otimes (a : \tau_2) = \{a : \tau_1 \times \tau_2 \mid a \in a_1 \times a_2\} \subseteq \mathcal{E}(Act)$.

Expansion rules

Formulas with extended actions are introduced by the following rules.

$$\frac{\ell :: p_1 \times p_2 \vdash < \alpha > A}{\ell :: p_1 \times p_2 \vdash < \beta > A} \mathcal{E} <> \qquad \beta \in \mathcal{E}(\alpha)$$

$$\frac{\ell :: p_1 \times p_2 \vdash [\alpha]A}{\{\ell :: p_1 \times p_2 \vdash [\beta_i]A\}_{i \in \{1,n\}}} \mathcal{E}[] \qquad \{\beta_1, \ldots, \beta_n\} = \mathcal{E}(\alpha)$$

where, for $\alpha = a : \tau \in \mathcal{S}(Act)$, $\mathcal{E}(\alpha) = \{a : \tau' \in \mathcal{E}(Act)\}$, such that $\tau' \in \mathcal{E}(\tau)$ is an extended type, where

1. if $\tau = \epsilon$, then $\mathcal{E}(\tau) = \{\epsilon \times \epsilon, \bar{\epsilon} \times \epsilon, \epsilon \times \bar{\epsilon}\}$;

2. if $\tau = \bar{\epsilon}$, then $\mathcal{E}(\tau) = \{\bar{\epsilon} \times \bar{\epsilon}\}$.

These rules are based on the consideration that an arrow $\ell :: p_1 \times p_2 \stackrel{\alpha}{\mapsto} \ell :: p'_1 \times p'_2$ is obtained by rule *par* from $\{p_i \stackrel{\alpha_i}{\mapsto} p_i\}_{i \in \{1,2\}}$, with $\alpha \in \alpha_1 \times \alpha_2$. Thus, if $type(\alpha) = \epsilon$, then $type(\alpha_i) = \epsilon$, for some $i \in \{1, 2\}$. On the other hand, if $type(\alpha) = \bar{\epsilon}$, then $type(\alpha_i) = \bar{\epsilon}$, for each $i \in \{1, 2\}$.

Rules for nil

We define extended formulas $(A)_{\ell\times p_2}$ with the following semantics:

$$\| (A)_{\ell\times p_2} \| = \{p_1 \in \mathcal{CSP} \mid \ell :: p_1 \times p_2 \in \| A \|\}$$

The corresponding shift rule is:

$$\frac{p_1 \vdash (A)_{\ell\times p_2}}{\ell :: p_1 \times p_2 \vdash A}$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: nil \vdash < \alpha > A}{p_1 \vdash < \beta > (A)_{\ell\times\ell_1::\ell_2::nil}} \times <> nil \quad \begin{array}{l}\alpha \in \beta \otimes (b : \bar{\epsilon}) \\ b \in \#(*)\end{array}$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: nil \vdash [\alpha]A}{\{p_1 \vdash [\beta_i](A)_{\ell\times\ell_1::\ell_2::nil}\}_{i\in\{1,k\}}} \times [] nil \quad \begin{array}{l}\{\beta_i\}_{i\in\{1,k\}} = \\ \{\beta_i \mid \alpha \in \beta_i \otimes (b : \bar{\epsilon}) \\ b \in \#(*)\}\end{array}$$

These rules are based on the observation that for $p_2 = \ell_1 :: \ell_2 :: nil$, $p_2 \stackrel{\alpha_2}{\longmapsto} p_2'$ iff $p_2' = p_2$ and $\alpha_2 = b : \bar{\epsilon}$, for $b \in \#(*)$.

Rules for prefixing

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \vdash < \alpha > A}{p_1 \vdash < \beta > (A)_{\ell\times\ell_1::t}} \times <>_1 \quad \alpha \in \beta \otimes (a : \epsilon)$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \vdash < \alpha > A}{p_1 \vdash < \beta > (A)_{\ell\times\ell_1::\ell_2::a.t}} \times <>_2 \quad \begin{array}{l}\alpha \in \beta \times (b : \bar{\epsilon}) \\ b \in \#(a)\end{array}$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \vdash [\alpha]A}{\begin{array}{l}\{p_1 \vdash [\alpha_i](A)_{\ell\times\ell_1::t}\}_{i\in\{1,n\}} \\ \{p_1 \vdash [\beta_i](A)_{\ell\times\ell_1::\ell_2::a.t}\}_{i\in\{1,k\}}\end{array}} \times [] \quad \begin{array}{l}\{\alpha_i\}_{i\in\{1,n\}} = \\ \{\alpha_i \mid \alpha \in \alpha_i \otimes (a : \epsilon)\} \\ \{\beta_i\}_{i\in\{1,k\}} = \\ \{\beta_i \mid \alpha \in \beta_i \otimes (b : \bar{\epsilon}), b \in \#(a)\}\end{array}$$

These rules are based on the observation that, if $p_2 = \ell_1 :: \ell_2 :: a.t$, then $p_2 \stackrel{\alpha_2}{\mapsto} p_2'$ iff one of the following cases holds:

1. $\alpha_2 = a : \epsilon$ and $p_2' = \ell_1 : t$;

2. $\alpha_2 = b : \bar{\epsilon}$, for $b \in \#(a)$, and $p_2' = p_2$.

Rules for choice

Rules for $p_2 = \ell_1 :: \ell_2 :: b_1 + b_2$ are partitioned into two sets. The first set is given by rules applicable when the type of the action of p_2 is ϵ, while the second one when the type of the action of t_2 is $\bar{\epsilon}$. Assume $\alpha = a : \tau_1 \times \tau_2 \in \mathcal{E}(Act)$, we use the notation $proj_i(\alpha)$ to denote $a : \tau_i \in \mathcal{E}(Act)$. Moreover, we define extended formulas $(A)_{\times b+}$ with the semantics

$$\| (A)_{\times b+} \| = \{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 \in \mathcal{CSP} \mid \ell :: p_1 \times \ell_1 :: \ell_2 :: b + b_1 \in \| A \|\}$$

The shift rule is

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 \vdash (A)_{\times b+}}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b + b_1 \vdash A}$$

1. Rules applicable iff $type(proj_2(\alpha)) = \epsilon$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash < \alpha > A}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash < \alpha > A} \times <> +\epsilon$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash [\alpha]A}{\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash [\alpha]A\}_{i \in \{1,2\}}} \times [] + \epsilon$$

2. Rules applicable iff $type(proj_2(\alpha)) = \bar{\epsilon}$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash < \alpha > A}{\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash < \alpha > (A)_{\times b_j+}\}_{i,j \in \{1,2\}, i \neq j}} \times <> +\bar{\epsilon}$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash [\alpha]A}{\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash [\alpha](A)_{\times b_j+}\}_{i,j \in \{1,2\}, i \neq j}} \times [] + \bar{\epsilon}_1$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash [\alpha]A}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash [\alpha] false} \times [] + \bar{\epsilon}_2$$

These rules are based on the following observation. For $p_2 = \ell_1 :: \ell_2 :: b_1 + b_2$, $p_2 \stackrel{\alpha_2}{\mapsto} p_2'$ iff one of the following cases hold:

1. $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha_2}{\mapsto} p_2'$, for some $i \in \{1, 2\}$, and $type(\alpha_2) = \epsilon$;
2. $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha_2}{\mapsto} \ell_1 :: \ell_2 :: b_i$, for each $i \in \{1, 2\}$, and $type(\alpha_2) = \bar{\epsilon}$, $p_2' = p_2$.

In rules $\times <> +\epsilon$ and $\times [] + \epsilon$ we exploit the fact that case 1. occurs and that, by rule par, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha}{\mapsto} \ell :: p_1' \times p_2'$, since $\alpha \in \alpha_1 \times \alpha_2$.

In rules $\times <> +\bar{\epsilon}$ and $\times [] + \bar{\epsilon}_i$ we exploit the fact that case 2. occurs and that, by rule par, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha}{\mapsto} \ell :: p_1' \times p_2'$, since $\alpha \in \alpha_1 \times \alpha_2$, by the properties of $\times$.

Rules for process refinement

Rules for refinement are similar to the ones for choice with the feature that rules $\bigtriangledown_1$ and $\bigtriangledown_2$ are not symmetric. We define extended formulas $(A)_{\times \ell_1 :: t \bigtriangledown}$ and $(A)_{\times \bigtriangledown p}$ with the semantics

$$\| (A)_{\times \ell_1 :: t \bigtriangledown} \| = \{ \ell :: p_1 \times p_2 \in \mathcal{CSP} \mid \ell :: p_1 \times \ell_1 :: t \bigtriangledown p \in \| A \| \}$$

and

$$\| (A)_{\times \bigtriangledown p} \| = \{ \ell :: p_1 \times \ell_1 :: t \in \mathcal{CSP} \mid \ell :: p_1 \times \ell_1 :: t \bigtriangledown p \in \| A \| \}$$

The corresponding shift rules are

$$\frac{\ell :: p_1 \times p_2 \vdash (A)_{\times \ell_1 :: t \bigtriangledown}}{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p_2 \vdash A}$$

and

$$\frac{\ell :: p_1 \times \ell_1 :: t \vdash (A)_{\times \bigtriangledown p}}{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p \vdash A}$$

1. Rules applicable iff $type(proj_2(\alpha)) = \epsilon$.

$$\frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash < \alpha > A}{\ell :: p_1 \times \ell_2 :: t \vdash < \alpha > A} \times <> \bigtriangledown \epsilon_1$$

$$\frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash < \alpha > A}{\ell :: p_1 \times p \vdash < \alpha > (A)_{\times \ell_2 :: t \bigtriangledown}} \times <> \bigtriangledown \epsilon_2$$

$$\frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash [\alpha] A}{\ell :: p_1 \times p \vdash [\alpha](A)_{\times \ell_2 :: t \bigtriangledown} \quad \ell :: p_1 \times \ell_2 :: t \vdash [\alpha] A} \times [] \bigtriangledown \epsilon$$

2. Rules applicable iff $type(proj_2(\alpha)) = \bar{\epsilon}$.

$$\frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash < \alpha > A}{\ell :: p_1 \times \ell_2 :: t \vdash < \alpha > (A)_{\times \bigtriangledown p} \quad \ell :: p_1 \times p \vdash < \alpha > (A)_{\times \ell_2 :: t \bigtriangledown}} \times <> \bigtriangledown \bar{\epsilon}$$

$$\frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash [\alpha] A}{\ell :: p_1 \times \ell_2 :: t \vdash [\alpha](A)_{\times \bigtriangledown p} \quad \ell :: p_1 \times p \vdash [\alpha](A)_{\times \ell_2 :: t \bigtriangledown}} \times [] \bigtriangledown \bar{\epsilon}_1$$

$$\frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash [\alpha] A}{\ell :: p_1 \times \ell_2 :: t \vdash [\alpha] false} \times [] \bigtriangledown \bar{\epsilon}_2$$

$$\frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash [\alpha] A}{\ell :: p_1 \times p \vdash [\alpha] false} \times [] \bigtriangledown \bar{\epsilon}_3$$

Rules for recursion

$$\frac{\ell :: p_1 \times \ell_2 :: rec\ x.t \vdash < \alpha > A}{\ell :: p_1 \times \ell_2 :: t[rec\ x.t/x] \vdash < \alpha > A} \times <> rec$$

$$\frac{\ell :: p_1 \times \ell_2 :: rec\ x.t \vdash [\alpha]A}{\ell :: p_1 \times \ell_2 :: t[rec\ x.t/x] \vdash [\alpha]A} \times [] rec$$

These rules are trivially based on the operational semantics of $rec\ x.t$.

Rules for parallel composition

We define extended formulas $(A)_\times$ with the semantics

$$\| (A)_\times \| = \{\ell :: \ell_1 :: (p_1 \times p_2) \times p_3 \in \mathcal{CSP} \mid \ell :: p_1 \times \ell_1 :: (p_2 \times p_3) \in \| A \|\}$$

The shift rule is

$$\frac{\ell :: \ell_1 :: (p_1 \times p_2) \times p_3 \vdash (A)_\times}{\ell :: p_1 \times \ell_1 :: (p_2 \times p_3) \vdash A}$$

$$\frac{\ell :: p_1 \times \ell_1 :: (p_3 \times p_4) \vdash < \alpha > A}{\ell :: \ell_1 :: (p_1 \times p_3) \times p_4 \vdash < \beta > (A)_\times} \times\times <> \qquad \beta \in left(\alpha)$$

$$\frac{\ell :: p_1 \times \ell_1 :: (p_3 \times p_4) \vdash [\alpha]A}{\{\ell :: \ell_1 :: (p_1 \times p_3) \times p_4 \vdash [\beta_i](A)_\times\}_{i \in \{1,n\}}} \times\times [] \quad \begin{array}{l}\{\beta_1, \ldots \beta_n\} = \\ \{\beta_i \mid \in left(\alpha)\}\end{array}$$

where, for $a :: \tau_1 \times \tau_2 \in \mathcal{E}(Act)$, $left(a : \tau_1 \times \tau_2) = \{a : \tau\}$, such that $\tau = (\tau_1 \times \tau_{2,1}) \times \tau_{2,2}$, for $\tau_{2,1} \times \tau_{2,2} \in \mathcal{E}(\tau_2)$.

The underlying idea of these rules is that it is sufficient to associate on the left. The soundness of the rules is ensured by the associative property of parallel composition. However, the type of action α must be changed to reflect the shift. Therefore, we consider actions in $left(\alpha)$.

4.4 Soundness and completeness of the proof system

The proof system is sound for arbitrary $\mathcal{CSP}$ processes. Therefore, it is possible to apply the proof system also for infinite state processes.

Theorem 4.4.1 *(Soundness) Assume $p \in \mathcal{CSP}$ over Act and Loc. For a closed L_μ formula A over Act and Loc, $p \vdash A$ implies $p \models A$.*

Proof of this theorem is shown in appendix B.1 and is obtained by proving that each rule is backward sound. The soundness of fixpoints rules follows from the reduction lemma 2.1.3 (chapter 2).

The proof system is not complete in general, but it is complete for regular processes that are finite-state. The proof of completeness is based on the definition of an ordering $\prec$ over formulas, which is well-founded for regular processes.

Definition 4.4.2 *Let A and A' be closed L_μ formulas over Act and Loc. We define $A \prec A'$ iff either A is a proper subformula of A' or $A' = \sigma X\{U\}B$ and $A = B[\sigma X\{U,p\}B/X]$ for $p \notin U$ and $\sigma \in \{\mu,\nu\}$, $p \in \mathcal{CSP}$.*

It is quite obvious that for formulas with tags in a finite subset of $\mathcal{CSP}$ the relation $\prec$ is well-founded.

The proof of completeness exploits a main lemma. Assume $p \models < \alpha > A$ (resp. $p \models [\alpha]A$). The lemma shows that there exist sets of processes, $\{p_i\}_{i\in\{1,n\}}$ and $\{p'_j\}_{j\in\{1,k\}}$, such that $p_i \models A$, and p'_j, p_i are subprocesses of p. Moreover, $p \vdash < \alpha > A$ (resp. $p \vdash [\alpha]A$) can be derived from premises $\{p_i \vdash A\}_{i\in\{1,n\}}$ and $\{p'_j \vdash true\}_{j\in\{1,k\}}$.

Subprocesses of p are formally defined as follows.

Definition 4.4.3 *Let $p \in \mathcal{CSP}$. We say that $p' \in \mathcal{CSP}$ is* subprocess *of p iff one of the following cases holds:*

1. $p = p'$;
2. $p = \ell :: \ell_1 :: b_1 + b_2$ *and* $p' = \ell :: \ell_1 :: b_i$;
3. $p = \ell :: rec\ x.t_2$ *and* $p' = \ell :: t_2[rec\ x.t_2/x]$;
4. $p = \ell :: t \bigtriangledown p''$ *and* $p' = p''$ *or* $p' = \ell :: t$;
5. $p = \ell :: p_1 \times p_2$ *and* $p' = p_i$;
6. $p = \ell :: p_1 \times \ell_1 :: (p_2 \times p_3)$ *and* $p' = \ell :: \ell_1 :: (p_1 \times p_2) \times p_3$;
7. *p' is obtained by replacing in p a subprocess p'' with p''', where p''' is a subprocess of p''.*
8. *p' is a subprocess of p'' and p'' is a subprocess of p.*

Assume $p \in \mathcal{CSP}$. We denote by $\mathcal{S}_p$ the least subset of closed $\mathcal{CSP}$ processes such that, if p' is a subprocess of p, then $\mathcal{R}_{p'} \subseteq \mathcal{S}_p$.

Lemma 4.4.4 *Assume $p \in \mathcal{CSP}$ over Act and Loc and A a closed L_μ formula over Act and Loc. If $p \models < \alpha > A$ (resp. $p \models [\alpha]A$), with $\alpha \in \mathcal{E}(Act)$, then*

1. *there exist sets $\{p_i \in \mathcal{S}_p\}_{i\in\{1,n\}}$ and $\{p'_j \in \mathcal{S}_p\}_{j\in\{1,k\}}$, such that $\{p_i \models A\}_{i\in\{1,n\}}$;*

2. $p \vdash < \alpha > A$ *(resp.* $p \vdash [\alpha]A$*)) can be deduced from* $\{p_i \vdash A\}_{i\in\{1,n\}}$ *and* $\{p'_j \vdash true\}_{j\in\{1,k\}}$.

Moreover, for each $i \in \{1, n\}$*, there exists* $p \stackrel{\alpha_i}{\mapsto} p_i$ *that is consistent with* α.

The proof of this lemma is shown in appendix B.2 and is by induction on a measure $m(p)$, which is defined as follows

- $m(\ell_1 :: \ell_2 :: nil) = 0$;
- $m(\ell_1 :: \ell_2 :: a.p) = 0$;
- $m(\ell_1 :: \ell_2 :: b_1 + b_2) = 1 + max(m(\ell_1 :: \ell_2 :: b_1), m(\ell_1 :: \ell_2 :: b_1))$;
- $m(\ell :: t \bigtriangledown p) = 1 + max(m(\ell :: t), m(p))$;
- $m(\ell :: rec\ x.t) = 1 + m(\ell :: t[rec\ x.t/x])$;
- $m(\ell :: p_1 \times p_2) = 1 + m(p_1) + 2m(p_2)$.

Note that the measure is defined so that $m(\ell_1 :: \ell_2 :: (p_1 \times p_2) \times p_3)) \prec m(\ell_1 :: p_1 \times (\ell_2 :: p_2 \times p_3))$.

Theorem 4.4.5 *(Completeness) Assume* $p \in \mathcal{CSP}$ *be a regular process over Act and Loc. For a closed* L_μ *formula* A *over Act and Loc and with tags in* $\mathcal{S}_p$*,* $p \models A$ *implies* $p \vdash A$.

The proof of this theorem is shown in appendix B.2 and is by induction on $\prec$. Actually, for regular processes $\mathcal{S}_p$ is finite, so that $\prec$ is well-founded. We show that if $p \models A$, then there exists a set $\{p_i \models A_i\}_{i\in\{1,n\}}$, such that $p_i \in \mathcal{S}_p$ and $A_i \prec A$, for each $i \in \{1, n\}$, and such that a proof of $p \vdash A$ can be obtained from $\{p_i \vdash A_i\}_{i\in\{1,n\}}$. By induction hypothesis, proofs $\{p_i \vdash A_i\}_{i\in\{1,n\}}$ indeed exist. In the case of formulas $< \alpha > A$ and $[\alpha]A$, we exploit lemma 4.4.4.

4.5 An example

In this section we show the application of the proof system to statecharts. Consider the statechart $\bigtriangledown(S, \{G, H\})$ of figure 4.1, where transitions are labelled as follows

$t_1 : \bar{d}, a/nil, nil/a, t_2 : \bar{d}, a/nil$
$t_3 : a, \bar{d}/nil, t_4 : a, \bar{d}/nil$
$t_5 : d/nil, t_6 : d/nil$

The process corresponding this statechart is $\alpha(\bigtriangledown(S, \{G, H\}))$:

$$S :: rec\ x.(G :: lab(t_5).rec\ y.(H :: lab(t_6).x \bigtriangledown \alpha(H)) \bigtriangledown \alpha(G))$$

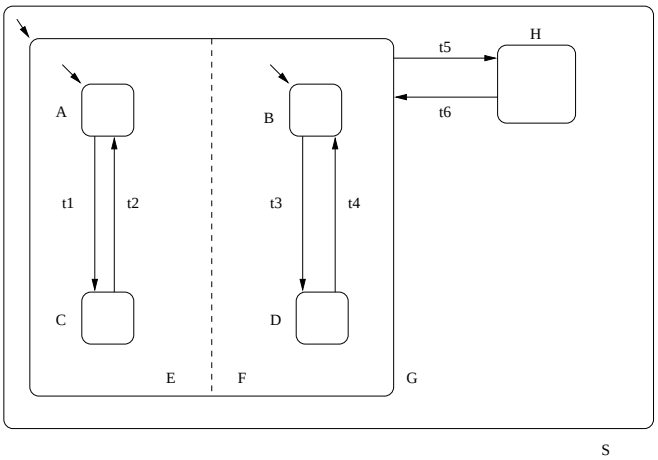

Figure 4.1: A statechart

where

$$\alpha(H) = H :: H :: nil$$
$$\alpha(G) = G :: \alpha(E) \times \alpha(F)$$
$$\alpha(E) = E :: rec\ x.A :: lab(t_1).rec\ y.C :: lab(t_2).x$$
$$\alpha(F) = F :: rec\ x.B :: lab(t_3).rec\ y.D :: lab(t_4).x.$$

In order to write more concise specifications we use the notation $[K]A$ and $< K > A$, where $K = \bigcup_{i\in\{1,n\}} \alpha_i \subseteq \mathcal{E}(Act)$. It is obvious that the proof system can easily be adapted to deal with formulas $[K]A$ and $< K > A$, where for each $\alpha_1, \alpha_2 \in K$, $type^*(\alpha_1) = type^*(\alpha_2)$.

Consider the L_μ formula $A \equiv \nu X.((\neg G \vee (G \wedge B)) \wedge [-]X)$, where $B \equiv [K_1]H \wedge [K_2]H$, for $K_1 = \bigcup_{i\in\{1,3\}} a_i : \epsilon$ and $K_2 = \bigcup_{i\in\{1,3\}} a_i : \bar{\epsilon}$ and $a_1 = (\{d, nil\}, \emptyset) \in \Sigma(\Pi)$, $a_2 = (\{d, a, nil\}, \emptyset) \in \Sigma(\Pi)$ and $a_3 = (\{d, \bar{a}, nil\}, \emptyset) \in \Sigma(\Pi)$.

This property asserts that always, if the system is in state G and if the environment produces d, then state H is entered. Note that it may be the case that the environment produces also a. In other words, transition t_5 behaves as an interrupt for the internal activity of state G.

We show the proof for this simple property. Let $p = \alpha(\bigtriangledown(S, \{G, H\}))$. Proof π_0 is as follows.

$$\dfrac{\dfrac{\dfrac{}{p \vdash A}}{p \vdash (\neg G \vee (G \wedge B)) \wedge [-]A_1}\,\nu_1}{\dfrac{\dfrac{}{p \vdash \neg G \vee (G \wedge B)}\,\pi_1 \qquad \dfrac{}{p \vdash [-]A_1}\,\pi_2}{}\,\wedge}$$

where $A_1 \equiv \nu X\{p\}.((\neg G \vee (G \wedge B)) \wedge [-]X)$.

1. Proof π_1 is as follows.

$$\dfrac{\dfrac{\dfrac{p \vdash \neg G \wedge (G \wedge B)}{p \vdash G \wedge B}\vee}{\dfrac{p \vdash [K_1]H \wedge [K_2]H}{\underline{p \vdash [K_1]H}_{\pi_{1,1}} \quad \underline{p \vdash [K_2]H}_{\pi_{1,2}}}\wedge \quad \dfrac{p \vdash G}{}\,prop_1}\wedge$$

1.1. The proof $\pi_{1,1}$ is as follows.

$$\dfrac{\dfrac{p \vdash [K_1]H}{S :: t_4 \bigtriangledown \alpha(G) \vdash [K_1]H}\,rec[]}{\underline{S :: t_4 \vdash [K_1]H}_{\pi_{1,1,1}} \quad \underline{\alpha(G) \vdash [K_1](H)_{S::t_4\bigtriangledown}}_{\pi_{1,1,2}}}\bigtriangledown[]\epsilon$$

where $t_4 = G :: lab(t_5).rec\ y.(H :: lab(t_6).t \bigtriangledown \alpha(H))$ for $t = rec\ x.(G :: lab(t_5).\ rec\ y.(H :: lab(t_6).\ x \bigtriangledown \alpha(H)) \bigtriangledown \alpha(G))$.

1.1.1. The proof $\pi_{1,1,1}$ is as follows.

$$\dfrac{\dfrac{S :: t_4 \vdash [\alpha_1]H \wedge [\alpha_2]H \wedge [\alpha_2]H}{\cdots}\wedge}{\dfrac{\dfrac{S :: t_4 \vdash [\alpha_1]H}{S :: t_5 \vdash H}\,prop_1}{}[]_1 \quad \dfrac{S :: t_4 \vdash [\alpha_2]H}{}[]_3 \quad \dfrac{S :: t_4 \vdash [\alpha_3]H}{}[]_3}$$

where $\alpha_1 = a_1 : \epsilon$, $\alpha_2 = a_2 : \epsilon$ and $\alpha_3 = a_3 : \epsilon$ and $t_5 = rec\ y.(H :: lab(t_6).p \bigtriangledown \alpha(H))$. Note that $lab(t_5) : \epsilon = \alpha_1$.

1.1.2. Proof $\pi_{1,1,2}$ is as follows.

$$\dfrac{\alpha(G) \vdash [K_1](H)_{S::t_4\bigtriangledown}}{\left\{\begin{array}{c}\underline{\alpha(G) \vdash [K_{1,i}](H)_{S::t_4\bigtriangledown}}_{\pi_{1,1,2,i}}\end{array}\right\}_{i\in\{1,3\}}}\,\mathcal{E}[]$$

where $K_{1,1} = \{a_i : \epsilon \times \epsilon\}_{i\in\{1,3\}}$, $K_{1,2} = \{a_i : \epsilon \times \bar{\epsilon}\}_{i\in\{1,3\}}$, $K_{1,3} = \{a_i : \bar{\epsilon} \times \epsilon\}_{i\in\{1,3\}}$.

1.1.2.1. Proof $\pi_{1,1,2,1}$ is as follows.

$$\dfrac{\dfrac{G :: \alpha(E) \times \alpha(F) \vdash [K_{1,1}](H)_{S::t_4\bigtriangledown}}{G :: \alpha(E) \times p_1 \vdash [K_{1,1}](H)_{S::t_4\bigtriangledown}} \times[]rec}{} \times[]$$

where $p_1 = F :: B :: lab(t_3).rec\, y.D :: lab(t_4).t_1$ and $t_1 = rec\, x.B :: lab(t_3).rec\, y.D :: lab(t_4).x$.

Actually, $a_i : \epsilon \times \epsilon \notin \alpha \otimes (lab(t_3) : \epsilon)$ for each α and for each $i \in \{1, 3\}$, since $\bar{d} \in trigger(lab(t_3))$ and $d \in trigger(a_i)$.

1.1.2.2 The proof $\pi_{1,1,2,2}$ is as follows.

$$\dfrac{\dfrac{\dfrac{G :: \alpha(E) \times \alpha(F) \vdash [K_{1,2}](H)_{S::t_4\bigtriangledown}}{G :: \alpha(E) \times p_1 \vdash [K_{1,2}](H)_{S::t_4\bigtriangledown}} \times[]rec}{\alpha(E) \vdash [K'_{1,2}]((H)_{S::t_4\bigtriangledown})_{G\times p_1}} \times[]}{} []_3$$

where $K'_{1,2} = \{\alpha \mid (a_i : \epsilon \times \bar{\epsilon}) \in \alpha \otimes (a : \bar{\epsilon}), a \in \#(lab(t_3))\}$. We have $a \in \#(lab(t_3))$ iff either $trigger(a) = \{d, a\}$ or $trigger(a) = \{\bar{a}, \bar{d}\}$ or $trigger(a) = \{\bar{a}, d\}$ and $action(a) = \emptyset$. We can apply rule $[]_3$, since $\bar{d} \in trigger(lab(t_1) : \epsilon)$, $d \in trigger(a_i)$ so that $(lab(t_1) : \epsilon) \notin K_3$.

1.1.2.3. The proof $\pi_{1,1,2,3}$ is as follows.

$$\dfrac{\dfrac{G :: \alpha(E) \times \alpha(F) \vdash [K_{1,3}](H)_{S::t_4\bigtriangledown}}{G :: \alpha(E) \times p_1 \vdash [K_{1,3}](H)_{S::t_4\bigtriangledown}} \times[]rec}{} \times[]$$

Actually, for each α, it is not the case that $(a_i : \bar{\epsilon} \times \epsilon) \in \alpha \otimes (lab(t_3) : \epsilon)$, because $\bar{d} \in trigger(lab(t_3))$, while $d \in trigger(a_i)$.

1.2. The proof $\pi_{1,2}$ is as follows.

$$\dfrac{\dfrac{\dfrac{p \vdash [K_2]H}{S :: t_4 \bigtriangledown \alpha(G) \vdash [K_2]H} rec[]}{S :: t_4 \vdash [K_2]false} \bigtriangledown[]\bar{\epsilon}_3}{} []_4$$

Actually, for each $a \in \#(lab(t_5))$, $\bar{d} \in trigger(a)$, while $d \in trigger(a_i)$, so that $a : \bar{\epsilon} \notin K_2$.

2. Proof π_2 is as follows.

$$\dfrac{\dfrac{\dfrac{p \vdash [-]A_1}{S :: t_4 \bigtriangledown \alpha(G) \vdash [-]A_1}\, rec[]}{\dfrac{S :: t_4 \bigtriangledown \alpha(G) \vdash [K_3]A_1}{\pi_{2,1}} \qquad \dfrac{S :: t_4 \bigtriangledown \alpha(G) \vdash [K_4]A_1}{\pi_{2,2}}}}$$

where $K_3 = \{\alpha \in \mathcal{S}(\Sigma(\pi)) \mid type(\alpha) = \epsilon\}$ and $K_4 = \{\alpha \in \mathcal{S}(\Sigma(\pi)) \mid type(\alpha) = \bar{\epsilon}\}$.

2.1. Proof $\pi_{2,1}$ is as follows.

$$\dfrac{\dfrac{S :: t_4 \bigtriangledown \alpha(G) \vdash [K_3]A_1}{\dfrac{S :: t_4 \vdash [K_3]A_1}{\pi_{2,1,1}} \qquad \dfrac{\alpha(G) \vdash [K_3](A_1)_{S::t_4\bigtriangledown}}{\pi_{2,1,2}}}\, \bigtriangledown[]\epsilon}$$

2.1.1 Proof $\pi_{2,1,1}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{S :: t_4 \vdash [K_3]A_1}{S :: t_5 \vdash A_1}\, []_1}{S :: t_5 \vdash (\neg G \vee (G \wedge B)) \wedge [-]A_2}\, \nu_1}{S :: t_5 \vdash \neg G \vee (G \wedge B)}\, \wedge}{S :: t_5 \vdash \neg G}\, prop_2 \qquad \dfrac{S :: t_5 \vdash [-]A_2}{\pi_{2,1,1,1}}}$$

where $A_2 \equiv \nu X\{p, S :: t_5\}.((\neg G \vee (G \wedge B)) \wedge [-]X)$. Rule $[]_1$ has been applied, since $lab(t_5) : \epsilon \in K_3$.

2.1.1.1 Proof $\pi_{2,1,1,1}$ is as follows.

$$\dfrac{S :: t_5 \vdash [-]A_2}{\dfrac{\dfrac{S :: t_5 \vdash [K_3]A_2}{p \vdash A_2}\, []_1}{\nu_2} \qquad \dfrac{\dfrac{S :: t_5 \vdash [K_4]A_2}{S :: t_5 \vdash A_2}\, []_1}{\nu_2}}$$

2.1.2. Proof $\pi_{2,1,2}$ is as follows.

$$\dfrac{\alpha(G) \vdash [K_3](A_1)_{S::t_4\bigtriangledown}}{\left\{ \dfrac{\alpha(G) \vdash [K_{3,i}](A_1)_{S::t_4\bigtriangledown}}{\pi_{2,1,2,i}} \right\}_{i\in\{1,3\}}}\, \mathcal{E}[]$$

where $K_{3,1} = \{\alpha \in \mathcal{E}(\Sigma(\Pi)) \mid \alpha \in \mathcal{E}(\alpha'), \alpha' \in K_3, \alpha = a : \epsilon \times \epsilon\}$, $K_{3,2} = \{\alpha \in \mathcal{E}(\Sigma(\Pi)) \mid \alpha \in \mathcal{E}(\alpha'), \alpha' \in K_3, \alpha = a : \epsilon \times \bar{\epsilon}\}$, $K_{3,3} = \{\alpha \in \mathcal{E}(\Sigma(\Pi)) \mid \alpha \in \mathcal{E}(\alpha'), \alpha' \in K_3, \alpha = a : \bar{\epsilon} \times \epsilon\}$.

2.1.2.1 Proof $\pi_{2,1,2,1}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\alpha(G) \vdash [K_{3,1}](A_1)_{S::t_4 \bigtriangledown}}{G :: \alpha(E) \times p_1 \vdash [K_{3,1}](A_1)_{S::t_4 \bigtriangledown}} \times[]rec}{\alpha(E) \vdash [K'_{3,1}]((A_1)_{S::t_4 \bigtriangledown})_{G \times p'_1}} \times[]}{p_2 \vdash [K'_{3,1}]((A_1)_{S::t_4 \bigtriangledown})_{G \times p'_1}} rec[]}{p'_2 \vdash ((A_1)_{S::t_4 \bigtriangledown})_{G \times p'_1}} []_1}{G :: p'_2 \times p'_1 \vdash (A_1)_{S::t_4 \bigtriangledown}} shift}{S :: t_4 \bigtriangledown G :: p'_2 \times p'_1 \vdash A_1}}{\pi_3}$$

where $p'_1 = F :: rec\, y.D :: lab(t_4).t_1$, $p_2 = E :: A :: lab(t_1).rec\, y.C :: lab(t_2).t_2$ $p'_2 = E :: rec\, y.C :: lab(t_2).t_2$, $t_2 = rec\, x.A :: lab(t_1).rec\, y.C :: lab(t_2).x$ and $K'_{3,1} = \{\alpha \mid \alpha' \in \alpha \otimes (lab(t_3) : \epsilon), \alpha' \in K_{3,1}\}$.

We omit proof π_3 for sake of brevity. It is obvious that it is analogous to proof π_0 since $\alpha(G)$ and $G :: p'_2 \times p'_1$ are symmetric. Therefore, by repeating a similar reasoning $S :: t_4 \bigtriangledown G :: p'_2 \times p'_1 \vdash A_1$ can be derived from $p \vdash A_3$, where $A_3 \equiv \nu X.\{p, S :: t_4 \bigtriangledown p'_2 \times p'_1\}.((\neg G \vee (G \wedge B)) \wedge [-]X)$. It is obvious that $p \vdash A_3$ by ν_2.

2.1.2.2 Proof $\pi_{2,1,2,2}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\alpha(G) \vdash [K_{3,2}](A_1)_{S::t_4 \bigtriangledown}}{G :: \alpha(E) \times p_1 \vdash [K_{3,2}](A_1)_{S::t_4 \bigtriangledown}} \times[]rec}{\alpha(E) \vdash [K'_{3,2}]((A_1)_{S::t_4 \bigtriangledown})_{G \times p'_1}} \times[]}{p_2 \vdash [K'_{3,2}]((A_1)_{S::t_4 \bigtriangledown})_{G \times p'_1}} rec[]}{[]_1}$$

where $K'_{3,2} = \{\alpha \mid \alpha' \in \alpha \otimes (a : \bar{\epsilon}), \alpha' \in K_{3,2}, a \in \#(lab(t_3))\}$. Actually, $lab(t_1) : \epsilon \notin K'_{3,2}$ since $lab(t_1) = lab(t_3)$.

2.1.2.3 Proof $\pi_{2,1,2,3}$ is similar to $\pi_{2,1,2,2}$.

2.2 Proof $\pi_{2,2}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{S :: t_4 \bigtriangledown \alpha(G) \vdash [K_4]A_1}{S :: t_4 \bigtriangledown \alpha(G) \vdash A_1} \bigtriangledown[]\bar{\epsilon}_1}{S :: t_4 \bigtriangledown \alpha(G) \vdash (\neg G \vee (G \wedge B)) \wedge [-]A_4} \nu_1}{S :: t_4 \bigtriangledown \alpha(G) \vdash \neg G \vee (G \wedge B)} \wedge \quad \dfrac{}{S :: t_4 \bigtriangledown \alpha(G) \vdash [-]A_4}$$

π_5 $\qquad$ π_4

where $A_4 \equiv \nu X\{p, S :: t_4 \bigtriangledown \alpha(G)\}.((\neg G \vee (G \wedge B)) \wedge [-]X)$.

Proof π_5 is obviously analogous to π_1, while proof π_4 is analogous to proof π_2. Therefore, we obtain $S :: t_4 \bigtriangledown \alpha(G) \vdash [-]A_4$ from $S :: t_4 \bigtriangledown \alpha(G) \vdash A_4$. It is obvious that $S :: t_4 \bigtriangledown \alpha(G) \vdash A_4$ by ν_2.

The proof is quite complex, since it requires to take into account the causal ordering between events and the type of actions. However, these drawbacks are intrinsic to the semantics of statecharts. However, the proof system has the advantage that subproofs can be reused and that, if we replace a component, for instance G, we must redone only the part of the proof concerning $\alpha(G)$. Since only very simple proofs can be done by hand, the application of the proof system would benefit of the use of an interactive proof checker for providing the user with a mechanized assistance.

4.6 Related works

We have discussed the relation of our proof system with previous works for simpler languages [13, 61, 90, 91, 10, 12]. An alternative approach to compositional reasoning for μ-calculus and CCS is the one proposed by [25]. Compositionality is adressed by considering judgements of the proof system with assertions on process variables. More in details judgements are of the form $x_1 : \phi_1, \ldots, x_n : \phi_n; p(x_1, \ldots, x_n) \vdash \phi$, where ϕ_i, ϕ are μ-calculus assertions and $p(x_1, \ldots, x_n)$ is an open process over variables $x_1, \ldots, x_n$. The idea is that properties of the open process $p(x_1, \ldots, x_n)$ are relativised to properties of its free variables. Thus, the previous statement means that $p(p_1, \ldots, p_n) \models \phi$, for each set of processes $p_1, \ldots, p_n$, such that $p_i \models \phi_i$. A *cut* rule is exploited for introducing variables and for checking the assumed properties of subprocesses. This approach permits a very elegant treatment of parallel composition.

With regard to statecharts, it is important to note that the non modularity of the semantics is a limitation for the application of many methods of compositional verification to statecharts. For instance the method of modular model checking suggested by [21] for CTL. The main idea is to find the conditions relating formulas ψ, ψ_1 and ψ_2 such that, if model M_1 satisfies ψ_1 and model M_2 satisfies ψ_2, then the parallel composition $M_1 \times M_2$ satisfies ψ. This kind of reasoning can be naturally formalized in the classical assumption-commitment paradigm [51], where a specification includes two parts. One describes the behaviour of the module, while the other describes the assumed behaviour of the environment. This approach has been extensively studied for branching temporal logic [34, 52, 53, 54, 1]. Specifications are formalized as $\langle \psi_1, \psi_2 \rangle$, where ψ_1 represents the assumption on the environment and ψ_2 the behaviour of the system. Hence, $M \models \langle \psi_1, \psi_2 \rangle$ is interpreted as M satisfies ψ_2 under the assumption that the environment satisfies ψ_1. These specifications are

applied for modular verification by defining proof rules such as the following.

$$\frac{\begin{array}{ll} M_1 \models \langle \psi_1, \phi_1 \rangle & M_2 \models \langle true, \psi_1 \rangle \\ M_2 \models \langle \psi_2, \phi_2 \rangle & M_1 \models \langle true, \psi_2 \rangle \end{array}}{M_1 \times M_2 \models \langle true, \phi_1 \wedge \phi_2 \rangle}$$

The key step is the verification of an assumption-commitment assertion, i.e. $M \models \langle \psi_1, \phi_1 \rangle$. It usually requires the use of tableau constructions for obtaining a general model representing the environments behaving as ψ_1.

This method of compositional verification seems to be not adequate for statecharts, since it would be necessary to express in classical temporal logics formulas describing assumptions on the environment that represent both the observable behaviour and the causality relation between input and output.

These difficulties are pointed out also in [48], where a compositional axiomatization of statecharts in a first order logic is proposed. Due to the well known problems of the Pnueli and Shalev semantics, the proof system is quite complex and the causal ordering between inputs and outputs is explicitely represented.

Chapter 5

Timed Statecharts and the language $\mathcal{TSP}$

Real-time systems are reactive systems that are required to meet hard timing constraints and whose correctness depends on the actual values of delays. Synchronous languages can be used for specifying real-time systems by considering time as an ordinary input event. For instance, a timeout can be realized by counting the occurrences of a particular input event representing clock ticks. This is the so called *multiform time* point of view. It has the advantage that the semantics is abstract, since does not refer to physical time and can be instantiated with different time scales. However, neither classical modal and temporal logics nor real-time logics are adequate for expressing and verifying quantitative properties about time over this abstract model. A possible solution is to adapt the logic to deal with multiform time. For instance [74] presents a version of linear temporal logic with multiform time for verifying Lustre programs. Another approach consists of introducing in the language explicit real-time features and of giving a real-time semantics with respect to a concrete domain of time. Quantitative properties can be expressed by a real-time logic and the verification can be performed via model checking over this real-time model. In the last years real-time extensions have been proposed both for Statecharts [56, 73, 72] and for the similar language Argos [55].

In this chapter we propose a real-time version of statecharts, where the syntax is extended by attaching a minimal and a maximal delay to transitions. The idea is that the transitions can be performed nondeterministically within the minimal and the maximal delay and cannot be delayed, when the maximal delay is reached. We give a semantics to timed statecharts with respect to the discrete time domain. This choice relies on the following assumptions:

- The environment can be described as a discrete process, namely as a sequence of inputs $I_1, I_2, \ldots, I_i \ldots$ occurring at time $i \in N$.

- The system is infinitely faster than the environment.

These assumptions allow us to retain the synchrony hypothesis, namely that the reaction is instantaneous, and to reason in *abstract time*. It is obvious that the adequacy of such a model depends on the possibility to determine a clock period that is an upperbound for the reaction time of the system and for the distance between two successive inputs of the environment.

We extend the Pnueli and Shalev semantics to timed statecharts. In contrast to the classical semantics, a step is caused either by the inputs of the environment or by the flow of time, which enables transitions depending on time. The definition of step satisfies the synchrony hypothesis and the principles of causality and of global consistency.

We extend also the process language $\mathcal{SP}$ by considering timed actions whose performance is constrained by a minimal and a maximal delay. We define a semantics for closed $\mathcal{TSP}$ (Timed $\mathcal{SP}$) processes by using a discrete version of timed graphs [5]. Timed graphs are a widely used model of real-time systems, that is obtained by extending the finite automaton model with a finite set of clocks, which proceed at a uniform rate on the time domain (either discrete or dense) and measure the amount of time since they were started. The performance of transitions may depend on the values of clocks and may have the effect of resetting some clocks. The behaviour of a timed graph is described by a generally infinite labelled transition system, where states correspond to nodes of the graph together with the values of clocks (timed states), and transitions are of two types: instantaneous transitions corresponding to an action of the system and transitions representing the progress of time of some amount. In our framework, since the time domain is discrete and the language is synchronous, we consider a discrete version of timed graphs where one action of the system is executed at each instant and where the amount of time between two successive transitions is exactly one unit. This choice allows us to consider transitions representing an action of the system together with the increment of time. Note that an action may correspond to the parallel composition of basic actions. The semantics is compositional and is obtained by considering semantics actions, idle and non-idle. Therefore, the proposed rules are very similar to the ones of the semantics of chapter 3.

Finally, we show the generalization of the translation α of statecharts into closed $\mathcal{SP}$ processes of chapter 3 to the real-time case. Only the translation for basic components must be modified by taking into account timed transitions. As in the untimed case we adopt actions corresponding to transitions representing both the input-output and the causal relation between inputs and outputs. The obtained semantics is equivalent to that of timed statecharts.

Such a timed graphs semantics is mainly motivated by our interest for developing methods of verification. Actually, time leads to an infinite model also for finite-state systems. On the other hand, for finite-state systems the timed graph is finite and has been successfully used as the basis for model checking methods also with respect to a dense time domain [3, 45, 82].

5.1 Timed statecharts

We extend the syntax of statecharts by adding transitions depending on time. A *timed statechart* is formally defined as in definition 1.1.1, where for basic statecharts we adopt the following syntax.

Definition 5.1.1 *Let* Π *be a set of primitive events. A* basic timed statechart *is defined as* $S : (\{S_1, \ldots, S_k), T_1, T_2, in, out, \chi, \tau, \delta)$ *where*

- $\{S_1, \ldots, S_k)\}$ *and* δ *as in definition 1.1.1;*
- T_1 *and* T_2 *are sets of transitions and* in, $out : T_1 \cup T_2 \to boxes(S)$ *give the target and source states of transitions;*
- $\chi : T_1 \cup T_2 \to 2^{\Pi \cup \bar{\Pi}} \times 2^{\Pi}$ *is the labelling function, such that* $trigger(t) = nil$ *for* $t \in T_2$*;*
- $\tau : T_2 \to (N \times (N^+ \cup \{\infty\}))$ *is the timing labelling function of transitions, such that for each transition* $t \in T_2$*, if* $min(t)$ *and* $max(t)$ *denote the first and the second component of* $\tau(t)$*, respectively, then* $min(t) \leq max(t)$*.*

We call *immediate* the transitions in T_1 and *timed* the ones in T_2. Timed transitions do not require inputs and can be performed nondeterministically in the interval between $min(t)$ and $max(t)$. These transitions may produce outputs. On the other hand, immediate transitions depend on the environment and are performed as soon as enabled by the environment. It is important to stress that we do not allow the maximal delay to be equal to zero. This condition is needed for maintain the principle of the untimed step semantics, where two consecutive transitions cannot be taken in the same step.

This formalism permits to specify classical constructs of real-time systems such as timeouts. Consider the statechart of figure 5.1 and the configuration $\{A, B\}$. Transition t_1 is immediate and is performed as soon as the environment produces a. On the other hand, transition t_2 is a timed transition and behaves as a timeout on the waiting for the signal a. Intuitively, if the environment does not produce signal a before two units of time, since the system was in configuration $\{A, B\}$, transition t_2 is performed.

5.1.1 The semantics of Timed Statecharts

We give the semantics of timed statecharts in the style of the one of Pnueli and Shalev. First of all, we extend the definition of configuration.

Definition 5.1.2 *Let* S *be a timed statechart. A* timed configuration *is a pair* (c, age)*, where* $c \in C(S)$ *and* $age : c \cap boxes(S) \to N$*.*

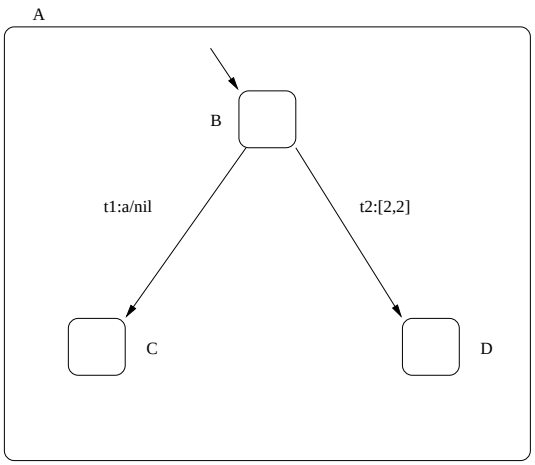

Figure 5.1: A timeout

The idea is that the function *age* gives for each box in the configuration its duration, that is the time since it was entered.

We denote by $TC(S)$ the set of timed configurations of S. The *initial timed configuration* of S is $(\Delta(S), age_0)$, where $age_0(S') = 0$, for $S' \in \Delta(S)$.

For a statechart S, we denote by $T_1(S)$ and $T_2(S)$ the set of immediate and timed transitions of S, namely the union of sets T_i such that $S' : (\{S_1, \ldots, S_k\}, T_1, T_2, in, out, \chi, \tau, \delta) \in boxes(S)$ and by $T(S) = T_1(S) \cup T_2(S)$.

Definition 5.1.3 *Let S be a timed statechart, $I \subseteq \Pi$ and $(c, age) \in TC(S)$.*

- *We define $enabled(age) = \{t \in T_2(S) \mid min(t) \leq age(out(t)) \leq max(t)\}$.*
- *We define $En_{(c,age),I}(T) = rel(c) \cap con(T) \cap (trig(I \cup gen(T)) \cup enabled(age))$, where $gen(T) = \bigcup_{t \in T} action(t)$.*

The function En is adapted to timed statecharts by taking into account both the immediate transitions that are enabled with respect to the generated signals ($trig(I \cup gen(T))$) and the timed transitions that are enabled by the flow of time ($enabled(age)$). Note that a timed transition is enabled iff its source state has been continously entered for a time between its minimal and maximal delay.

The definition of step is given as follows.

Definition 5.1.4 *Let S be a timed statechart over Π. Given a timed configuration $(c, age) \in TC(S)$, a set of input signals $I \subseteq \Pi$, the set of transitions $T \subseteq T(S)$ is an* admissible step *from (c, age) with respect to I iff*

- $T \subseteq En_{(c,age),I}(T)$;
- $En_{(c,age),I}(T) \cap T_1(S) \subseteq T$;

- *for each* $t \in T_2(S)$, *if* $t \notin T$ *and* $t \in rel(c)$ *and* $max(t) = age(out(t))$, *then* $t \notin rel(c')$;
- T *is* inseparable;

where $c' \in C(S)$ *is the configuration,*

$$c' = c - (\bigcup_{t \in T} boxes(out(t))) \cup (\bigcup_{t \in T} \Delta(in(t))).$$

The reached timed configuration is (c', age'), *where* $age'(S') = 0$, *for* $S' \in c' \setminus c$, *and* $age'(S') = age(S')$, *for* $S' \in c \cap c'$.

The definition of admissible step is quite different from the one for the untimed case. Condition $T \subseteq En_{(c,age),I}(T)$ ensures that all the transitions are enabled. Condition $En_{(c,age),I}(T) \cap T_1(S) \subseteq T$ ensures that all the immediate transitions, that are enabled and are not in conflict, are performed. On the other hand, a timed transition may be performed nondeterministically within the minimal and the maximal delay. Hence, we do not require maximality for this class of transitions. However, a timed transition must be performed when the maximal delay is reached, unless it is disabled. This is guaranteed by the condition that, for each $t \in T_2(S)$, if $t \notin T$ and $t \in rel(c)$ and $max(t) = age(out(t))$, then $t \notin rel(c')$. The condition of inseparability ensures the causality relation between transitions. Note that a step is assumed to be instantaneous. Actually, the function age is not incremented.

Consider the statechart of figure 5.1 and the timed configuration $(\{B, A\}, age)$, where $age(B) = 2$. Both $\{t_1\}$ and $\{t_2\}$ are admissible steps with respect to the input $\{a\}$. Actually, transition t_1 is enabled and its performance disables t_2. The reached timed configuration is $(\{A, C\}, age')$, where $age'(C) = 0$. On the other hand, the timed configuration reached via $\{t_2\}$ is $(\{A, D\}, age')$, where $age'(D) = 0$. Consider the timed configuration $(\{B, A\}, age)$, where $age(B) = 1$ and suppose that the input does not contain a. Both $\{t_2\}$ and the empty step are admissible, since the maximal delay of transition $\{t_2\}$ has not been reached and it can be delayed.

A computation of a statechart is an infinite sequence of steps occurring at times i, for each $i \in N$. We use the notation $age + \delta$, for $\delta \in N$, to denote age', such that $age'(S) = age(S) + \delta$, for each S for which age is defined.

Definition 5.1.5 *A* computation *of a statechart* S *is an infinite sequence of steps* $T_1, T_2 \ldots$ *such that for each* $i \in N$,

1. (c_0, age_0) *is the initial timed configuration of* S;
2. T_i *is an admissible step from* (c_i, age_i) *to* (c_{i+1}, age'_{i+1}) *with respect to* I_i;
3. $age_i = age'_i + 1$.

We use the notation $(c_i, age_i) \stackrel{I_i,O_i}{\Longrightarrow} (c_{i+1}, age_{i+1})$ to denote the step T_i with respect to input I_i from timed configuration (c_i, age_i) to timed configuration (c_{i+1}, age_{i+1}), that produces $O_i = gen(T_i)$. In other words, each arrow $\Longrightarrow$ represents a step with respect to input I_i followed by an increment of one unit of time.

Note that the increment of unit of time does not violate the maximal delay for each transition. Actually, the condition $max(t) \neq 0$ ensures that, for each $t \in T_1(S) \cap rel(c_{i+1})$, $max(t) \neq age_{i+1}(out(t))$. Actually, if $t \notin T_i$ and $t \in rel(c_i)$, then $age_{i+1}(out(t)) = age'_i(out(t))$, so that $max(t) = age'_i(out(t))$. Hence, by the definition of step either $t \in T_i$ or $t \notin rel(c_{i+1})$. On the other hand, if $t \in rel(c_{i+1})$ and $t \notin T_i$, then $age_{i+1}(out(t)) = 0$.

Statechart of figure 5.2 specifies a teller-machine. When the statechart is in state *ready* it waits for a client to insert the card and moves to state *control*. In this state the check for the code is performed. If the control of code has not success before three units of time, the state *control* is left and the statechart reaches state *ready* by emitting the signal $suspend - card$. On the other hand, if the control of code has success before three units of time, the state *operation* is entered.

Consider the initial timed configuration $(\{ready\}, age_0)$, where $age_0(ready) = 0$. We omit for sake of brevity the state $teller - machine$ from configurations. Suppose that the environment produces signal *card* after one unit of time, signal *code* after two units of time, signal $not - cor$ after three units of time and signal *code* after four units of time. We have the following sequence of steps:

$$(\{ready\}, age_0) \stackrel{\emptyset,\emptyset}{\Longrightarrow} (\{ready\}, age_1)$$

with $age_1(ready) = 1$.

$$(\{ready\}, age_1) \stackrel{\{card\},\emptyset}{\Longrightarrow} (\{control, code\}, age_2)$$

with $age_2(code) = age_2(control) = 1$.

$$(\{control, code\}, age_2) \stackrel{\{code\},\emptyset}{\Longrightarrow} (\{control, check - code\}, age_3)$$

with $age_3(check - code) = 1$ and $age_3(control) = 2$.

$$(\{control, check - code\}, age_3) \stackrel{\{not-cor\},\emptyset}{\Longrightarrow} (\{control, code\}, age_4)$$

with $age_4(code) = 1$ and $age_4(control) = 3$.

$$(\{control, code\}, age_4) \stackrel{\{code\},\{suspend-card\}}{\Longrightarrow} (\{ready\}, age_5)$$

with $age_5(ready) = 0$.

Transition t_2 is enabled in the timed configuration ($\{control, code\}$, age_4), since the environment produces *code*. However, it cannot be performed, since the maximal delay of t_4 is reached and the performance of t_2 does not disable t_4. Note that, when $age(control) = 3$, the internal activity of state *control* is suspended. This is well known as a watchdog construct.

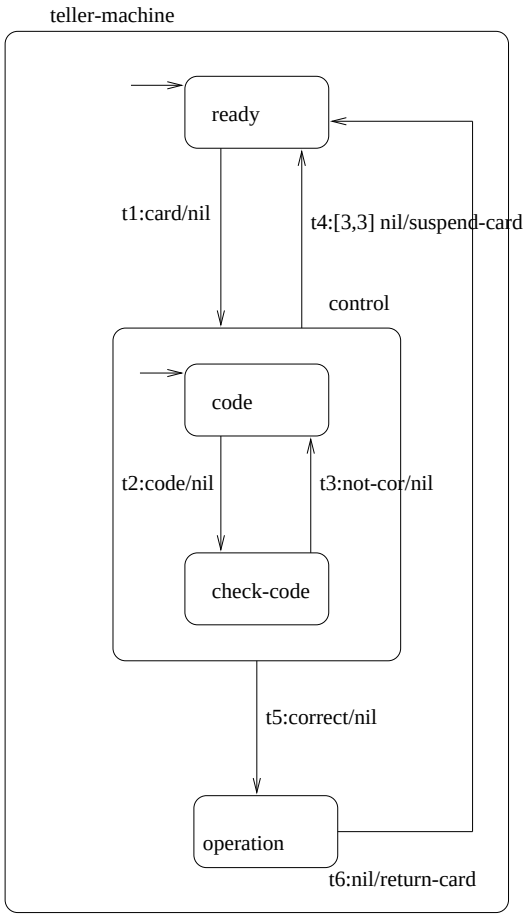

Figure 5.2: The teller-machine

5.2 The language $\mathcal{TSP}$

In this section we introduce the timed process language $\mathcal{TSP}$ and we define its timed graphs semantics.

We assume a set Act of basic actions, a set Loc of locations and a set Var of variables. On Act we assume an operation $\times : Act \times Act \to 2^{Act}$ of parallel composition and an operation $\# : Act \cup \{*\} \to 2^{Act}$ as in chapter 3.

Definition 5.2.1 *We define processes $\mathcal{TSP}$ as follows:*

$$b ::= nil \mid a.t \mid b_1 + b_2 \mid [m, M]a.t$$
$$t ::= \ell :: b \mid rec\ x.t \mid t \bigtriangledown p \mid x$$
$$p ::= \ell :: t \mid \ell :: p_1 \times p_2$$

where $a \in Act$, $x \in Var$, $\ell \in Loc$, $m, M \in N$ and $m \leq M$, $M \neq 0$. Moreover, we require $\#(a) = \emptyset$ for $[m, M]a.t$.

The syntax of $\mathcal{TSP}$ processes is the obvious counterpart of the one for timed statecharts. A timed action $[m, M]a$ can be performed within time m and time M and cannot be delayed after M.

Definitions 3.1.2 and 3.2.1 of $loc(p)$, $Loc(p)$ and of well-formed process can be trivially extended to timed processes. As usual we consider well-formed $\mathcal{TSP}$ processes.

5.2.1 The semantics of $\mathcal{TSP}$ processes

We define the semantics of closed $\mathcal{TSP}$ processes by adopting a discrete version of timed graphs. A timed graph is defined with respect to a set of clocks and has edges that are labelled by an action and by a constraint over clocks. The idea is that the corresponding action can be performed iff the valuation of the clock constraint in the timed state is true.

Definition 5.2.2 *Let C be a set of clocks. A* clock constraint *is inductively defined by the following grammar*

$$\omega ::= x \mathcal{R} n \mid x + n \mathcal{R} y + m \mid \omega_1 \wedge \omega_2 \mid \neg\omega$$

with $x, y \in C$ and $n, m \in N$, $\mathcal{R} \in \{\geq, \leq\}$.

We denote by $\Omega(C)$ the set of clock constraints over clocks C. Clock constraints are interpreted over *clock assignments*, namely $\pi : C \to N$. If π be a clock assignment over C and $\delta \in N$, we denote by $\pi + \delta$ the clock assignment, which assigns to each $x \in C$, $\pi(x) + \delta$ (and analogously for $\pi - \delta$). Moreover, for $C_1 \subseteq C$, $\pi[C_1 := 0]$ denotes the assignment such that, $\pi(y) = \pi[C_1 := 0](y)$, for $y \notin C_1$, and $\pi[C_1 := 0](x) = 0$ for $x \in C_1$.

Definition 5.2.3 *A* Discrete Timed Graph *is* $(S, s_0, Act, C, \longmapsto, delay, clock)$ *where*

- S *is a set of states, with* $s_0 \in S$ *the initial state;*
- Act *is a set of actions and* C *is a set of clocks;*
- $clock : S \to 2^C$ *is the function, which gives the subset of clocks that are defined in each state;*
- $delay : S \to \Omega(C)$ *is a function assigning to each state a past-closed clock constraint (such that if* $\pi \models \omega$ *then* $\pi - \delta \models \omega$ *for* $\delta \in N$*), such that for* $s \in S$, $delay(s) \in \Omega(clock(s))$*;*
- $\longmapsto \subseteq S \times (Act \times \Omega(C)) \times S$ *is the transition relation, such that for each* $s \in S$, $(s, (\alpha, \omega), s')$ *iff* $\omega \in \Omega(clock(s))$.

This definition is different from the classical one, where transitions are labelled also by a set of clocks that are reset when the transition is performed. We consider a different formulation where each state $s \in S$ has a subset of clocks $clock(s) \subseteq C$ which are defined in s. The idea is that, when a transition $s \stackrel{\alpha,\omega}{\longmapsto} s'$ is performed, the clocks in $clock(s') \setminus clock(s)$ are reset. This approach is more adequate for $\mathcal{TSP}$ processes.

The behaviour of a timed graph is described by a labelled transition system, where states represent a state of the timed graph together with an assignment for the corresponding clocks and transitions represent an action of the system together with a time progress of one unit.

We need some preliminary concepts. Let C be a set of clocks, $\pi : C \to N$ a clock assignment over C and $C_1 \subseteq C$. We define the clock assignment $pro_{C_1}(\pi) : C_1 \to N$, such that $\pi(x) = pro_{C_1}(\pi)(x)$, for each $x \in C_1$. For two sets of clocks C_1 and C_2 with $C_1 \cap C_2 = \emptyset$, and for $\pi_1 : C_1 \to N$ and $\pi_2 : C_2 \to N$, we define $\pi_1 \cup \pi_2 : C_1 \cup C_2 \to N$, such that $\pi_1 \cup \pi_2(x) = \pi_1(x)$, for $x \in C_1$, and $\pi_1 \cup \pi_2(x) = \pi_2(x)$, otherwise. It is obvious that $\pi = pro_{c_1}(\pi) \cup pro_{C_2}(\pi)$ for $C_1 \cup C_2 = C$.

Moreover, for $C_1 \subseteq C$ and C_2, such that $C_1 \cap C_2 = \emptyset$, and for $\pi : C \to N$, we denote by $reset(\pi, C_1, C_2) : C_1 \cup C_2 \to N$ the clock assignment, such that $reset(\pi, C_1, C_2) = pro_{C_1}(\pi) \cup \{C_2 := 0\}$. Intuitively, $reset(\pi, C_1, C_2)$ gives the clock assignment over $C_1 \cup C_2$, that agrees with π for C_1 and where clocks of C_2 have value zero.

Definition 5.2.4 *Let* $\mathcal{G} = (S, s_0, Act, C, \longmapsto, delay, clock)$ *be a discrete timed graph. We define* $LTS(\mathcal{G}) = (TS, ts_0, Act, \mapsto)$ *where*

- TS *is the set of* timed states*, which are pairs of the form* (s, π)*, where* $s \in S$ *and* π *is a clock assignment over* $clock(s)$*, such that* $\pi \models delay(s)$*;*
- $ts_0 = (s_0, \pi_0)$ *with* $\pi_0(x) = 0$*, for each* $x \in clock(s_0)$*;*

- *the transition relation* $\mapsto \subseteq S \times Act \times S$ *is defined as follows. There is a transition* $(s_1, \pi_1) \stackrel{\alpha}{\mapsto} (s_2, \pi_2)$ *iff*

  1. *there exists* $s_1 \stackrel{\alpha,\omega}{\longmapsto} s_2$, *such that* $\pi_1 \models \omega$;
  2. $\pi_2 = reset(\pi_1, C_1, C_2) + 1$, *where* $C_1 = clock(s_1) \cap clock(s_2)$, $C_2 = clock(s_2) \setminus clock(s_1)$, *and* $\pi_2 \models delay(s_2)$.

The transition relation of the labelled transition system represents two steps. In the first one, a transition is performed such that the corresponding clock constraint is satisfied. The resulting state is obtained by starting the new clocks of the entered state. In the second one, time increases by one unit iff the maximal delay of the target timed process is not violated.

Note that we consider a discrete strongly monotonic semantics, where two successive actions cannot be taken at the same time. Moreover, we assume the increment of time to be equal to one so that next-time corresponds to next-state. This choice is motivated by the synchrony of language. On the other hand, in an interleaving approach weakly monotonic time should be considered. In this case, the increment of time could be of any amount (also zero). It is important to stress that the labelled transition system is infinite.

The underlying idea for defining a timed graphs semantics of closed $\mathcal{TSP}$ processes is to consider a timed graph with states corresponding to closed processes, clocks corresponding to locations and transitions to composite actions. The clock constraints of the timed graph are used for representing the minimal and maximal delays of timed actions.

Since we want a compositional semantics, we consider semantic actions $\mathcal{S}(Act)$ as in the labelled transition system semantics of chapter 3. In this way, idle actions can be distinguished from non-idle ones and we can achieve compositionality as in the untimed case.

Definition 5.2.5 *Assume* $p \in \mathcal{TSP}$ *be a closed process over actions Act and locations Loc. We define the discrete timed graph of* p *as*

$$\mathcal{G}(p) = (\mathcal{CTSP}, p, \mathcal{S}(Act), Loc, \longmapsto, delay, clock)$$

where

- $\mathcal{CTSP}$ is the set of closed $\mathcal{TSP}$ processes;
- For $p \in \mathcal{CTSP}$,

$$\begin{array}{l} delay(\ell :: t) \equiv delay(t) \\ delay(\ell :: p_1 \times p_2) \equiv delay(p_1) \wedge delay(p_2) \\ delay(rec\ x.t) \equiv delay(t[rec\ x.t/x]) \\ delay(t \bigtriangledown p) \equiv delay(t) \wedge delay(p) \\ delay(\ell :: b) \equiv \left\{ \begin{array}{ll} \ell \leq delay(b) & delay(b) \neq \infty \\ true & \text{otherwise} \end{array} \right\} \end{array}$$

$$delay(nil) \equiv \infty$$
$$delay(a.t) \equiv \infty$$
$$delay([m, M]a.t) \equiv M$$
$$delay(b_1 + b_2) \equiv min(delay(b_1), delay(b_2));$$

- $clock(p) = loc(p)$;
- The transition relation $\stackrel{\alpha,\omega}{\longmapsto}$ is obtained by the following rules.

$$\frac{t \stackrel{\alpha,\omega}{\longmapsto} t'}{\ell :: t \stackrel{\alpha,\omega}{\longmapsto} \ell :: t'}$$

$$\frac{\{p_i \stackrel{\alpha_i,\omega_i}{\longmapsto} p_i'\}_{i\in\{1,2\}} \quad \alpha \in \alpha_1 \times \alpha_2}{\ell :: p_1 \times p_2 \stackrel{\alpha,\omega_1\wedge\omega_2}{\longmapsto} \ell :: p_1' \times p_2'}\ par$$

$$\ell :: a.t \stackrel{a:\epsilon,true}{\longmapsto} t$$

$$\ell :: a.t \stackrel{b:\bar{\epsilon},true}{\longmapsto} \ell :: a.t \quad b \in \#(a)$$

$$\ell :: nil \stackrel{b:\bar{\epsilon},true,\emptyset}{\longmapsto} \ell :: nil \quad b \in \#(*)$$

$$\ell :: [m, M]a.t \stackrel{a:\epsilon,m\le\ell\le M}{\longmapsto} t$$

$$\ell :: [m, M]a.t \stackrel{b:\bar{\epsilon},true}{\longmapsto} \ell :: [m, M]a.t \quad b \in \#(*)$$

$$\frac{\ell :: b_i \stackrel{\alpha,\omega}{\longmapsto} t \quad type(\alpha) = \epsilon}{\ell :: b_1 + b_2 \stackrel{\alpha,\omega}{\longmapsto} t}\ choice_1$$

$$\frac{\ell :: b_1 \stackrel{\alpha,\omega_1}{\longmapsto} \ell :: b_1 \quad \ell :: b_2 \stackrel{\alpha,\omega_2}{\longmapsto} \ell :: b_2 \quad type(\alpha) = \bar{\epsilon}}{\ell :: b_1 + b_2 \stackrel{\alpha,\omega_1\wedge\omega_2}{\longmapsto} \ell :: b_1 + b_2}\ choice_2$$

$$\frac{t \stackrel{\alpha,\omega}{\longmapsto} t' \quad type(\alpha) = \epsilon}{t \bigtriangledown p \stackrel{\alpha,\omega}{\longmapsto} t'}\ \bigtriangledown_1$$

$$\frac{p \stackrel{\alpha,\omega}{\longmapsto} p' \quad type(\alpha) = \epsilon}{t \bigtriangledown p \stackrel{\alpha,\omega}{\longmapsto} t \bigtriangledown p'}\ \bigtriangledown_2$$

$$\frac{\begin{array}{c} t \stackrel{\alpha,\omega_1}{\longmapsto} t \\ p \stackrel{\alpha,\omega_2}{\longmapsto} p \\ type(\alpha) = \bar{\epsilon} \end{array}}{t \bigtriangledown p \stackrel{\alpha,\omega_1 \wedge \omega_2}{\longmapsto} t \bigtriangledown p} \bigtriangledown_3$$

$$\frac{t[rec\ x.t/x] \stackrel{\alpha,\omega}{\longmapsto} t' \quad type(\alpha) = \epsilon}{rec\ x.t \stackrel{\alpha,\omega}{\longmapsto} t'} rec_1$$

$$\frac{t[rec\ x.t/x] \stackrel{\alpha,\omega}{\longmapsto} t[rec\ x.t/x] \quad type(\alpha) = \bar{\epsilon}}{rec\ x.t \stackrel{\alpha,\omega}{\longmapsto} rec\ x.t} rec_2$$

Let us explain the semantics. The rules are very similar to the ones of the labelled transition system semantics of $\mathcal{SP}$ processes proposed in chapter 3.

The case of *nil* and of *prefixing* are quite simple. Consider a timed process

$$p = \ell_1 :: \ell_2 :: a.t.$$

We have $p \stackrel{\alpha,\omega}{\longmapsto} p'$ iff one of the following cases holds:

1. $\alpha = a : \epsilon$ and $\omega \equiv true$, $p' = \ell_1 :: t$;
2. $\alpha = b : \bar{\epsilon}$, for $b \in \#(a)$, and $\omega \equiv true$, $p' = p$.

In the first case, since $clock(p) = loc(p)$ and the process is well-formed, $clock(p') \setminus clock(p) = loc(t)$ and $clock(p) \cap clock(p') = \{\ell_1\}$. Therefore, for each $\pi : loc(p) \to N$, we have

$$(p, \pi) \stackrel{\alpha}{\mapsto} (p', \pi')$$

where $\pi' : \{\ell_1, loc(t)\} \to N$ and $\pi'(\ell_1) = \pi(\ell_1) + 1$ and $\pi'(loc(t)) = 1$, iff $\pi' \models delay(\ell_1 :: t)$. In the second case, for each $\pi : loc(p) \to N$, we have

$$(p, \pi) \stackrel{\alpha}{\mapsto} (p, \pi + 1).$$

Actually, $delay(p) \equiv true$.

Consider a timed process $\ell_1 :: \ell :: [m, M]a.t$. The semantics is similar to the one of prefixing. However, the transition representing the non-idle action a is labelled by the clock constraint $m \leq \ell \leq M$ in order to ensure the action a to be performed within its maximal and minimal delay. Since the action a can be delayed until its maximal delay expires independently of the environment, we consider transitions $p \stackrel{b:\bar{\epsilon},true}{\longmapsto} p$ for each $b \in \#(*)$. It is worth noting that these idle actions cannot be performed when $\ell = M$, since at time M action a must be performed. Actually, by definition of the labelled transition system semantics,

$$(\ell_1 :: \ell :: [m, M]a.t, \pi_1) \stackrel{b:\bar{\epsilon}}{\mapsto} (\ell_1 :: \ell :: [m, M]a.t, \pi_2)$$

iff $\pi_2 \models delay(\ell_1 :: \ell :: [m, M]a.t) \equiv \ell \leq M$ and $\pi_2 = \pi_1 + 1$. Therefore, if $\pi_1 \models \ell = M$, $\pi_2 \not\models \ell \leq M$.

The case of *rec* is analogous to the untimed one. The case of *choice* is similar to the one for the untimed case. There are two rules, one for idle and one for non-idle actions. As usual the two processes must agree on idle actions.

Consider for instance the $\mathcal{TSP}$ process

$$p = \ell_1 :: \ell_2 :: a_1.\ell_3 :: nil + [1, 2]a_2.\ell_4 :: nil$$

where $a_3 \in \#(a_1)$ and $a_3 \in \#(*)$. Transitions of p are obtained either by rule $choice_1$ or by rule $choice_2$.

Since $\ell_1 :: \ell_2 :: a_1.\ell_3 :: nil \overset{\alpha_1, true}{\longmapsto} p_2$, where $\alpha_1 = a_1 : \epsilon$ and $p_2 \equiv \ell_1 :: \ell_3 :: nil$, then $p \overset{\alpha_1, true}{\longmapsto} p_2$ by $choice_1$.

Since $\ell_1 :: \ell_2 :: [1, 2]a_2.\ell_4 :: nil \overset{\alpha_2, 1 \leq \ell_2 \leq 2}{\longmapsto} p_3$, where $\alpha_2 = a_2 : \epsilon$ and $p_3 = \ell_1 :: \ell_4 :: nil$, then $p \overset{\alpha_2, 1 \leq \ell_2 \leq 2}{\longmapsto} p_3$ by $choice_1$.

Moreover, $\ell_1 :: \ell_2 :: [1, 2]a_2.\ell_4 :: nil \overset{\alpha_3, true}{\longmapsto} \ell_1 :: \ell_2 :: [1, 2]a_2.\ell_4 :: nil$ and $\ell_1 :: \ell_2 :: a_1.\ell_3 :: nil \overset{\alpha_3, true}{\longmapsto} \ell_1 :: \ell_2 :: a_1.\ell_3 :: nil$, where $\alpha_3 = a_3 : \bar{\epsilon}$. Therefore, by rule $choice_2$, $p \overset{\alpha_3, true}{\longmapsto} p$.

Note that $delay(p) \equiv \ell_2 \leq 2$ is obtained by the and of the two delays and ensures that action a_2 is not delayed, when two units of time are reached by the clock ℓ_2. Actually,

$$(p, \pi) \overset{\alpha_3}{\mapsto} (p, \pi + 1)$$

iff $\pi + 1 \models delay(p) \equiv \ell_2 \leq 2$. Therefore, if $\pi \models \ell_2 = 2$, $\pi + 1 \not\models delay(p)$.

The rules for *refinement* are similar to the ones for the untimed case. Consider a process $\ell :: t \bigtriangledown p$. The function *delay* is defined so that the two processes must agree on the increment of time. Transitions of $\ell :: t \bigtriangledown p$ are obtained by rules $\bigtriangledown_1$, $\bigtriangledown_2$ and $\bigtriangledown_3$. By rules $\bigtriangledown_1$ and $\bigtriangledown_2$ transitions corresponding to non-idle actions of both t and of p give transitions of $\ell :: t \bigtriangledown p$.

Consider the process

$$p_1 = \ell_5 :: (\ell_1 :: [1, 1]a_4.\ell_6 :: nil) \bigtriangledown p.$$

We have $delay(p_1) \equiv \ell_1 \leq 1 \wedge \ell_2 \leq 2$. Since $p \overset{\alpha_2, 1 \leq \ell_2 \leq 2}{\longmapsto} p_3$, then by rule $\bigtriangledown_2$, we obtain $p_1 \overset{\alpha_2, 1 \leq \ell_2 \leq 2}{\longmapsto} p_4$ with $p_4 = \ell_5 :: (\ell_1 :: [1, 1]a_4.\ell_6 :: nil) \bigtriangledown p_3$ and $delay(p_4) \equiv \ell_1 \leq 1$. By definition of the labelled transition system semantics,

$$(p_1, \pi_1) \overset{\alpha_2}{\mapsto} (p_4, \pi_2)$$

iff $\pi_1 \models 1 \leq \ell_2 \leq 2$, $\pi_2 = reset(\pi_1, \{\ell_5, \ell_1\}, \{\ell_4\}) + 1$ and $\pi_2 \models delay(p_4)$.

Suppose $\pi_1(\ell_1) = 1$ and $\pi_1(\ell_2) = 2$, then $(p_1, \pi_1) \overset{\alpha_2}{\not\mapsto}$. Actually, $\pi_2 \not\models \ell_1 \leq 1 \equiv delay(p_4)$. In other words, the action a_4 has priority over action a_2, when the maximal delay of a_4 is reached.

On the other hand, rule $\bigtriangledown_3$ allows one to find the transitions corresponding to idle actions. As in the untimed case, an idle action must be an idle action both of p and of t.

The rule *par* for parallel composition is quite obvious. The transitions are obtained by composing transitions of the two components and the communication is realized by the operation of parallel composition over actions.

It is important to stress that the timed graph corresponding to a closed $\mathcal{TSP}$ process is finite for regular processes.

5.3 The Translation

The translation α of chapter 3 can be trivially extended to the real-time case. It is sufficient to modify definition 3.2.6 for basic statecharts.

Definition 5.3.1 *Let $S : (\{S_1, \ldots, S_n\}, T_1, T_2, , in, out, \chi, \tau, \delta)$ be a basic non-empty timed statechart.*

For each $i \in \{1, n\}$, let $T_{i,j} = \{t \in T_j \mid out(t) = S_i\}$ for $j \in \{1, 2\}$. Let $t_{S_i} = rec\ s_i.S_i :: (\sum_{t \in T_{i,1}} lab(t).in(t)) + (\sum_{t \in T_{i,2}} [min(t), max(t)]lab(t).in(t))$, if $T_{i,1} \cup T_{i,2} \neq \emptyset$. Let $t_{S_i} = S_i :: nil$, otherwise.

Process $\alpha(S)$ is obtained by repeating the following substitutions, where initially $\alpha(S) = S :: t_{\delta(S)}$:

- *If $\alpha(S)$ is closed then stop;*
- *If $\alpha(S)$ is open, then for each $s_j \in free(\alpha(S))$, $\alpha(S) = \alpha(S)[t_{S_j}/s_j]$.*

The semantics of $\alpha(S)$ agrees with the semantics of S, namely computations of S correspond to computations in the labelled transition system of $\mathcal{G}(\alpha(S))$ and vice-versa.

Theorem 5.3.2 *Let S be a timed statechart over Π.*

- *For each $(c, age) \in TC(S)$ and for each $I \subseteq \Pi$, such that (c, age) is reachable from the initial timed configuration $(\Delta(S), \pi_0)$, if $(c, age) \stackrel{I,O}{\Longrightarrow} (c', age')$, then*
  1. *there exists $p \in \mathcal{R}_{\alpha(S)}$, such that $loc(p) = c$;*
  2. *there exists $(p, \pi) \stackrel{\alpha}{\mapsto} (p', \pi')$, where $loc(p') = c'$, $I \rightsquigarrow trigger(\alpha)$, $action(\alpha) = O - trigger(\alpha)$ and $\pi' = age'$, $\pi = age$;*
- *For each $p \in \mathcal{R}_{\alpha(S)}$, if $(p, \pi) \stackrel{\alpha}{\mapsto} (p', \pi')$, then*
  1. *there exist $(c, age), (c', age') \in TC(S)$, where $loc(p) = c$, $age = \pi$, $loc(p') = c'$, $age' = \pi'$;*

2. *there exists* $(c, age) \stackrel{I,O}{\Longrightarrow} (c', age')$, *for each* $I, O \subseteq \Pi$, *such that* $I \rightsquigarrow trigger(\alpha)$ *and* $O - trigger(\alpha) = action(\alpha)$.

The proof of this theorem is a straightforward generalization of theorem 3.2.7.

5.4 Related works

The formalism of statecharts has been extended with real-time features both in [73] and in [56] by attaching minimal and maximal delays to transitions. The main difference with our approach is that the environment is assumed to be dense, i.e. it can produce inputs at arbitrary close instants of time. Therefore, the semantics is given with respect to a dense time domain. Moreover, these semantics are so called *chain* semantics, where sequential transitions are allowed in the same step. The instantaneous reaction is more complex and additional conditions are needed for ensuring its finiteness.

In the approach of [73] negated events are not allowed, but events may have duration. The semantics is based on the synchrony hypothesis and is given in an operational style similar to the one of [37], namely a step is a maximal sequence of microsteps. Maximal and minimal delays are interpreted in a different way, since transitions are forced to be performed as soon as possible between the minimal and the maximal delay. The finiteness of a step is ensured by requiring that a state cannot be exited and entered in the same step. This approach seems to be not adequate to cope with negated events, since it is not clear when the condition "*a is absent*" must be tested with respect to a dense environment.

On the other hand, the semantics of [56] is an asynchronous (interleaving) semantics based on timed transitions systems [44]. The main difference with the semantics of [73] is that transitions may be performed (nondeterministically) within the minimal and the maximal delay. In this approach negated events are allowed, but a step is not required to be globally consistent. Therefore, the interpretation of causal paradoxes is not consistent with the one of [77]. We have chosen to adopt the interpretation of [56] for minimal and maximal delays, since the formalism of statecharts is a specification language, where nondeterminism is very useful.

Note that in a dense framework the synchrony hypothesis implies the reaction time of the system to be less than an arbitrary small quantum of time. Since the implementability of this model requires complex techniques of retiming [66] for ensuring the reaction time to fulfill the timing constraints that are given with respect to abstract time, the advantages of this abstraction are limited.

Moreover, the dense domain of time leads to other difficulties. Actually, it may be the case that an infinite number of transitions is performed in a finite quantum of time (Zeno paradox). In order to avoid this possibility, it is necessary to consider only those computations, where time advances beyond any bound.

Argos has been extended with real-time features [55] by attaching delays to states and special timeout transitions, which are performed whenever the delay of the source state is reached. The semantics is given as timed graphs and is parametric in the time domain. However, the translation is quite different from ours since the basic principles of the semantics of Argos are retained. For instance, nondeterminism is considered as an error of the system.

Chapter 6

A Proof System for Real-time μ-Calculus

In this chapter we tackle the problem of verifying the correctness of timed statecharts. For expressing quantitative temporal properties we define a real-time extension of μ-calculus with respect to the discrete time domain. The logic, called RTL_μ, has a next time operator, least and greatest fixpoints and has freeze quantifiers and timing constraints for relating the times of different states. Since we are dealing with synchronous languages, we assume next time to be equal to next state. The logic permits to express properties such as bounded invariance, bounded response, and so on. Actually, the discrete logic $RTCTL$ [31, 29] is trivially subsumed by RTL_μ.

We propose a proof system for local model checking RTL_μ. It checks whether a set of timed states of a discrete timed graph satisfies an RTL_μ formula. Judgments of the proof system are of the form $s, \omega \vdash A$, where s is a state of the timed graph, ω is a clock constraint and A is a formula. The pair (s, ω) represents a set of timed states, namely the ones for which ω holds. This kind of judgments allows us to extend the technique of [92] for computing fixpoints by tagging fixpoints formulas with sets of timed states. The proof system is proved to be sound in general and complete for timed graphs with a finite number of states. Therefore, due to the correspondence between timed statecharts and $\mathcal{TSP}$ processes, the proof system is complete for arbitrary timed statecharts.

The proof system provides a formal basis for extending the compositional method of chapter 4 to $\mathcal{TSP}$ processes.

6.1 The logic RTL_μ

We introduce a discrete version of real-time μ-calculus that is negation-free and has freeze quantification over a set of specification clocks. Moreover, as in the untimed case we adopt tagged fixpoints formulas.

Assume a set of specification clocks C, a set of logical variables VAR, a set of propositional symbols $PROP$ and a set of actions $\mathcal{A}$. Moreover, assume a set of clocks C', with $C \cap C' = \emptyset$, a set of states S and a function $clock : S \to 2^{C'}$. Formulas of the real-time μ-calculus RTL_μ are defined as follows

$$X \mid P \mid \neg P \mid A_1 \vee A_2 \mid A_1 \wedge A_2 \mid \omega \mid < \alpha > A \mid [\alpha]A \mid$$
$$x.A \mid \mu X\{U\}.A \mid \nu X\{U\}.A.$$

where $\omega \in \Omega(C)$ is a clock constraint, $x \in C$ is a specification clock, $\alpha \in \mathcal{A}$ is an action, $P \in PROP$ is a propositional symbol, $X \in VAR$ is a logical variable and $U = \{(s, \omega) \mid s \in S, \omega \in \Omega(C \cup clock(s))\}$ is the set of tags.

The modal operator $< \alpha >$ is the next time, which is equivalent in this framework to next state, while $[\alpha]$ is its dual. The quantifier $x.$ is called *reset* and is inspired by the freeze quantification [7].

Formulas are interpreted over a model $\mathcal{M} = ((TS, ts_o, \mathcal{A}, \mapsto), \gamma)$, where $\gamma : PROP \to 2^{TS}$ is the valuation function of propositional symbols and $(TS, ts_o, \mathcal{A}, \mapsto)$ is a labelled transition system, such that

- TS is a set of timed states (s, π), where $s \in S$ and $\pi : C \cup C' \to N \cup \{und\}$ is a partial assignment for the specification clocks $C' \cup C$, such that $\pi = \pi_1 \cup \pi_2$, where $\pi_1 : C \to N \cup \{und\}$ and $\pi_2 : C' \to N$;
- $ts_0 = (s_0, \pi_0)$, where $\pi_0(x) = und$, for each $x \in C$;
- $\mapsto \subseteq TS \times \mathcal{A} \times TS$ is the next state relation such that, for each $(s_1, \pi_1) \stackrel{\alpha}{\mapsto} (s_2, \pi_2)$, $\pi_2(x) = \pi_1(x) + 1$, for $x \in C$ $(und + 1 \equiv und)$.

Moreover, in order to give the semantics to open formulas we must consider the valuation function $\rho : VAR \to 2^{TS}$ for logical variables VAR. The semantics of formulas with respect to a model $\mathcal{M} = ((TS, ts_o, \mathcal{A}, \mapsto), \gamma)$ and to a valuation ρ is defined as follows:

$$\| P \|_{\mathcal{M},\rho} = \gamma(P)$$
$$\| \neg P \|_{\mathcal{M},\rho} = TS \setminus \gamma(P)$$
$$\| \omega \|_{\mathcal{M},\rho} = \{(s, \pi) \in TS \mid \pi \models \omega\}$$
$$\| x.A \|_{\mathcal{M},\rho} = \{(s, \pi) \in TS \mid (s, \pi[x := 0]) \in \| A \|_{\mathcal{M},\rho}\}$$
$$\| < \alpha > A \|_{\mathcal{M},\rho} = \{(s, \pi) \in TS \mid (s, \pi) \stackrel{\alpha}{\mapsto} (s', \pi'), (s', \pi') \in \| A \|_{\mathcal{M},\rho}\}$$
$$\| [\alpha]A \|_{\mathcal{M},\rho} = \{(s, \pi) \in TS \mid \forall (s, \pi) \stackrel{\alpha}{\mapsto} (s', \pi'), (s', \pi') \in \| A \|_{\mathcal{M},\rho}\}$$
$$\| \mu X\{U\}.A \|_{\mathcal{M},\rho} = \mu V.(\| A \|_{\mathcal{M},\rho[V/X]} \setminus \| U \|_{\mathcal{M},\rho[V/X]})$$
$$\| \nu X\{U\}.A \|_{\mathcal{M},\rho} = \nu V.(\| A \|_{\mathcal{M},\rho[V/X]} \cup \| U \|_{\mathcal{M},\rho[V/X]})$$

where $\| U \|_{\mathcal{M},\rho} = \{(s, \pi) \in TS \mid (s, \omega) \in U, \pi \models \omega\}$.

We use the notation $(s, \pi) \models_{\mathcal{M}} A$ for $(s, \pi) \in \| A \|_{\mathcal{M},\rho}$ for each valuation ρ.

The discrete extension of CTL with bounded operators $(RTCTL)$ [31] is subsumed by RTL_μ. Consider the time bounded property $E\Diamond_{\leq 3}A$. It is expressed by the formula

$$x.\mu X.((A \wedge x \leq 3) \vee < - > X).$$

Analogously, $A\Diamond_{\leq 3}A$ is expressed by the formula

$$x.\mu X.((A \wedge x \leq 3) \vee [-]X).$$

On the other hand, $A\Box_{\leq 3}A$ is expressed by the formula

$$x.\nu X.(((A \wedge x \leq 3) \vee (x > 3)) \wedge [-]X).$$

Hence, the time bounded response property $A\Box(P \supset A\Diamond_{\leq 5}Q)$ is expressed by the formula

$$\nu X.(\neg P \vee x.(\mu Z.((Q \wedge x \leq 5) \vee [-]Z)) \wedge [-]X).$$

Let $\mathcal{G} = (S, s_0, Act, C', \longmapsto, delay, clock)$ be a discrete timed graph and let $\gamma : PROP \rightarrow 2^{TS}$ be a valuation for propositional symbols $PROP$. We say that an RTL_μ formula A is defined over $\mathcal{G}$, $PROP$ and C iff it is defined over actions Act, propositional symbols $PROP$, specification clocks C, with $C \cap C' = \emptyset$, and over S and $clock$.

Suppose $LTS(\mathcal{G}) = (TS', ts'_0, Act, \mapsto)$. For a set of specification clocks C, such that $C' \cap C = \emptyset$, we define the model of $\mathcal{G}$ with respect to C and γ as $\mathcal{M}(\mathcal{G}, \gamma, C) = ((TS, ts_0, Act, \mapsto), \gamma)$, where

- $TS = \{(s, \pi) \mid (s, \pi') \in TS'$ and $\pi = \pi' \cup \pi''$, where $\pi'' : C \rightarrow N \cup \{und\}\}$.
- $ts_0 = (s_0, \pi_0)$ where $\pi_0 = \pi'_0 \cup \pi''_0$, where $\pi''_0(x) = und$, for $x \in C$, and $ts_0 = (s_0, \pi'_0)$;
- $(s_1, \pi_1) \stackrel{\alpha}{\mapsto} (s_2, \pi_2)$ iff $(s_1, \pi'_1) \stackrel{\alpha}{\mapsto} (s_2, \pi'_2)$, where $\pi_i = \pi'_i \cup \pi''_i$, where $\pi''_2 = \pi''_1 + 1$.

The idea is that $\mathcal{M}(\mathcal{G}, \gamma, C)$ represents the extension of $LTS(\mathcal{G})$ obtained by considering the additional assignments for the specification clocks C. For a closed RTL_μ formula A over $\mathcal{G}$, $PROP$ and C, we say that $(s, \pi) \models A$ iff $(s, \pi) \models_{\mathcal{M}(\mathcal{G},\gamma,C)} A$.

6.2 The proof system

We introduce the proof system for checking whether a set of states of a given discrete timed graph satisfies an RTL_μ assertion. The judgments of the proof system are of the form $s, \omega \vdash A$, where $s \in S$ is a state, ω is a clock constraint over $clock(s) \cup C$ and A is a closed RTL_μ formula. The meaning of this kind of judgments is that the timed state (s, π) satisfies A, for each clock assignment π, such that $\pi \models \omega$.

In the following, let $\mathcal{G} = (S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph and γ be the corresponding valuation function for propositional symbols $PROP$.

We need some preliminary definitions. Assume $\omega \in \Omega(C)$. We denote by $\omega + \delta$ ($\omega - \delta$) for $\delta \in N$, the clock constraint $\omega[x + \delta/x]$ ($\omega[x - \delta/x]$), for each clock $x \in C$.

For $C_1 \subseteq C$, we denote by $pro_{C_1}(\omega)$ the first order formula $\exists x_1, \ldots, x_n\ (\omega)$, where $C \setminus C_1 = \{x_1, \ldots x_n\}$.

Proposition 6.2.1 *Let C be a set of clocks, $\omega \in \Omega(C)$ and $C_1 \subseteq C$. There exists $\omega' \in \Omega(C_1)$, such that $pro_{C_1}(\omega) \equiv \omega'$.*

Let $\omega \in \Omega(C)$ and $C_1 \subseteq C$, $C_2 \cap C_1 = \emptyset$. We denote by $reset(\omega, C_1, C_2) \equiv pro_{C_1}(\omega) \wedge \bigwedge_{x \in C_2} x = 0$. Note that $reset(\omega, C_1, C_2) \in \Omega(C_1 \cup C_2)$ by proposition 6.2.1.

Intuitively, $pro_{C_1}(\omega)$ and $reset(\omega, C_1, C_2)$ are the counterparts of the corresponding operations over clock assignments. The following properties hold.

Proposition 6.2.2 *Let C be a set of clocks and $C_1 \subseteq C$, $C_1 \cap C_2 = \emptyset$. For each assignment π over C and for each $\omega \in \Omega(C)$:*

1. *$\pi \models \omega$ then $pro_{C_1}(\pi) \models pro_{C_1}(\omega)$. Moreover, if ω is satisfiable, then if $\pi' \models pro_{C_1}(\omega)$, there exists $\pi \models \omega$ and $\pi' = pro_{C_1}(\pi)$.*

2. *$\pi \models \omega$, then $reset(\pi, C_1, C_2) \models reset(\omega, C_1, C_2)$. Moreover, if ω is satisfiable and $\pi' \models reset(\omega, C_1, C_2)$, then there exists $\pi \models \omega$ and $reset(\pi, C_1, C_2,) = \pi'$.*

3. *$\pi \models \omega + \delta$ iff $\pi + \delta \models \omega$.*

4. *$\pi \models \omega - \delta$ iff $\pi - \delta \models \omega$.*

For $\omega \in \Omega(C)$, we use the notation $\forall(\omega)$ for $\forall x_1, \ldots, \forall x_n(\omega)$, where $C = \{x_1, \ldots, x_n\}$.

The proof system is composed by four sets of rules:

Logical Rules These rules work on the structure of the formula and modify neither the state nor the clock constraint;

Structural Rules These rules work on the clock constraint.

Rules for modal operators There is one rule for $x.A$, one for $< \alpha > A$ and one for $[\alpha]A$. These rules modify the state, the clock constraint and the formula. Rules for $< \alpha > A$ and $[\alpha]A$ explicitly refer to transitions of the graph.

Rules for fixpoints These rules realize the unfolding of fixpoints formulas.

6.2.1 Logical rules

$$\frac{}{s, \omega \vdash P}\, prop_1 \quad s \in \gamma(P) \qquad \frac{}{s, \omega \vdash \neg P}\, prop_2 \quad s \notin \gamma(P)$$

$$\frac{s, \omega \vdash A_1 \wedge A_2}{s, \omega \vdash A_1 \quad s, \omega \vdash A_2}\, \wedge \qquad \frac{s, \omega \vdash A_1 \vee A_2}{s, \omega \vdash A_i}\, \vee$$

6.2.2 Structural rules

$$\frac{s, \omega_1 \vdash A}{s, \omega_2 \vdash A}\, thin \qquad \models \forall(\omega_1 \supset \omega_2)$$

$$\frac{s, \omega_1 \vee \omega_2 \vdash A}{s, \omega_1 \vdash A \quad s, \omega_1 \vdash A}\, disj$$

$$\frac{s, \omega_1 \vdash \omega_2}{}\, taut \qquad \models \forall(\omega_1 \supset \omega_2)$$

$$\frac{s, \omega \vdash A}{}\, empty \qquad \models \forall(\omega \wedge delay(s) \supset false)$$

6.2.3 Rules for modal operators

$$\frac{s, \omega \vdash x.A}{s, reset(\omega, (clock(s) \cup C) \setminus \{x\}, x) \vdash A}\, reset$$

$$\frac{s, \omega_1 \vdash < \alpha > A}{s', \omega_2 \vdash A} <> \quad \begin{array}{l} \models \forall(\omega_1 \supset \omega) \text{ for } s \overset{\alpha,\omega}{\longmapsto} s' \\ \omega_2 \equiv reset(\omega_1, C_1, C_2) - 1 \\ C_1 = (clock(s') \cap clock(s)) \cup C \\ C_2 = clock(s') \setminus clock(s) \\ \models \forall(\omega_2 \supset delay(s')) \end{array}$$

$$\frac{s, \omega \vdash [\alpha]A}{\{s_i, \omega_i \vdash A\}_{i \in \{1,n\}}} [] \quad \begin{array}{l} \{s \overset{\alpha,\omega'_i}{\longmapsto} s_i \text{ such that } \not\models \forall(\omega \wedge \omega'_i \supset false)\}_{i \in \{1,n\}} \\ \omega_i \equiv reset(\omega'_i \wedge \omega, C_{1,i}, C_{2,i}) - 1 \\ C_{1,i} = (clock(s_i) \cap clock(s)) \cup C \\ C_{2,i} = clock(s_i) \setminus clock(s) \end{array}$$

Let us explain this set of rules. Rule *reset* is quite obvious and is based on the semantics of $x.$. Actually, $(s, \pi) \models x.A$ iff $(s, \pi[x := 0]) \models A$. The idea is that, if $\pi \models \omega$, then $\pi[x := 0] \models reset(\omega, (clock(s) \cup A) \setminus \{x\}, x)$, by proposition 6.2.2. Suppose $\omega \equiv y \leq x \wedge x \leq 3$. In order to prove $s, \omega \vdash x.A$, we must prove $s, x = 0 \wedge y \leq 3 \vdash A$.

Rule $<>$ is more involved. In order to prove $s, \omega \vdash < \alpha > A$, we must show that, for each $\pi \models \omega$, there exists $(s, \pi) \overset{\alpha}{\mapsto} (s', \pi')$, such that $(s', \pi') \models A$.

According to the labelled transition system semantics of timed graphs, it is sufficient to find a transition $s \overset{\alpha,\omega}{\longmapsto} s'$ such that, for each $\pi \models \omega_1$, $\pi \models \omega$ and $(s', \pi') \models A$, $\pi' \models delay(s')$, where $\pi' = reset(\pi, C_1, C_2) + 1$ for $C_1 = (clock(s) \cap clock(s') \cup C)$ and $C_2 = clock(s') \setminus clock(s)$. The conditions $\models \forall(\omega_1 \supset \omega)$ and

$\omega_2 \equiv reset(\omega_1, C_1, C_2) - 1$ and $\models \forall(\omega_2 \supset delay(s'))$ ensure that these requirements are fulfilled, by proposition 6.2.2.

Consider for instance $\omega_1 \equiv 1 \leq x \leq 4 \wedge y \leq 6$ and suppose $s \overset{\alpha, x \leq 7}{\longmapsto} s'$, where $delay(s) \equiv true$ and $delay(s') \equiv y \leq 7$, $clock(s') \setminus clock(s) = \emptyset$ and $x \notin clock(s')$, $y \in clock(s) \cap clock(s')$. Since $\models \forall(\omega_1 \supset x \leq 7)$, $reset(\omega_1, \{y\}, \emptyset) - 1 \equiv y - 1 \leq 6$ and $\models \forall(reset(\omega_1, \{y\}, \emptyset) - 1 \supset delay(s'))$, then it is sufficient to prove $s', y \leq 7 \vdash A$. On the other hand, if either $delay(s') = y \leq 6$ or $\omega_1 \equiv x > 7$, then the rule is obviously not applicable.

In order to prove $s, \omega \vdash [\alpha]A$, we must show that, for each $\pi \models \omega$, if $(s, \pi) \overset{\alpha}{\mapsto} (s', \pi')$, then $(s', \pi') \models A$. Let $\{s \overset{\alpha, \omega_i'}{\longmapsto} s_i\}_{i \in \{1,n\}}$ be the set of transitions, such that $\omega \wedge \omega_i'$ is satisfiable. According to the labelled transition system semantics of timed graphs, for each $\pi \models \omega$, $(s, \pi) \overset{\alpha}{\mapsto} (s', \pi')$ iff there exists $i \in \{1, n\}$, such that $\pi \models \omega_i'$ and $s' = s_i$, $\pi' = (reset(\pi, (clock(s) \cap clock(s_i)) \cup C, clock(s_i) \setminus clock(s))) + 1$. Moreover, $\pi' \models delay(s_i)$. The rule is based on the observation that $\pi' \models \omega_i$, by proposition 6.2.2.

Consider for instance $\omega \equiv 3 \leq x \leq 5 \wedge y \leq x$ and suppose that the transitions corresponding to α are $s \overset{\alpha, x < 1}{\longmapsto} s_1$ and $s \overset{\alpha, y \leq 4}{\longmapsto} s_2$ and $delay(s) = delay(s_1) = delay(s_2) \equiv true$ and $clock(s) = clock(s_1) = clock(s_2)$. We have that $3 \leq x \leq 5 \wedge y \leq x \wedge x < 1$ is not satisfiable. On the other hand, for $s \overset{\alpha, y \leq 4, y}{\longmapsto} s_2$, $y \leq 4 \wedge 3 \leq x \leq 5 \wedge y \leq x$ is satisfiable. Hence, it is sufficient to prove $s_2, y \leq 5 \wedge 4 \leq x \leq 6 \wedge y \leq x \vdash A$.

6.2.4 Rules for fixpoints

$$\frac{s, \omega \vdash \nu X\{U\}.A}{} \; \nu_1 \qquad \begin{array}{l} \{(s, \omega_i) \subseteq U\}_{i \in \{1,n\}} \\ \models \forall(\omega \supset \bigvee_{i \in \{1,n\}} \omega_i) \end{array}$$

$$\frac{s, \omega \vdash \nu X\{U\}.A}{s, \omega \vdash A[\nu X\{U \cup \{(s, \omega)\}\}.A/X]} \; \nu_2$$

$$\frac{s, \omega \vdash \mu X\{U\}.A}{s, \omega \vdash A[\mu X\{U \cup \{(s, \omega)\}\}.A/X]} \; \mu \qquad \begin{array}{l} \{(s, \omega_i) \subseteq U\}_{i \in \{1,n\}} \\ \forall i \in \{1, n\} \models \forall(\omega \wedge \omega_i \supset false) \end{array}$$

These rules are the obvious generalization of the untimed ones. Actually, the side condition of rule ν_1 ensures that $\| (s, \omega) \| \subseteq \| U \|$. On the other hand, the side condition of rule μ ensures that $\| (s, \omega) \| \cap \| U \| = \emptyset$.

6.3 Soundness and completeness of the proof system

The proof system is sound in general and complete for timed graphs with a finite number of states.

Soundness means that, if $s, \omega \vdash A$ then $(s, \pi) \models A$, for each clock assignment π over $C \cup clock(s)$, such that $\pi \models \omega$. We use the notation $s, \omega \models A$, for $(s, \pi) \models A$, for each assignment π over $C \cup clock(s)$, such that $\pi \models \omega$.

For proving the soundness of fixpoints rules we need the following lemma, that is an extension of the reduction lemma 2.1.3.

Lemma 6.3.1 *Let ψ be a monotonic function on the lattice $\mathcal{P}(D)$. For $P \in \mathcal{P}(D)$,*

1. $P \subseteq \psi(\mu V.(\psi(V) \setminus P)) \Rightarrow P \subseteq \mu V.(\psi(V))$;
2. $P \subseteq \psi(\nu V.(\psi(V) \cup P)) \Rightarrow P \subseteq \nu V.(\psi(V))$.

The proof of this lemma can be found in appendix C.1.

Theorem 6.3.2 *(Soundness) Let $\mathcal{G} = ((S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph, γ be a corresponding valuation for propositional symbols $PROP$ and A be a closed RTL_μ formula over $\mathcal{G}$, $PROP$ and C. For $s \in S$ and $\omega \in \Omega(clock(s) \cup C)$, if $s, \omega \vdash A$ then $s, \omega \models A$.*

The proof of this theorem is based on showing that each rule is backward sound, and it can be found in appendix C.1. The most difficult cases are those of rules *reset*, $<>$ and $[]$, where the thesis follows by proposition 6.2.2.

The proof system is complete for the class of timed graphs with a finite number of states. The proof is based on the observation that there exists a finite quotient of the model $\mathcal{M}(\mathcal{G}, \gamma, C)$ that is sufficient for establishing A. A similar result is used by [45, 2] for proving that fixpoints can be computed in a finite number of iterations with respect to a dense domain of time. The underlying idea is that the set of timed states satisfying a formula can be characterized as a *region*.

Definition 6.3.3 *Let $\mathcal{G} = (S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph, γ be a corresponding valuation over $PROP$ and A be a RTL_μ formula over $\mathcal{G}$, $PROP$ and C. For $\omega \in \Omega(C' \cup C)$, we say that ω is a* $(\mathcal{G}, A)$-constraint *iff the greatest constant of ω is k, where k is the greatest constant that occurs in $\mathcal{G}$ or in A. A set of timed states $\mathcal{R}$ is a* $(\mathcal{G}, A)$-region *iff, for each $s \in S$, there exists a $(\mathcal{G}, A)$-constraint ω, such that $\omega \in \Omega(clock(s) \cup C)$ and $(s, \pi) \in \mathcal{R}$ iff $\pi \models \omega$.*

We denote by $\mathcal{R}_{(\mathcal{G},A)}$ the set of all $(\mathcal{G}, A)$-regions. The basic observation is that, if the set of states of $\mathcal{G}$ is finite, there are only finitely many $(\mathcal{G}, A)$-regions. Moreover, the set of states satisfying a formula A is a $(\mathcal{G}, A)$-region.

Lemma 6.3.4 *Let $\mathcal{G} = (S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph, γ be a corresponding valuation over $PROP$ and A be a fixpoint-quantifier free RTL_μ formula over $\mathcal{G}$, $PROP$ and C with free formula variables $X_1, \ldots, X_n$. If $\rho(X_i) \in \mathcal{R}_{(\mathcal{G},A)}$, for each $i \in \{1, n\}$, then $\| A \|_{\mathcal{M}(\mathcal{G},\gamma,C),\rho} \in \mathcal{R}_{(\mathcal{G},A)}$.*

The proof of this lemma is by induction on the structure of A and is shown in appendix C.2.

Lemma 6.3.5 *Let $\mathcal{G} = (S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph, γ be a corresponding valuation over $PROP$ and A be a closed RTL_μ formula over $\mathcal{G}$, $PROP$ and C. Then $\| A \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \in \mathcal{R}_{(\mathcal{G},A)}$.*

Proof. Straightforward by lemma 6.3.4. ∎

Note that, if S is finite, then the number of regions is finite and, by lemma 6.3.4, fixpoints can be computed within a finite number of steps. This ensures the following properties.

Lemma 6.3.6 *Let ψ be a monotonic function on the lattice $\mathcal{P}(D)$. For $P \in \mathcal{P}(D)$,*

$$P \subseteq \nu V.\psi(V) \Rightarrow P \subseteq \psi(\nu V.(\psi(V) \cup P)).$$

A result similar to the one of lemma 6.3.6 does not hold for least fixed points. However, the following property holds.

Lemma 6.3.7 *Let ψ be a monotonic function on the lattice $\mathcal{P}(D)$ such that $\mu V.\psi(V) = \psi^k(\emptyset)$ for some integer k. For $P \in \mathcal{P}(D)$, there exists $\{P_i\}_{i\in\{1,n\}}$, such that $P = \bigcup_{i\in\{1,n\}} P_i$, and, for each $i \in \{1, n\}$,*

$$P_i \subseteq \mu V.\psi(V) \Rightarrow P_i \subseteq \psi(\mu V.(\psi(V) \setminus P_i)).$$

The proofs of lemmata 6.3.6 and 6.3.7 are shown in appendix C.2. The completeness of the proof system for timed graphs with a finite number of states is proved by induction on a relation $\prec$ on closed formulas.

Definition 6.3.8 *Let $\mathcal{G} = (S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph, γ be a corresponding valuation over $PROP$ and A, A' be closed RTL_μ formulas over $\mathcal{G}$, $PROP$ and C, with tags (s, ω), where ω is a $(\mathcal{G}, A)$-constraint. We define $A' \prec A$ iff either A' is a proper subformula of A or $A = \sigma X\{U\}B$, for $\sigma \in \{\mu, \nu\}$, and $A' = B[\sigma X\{U \cup \{(s, \omega)\}\}B/X]$ where $\omega \in \Omega(clock(s) \cup C)$ is a $(\mathcal{G}, A)$-constraint, such that $\| (s, \omega) \| \cap \| U \| = \emptyset$.*

Since the set of $(\mathcal{G}, A)$-regions is finite, if S is finite, then $\prec$ is well-founded for this class of timed graphs.

Theorem 6.3.9 *(Completeness) Let $\mathcal{G} = (S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph with S finite, γ be a corresponding valuation over $PROP$ and A be a closed RTL_μ formula over $\mathcal{G}$, $PROP$ and C. For $s \in S$ and $\omega \in \Omega(clock(s) \cup C)$, if $s, \omega \models A$ then $s, \omega \vdash A$.*

The proof of this theorem is by induction on $\prec$ and is shown in appendix C.2.

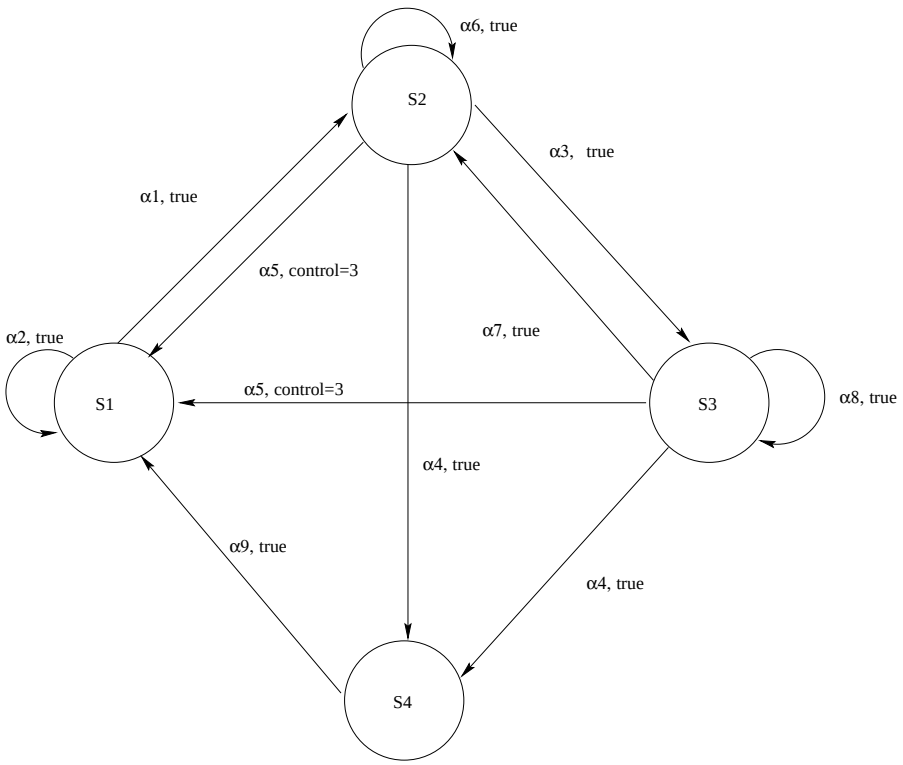

Figure 6.1: The timed graph

6.4 An example

In this section we show the use of the proof system for the statechart of figure 5.2 of chapter 5.

The timed graph of this statechart obtained via the translation α is shown in figure 6.1. The states correspond to the following configurations.

$$\begin{aligned} s_1 &\equiv \{teller - machine, ready\} \\ s_2 &\equiv \{teller - machine, control, code\} \\ s_3 &\equiv \{teller - machine, control, check - code\} \\ s_4 &\equiv \{teller - machine, operation\} \end{aligned}$$

Moreover, $delay(s_1) \equiv true$, $delay(s_2) \equiv control \leq 3$, $delay(s_3) \equiv control \leq 3$ and $delay(s_4) \equiv true$. The labels of transitions are as follows:

$$\begin{aligned} &\alpha_1 = lab(t_1) : \epsilon \quad \alpha_2 = \#(lab(t_1)) : \bar{\epsilon} \\ &\alpha_3 = lab(t_2) : \epsilon \quad \alpha_4 = lab(t_5) : \epsilon \\ &\alpha_5 = lab(t_4) : \epsilon \quad \alpha_6 = \#(lab(t_2)) \cap \#(lab(t_5))) : \bar{\epsilon} \\ &\alpha_7 = lab(t_3) : \epsilon \quad \alpha_8 = \#(lab(t_3)) \cap \#(lab(t_5))) : \bar{\epsilon} \\ &\alpha_9 = lab(t_6) : \epsilon. \end{aligned}$$

Consider the property

$$A \equiv \nu X.(((\neg control \vee (control \wedge B)) \wedge [-]X)$$

where

$$B \equiv x.(\mu Y.((\neg control \wedge x \leq 3) \vee [-]Y)).$$

This property states that, for each computation, the duration of state *control* is less or equal to three units of time.

We show the proof for s_1, ω_0, where $\omega_0 \equiv teller - machine = 0 \wedge ready = 0$.

$$\dfrac{\dfrac{\dfrac{s_1, \omega_0 \vdash A}{s_1, true \vdash A}\, thin}{s_1, true \vdash (\neg control \vee (control \wedge B)) \wedge [-]A_1}\, \nu_2}{\pi_0}$$

where $A_1 = \nu X\{(s_1, true)\}.(((\neg control \vee (control \wedge B)) \wedge [-]X)$.

Proof π_0 is as follows.

$$\dfrac{\dfrac{s_1, true \vdash (\neg control \vee (control \wedge B)) \wedge [-]A_1}{s_1, true \vdash \neg control \vee (control \wedge B)}\, \wedge}{\dfrac{s_1, true \vdash \neg control}{}\, prop_2 \qquad \dfrac{s_1, true \vdash [-]A_1}{\pi_1}}\, \vee$$

The proof π_1 is as follows.

$$\dfrac{\dfrac{}{s_1, true \vdash [-]A_1}\,[]}{\dfrac{}{s_1, \omega_{0,1} \vdash A_1}\,\nu_1 \qquad \dfrac{}{s_2, \omega_{0,2} \vdash A_1}\,\pi_2}$$

where $\omega_{0,1} \equiv ready \geq 1 \wedge teller - machine \geq 1$, $\omega_{0,2} \equiv code = 1 \wedge control = 1 \wedge teller - machine \geq 1$.

Actually, for $s_1 \overset{\alpha_2, true}{\longmapsto} s_1$, $true - 1 \equiv ready \geq 1 \wedge teller - machine \geq 1$. On the other hand, for $s_1 \overset{\alpha_1, true}{\longmapsto} s_2$ we have $C_1 = clock(s_1) \cap clock(s_2) = \{teller - machine\}$ and $C_2 = clock(s_2) \backslash clock(s_1) = \{control, code\}$ so that $reset(true, C_1, C_2) \equiv code = 0 \wedge control = 0$ and $reset(true, C_1, C_2) - 1 \equiv code = 1 \wedge control = 1$. Note that we can omit the values for the clock $teller - machine$ since they are not significant.

The proof π_2 is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{s_2, \omega_{0,2} \vdash A_1}}{s_2, \omega_1 \vdash A_1}\,thin}{s_2, \omega_1 \vdash (\neg control \vee (control \wedge B)) \wedge [-]A_2}\,\nu_2}{\dfrac{}{s_2, \omega_1 \vdash \neg control \vee (control \wedge B)}\,\pi_{2,1} \qquad \dfrac{}{s_2, \omega_1 \vdash [-]A_2}\,\pi_{2,2}}\,\wedge$$

where $A_2 = \nu X\{(s_1, true), (s_2, \omega_1)\}.(((\neg control \vee (control \wedge B)) \wedge [-]X)$ and $\omega_1 \equiv 1 \leq control \leq 3$.

1. The proof $\pi_{2,1}$ is as follows.

$$\dfrac{\dfrac{\dfrac{}{s_2, \omega_1 \vdash \neg control \vee (control \wedge B)}}{s_2, \omega_1 \vdash control \wedge B}\,\vee}{\dfrac{}{s_2, \omega_1 \vdash control}\,prop_1 \qquad \dfrac{}{s_2, \omega_1 \vdash B}\,\pi_3}\,\wedge$$

The proof π_3 is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{s_2, \omega_1 \vdash B}}{s_2, \omega_1 \wedge x = 0 \vdash C}\,reset}{s_2, 1 \leq control \leq 3 \wedge x \leq control - 1 \vdash C}\,thin}{\dfrac{}{s_2, \omega_2 \vdash C}\,\pi_{3,1} \qquad \dfrac{}{s_2, \omega_3 \vdash C}\,\pi_{3,2} \qquad \dfrac{}{s_2, \omega_4 \vdash C}\,\pi_{3,3}}\,disj$$

where

$\omega_2 \equiv control = 3 \wedge x \leq control - 1$
$\omega_3 \equiv 2 \leq control < 3 \wedge x \leq control - 1$
$\omega_4 \equiv 1 \leq control < 2 \wedge x \leq control - 1$
$C \equiv \mu Y.((\neg control \wedge x \leq 3) \vee [-]Y)$.

1.1. Proof $\pi_{3,1}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{s_2, \omega_2 \vdash C}{s_2, \omega_2 \vdash (\neg control \wedge x \leq 3) \vee [-]C_1}\,\mu}{s_2, \omega_2 \vdash [-]C_1}\,\vee}{\dfrac{s_2, \omega_{2,1} \vdash C_1}{}\quad \dfrac{s_3, \omega_{2,2} \vdash C_1}{}\quad \dfrac{s_4, \omega_{2,3} \vdash C_1}{\pi_{3,1,1}}\quad \dfrac{s_1, \omega_{2,4} \vdash C_1}{\pi_{3,1,2}}}\,[]$$

where

$C_1 \equiv \mu Y\{(s_2, \omega_2)\}.((\neg control \wedge x \leq 3) \vee [-]Y)$
$\omega_{2,1} \equiv control = 4 \wedge x \leq control - 1$
$\omega_{2,2} \equiv \omega_{2,1} \wedge check - code = 1$
$\omega_{2,3} \equiv x \leq 3 \wedge operation = 1$
$\omega_{2,4} \equiv x \leq 3 \wedge ready = 1$.

Actually, for $s_2 \overset{\alpha_6, true}{\longmapsto} s_2$, $\omega_2 - 1 \equiv control = 4 \wedge x \leq control - 1 \equiv \omega_{2,1}$ and $\omega_{2,1} \wedge delay(s_2) \equiv \omega_{2,1} \wedge control \leq 3$ is not satisfiable. Therefore, $s_2, \omega_{2,1} \vdash C_1$ by rule *empty*.

Analogously, for $s_2 \overset{\alpha_3, true}{\longmapsto} s_3$, we have $C_1 = \{x\} \cup (clock(s_3) \cap clock(s_2)) = \{teller - machine, control\}$ and $C_2 = clock(s_3) \setminus clock(s_2) = \{check - code\}$. Therefore, $reset(\omega_2, C_1, C_2) - 1 \equiv \omega_{2,1} \wedge check - code = 1$ and $\omega_{2,2} \wedge delay(s_3) \equiv \omega_{2,2} \wedge control \leq 3$ is not satisfiable. Therefore, $s_3, \omega_{2,2} \vdash C_1$ by rule *empty*.

For $s_2 \overset{\alpha_4, true}{\longmapsto} s_4$ we have $C_1 = \{x\} \cup (clock(s_2) \cap clock(s_4)) = \{teller - machine\}$ and $C_2 = clock(s_4) \setminus clock(s_2) = \{operation\}$. Therefore, $reset(\omega_2, C_1, C_2) \equiv x \leq 2 \wedge operation = 0$ and $x - 1 \leq 2 \wedge operation - 1 = 0 \equiv \omega_{2,3}$.

Finally, For $s_2 \overset{\alpha_5, control=3}{\longmapsto} s_1$, we have $C_1 = \{x\} \cup (clock(s_2) \cap clock(s_1)) = \{teller - machine\}$ and $C_2 = clock(s_1) \setminus clock(s_2) = \{ready\}$.
Therefore, $reset(\omega_2, C_1, C_2) - 1 \equiv \omega_{2,4}$.

Proof $\pi_{3,1,1}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{s_4, \omega_{2,3} \vdash C_1}{s_4, \omega_{2,3} \vdash (\neg control \wedge x \leq 3) \vee [-]C_2}\,\mu}{s_4, \omega_{2,3} \vdash \neg control \wedge x \leq 3}\,\vee}{\dfrac{s_4, \omega_{2,3} \vdash \neg control}{}\,prop_2 \qquad \dfrac{s_4, \omega_{2,3} \vdash x \leq 3}{}\,taut}\,\wedge$$

where $C_2 \equiv \mu Y\{(s_2, \omega_2), (s_4, \omega_{2,3})\}.(\neg control \wedge x \leq 3) \vee [-]Y$.

Proof $\pi_{3,1,2}$ is analogous to $\pi_{3,1,1}$, since $control \notin \gamma(s_1)$.

1.2. Proof $\pi_{3,2}$ is as follows.

$$\dfrac{\dfrac{\dfrac{s_2, \omega_3 \vdash C}{s_2, \omega_3 \vdash (\neg control \wedge x \leq 3) \vee [-]C_3}\,\mu}{s_2, \omega_3 \vdash [-]C_3}\,\vee}{\dfrac{s_2, \omega_{3,1} \vdash C_3}{\pi_{3,2,1}} \quad \dfrac{s_3, \omega_{3,2} \vdash C_3}{\pi_{3,2,2}} \quad \dfrac{s_4, \omega_{2,3} \vdash C_3}{\pi_{3,1,1}}}\,[]$$

where

$C_3 \equiv \mu Y\{(s_2, \omega_3)\}.((\neg control \wedge x \leq 3) \vee [-]Y)$
$\omega_{3,1} \equiv 3 \leq control < 4 \wedge x \leq control - 1$
$\omega_{3,2} \equiv \omega_{3,1} \wedge check - code = 1.$

Actually, for $s_2 \overset{\alpha_5, control=3}{\longmapsto} s_1$, $\omega_3 \wedge control = 3$ is not satisfiable.
Proof $\pi_{3,2,1}$ is derivable from $\pi_{3,1}$, since

1. $\omega_{3,1} \equiv \omega_2 \vee 3 < control < 4$ and $3 < control < 4 \wedge delay(s_2)$ is not satisfiable. By rule *disj* and *empty*, $s_2, \omega_{3,1} \vdash C_3$ is derivable from $s_2, \omega_2 \vdash C_3$;

2. $\omega_3 \wedge \omega_2$ is not satisfiable so that the proof for C and for C_3 is the same.

We omit proof $\pi_{3,2,2}$, since it is indeed similar to $\pi_{3,1}$. Actually, s_2 and s_3 are symmetric.

1.3. Proof $\pi_{3,3}$ is as follows.

$$\dfrac{\dfrac{\dfrac{s_2, \omega_4 \vdash C}{s_2, \omega_4 \vdash (\neg control \wedge x \leq 3) \vee [-]C_4}\,\mu}{s_2, \omega_4 \vdash [-]C_4}\,\vee}{\dfrac{s_2, \omega_{4,1} \vdash C_4}{\pi_{3,3,1}} \quad \dfrac{s_3, \omega_{4,2} \vdash C_4}{\pi_{3,3,2}} \quad \dfrac{s_4, \omega_{2,3} \vdash C_4}{\pi_{3,1,1}}}\,[]$$

where

$C_4 \equiv \mu Y\{(s_2, \omega_4)\}.((\neg control \wedge x \leq 3) \vee [-]Y)$
$\omega_{4,1} \equiv 2 \leq control < 3 \wedge x \leq control - 1$
$\omega_{4,2} \equiv \omega_{4,1} \wedge check - code = 1$

Proof $\pi_{3,3,1}$ is similar to $\pi_{3,2}$, since

1. $\omega_3 \equiv \omega_{4,1}$;

2. $\omega_3 \wedge \omega_4$ is not satisfiable so that the proof for C and for C_4 is the same.

We omit the proof of $\pi_{3,3,2}$ that is indeed similar to $\pi_{3,2}$, since s_2 and s_3 are symmetric.

2. Proof $\pi_{2,2}$ is as follows.

$$\dfrac{s_2, \omega_1 \vdash [-]A_2}{\dfrac{s_2, \omega_5 \vdash A_2}{\pi_{4,3}} \quad \dfrac{s_1, \omega_7 \vdash A_2}{}\,\nu_1 \quad \dfrac{s_3, \omega_6 \vdash A_2}{\pi_{4,1}} \quad \dfrac{s_4, \omega_8 \vdash A_2}{\pi_{4,2}}}\,[]$$

where $\omega_5 \equiv 2 \leq control \leq 4$ and $\omega_6 \equiv \omega_5 \wedge check - code = 1$, $\omega_7 \equiv ready = 1$ and $\omega_8 \equiv operation = 1$.

2.1. Proof $\pi_{4,3}$ is as follows.

$$\dfrac{s_2, \omega_5 \vdash A_2}{\dfrac{s_2, \omega_{5,1} \vdash A_2}{}\,\nu_1 \quad \dfrac{s_2, \omega_{5,2} \vdash A_2}{}\,empty}\,disj$$

where $\omega_{5,1} \equiv 2 \leq control \leq 3$ and $\omega_{5,2} \equiv 3 < control \leq 4$. Actually, $\omega_{5,2} \wedge delay(s_2)$ is not satisfiable.

2.2. Proof $\pi_{4,2}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{s_4, \omega_8 \vdash A_2}{s_4, true \vdash A_2}\,thin}{s_4, true \vdash (\neg control \vee (control \wedge B)) \wedge [-]A_4}\,\nu_2}{\dfrac{\dfrac{\dots}{s_4, true \vdash \neg control}\,prop_2 \quad \dfrac{s_4, true \vdash [-]A_4}{s_1, ready = 1 \vdash A_4}\,[]\ \nu_1}{}\,\wedge}$$

$A_4 \equiv \nu x.\{(s_1, true), (s_2, \omega_1), (s_4, true)\}.(\neg control \vee (control \wedge B) \wedge [-]X)$.

2.3. Proof $\pi_{4,1}$ is as follows.

$$\dfrac{s_3, \omega_6 \vdash A_2}{\dfrac{s_3, \omega_{6,1} \vdash A_2}{}\,empty \quad \dfrac{s_3, \omega_{6,2} \vdash A_2}{\pi'_4}}\,disj$$

where

$$\omega_{6,1} \equiv 3 < control \leq 4 \wedge check - code = 1$$
$$\omega_{6,2} \equiv 2 \leq control \leq 3 \wedge check - code = 1.$$

Proof π_4' is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{s_3, \omega_{6,2} \vdash A_2}{s_3, \omega_{6,2} \vdash (\neg control \vee (control \vee B)) \wedge [-]A_3}\,\nu_2}{s_3, \omega_{6,2} \vdash \neg control \wedge (control \vee B)}\,\wedge}{s_3, \omega_{6,2} \vdash control \wedge B}\,\vee}{\pi_{4,1,1}} \qquad \dfrac{s_3, \omega_{6,2} \vdash [-]A_3}{\pi_{4,1,2}}$$

where $A_3 \equiv \nu x.\{(s_1, true), (s_2, \omega_1), (s_3, \omega_{6,2})\}.(\neg control \vee (control \wedge B) \wedge [-]X)$.

2.3.1. Proof $\pi_{4,1,1}$ is as follows.

$$\dfrac{\dfrac{s_3, \omega_{6,2} \vdash control \wedge B}{s_3, \omega_{6,2} \vdash control}\,prop_1 \qquad \dfrac{s_3, \omega_{6,2} \vdash B}{\pi_5}}{}\,\wedge$$

Proof π_5 is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{s_3, \omega_{6,2} \vdash B}{s_3, \omega_{6,2} \wedge x = 0 \vdash C}\,reset}{s_3, \omega_{6,2} \wedge x \leq control - 1 \vdash C}\,thin}{s_3, \omega_2 \vdash C}\,disj \qquad \dfrac{s_3, \omega_3 \vdash C}{\pi_{5,2}}}{\pi_{5,1}}$$

It is obvious that proofs $\pi_{5,1}$ and $\pi_{5,2}$ are analogous to $\pi_{3,1}$ and $\pi_{3,2}$. We omit these proofs for sake of brevity.

2.3.2. Proof $\pi_{4,1,2}$ is as follows.

$$\dfrac{s_3, \omega_{6,2} \vdash [-]A_3}{\dfrac{s_2, \omega_9 \vdash A_3}{}\,\nu_1 \qquad \dfrac{s_3, \omega_{10} \vdash A_3}{}\,\nu_1 \qquad \dfrac{s_1, \omega_7 \vdash A_3}{}\,\nu_1 \qquad \dfrac{s_4, \omega_8 \vdash A_3}{\pi_6}}\,[]$$

where

$$\omega_9 \equiv 3 \leq control \leq 4$$
$$\omega_{10} \equiv 3 \leq control \leq 4 \wedge check - code = 1$$

Proof π_6 is obviously analogous to $\pi_{4,2}$.

6.5 Conclusions and related works

We have defined a technique for local model checking RTL_μ. The proof system checks whether a set of timed states of a timed graph satisfies an RTL_μ formula and exploits the technique of Winskel for fixpoints formulas. The logic is defined over the discrete time domain and is designed for synchronous systems, since next state is equal to next time. The proof system allows us to verify RTL_μ properties of a closed $\mathcal{TSP}$ process p by using its timed graphs semantics $\mathcal{G}(p)$. It is worth mentioning that we could also follow another approach: directly construct a finite quotient of $LTS(\mathcal{G}(p))$ and define a proof system that works on this model. The finite quotient can be obtained by taking the sets of timed states corresponding to minimal regions. However, our approach has the advantage that transitions are generated locally only if it is necessary for the proof. Therefore, it may prevent the construction of the whole quotient of $LTS(\mathcal{G}(p))$.

A real-time version of μ-calculus with respect to discrete time has been proposed by Emerson [29]. The main idea of this proposal is that of expressing bounded operators by counting the number of fixpoints iterations. More in details $p \models \mu(i)X.A(X)$ iff $p \in\| A \|^i (\emptyset)$, namely i iterations are sufficient for establishing $p \in \mu X.A(X)$. Hence, the $RTCTL$ property $EF_{\leq 3}P$ is expressed as $\mu(4)X.(P \vee [-]X)$. This logic is trivially subsumed by RTL_μ.

In the last few years also extensions of μ-calculus with respect to dense time have been proposed. However, the logic suffers of some intrinsic drawbacks due to the Zeno paradox. As we have discussed in chapter 5 it is necessary to consider only the computations with diverging time. Thus, a property such as " for all computations P eventually holds" must be verified over diverging computations. In the framework of μ-calculus an eventuality property is expressed as the iteration of the next modality. Therefore, we should have a next operator that does not force time to increase by more than an infinitesimal amount, but whose iteration forces time to advance beyond any bound. In this case, we could express the property above as $\mu X.(P \vee <> X)$ as in the untimed case. Unfortunately, in general diverging and non-diverging computations cannot be distinguished by a next operator. These problems have been pointed out in [45]. The syntax of the proposed logic is similar to ours. The main difference regards the *next* operator. Since in a dense-time framework, next state is not equal to next time, they propose a binary next time $A_1 <> A_2$, which can be seen as a single step until, namely A_1 is true and stays true until a transition is taken and this transition establishes A_2. The transition may be taken after any increment of time. The paper presents a translation of $TCTL$ into real-time μ-calculus, which is correct for a particular class of models for which inevitability corresponds to time bounded inevitability. The result exploits the equivalence between time-bounded properties and safety properties. For instance, $z.A\Diamond(q \wedge z \leq 5)$ can be expressed as $\neg z.((\neg q)E\mathcal{U}z > 5)$. The case of modalities such as $E\mathcal{U}$ does not present difficulties.

Such a translation is exploited for defining a global and symbolic algorithm

of model checking for a version of $TCTL$ [3] with freeze quantification instead of bounded operators. The problem adressed is that of checking whether a timed graph satisfies a formula. The algorithm is symbolic, since sets of states (regions) are represented as predicates over clocks and fixpoints are computed iteratively. The completeness of the algorithm is based on the observation that the limit of the approximation sequence is reached within a finite number of iterations, since the number of regions is finite. Note that this result does not hold if clock constraints with addition of variables are considered. It is important to stress that the procedure limits the state explosion problem of the algorithm proposed by [3] for CTL, that works on the whole region graph. A further improvement is obtained in [42], where the classical automata-theoretic approach is generalized to real-time for achieving an on-the-fly and space efficient algorithm for model checking $TCTL$.

A different extension of μ-calculus with respect to a dense time domain is presented in [82]. The main difference is that there are two kinds of next operators. One corresponds to time successor, while the other corresponds to transition successor. In this paper, a model checking algorithm is proposed for the alternation-free fragment of μ-calculus by assuming the system to be modeled as timed graph. The algorithm is local (on-the-fly) and exploits the graph techniques of [81, 11].

Chapter 7

A Compositional Proof System for $\mathcal{TSP}$

The proof system proposed in chapter 6 permits to prove RTL_μ properties of $\mathcal{TSP}$ processes by exploiting their timed graph semantics. This approach has the advantage to be local, but is not compositional. Therefore, it is not adequate to be used during the process development, since it is applicable only after the process has been completely specified. In this chapter we extend the compositional technique of chapter 4 to $\mathcal{TSP}$ processes. The main idea is that of replacing in the proof system of chapter 6 the rules [] and <> with a set of rules, that reduce the satisfaction problems $p, \omega \vdash < \alpha > A$ and $p, \omega \vdash [\alpha]A$ to satisfaction problems for the components of p. The rules depend on the structure of the process p and are very similar to the ones for the untimed case. The proof system is proved to be sound in general and complete for regular processes. It is indeed complete for arbitrary timed statecharts.

7.1 Introduction to the proof system

We explain the main ideas underlying the definition of rules for proving both $< \alpha > A$ and $[\alpha]A$ in a compositional way. Suppose to have the following proof in the proof system of chapter 6.

$$\dfrac{\dfrac{p, \omega_1 \vdash < \alpha > A}{p', reset(\omega_1, C_1, C_2) - 1 \vdash A}\ <>}{\pi}$$

By definition of <>, there exists $p \xmapsto{\alpha,\omega} p'$ such that

- $\models \forall(\omega_1 \supset \omega)$;
- $C_1 = C \cup (loc(p) \cap loc(p'))$ and $C_2 = loc(p') \setminus loc(p)$;

- $\models \forall(reset(\omega_1, C_1, C_2) - 1 \supset delay(p'))$.

Our goal is to replace rule <> by a set of rules, that allow us to deduce $p, \omega_1 \vdash < \alpha > A$ from $p', reset(\omega_1, C_1, C_2) - 1 \vdash A$ without referring to the transition $p \stackrel{\alpha,\omega}{\longmapsto} p'$ of $\mathcal{G}(p)$. Moreover, this set of rules must be sound, namely, if we deduce $p, \omega_1 \vdash < \alpha > A$ from $p'', \omega_1' \vdash A$ in a compositional way, then there exists $p \stackrel{\alpha,\omega}{\longmapsto} p'$, such that

- rule <> is applicable with respect to $p \stackrel{\alpha,\omega}{\longmapsto} p'$;
- $\omega_1' \equiv reset(\omega_1, C_1, C_2) - 1$ and $p' = p''$.

An analogous reasoning applies for the case of rule [].

Consider a basic process $p = \ell_1 :: \ell_2 :: b$, where b is not further decomposable. Since p is a basic process, we can deduce from the syntax of p the transitions $p \stackrel{\alpha,\omega}{\longmapsto} p'$. Therefore, it is sufficient to instantiate rule <> with these informations. Suppose $b = a.t$. By definition, there are two cases:

1. $\alpha = a : \epsilon$ and $\omega \equiv true$, $p' = \ell_1 :: t$;
2. $\alpha = b : \bar{\epsilon}$, with $b \in \#(a)$, and $\omega \equiv true$, $p' = p$.

We propose the following rules:

1. $$\frac{\ell_1 :: \ell_2 :: a.t, \omega_1 \vdash < a : \epsilon > A}{\ell_1 :: t, \omega_2 \vdash A}$$

   where $\omega_2 \equiv reset(\omega_1, C \cup \{\ell_1\}, loc(t)) - 1$ and $\models \forall(\omega_2 \supset delay(\ell_1 :: t))$. This rule corresponds to rule <>, where we know that $\omega \equiv true$ and $clock(\ell_1 :: \ell_2 :: a.t) \cap clock(\ell_1 :: t) = \{\ell_1\}$, while $clock(\ell_1 :: t) \setminus clock(\ell_1 :: \ell_2 :: a.t) = loc(t)$, by definition of well-formed processes.

2. $$\frac{\ell_1 :: \ell_2 :: a.t, \omega_1 \vdash < b : \bar{\epsilon} > A}{\ell_1 :: \ell_2 :: a.t, \omega_1 - 1 \vdash A}$$

   This rule corresponds to rule <>, where $p = p'$ and $reset(\omega_1, C \cup (clock(p) \cap clock(p')), clock(p') \setminus clock(p)) \equiv \omega_1$. Moreover $delay(p) \equiv true$.

Suppose $b = [m, M]a.t$. By definition, there are two cases:

1. $\alpha = a : \epsilon$ and $\omega \equiv m \leq \ell_2 \leq M$, $p' = \ell_1 :: t$;
2. $\alpha = b : \bar{\epsilon}$, with $b \in \#(a)$, and $\omega \equiv true$, $p' = p$.

We propose the following rules:

1. $$\frac{\ell_1 :: \ell_2 :: [m, M]a.t, \omega_1 \vdash < a : \epsilon > A}{\ell_1 :: t, \omega_2 \vdash A}$$

   where $\models \forall(\omega_1 \supset m \leq \ell_2 \leq M)$, $\omega_2 \equiv reset(\omega_1, C \cup \{\ell_1\}, loc(t)) - 1$ and $\models \forall(\omega_2 \supset delay(\ell_1 :: t))$.

   This rule is similar to the corresponding one for prefixing. However, in this case $\omega \equiv m \leq \ell_2 \leq M$. Therefore, we require the side condition $\models \forall(\omega_1 \supset m \leq \ell_2 \leq M)$.

2. $$\frac{\ell_1 :: \ell_2 :: [m, M]a.t, \omega_1 \vdash < b : \bar{\epsilon} > A}{\ell_1 :: \ell_2 :: [m, M]a.t, \omega_1 - 1 \vdash A}$$

   where $\models \forall(\omega_1 - 1 \supset \ell_2 \leq M)$. This rule is similar to the corresponding one for prefixing. However, in this case $delay(p) \equiv \ell_2 \leq M$. Therefore, we require the side condition $\models \forall(\omega_1 - 1 \supset \ell_2 \leq M)$.

Consider a process $p = \ell :: t \bigtriangledown p_1$. By definition of the semantics, we have $p \overset{\alpha,\omega}{\longmapsto} p'$ iff one of the following cases holds:

1. $\ell :: t \overset{\alpha,\omega}{\longmapsto} p'$ and $type(\alpha) = \epsilon$ (rule $\bigtriangledown_1$);
2. $p_1 \overset{\alpha,\omega}{\longmapsto} p_1'$ and $type(\alpha) = \epsilon$, $p' = \ell :: t \bigtriangledown p_1'$ (rule $\bigtriangledown_2$).
3. $p_1 \overset{\alpha,\omega_1}{\longmapsto} p_1$ and $type(\alpha) = \bar{\epsilon}$, $\ell :: t \overset{\alpha,\omega_2}{\longmapsto} \ell :: t$ and $p' = p$, $\omega \equiv \omega_1 \wedge \omega_2$ (rule $\bigtriangledown_3$).

Consider for instance

$t = \ell_1 :: [2, 4]a.t_1$
$p_1 = \ell_1 :: \ell_3 :: b.t_2$
$\omega_1 \equiv 3 \leq \ell_1 \leq 4 \wedge 1 \leq \ell_3 \leq 5.$

1. Consider $\ell :: t \overset{a:\epsilon, 2 \leq \ell_1 \leq 4}{\longmapsto} \ell :: t_1$ and suppose

   $$\cfrac{\cfrac{\ell :: t \bigtriangledown p_1, \omega_1 \vdash < a : \epsilon > A}{p', reset(\omega_1, C_1, C_2) - 1 \vdash A}\ <>}{\pi}$$

   where $p' = \ell :: t_1$.

   By repeating a reasoning similar to the one of the untimed case, we could think to deduce $\ell :: t \bigtriangledown p_1, \omega_1 \vdash < a : \epsilon > A$ from $\ell :: t, \omega_1 \vdash < a : \epsilon > A$. However, it may be the case that $\omega_1 \notin \Omega(C \cup loc(\ell :: t))$. Actually, since the process is well-formed $Loc(\ell :: t \bigtriangledown p_1) = Loc(\ell :: t) \cup Loc(p_1)$ and $Loc(\ell :: t) \cap Loc(p_1) = main(t)$. This drawback can be solved by considering the projection of ω_1 with respect to clocks $C \cup loc(\ell :: t)$. For instance, $pro_{C \cup loc(\ell :: t)}(\omega_1) \equiv 3 \leq \ell_1 \leq$

$4 \equiv \omega_2$. The idea is that the value of clock ℓ_3 regards process p_1 and does not regard process $\ell :: t$. We have a compositional proof as the following.

$$\dfrac{\dfrac{p, \omega_1 \vdash < a : \epsilon > A}{\ell :: t, \omega_2 \vdash < a : \epsilon > A}}{\ell :: t_1, reset(\omega_2, C_1', C_2') - 1 \vdash A}$$

where $C_1' = C \cup (loc(\ell :: t_1) \cap loc(\ell :: t))$ and $C_2' = loc(\ell :: t_1) \setminus loc(\ell :: t)$.

Since $Loc(p_1) \cap Loc(\ell :: t) = main(t) = \{\ell_1\}$, then $\ell_1 \notin loc(p')$ so that

$$reset(pro_{C \cup loc(\ell::t)}(\omega_1), C_1', C_2') \equiv reset(\omega_1, C_1, C_2).$$

We have obtained $p, \omega_1 \vdash < a : \epsilon > A$ from $p', reset(\omega_1, C_1, C_2) - 1 \vdash A$ in a compositional way. The soundness of this rule is based on similar considerations.

2. This case is similar to the previous one. However, the operator of $\bigtriangledown$ is not symmetric, since $p' = \ell :: t \bigtriangledown p_1'$. As in untimed case we must use an extended formula for recovering the whole structure of p'.

   Consider $p_1 \overset{b:\epsilon, true}{\longmapsto} \ell_1 :: t_2$ and suppose

   $$\dfrac{\dfrac{\ell :: t \bigtriangledown p_1 \vdash < b : \epsilon > A}{p', reset(\omega_1, C_1, C_2) - 1 \vdash A} <>}{\pi}$$

   where $p' = \ell :: t \bigtriangledown p_1'$ and $p_1' = \ell_1 :: t_2$.

   By repeating a reasoning similar to the one for the untimed case, we could think to deduce $p, \omega_1 \vdash < b : \epsilon > A$ from $p_1, pro_{loc(p_1) \cup C}(\omega_1) \vdash < b : \epsilon > (A)_{(\ell::t, \omega_2)\bigtriangledown}$, where $\omega_2 \in \Omega(loc(\ell :: t))$. Note that $pro_{loc(p_1) \cup C}(\omega_1) \equiv \omega_1$, since $\ell_1 \in loc(p_1)$. We would obtain the following proof.

   $$\dfrac{\dfrac{\dfrac{p, \omega_1 \vdash < b : \epsilon > A}{p_1, pro_{C \cup loc(p_1)}(\omega_1) \vdash < b : \epsilon > (A)_{(\ell::t, \omega_2)\bigtriangledown}}}{\ell_1 :: t_2, reset(pro_{C \cup loc(p_1)}(\omega_1), C_1', C_2') - 1 \vdash (A)_{(\ell::t, \omega_2)\bigtriangledown}}}{\ell :: t \bigtriangledown \ell_1 :: t_2, reset(pro_{C \cup loc(p_1)}(\omega_1), C_1', C_2') - 1 \wedge \omega_2 \vdash A} \; shift$$

   where $C_1' = C \cup (loc(p_1') \cap loc(p_1))$ and $C_2' = loc(p_1') \setminus loc(p_1)$.

   This reasoning is correct iff $reset(pro_{C \cup loc(p_1)}(\omega_1), C_1', C_2') - 1 \wedge \omega_2$ is equivalent to $reset(\omega_1, C_1, C_2) - 1$, so that $p, \omega_1 \vdash < b : \epsilon > A$ follows from $p', reset(\omega_1, C_1,$

$C_2)-1 \vdash A$ by $<>$. It is possible to find $\omega_2 \in \Omega(loc(\ell :: t))$ that fulfills these requirements?

We exploit the fact that that under some assumptions on the syntax of $\omega \in \Omega(C)$, $pro_{C_1}(\omega) \wedge pro_{C_2}(\omega) \equiv \omega$, for $C_1 \cap C_2 = \emptyset$ and $C_1 \cup C_2 = C$. Since the process is well-formed $C_1 = (C_1' \setminus main(t)) \cup C_1''$ and $C_2 = C_2'$, where $C_1'' = loc(\ell :: t)$. Moreover, $(C_1' \setminus main(t)) \cap C_1'' = \emptyset$. Therefore, if we consider $\omega_2 \equiv reset(\omega_1, C_1'', \emptyset)-1$ and $pro_{(loc(p_1)\setminus main(t))\cup C}(\omega_1)$ instead of $pro_{loc(p_1)\cup C}(\omega_1)$, we have $reset(pro_{C\cup(loc(p_1)\setminus main(t))}(\omega_1), C_1', C_2') - 1 \wedge \omega_2 \equiv reset(\omega_2, C_1, C_2) - 1$.

It is quite obvious that the fact of considering $pro_{(loc(p_1)\setminus main(t))\cup C}(\omega_1)$ instead of $pro_{loc(p_1)\cup C}(\omega_1)$ is equivalent for process p_1, since the clock ℓ_1 does not constrain the behaviour of process p_1. In this case we have $\omega_2 \equiv 4 \leq \ell_1 \leq 5$ and $pro_{(loc(p_1)\setminus main(t))\cup C}(\omega_1) \equiv 1 \leq \ell_3 \leq 5$.

However, this kind of rule is not in general sound. Suppose $\omega_1 \equiv \ell_1 = 4$. By definition, action a must be performed, since it cannot be delayed. Therefore, the previous proof is not correct. The problem is that, $\not\models \forall(reset(\omega_1, C_1, C_2) - 1 \supset delay(p'))$ so that we cannot apply rule $<>$ for deducing $p, \omega_1 \vdash < b : \epsilon > A$. In order to have a sound rule it is sufficient to require in addition the side condition $\models \forall(\omega_2 \supset delay(\ell :: t))$.

3. It is obvious that $\omega_i \equiv true$, for each $i \in \{1, 2\}$ so that $reset(\omega_1, C_1, C_2) - 1 \equiv \omega_1 - 1$. This allows us to define a rule very similar to the one for the untimed case. For $type(\alpha) = \bar{\epsilon}$.

$$\frac{p, \omega_1 \vdash < \alpha > A}{\begin{array}{l} p_1, pro_{C\cup(loc(p_1)\setminus main(t))}(\omega_1) \vdash < \alpha > true \\ \ell :: t, pro_{C\cup loc(\ell::t)}(\omega_1) \vdash < \alpha > true \\ p, \omega_1 - 1 \vdash A \end{array}}$$

Actually, $\omega_1 - 1 \equiv pro_{C\cup(loc(p_1)\setminus main(t))}(\omega_1) - 1 \wedge pro_{C\cup loc(\ell::t)}(\omega_1) - 1$.

Note that the soundness of this rule is guaranteed by the fact that if

$$\begin{array}{l} p_1, pro_{C\cup(loc(p_1)\setminus main(t))}(\omega_1) \models < \alpha > true \\ \ell :: t, pro_{C\cup loc(\ell::t)}(\omega_1) \models < \alpha > true \end{array}$$

then $\models \forall(pro_{C\cup(loc(p_1)\setminus main(t))}(\omega_1) - 1 \supset delay(p_1))$ and $\models \forall(pro_{C\cup loc(\ell::t)}(\omega_1) - 1 \supset delay(\ell :: t))$ and $delay(\ell :: t \bigtriangledown p_1) \equiv delay(\ell :: t) \wedge delay(p_1)$.

Consider a process $p = \ell :: p_1 \times p_2$. As in the untimed case the rules depend on the structure of p_2. By definition, $p \stackrel{\alpha,\omega}{\longmapsto} p'$ iff, by rule par, $p_1 \stackrel{\alpha_1,\omega_1}{\longmapsto} p_1'$ and $p_2 \stackrel{\alpha_2,\omega_2}{\longmapsto} p_2'$, where $\alpha \in \alpha_1 \times \alpha_2$, $\omega \equiv \omega_1 \wedge \omega_2$ and $p' = \ell :: p_1' \times p_2'$.

Suppose $p_2 = \ell_1 :: \ell_2 :: a.t.$ Since p_2 is a basic process, we can deduce the transitions $p_2 \stackrel{\alpha_2,\omega_2}{\longmapsto} p_2'$ from its syntax. Consider a transition $p \stackrel{\alpha,\omega}{\longmapsto} p'$, that is

obtained from $p_1 \stackrel{\alpha_1,\omega_1}{\longmapsto} p_1'$, where $\alpha \in \alpha_1 \times (a:\epsilon)$, $\omega \equiv \omega_1$, $p_2' = \ell_1 :: t$. Moreover suppose to have a proof

$$\dfrac{\dfrac{p, \omega_1 \vdash < \alpha > A}{p', reset(\omega_1, C_1, C_2) - 1 \vdash A}<>}{\pi}$$

By repeating a reasoning similar to the one for the untimed case, we can deduce $p, \omega_1 \vdash < \alpha > A$ from $p_1, pro_{C \cup loc(p_1)}(\omega_1) \vdash < \alpha_1 > (A)_{\ell \times (\ell_1 :: t, \omega_2)}$, where $\omega_2 \in \Omega(loc(\ell_1 :: t))$ is $reset(\omega_1, C_{2,1}, C_{2,2}) - 1$, for $C_{2,1} = loc(p_2) \cap loc(\ell_1 :: t) = \{\ell_1\}$, $C_{2,2} = loc(\ell_1 :: t) \setminus loc(p_2) = loc(t)$.

We obtain a proof as the following, where $\omega_3 \equiv pro_{C \cup loc(p_1)}(\omega_1)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p, \omega_1 \vdash < \alpha > A}{p_1, \omega_3 \vdash < \alpha_1 > (A)_{\ell \times (\ell_1 :: t, \omega_2)}}}{\cdots}}{p_1', reset(\omega_3, C_{1,1}, C_{1,2}) - 1 \vdash (A)_{\ell \times (\ell_1 :: t, \omega_2)}}}{\ell :: p_1' \times \ell_1 :: t, reset(\omega_3, C_{1,1}, C_{1,2}) - 1 \wedge \omega_2 \vdash A}\, shift}{\pi}$$

where $C_{1,1} = C \cup (loc(p_1) \cap loc(p_1'))$, $C_{1,2} = loc(p_1') \setminus loc(p_1)$.

Since the process is well-formed, then $Loc(p_1) \cap Loc(p_2) = \emptyset$ so that $C_{1,1} \cap C_{2,1} = \emptyset$ and $C_{1,2} \cap C_{2,2} = \emptyset$. Moreover, $C_1 = C_{1,1} \cup C_{2,1}$ and $C_2 = C_{1,2} \cup C_{2,2}$. Therefore, $reset(pro_{C \cup loc(p_1)}(\omega_1), C_{1,1}, C_{1,2}) - 1 \wedge \omega_2 \equiv reset(\omega_1, C_1, C_2) - 1$ and we deduce $p, \omega_1 \vdash < \alpha > A$ from $p', reset(\omega_1, C_1, C_2) - 1 \vdash A$.

Note that in order to ensure the soundness of this kind of rule, we must in addition require $\models \forall(\omega_2 \supset delay(\ell_1 :: t))$ so that rule $<>$ is indeed applicable.

7.2 The proof system

In this section we introduce the compositional rules for establishing $< \alpha > A$ and $[\alpha]A$. We would like to have judgments of the proof system such as $p, \omega \vdash A$, where $p \in \mathcal{CTSP}$ over Loc and Act, $\omega \in \Omega(loc(p) \cup C)$ and A is a closed RTL_μ formula over propositional symbols Loc, actions $\mathcal{S}(Act)$ and specification clocks C, such that $C \cap Loc = \emptyset$.

However, as we have explained in section 7.1 it is not possible to consider arbitrary clock constraints. Therefore, we consider judgments $p, \omega \vdash A$, where $\omega \in \Omega(loc(p) \cup C)$ such that, if $x + m \mathcal{R} y + m \in \omega$, then $x, y \in C$.

Note that this is not a limitation, since the clock constraints of $\mathcal{G}(p)$ satisfy this property so that the results of chapter 6 are not affected. This assumption permits to exploit the following properties.

We extend the definition of $\cup$ to arbitrary assignment. Let $\pi_1 : C_1 \to N$ and $\pi_2 : C_2 \to N$. We define $\pi_1 \cup \pi_2 : C_1 \cup C_2 \to N$, such that $\pi_1 \cup \pi_2(x) = \pi_i(x)$, if $x \in C_i$, where $\pi_1(x) = \pi_2(x)$, for each $x \in C_1 \cap C_2$.

Proposition 7.2.1 *Let $\pi : C \to N$ and $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C$. Let $C_{i,1}, C_{i,2}$, such that $C_{j,i} \cup C_{k,i} = C_i$, for each $i, j, k \in \{1, 2\}$ with $j \neq k$. We have that*

$$reset(\pi, C_1, C_2) = reset(\pi, C_{1,1}, C_{1,2}) \cup reset(\pi, C_{2,1}, C_{2,2}).$$

Proposition 7.2.2 *Let $C_1 \cap C_2 = \emptyset$, $C_1 \subseteq C$ and $C_{i,1}, C_{i,2}$, such that $C_{j,i} \cup C_{k,i} = C_i$, for each $i, j, k \in \{1, 2\}$ with $j \neq k$ and $C_{1,i} \cap C_{2,i} = \emptyset$. Let $\omega \in \Omega(C)$ such that, for each $x + n\mathcal{R}y + m \in \omega$, either $x, y \in C_{1,1}$ or $x, y \in C_{2,1}$. We have that*

$$reset(\omega, C_1, C_2) \equiv reset(\omega, C_{1,1}, C_{1,2}) \wedge reset(\omega, C_{2,1}, C_{2,2}).$$

Let $p \in \mathcal{CTSP}$ be a closed process over Act and Loc and let A be a closed RTL_μ formula over $\mathcal{G}(p)$, Loc and C. We say that A is an RTL_μ formula over Act, Loc and C and we use the notation $p, \omega \models A$ to denote $p, \omega \models_{\mathcal{M}(\mathcal{G}(p),\gamma,C)} A$, where γ is the valuation $\gamma : Loc \to \mathcal{CTSP}$, such that $p \in \gamma(P)$ iff $P \in loc(p)$.

We present the rules for establishing $< \alpha > A$ and $[\alpha]A$ for each process constructor.

7.2.1 Rules for *nil*

$$\frac{\ell :: \ell_1 :: nil, \omega_1 \vdash < a : \bar{\epsilon} > A}{\ell :: \ell_1 :: nil, \omega_1 - 1 \vdash A} <> nil \quad a \in \#(*)$$

$$\frac{\ell :: \ell_1 :: nil, \omega_1 \vdash [a : \epsilon]A}{} [] nil_1$$

$$\frac{\ell :: \ell_1 :: nil, \omega_1 \vdash [a : \bar{\epsilon}]A}{\ell :: \ell_1 :: nil, \omega_1 - 1 \vdash A} [] nil_2 \quad a \in \#(*)$$

$$\frac{\ell :: \ell_1 :: nil, \omega_1 \vdash [a : \bar{\epsilon}]A}{} [] nil_3 \quad a \notin \#(*)$$

These rules are quite obvious. Actually, by definition of the timed graph semantics, $\ell :: \ell_1 :: nil \xrightarrow{\alpha,\omega} p$ iff $\alpha = a : \bar{\epsilon}$ and $a \in \#(*)$ and $\omega \equiv true$ and $p = \ell :: \ell_1 :: nil$.

As explained in section 7.1, the rules are trivially obtained by properly instantiating rules $<>$ and $[]$ of chapter 6, where $clock(p) = loc(p)$ and $delay(\ell :: \ell_1 :: nil) \equiv true$. Moreover, note that for each transition $p \xrightarrow{\alpha,\omega} p'$, if $type(\alpha) = \bar{\epsilon}$, then $p' = p$ and $\omega \equiv true$.

7.2.2 Rules for prefixing

Let $\omega_2 \equiv reset(\omega_1, C_1, C_2) - 1$ for $C_1 = \{\ell\} \cup C$ and $C_2 = loc(t)$.

$$\frac{\ell :: \ell_1 :: a.t, \omega_1 \vdash < a : \epsilon > A}{\ell :: t, \omega_2 \vdash A} <> \epsilon \quad \models \forall(\omega_2 \supset delay(\ell :: t))$$

$$\frac{\ell :: \ell_1 :: a_1.t, \omega_1 \vdash < a_2 : \bar{\epsilon} > A}{\ell :: \ell_1 :: a_1.t, \omega_1 - 1 \vdash A} <> \bar{\epsilon} \quad a_2 \in \#(a_1)$$

$$\frac{\ell :: \ell_1 :: a.t, \omega_1 \vdash [a : \epsilon]A}{\ell :: t, \omega_2 \vdash A} []\epsilon_1$$

$$\frac{\ell :: \ell_1 :: a_1.t, \omega_1 \vdash [a_2 : \bar{\epsilon}]A}{\ell :: \ell_1 :: a_1.t, \omega_1 - 1 \vdash A} []\bar{\epsilon}_1 \quad a_2 \in \#(a_1)$$

$$\frac{\ell :: \ell_1 :: a_1.t, \omega_1 \vdash [a_2 : \epsilon]A}{} []\epsilon_2 \quad a_1 \neq a_2$$

$$\frac{\ell :: \ell_1 :: a_1.t, \omega_1 \vdash [a_2 : \bar{\epsilon}]A}{} []\bar{\epsilon}_2 \quad a_2 \notin \#(a_1)$$

As explained in section 7.1, these rules are based on the observation that $\ell :: \ell_1 :: a_1.t \overset{\alpha,\omega}{\mapsto} p$ iff one of the following cases holds:

1. $\alpha = a_1 : \epsilon$, $\omega \equiv true$ and $p = \ell :: t$;
2. $\alpha = a_2 : \bar{\epsilon}$, $\omega \equiv true$ and $p = \ell :: \ell_1 :: a_1.t$ for $a_2 \in \#(a_1)$.

Moreover, $delay(\ell :: \ell_1 :: a_1.t) \equiv true$ and $clock(\ell :: \ell_1 :: a_1.t) \cap clock(\ell :: t) = \{\ell\}$, $clock(\ell :: t) \setminus clock(\ell :: \ell_1 :: a_1.t) = loc(t)$, by definition of well-formed process.

7.2.3 Rules for timed prefixing

Let $\omega_2 \equiv reset(\omega_1, C_1, C_2) - 1$ and $\omega_3 \equiv reset(\omega_1 \wedge m \leq \ell_1 \leq M, C_1, C_2) - 1$ for $C_1 = \{\ell\} \cup C$ and $C_2 = loc(t)$.

$$\frac{\ell :: \ell_1 :: [m, M]a.t, \omega_1 \vdash < a : \epsilon > A}{\ell :: t, \omega_2 \vdash A} \tau <> \epsilon \quad \begin{array}{l} \models \forall(\omega_1 \supset m \leq \ell_1 \leq M) \\ \models \forall(\omega_2 \supset delay(\ell :: t)) \end{array}$$

$$\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash < a_2 : \bar{\epsilon} > A}{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 - 1 \vdash A} \tau <> \bar{\epsilon} \quad \begin{array}{l} a_2 \in \#(*) \\ \models \forall(\omega_1 - 1 \supset \ell_1 \leq M) \end{array}$$

$$\frac{\ell :: \ell_1 :: [m, M]a.t, \omega_1 \vdash [a : \epsilon]A}{\ell :: t, \omega_3 \vdash A} \tau[]\epsilon_1 \quad \not\models \forall(\omega_1 \wedge m \leq \ell_1 \leq M \supset false)$$

$$\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash [a_2 : \bar{\epsilon}]A}{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 - 1 \vdash A}\,\tau[]\bar{\epsilon}_1 \quad a_2 \in \#(*)$$

$$\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash [a_2 : \epsilon]A}{}\,\tau[]\epsilon_2 \quad a_1 \neq a_2$$

$$\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash [a_2 : \bar{\epsilon}]A}{}\,\tau[]\bar{\epsilon}_2 \quad a_2 \notin \#(*)$$

$$\frac{\ell :: \ell_1 :: [m, M]a_1.t, \omega_1 \vdash [a_1 : \epsilon]A}{}\,\tau[]\epsilon_3 \quad \models \forall(\omega_1 \wedge m \leq \ell_1 \leq M \supset false)$$

Rules for timed prefixing are similar to the ones for prefixing. However, in this case $delay(\ell :: \ell_1 :: [m, M]a_1.t) \equiv \ell_1 \leq M$ and the action a_1 can be performed iff $m \leq \ell_1 \leq M$.

Therefore, in rule $\tau <> \epsilon$, we require $\models \forall(\omega_1 \supset m \leq \ell_1 \leq M)$ and in rule $\tau <> \bar{\epsilon}$ we require $\models \forall(\omega_1 - 1 \supset \ell_1 \leq M)$. Moreover, in $\tau[]\epsilon_1$, we consider $\omega_3 \equiv reset(\omega_1 \wedge m \leq \ell_1 \leq M, C_1, C_2) - 1$. We have also an additional rule: $\tau[]\epsilon_3$. The underlying idea is that, if $\omega_1 \wedge m \leq \ell_1 \leq M$ is not satisfiable, then the action a_1 cannot be performed.

7.2.4 Rules for choice

Rules for *choice* are similar to the ones for the untimed case. Actually, there are two sets of rules: rules applicable when the type of the action is non-idle and rules applicable when the type of the action is idle. Moreover, we exploit the fact that $delay(\ell :: \ell_1 :: b_1 + b_2) \equiv delay(\ell :: \ell_1 :: b_1) \wedge delay(\ell :: \ell_1 :: b_2)$ and that $clock(\ell :: \ell_1 :: b_1 + b_2) = clock(\ell :: \ell_1 :: b_i)$, for each $i \in \{1, 2\}$.

1. Rules applicable iff $type(\alpha) = \epsilon$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\ell :: \ell_1 :: b_i, \omega_1 \vdash < \alpha > A} + <> \epsilon$$

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\{\ell :: \ell_1 :: b_i, \omega_1 \vdash [\alpha]A\}_{i \in \{1,2\}}} + []\epsilon$$

2. Rules applicable iff $type(\alpha) = \bar{\epsilon}$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\{\ell :: \ell_1 :: b_i, \omega_1 \vdash < \alpha > true\}_{i \in \{1,2\}} \quad \ell :: \ell_1 :: b_1 + b_2, \omega_1 - 1 \vdash A} + <> \bar{\epsilon}$$

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\ell :: \ell_1 :: b_1 + b_2, \omega_1 - 1 \vdash A} + [] \bar{\epsilon}_1$$

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\ell :: \ell_1 :: b_i, \omega_1 \vdash [\alpha] false} + [] \bar{\epsilon}_2$$

The rules are based on the following properties:

1. if $type(\alpha) = \epsilon$, then $\ell :: \ell_1 :: b_1 + b_2 \stackrel{\alpha,\omega}{\longmapsto} \ell :: t$ iff, by $choice_1$, $\ell :: \ell_1 :: b_i \stackrel{\alpha,\omega}{\longmapsto} \ell :: t$, for some $i \in \{1, 2\}$.

2. if $type(\alpha) = \epsilon$, then $\ell :: \ell_1 :: b_1 + b_2 \stackrel{\alpha,\omega}{\longmapsto} \ell :: t$ iff, by $choice_2$, $\ell :: \ell_1 :: b_i \stackrel{\alpha,\omega_i}{\longmapsto} \ell :: \ell_1 :: b_i$, for each $i \in \{1, 2\}$ and $t = \ell_1 :: b_1 + b_2$, $\omega \equiv \omega_1 \wedge \omega_2$. Moreover, $\omega_i \equiv true$.

7.2.5 Rules for refinement

Rules for refinement are similar to the ones for choice. However, in this case $clock(\ell :: t \bigtriangledown p) = clock(\ell :: t) \cup clock(p)$ and $clock(p) \cap clock(\ell :: t) = main(t)$, by definition of well-formed process. Therefore, we consider the projections with respect to $clock(p) \setminus main(t)$ and $clock(\ell :: t)$.

Since rules $\bigtriangledown_1$ and $\bigtriangledown_2$ are not symmetric, we use extended formulas as in the untimed case. We define extended formulas $(A)_{(\ell::t,\omega_2)\bigtriangledown}$ for $\omega_2 \in \Omega(clock(\ell :: t))$ with the following semantics

$$\| (A)_{(\ell::t,\omega_2)\bigtriangledown} \| = \{(p, \pi) \mid (\ell :: t \bigtriangledown p, \pi \cup \pi_2) \in \| A \|, \forall \pi_2 \models \omega_2\}$$

The shift rule is defined as follows.

$$\frac{p, \omega_1 \vdash (A)_{(\ell::t,\omega_2)\bigtriangledown)}}{\ell :: t \bigtriangledown p, \omega_1 \wedge \omega_2 \vdash A}$$

Let $\omega_2 \equiv pro_{loc(\ell::t)}(\omega_1) - 1$, $\omega_p \equiv pro_{C \cup (loc(p) \setminus main(t))}(\omega_1)$ and $\omega_t \equiv pro_{C \cup loc(\ell::t)}(\omega_1)$.

1. Rules applicable iff $type(\alpha) = \epsilon$.

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: t, \omega_t \vdash < \alpha > A} \bigtriangledown <> \epsilon_1$$

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{p, \omega_p \vdash < \alpha > (A)_{(\ell::t,\omega_2)\bigtriangledown}} \bigtriangledown <> \epsilon_2 \qquad \models \forall(\omega_2 \supset delay(\ell :: t))$$

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: t, \omega_t \vdash [\alpha]A \quad p, \omega_p \vdash [\alpha](A)_{(\ell::t,\omega_2)\bigtriangledown}} \bigtriangledown [] \epsilon$$

2. Rules applicable iff $type(\alpha) = \bar{\epsilon}$.

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: t, \omega_t \vdash < \alpha > true \quad p, \omega_p \vdash < \alpha > true \quad \ell :: t \bigtriangledown p, \omega_1 - 1 \vdash A} \bigtriangledown <> \bar{\epsilon}$$

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: t \bigtriangledown p, \omega_1 - 1 \vdash A} \bigtriangledown [] \bar{\epsilon}_1$$

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: t, \omega_t \vdash [\alpha] false} \bigtriangledown [] \bar{\epsilon}_2$$

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{p, \omega_p \vdash [\alpha] false} \bigtriangledown [] \bar{\epsilon}_3$$

These rules are based on the observation that $\ell :: t \bigtriangledown p \stackrel{\alpha,\omega}{\longmapsto} p'$ iff one of the following cases holds:

1. $\ell :: t \stackrel{\alpha,\omega}{\longmapsto} p'$ and $type(\alpha) = \epsilon$ (rule $\bigtriangledown_1$). Since $loc(p') \cap loc(p) = \emptyset$, then $reset(\omega_1, C \cup (loc(\ell :: t \bigtriangledown p) \cap loc(p')), loc(p') \setminus loc(\ell :: t \bigtriangledown p)) \equiv reset(\omega_t, C \cup (loc(\ell :: t) \cap loc(p')), loc(p') \setminus loc(\ell :: t))$.

2. $p \stackrel{\alpha,\omega}{\longmapsto} p'_1$ and $type(\alpha) = \epsilon$ and $p' = \ell :: t \bigtriangledown p'_1$ (rule $\bigtriangledown_2$).

   We use in rules $\bigtriangledown <> \epsilon_2$ and $\bigtriangledown[]\epsilon$, the extended formula $(A)_{(\ell::t,\omega_2)}$. Since $loc(p'_1) \cap loc(p) = main(t)$ and $main(t) \notin loc(p'_1) \setminus loc(p)$, then $reset(\omega_1, C \cup (loc(\ell :: t \bigtriangledown p) \cap loc(p')), loc(p') \setminus loc(\ell :: t \bigtriangledown p)) \equiv reset(\omega_p, C \cup (loc(p) \cap loc(p'_1)), loc(p'_1) \setminus loc(p)) \wedge reset(\omega_1, loc(\ell :: t), \emptyset)$, by proposition 7.2.2. Note that we require in $\bigtriangledown <> \epsilon_2$, $\models \forall(\omega_2 \supset delay(\ell :: t))$.

3. $type(\alpha) = \bar{\epsilon}$ and both $\ell :: t \stackrel{\alpha,\omega_1}{\longmapsto} \ell :: t$ and $p \stackrel{\alpha,\omega_2}{\longmapsto} p$ and $p' = \ell :: t \bigtriangledown p$, $\omega \equiv \omega_1 \wedge \omega_2$ (Rule $\bigtriangledown_3$). Moreover, $\omega_i \equiv true$.

7.2.6 Rules for recursion

Rules for recursion are similar to the ones for the untimed case. Actually, $delay(\ell :: rec\ x.t) = delay(\ell :: t[rec\ x.t/x])$ and $clock((\ell :: rec\ x.t) = clock(\ell :: t[rec\ x.t/x])$.

$$\frac{\ell :: rec\ x.t, \omega_1 \vdash [\alpha]A}{\ell :: t[rec\ x.t/x], \omega_1 \vdash [\alpha]A} rec[] \qquad \alpha \in \mathcal{S}(Act)$$

$$\frac{\ell :: rec\ x.t, \omega_1 \vdash < \alpha > A}{\ell :: t[rec\ x.t/x], \omega_1 \vdash < \alpha > A} rec <> \quad \alpha \in \mathcal{S}(Act)$$

7.2.7 Rules for parallel composition

Rules for parallel composition are similar to the ones for the untimed case. It is obvious that also in this case we must consider extended actions. The semantics of formulas with extended actions requires a slight modification. In particular, we must adapt the definition of *consistency*.

Assume $\alpha \in \mathcal{E}(Act)$. We say that $(p,\pi) \stackrel{\alpha'}{\mapsto} (p',\pi')$ is consistent with α iff it has been obtained from $p \stackrel{\alpha',\omega}{\longmapsto} p'$ and $p \stackrel{\alpha',\omega}{\longmapsto} p'$ is consistent with α, i.e.

1. $\alpha = a:\tau$ and $\alpha' = a:\tau'$, where $type(\alpha') = type^*(\alpha)$;
2. If $\alpha = a:\tau_1 \times \tau_2 \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$, then $p \stackrel{\alpha',\omega}{\longmapsto} p'$ has been been obtained by rule *par* from $p_i \stackrel{\alpha_i'}{\longmapsto} p_i'$, such that $p = \ell :: p_1 \times p_2$, $p' = \ell :: p_1' \times p_2'$, $\alpha' \in \alpha_1' \times \alpha_2'$, and $p_i \stackrel{\alpha_i'}{\longmapsto} p_i'$ is consistent with α_i, where $\alpha_i = a_i:\tau_i$ and $\alpha_i' = a_i:\tau_i'$.

Expansion rules

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times p_2, \omega_1 \vdash < \beta > A}\,\mathcal{E}\,<> \qquad \beta \in \mathcal{E}(\alpha)$$

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash [\alpha]A}{\{\ell :: p_1 \times p_2, \omega_1 \vdash [\beta_i]A\}_{i\in\{1,n\}}}\,\mathcal{E}[] \qquad \{\beta_1,\ldots,\beta_n\} = \mathcal{E}(\alpha)$$

Rules for *nil*

The rules are similar to the ones for the untimed case. We define extended formulas $(A)_{\ell\times(p_2,\omega_2)}$ with $\omega_2 \in \Omega(clock(p_2))$ with the following semantics

$$\| (A)_{\ell\times(p_2,\omega_2)} \| = \{(p_1,\pi) \mid (\ell :: p_1 \times p_2, \pi \cup \pi_2) \in \| A \|, \forall \pi_2 \models \omega_2\}$$

The shift rule is defined as

$$\frac{p_1, \omega_1 \vdash (A)_{\ell\times(p_2,\omega_2)}}{\ell :: p_1 \times p_2, \omega_1 \wedge \omega_2 \vdash A}$$

Let $p_2 = \ell_1 :: \ell_2 :: nil$ and $\omega_2 \equiv pro_{\{\ell_1,\ell_2\}}(\omega_1) - 1$.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, pro_{C\cup loc(p_1)}(\omega_1) \vdash < \beta > (A)_{\ell\times(p_2,\omega_2)}} \times <> nil \qquad \begin{array}{l}\alpha \in \beta \otimes (b:\bar{\epsilon}) \\ b \in \#(*)\end{array}$$

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash [\alpha]A}{\{p_1, pro_{C\cup loc(p_1)}(\omega_1) \vdash [\beta_i](A)_{\ell\times(p_2,\omega_2)}\}_{i\in\{1,k\}}} \times [] nil \qquad \begin{array}{l}\{\beta_i\}_{i\in\{1,k\}} = \\ \{\beta_i \mid \alpha \in \beta_i \otimes (b:\bar{\epsilon}) \\ b \in \#(*)\end{array}$$

These rules are based on the fact that $\ell :: p_1 \times p_2 \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times p_2'$ iff, by rule *par*, $\ell :: p_1 \stackrel{\alpha_1,\omega}{\longmapsto} p_1'$ and $\alpha \in \alpha_1 \times (b : \bar{\epsilon})$, for $b \in \#(*)$, and $p_2' = p_2$. Moreover, $loc(p_2) = \{\ell_1, \ell_2\}$ and $delay(p_2) \equiv true$.

Note that, since the process is well-formed, $Loc(p_1) \cap Loc(p_2) = \emptyset$ and $loc(p_i') \cap loc(p_j') = \emptyset$ for each $i \neq j$. By proposition 7.2.2

$$reset(\omega_1, C_1, C_2) \equiv reset(\omega_1, C_{1,1}, C_{1,2}) \wedge reset(\omega_1, C_{2,1}, C_{2,2})$$

where

$$\begin{array}{l} C_1 = C \cup (loc(\ell :: p_1 \times p_2) \cap loc((\ell :: p_1' \times p_2')) \\ C_2 = loc((\ell :: p_1' \times p_2') \setminus loc(\ell :: p_1 \times p_2) \\ C_{i,1} = C \cup (loc(p_i) \cap loc(p_i')) \\ C_{i,2} = loc(p_i') \setminus loc(p_i) \end{array}$$

Since $p_2 = p_2'$, then $reset(\omega_1, C_{2,1}, C_{2,2}) \equiv pro_{\{\ell_1,\ell_2\}}(\omega_1)$.

Rules for prefixing

Let $p_2 = \ell_1 :: \ell_2 :: a.t$ and $\omega_2 \equiv reset(\omega_1, \{\ell_1\}, loc(t)) - 1$, $\omega_3 \equiv pro_{loc(p_2)}(\omega_1) - 1$.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, pro_{C \cup loc(p_1)}(\omega_1) \vdash < \beta > (A)_{\ell \times (\ell_1 :: t, \omega_2)}} \times <>_1 \quad \begin{array}{l} \alpha \in \beta \otimes (a : \epsilon) \\ \models \forall(\omega_2 \supset delay(\ell_1 :: t)) \end{array}$$

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, pro_{C \cup loc(p_1)}(\omega_1) \vdash < \beta > (A)_{\ell \times (p_2, \omega_3)}} \times <>_2 \quad \begin{array}{l} \alpha \in \beta \times (b : \bar{\epsilon}) \\ b \in \#(a) \end{array}$$

Let

$$\begin{array}{l} \{\alpha_i\}_{i \in \{1,n\}} = \{\alpha_i \mid \alpha \in \alpha_i \otimes (a : \epsilon)\} \\ \{\beta_i\}_{i \in \{1,k\}} = \{\beta_i \mid \alpha \in \beta_i \otimes (b : \bar{\epsilon}), b \in \#(a)\} \end{array}$$

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash [\alpha]A}{\begin{array}{l} \{p_1, pro_{C \cup loc(p_1)}(\omega_1) \vdash [\alpha_i](A)_{\ell \times (\ell_1 :: t, \omega_2)}\}_{i \in \{1,n\}} \\ \{p_1, pro_{C \cup loc(p_1)}(\omega_1) \vdash [\beta_i](A)_{\ell \times (p_2, \omega_3)}\}_{i \in \{1,k\}} \end{array}} \times []$$

These rules are based on the fact that $\ell :: p_1 \times p_2 \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times p_2'$ iff, by rule *par*, one of the following cases hold:

1. $\ell :: p_1 \stackrel{\alpha_1,\omega}{\longmapsto} p_1'$ and $\alpha \in \alpha_1 \times (b : \bar{\epsilon})$, for $b \in \#(a)$ and $p_2' = p_2$;

2. $\ell :: p_1 \stackrel{\alpha_1,\omega}{\longmapsto} p_1'$ and $\alpha \in \alpha_1 \times (a : \epsilon)$ and $p_2' = \ell_1 :: t$.

Moreover, $loc(p_2) = \{\ell_1, \ell_2\}$ and $delay(p_2) \equiv true$.

Note that, since the process is well-formed, $Loc(p_1) \cap Loc(p_2) = \emptyset$ and $loc(p'_i) \cap loc(p'_j) = \emptyset$, for each $i \neq j$. By proposition 7.2.2

$$reset(\omega_1, C_1, C_2) \equiv reset(\omega_1, C_{1,1}, C_{1,2}) \wedge reset(\omega_1, C_{2,1}, C_{2,2})$$

where

$$C_1 = C \cup (loc(\ell :: p_1 \times p_2) \cap loc((\ell :: p'_1 \times p'_2))$$
$$C_2 = loc((\ell :: p'_1 \times p'_2) \setminus loc(\ell :: p_1 \times p_2)$$
$$C_{i,1} = C \cup (loc(p_i) \cap loc(p'_i))$$
$$C_{i,2} = loc(p'_i) \setminus loc(p_i)$$

If $p'_2 = p_2$, then $reset(\omega_1, C_{2,1}, C_{2,2}) \equiv pro_{\{\ell_1,\ell_2\}}(\omega_1)$ and, if $p'_2 = \ell_1 :: t$, then $reset(\omega_1, C_{2,1}, C_{2,2}) \equiv reset(\omega_1, \{\ell_1\}, loc(t))$.

Rules for timed prefixing

Let $p_2 = \ell_1 :: \ell_2 :: [m, M]a.t$, $\omega_2 \equiv reset(\omega_1, \{\ell_1\}, loc(t)) - 1$, $\omega_3 \equiv reset(\omega_1 \wedge m \leq \ell_2 \leq M, \{\ell_1\}, loc(t)) - 1$, $\omega_4 \equiv pro_{loc(p_2)}(\omega_1) - 1$ and $\omega_{p_1} \equiv pro_{C \cup loc(p_1)}(\omega_1)$.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, \omega_{p_1} \vdash < \beta > (A)_{\ell \times (\ell_1 :: t, \omega_2)}} \tau\times <>_1 \qquad \begin{array}{l} \alpha \in \beta \otimes (a : \epsilon) \\ \models \forall(\omega_1 \supset m \leq \ell_2 \leq M) \\ \models \forall(\omega_2 \supset delay(\ell_1 :: t)) \end{array}$$

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, \omega_{p_1} \vdash < \beta > (A)_{\ell \times (p_2, \omega_4)}} \tau\times <>_2 \qquad \begin{array}{l} \alpha \in \beta \times (b : \bar{\epsilon}) \\ b \in \#(*) \\ \models \forall(\omega_4 \supset \ell_2 \leq M) \end{array}$$

Let

$$\{\alpha_i\}_{i \in \{1,n\}} = \{\alpha_i \mid \alpha \in \alpha_i \otimes (a : \epsilon)\}$$
$$\{\beta_i\}_{i \in \{1,k\}} = \{\beta_i \mid \alpha \in \beta_i \otimes (b : \bar{\epsilon}), b \in \#(*)\}$$

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash [\alpha]A}{\begin{array}{l} \{p_1, \omega_{p_1} \vdash [\alpha_i](A)_{\ell \times (\ell_1 :: t, \omega_3)}\}_{i \in \{1,n\}} \\ \{p_1, \omega_{p_1} \vdash [\beta_i](A)_{\ell \times (p_2, \omega_4)}\}_{i \in \{1,k\}} \end{array}} \tau \times []_1 \qquad \not\models \forall(\omega_1 \wedge m \leq \ell \leq M \supset false)$$

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash [\alpha]A}{\{p_1, \omega_{p_1} \vdash [\beta_i](A)_{\ell \times (p_2, \omega_4)}\}_{i \in \{1,k\}}} \tau \times []_2 \qquad \models \forall(\omega_1 \wedge m \leq \ell \leq M \supset false)$$

These rules are based on the fact that $\ell :: p_1 \times p_2 \stackrel{\alpha,\omega}{\longmapsto} \ell :: p'_1 \times p'_2$ iff, by rule par, one of the following cases hold:

1. $\ell :: p_1 \stackrel{\alpha_1,\omega}{\longmapsto} p'_1$ and $\alpha \in \alpha_1 \times (b : \bar{\epsilon})$, for $b \in \#(*)$ and $p'_2 = p_2$;
2. $\ell :: p_1 \stackrel{\alpha_1,\omega_1}{\longmapsto} p'_1$ and $\alpha \in \alpha_1 \times (a : \epsilon)$ and $p'_2 = \ell_1 :: t$, $\omega \equiv \omega_1 \wedge m \leq \ell_2 \leq M$.

Moreover, $loc(p_2) = \{\ell_1, \ell_2\}$ and $delay(p_2) \equiv \ell_2 \leq M$.

Rules for choice

The rules for choice are very similar to the untimed ones. We define extended formulas $(A)_{\times+b}$ with the following semantics

$$\| (A)_{\times+b} \| = \{(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1, \pi) \mid (\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b, \pi) \in \| A \|\}$$

The shift rule is

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1, \omega_1 \models (A)_{\times+b}}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b, \omega_1 \models A}$$

1. Rules applicable iff $type(proj_2(\alpha)) = \epsilon$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash < \alpha > A} \times <> +\epsilon$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash [\alpha]A\}_{i \in \{1,2\}}} \times [] + \epsilon$$

2. Rules applicable iff $type(proj_2(\alpha)) = \bar{\epsilon}$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash < \alpha > (A)_{\times+b_j}\}_{i,j \in \{1,2\}, i \neq j}} \times <> +\bar{\epsilon}$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash [\alpha](A)_{\times+b_j}\}_{i,j \in \{1,2\}, i \neq j}} \times [] + \bar{\epsilon}_1$$

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash [\alpha]A}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash [\alpha]false} \times [] + \bar{\epsilon}_2$$

These rules rely on the following reasoning. We have $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times p_2'$ iff $p_1 \stackrel{\alpha_1,\omega_1}{\longmapsto} p_1'$ and $\ell_1 :: \ell_2 :: b_1 + b_2 \stackrel{\alpha_2,\omega_2}{\longmapsto} p_2'$, where $\alpha \in \alpha_1 \times \alpha_2$ and $\omega \equiv \omega_1 \wedge \omega_2$. There are two cases:

1. if $type(\alpha_2) = \epsilon$, then $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha_2,\omega_2}{\longmapsto} p_2'$, for some $i \in \{1, 2\}$. Therefore, by rule *par*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times p_2'$. Moreover, $loc(\ell_1 :: \ell_2 :: b_i) = loc(\ell_1 :: \ell_2 :: b_1 + b_2)$.

2. if $type(\alpha_2) = \bar{\epsilon}$, then $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha_2,\omega_2}{\longmapsto} p_2'$, for each $i \in \{1, 2\}$ and $\omega_2 \equiv true$. Therefore, by rule *par*, for each $i \in \{1, 2\}$, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times p_2'$. Moreover, $loc(\ell_1 :: \ell_2 :: b_i) = loc(\ell_1 :: \ell_2 :: b_1 + b_2)$.

Rules for refinement

The rules for refinement are very similar to the ones for choice. However, the operator is not symmetric and $loc(\ell_1 :: t) \cap loc(p) = main(t)$.

We define extended formulas $(A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}$ and $(A)_{\times\bigtriangledown(p_1,\omega_2)}$ with the following semantics

$$\| (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown} \| = \{(\ell :: p_1 \times p_2, \pi) \mid (\ell :: p_1 \times \ell_1 :: t \bigtriangledown p_2, \pi \cup \pi_2) \in \| A \|, \forall \pi_2 \models \omega_2\}$$

and

$$\| (A)_{\times\bigtriangledown(p_1,\omega_2)} \| = \{(\ell :: p_2 \times \ell_1 :: t, \pi) \mid (\ell :: p_2 \times \ell_1 :: t \bigtriangledown p_2, \pi \cup \pi_2) \in \| A \|, \forall \pi_2 \models \omega_2\}$$

The shift rule are the following.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}}{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p_2, \omega_1 \wedge \omega_2 \vdash A}$$

$$\frac{\ell :: p_2 \times \ell_1 :: t, \omega_1 \vdash (A)_{\times\bigtriangledown(p_1,\omega_2)}}{\ell :: p_2 \times \ell_1 :: t \bigtriangledown p_1, \omega_1 \wedge \omega_2 \vdash A}$$

Let $\omega_p \equiv pro_{C\cup(loc(\ell::p_1\times p)\setminus main(t))}(\omega_1)$, $\omega_t \equiv pro_{C\cup loc(\ell::p_1\times\ell_1::t)}(\omega_1)$, $\omega_2 \equiv pro_{loc(\ell_1::t)}(\omega_1) - 1$ and $\omega_3 \equiv pro_{loc(p)\setminus main(t)}(\omega_1) - 1$.

1. Rules applicable iff $type(proj_2(\alpha)) = \epsilon$.

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times \ell_1 :: t, \omega_t \vdash < \alpha > A} \times <> \bigtriangledown\epsilon_1$$

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times p, \omega_p \vdash < \alpha > (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}} \times <> \bigtriangledown\epsilon_2 \quad \models \forall(\omega_2 \supset delay(\ell_1 :: t))$$

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: p_1 \times p, \omega_p \vdash [\alpha](A)_{\times(\ell_1::t,\omega_2)\bigtriangledown} \quad \ell :: p_1 \times \ell_1 :: t, \omega_t \vdash [\alpha]A} \times[] \bigtriangledown \epsilon_2$$

2. Rules applicable iff $type(proj_2(\alpha)) = \bar{\epsilon}$.

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\begin{array}{l}\ell :: p_1 \times \ell_1 :: t, \omega_t \vdash < \alpha > (A)_{\times\bigtriangledown(p,\omega_3)} \\ \ell :: p_1 \times p, \omega_p \vdash < \alpha > (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}\end{array}} \times <> \bigtriangledown\bar{\epsilon}$$

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\begin{array}{l}\ell :: p_1 \times \ell_1 :: t, \omega_t \vdash [\alpha](A)_{\times \bigtriangledown (p,\omega_3)} \\ \ell :: p_1 \times p, \omega_p \vdash [\alpha](A)_{\times (\ell_1 :: t, \omega_2)\bigtriangledown}\end{array}} \times [] \bigtriangledown \bar{\epsilon}_1$$

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: p_1 \times \ell_1 :: t, \omega_t \vdash [\alpha] false} \times [] \bigtriangledown \bar{\epsilon}_2$$

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash [\alpha]A}{\ell :: p_1 \times p, \omega_p \vdash [\alpha] false} \times [] \bigtriangledown \bar{\epsilon}_3$$

Rules for recursion

The rules are analogous to the ones for the untimed case.

$$\frac{\ell :: p_1 \times \ell_2 :: rec\ x.t, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times \ell_2 :: t[rec\ x.t/x], \omega_1 \vdash < \alpha > A} \times <> rec$$

$$\frac{\ell :: p_1 \times \ell_2 :: rec\ x.t, \omega_1 \vdash [\alpha]A}{\ell :: p_1 \times \ell_2 :: t[rec\ x.t/x], \omega_1 \vdash [\alpha]A} \times [] rec$$

Rules for parallel composition

The rules are analogous to the ones for the untimed case. We define extended formulas $(A)_\times$ with the obvious semantics and shift rule.

$$\frac{\ell :: p_1 \times \ell_2 :: (p_3 \times p_4), \omega_1 \vdash < \alpha > A}{\ell :: \ell_2 :: (p_1 \times p_3) \times p_4, \omega_1 \vdash < \beta > (A)_\times} \times \times <>_{\beta \in left(\alpha)}$$

$$\frac{\ell :: p_1 \times \ell_2 :: (p_3 \times p_4), \omega_1 \vdash [\alpha]A}{\{\ell :: \ell_1 :: (p_1 \times p_3) \times p_4, \omega_1 \vdash [\beta_i](A)_\times\}_{i \in \{1,n\}}} \times \times [] \qquad \begin{array}{l}\{\beta_1, \ldots \beta_n\} = \\ \{\beta_i \mid \in left(\alpha)\}\end{array}$$

7.3 Soundness and completeness of the proof system

The proof system is sound in general and complete for regular processes.

Theorem 7.3.1 *(Soundness) Assume $p \in \mathcal{CTSP}$ over Act and Loc. For a closed RTL_μ formula A over Act, Loc and C and for $\omega \in \Omega(C \cup loc(p))$, if $p, \omega \vdash A$, then $p, \omega \models A$.*

The proof of this theorem is quite similar to the one of theorem 4.4.1. It is sufficient to show that each rule for proving $< \alpha > A$ and $[\alpha]A$ is backward sound. Actually,

the soundness of the other rules has been proved in theorem 6.3.2. The proof can be found in appendix D.1 and exploits proposition 7.2.1.

The proof system is complete for regular processes. The proof is similar to the one of theorem 4.4.5. We generalize lemma 4.4.4 to timed processes.

For $p \in CTSP$, we denote by $\mathcal{R}_p$ the subset of closed processes such that

1. $p \in \mathcal{R}_p$;
2. if $p' \in \mathcal{R}_p$ and $p' \stackrel{\alpha,\omega}{\longmapsto} p''$, then $p'' \in \mathcal{R}_p$.

Moreover, we denote by $\mathcal{S}_p$ the subset of closed processes such that, if p' is a subprocess of p, then $\mathcal{R}_{p'} \subseteq \mathcal{S}_p$. As in the untimed case regular processes are characterized by the fact that $\mathcal{S}_p$ is finite.

Lemma 7.3.2 *Assume $p \in \mathcal{CTSP}$ over Act and Loc. For a closed RTL_μ formula A over Act, Loc and C and for $\omega \in \Omega(C \cup loc(p))$, if $p, \omega \models < \alpha > A$ (resp. $p, \omega \models [\alpha]A$), with $\alpha \in \mathcal{E}(Act)$, then*

1. *there exist sets $\{p_i \in \mathcal{S}_p\}_{i\in\{1,n\}}$ and $\{p'_j \in \mathcal{S}_p\}_{j\in\{1,k\}}$, such that $\{p_i, \omega_i \models A\}_{i\in\{1,n\}}$ and $\{p'_j, \omega'_j \models true\}_{j\in\{1,k\}}$;*
2. *$p, \omega \vdash < \alpha > A$ (resp. $p, \omega \vdash [\alpha]A$)) can be derived from $\{p_i, \omega_i \vdash A\}_{i\in\{1,n\}}$ and $\{p'_j, \omega'_j \vdash true\}_{j\in\{1,k\}}$.*

Moreover, for each $i \in \{1, n\}$ and $\pi \models \omega$, there exists $(p, \pi) \stackrel{\alpha_i}{\mapsto} (p_i, \pi')$, where $\pi' \models \omega_i$, such that it is consistent with α.

The proof is by induction on $m(p)$ and exploits proposition 7.2.2. It is shown in appendix D.2.

Theorem 7.3.3 *(Completeness) Assume $p \in \mathcal{CTSP}$ be a regular process over Act and Loc. For a closed RTL_μ formula A over Act, Loc and C and for $\omega \in \Omega(C \cup clock(p))$, if $p, \omega \models A$ then $p, \omega \vdash A$.*

This theorem follows trivially by lemma 7.3.2 and theorem 6.3.9 by induction on the relation $\prec$ (definition 6.3.8), that is well-founded for regular processes, since $\mathcal{S}_p$ is finite.

7.4 An example

In this section we show the proof of a simple property for the statechart that is depicted in figure 7.1.

The $\mathcal{TSP}$ process corresponding to this statechart is $p = A :: p_1 \times p_2$ where

$$p_1 = B :: rec\ x.D :: [0,1]lab(t_1).rec\ y.E :: lab(t_2).x$$
$$p_2 = C :: rec\ x.F :: (lab(t_4).rec\ y.G :: lab(t_3).x) + ([2,2]lab(t_5).H :: nil)$$

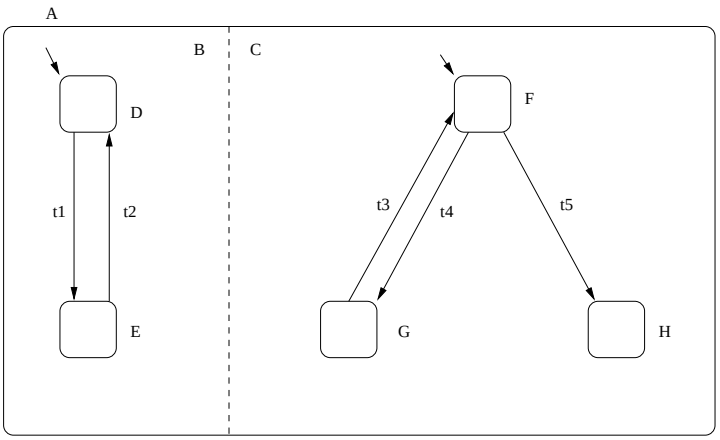

Figure 7.1: A timed statechart

Consider the RTL_μ formula

$$A \equiv \nu X.(\neg H \wedge [K]X)$$

where $K = \{\alpha_{j,i}\}_{j\in\{1,2\},i\in\{1,2\}}$ such that $\alpha_{j,1} = a_j : \epsilon$, $\alpha_{j,2} = a_j : \bar{\epsilon}$ and $a_1 = (\{\bar{a}\}, \emptyset), a_2 = (\{a, nil\}, \{nil\} \prec \{a\})$. This assertion express the property that, for each computation, state H is never entered under the assumption that event a is local, namely it is not produced by the environment.

Let $\omega_1 \equiv D = 0 \wedge F = 0$. We have the following proof.

$$\dfrac{\dfrac{\dfrac{p, \omega_1 \vdash A}{p, \omega_1 \vdash \neg H \wedge [K]A_1}\,\nu_2}{\dfrac{p, \omega_1 \vdash \neg H}{}\,prop_2 \qquad \dfrac{p, \omega_1 \vdash [K]A_1}{\pi_1}}\,\wedge}$$

where $A_1 \equiv \nu X\{(p, \omega_1)\}.(\neg H \wedge [K]X)$.

Proof π_1 is as follows.

$$\dfrac{p, \omega_1 \vdash [K]A_1}{\left\{ \dfrac{p, \omega_1 \vdash [\alpha_{j,i}]A_1}{\pi_{j,i}} \right\}_{j\in\{1,2\},i\in\{1,2\}}}\,\wedge$$

1. Proof $\pi_{1,1}$ is as follows.

$$\dfrac{\dfrac{\dfrac{A :: p_1 \times p_2, \omega_1 \vdash [\alpha_{1,1}]A_1}{A :: p_1 \times p_2', \omega_1 \vdash [\alpha_{1,1}]A_1}\times[]rec}{\left\{\dfrac{A :: p_1 \times p_2', \omega_1 \vdash [\alpha_{1,1,i}]A_1}{\pi_{1,1,i}}\right\}_{i\in\{1,3\}}}\mathcal{E}[]}$$

where

$p_2' = C :: F :: (lab(t_4).rec\ y.G :: lab(t_3).t_2) + ([2,2]lab(t_5).H :: nil)$
$t_2 = C :: recx.F :: (lab(t_4).rec\ y.G :: .lab(t_3).x) + ([2,2]lab(t_5).H :: nil)$
$\alpha_{1,1,1} = a_1 : \epsilon \times \epsilon$
$\alpha_{1,1,2} = a_1 : \epsilon \times \bar{\epsilon}$
$\alpha_{1,1,3} = a_1 : \bar{\epsilon} \times \epsilon.$

1.1. Proof $\pi_{1,1,1}$ is as follows.

$$\dfrac{A :: p_1 \times p_2', \omega_1 \vdash [\alpha_{1,1,1}]A_1}{\dfrac{A :: p_1 \times p_3, \omega_1 \vdash [\alpha_{1,1,1}]A_1}{}\times[] \qquad \dfrac{A :: p_1 \times p_4, \omega_1 \vdash [\alpha_{1,1,1}]A_1}{}\tau \times []_2}$$

where

$p_3 = C :: F :: lab(t_4).rec\ y.G :: lab(t_3).t_2$
$p_4 = C :: F :: [2,2]lab(t_5).H :: nil.$

We have applied rule $\times[]+\epsilon$, since $type(proj_2(\alpha_{1,1,1})) = \epsilon$. In rule $\times[]$, we exploit the fact that $\bar{a} \in trigger(a_1)$ and $a \in trigger(lab(t_4))$ so that $a_1 \notin \alpha \times lab(t_4)$, for each α. Moreover, in rule $\tau \times []_2$, $\omega_1 \wedge F = 2$ is not satisfiable.

1.2. Proof $\pi_{1,1,2}$ is as follows.

$$\dfrac{\dfrac{\dfrac{A :: p_1 \times p_2', \omega_1 \vdash [\alpha_{1,1,2}]A_1}{A :: p_1 \times p_3, \omega_1 \vdash [\alpha_{1,1,2}]false}\times[] + \bar{\epsilon}_2}{p_1', \omega_{1,1} \vdash [K_1](false)_{A\times(p_3,\omega_{1,2})}}\tau \times []_2}{}\tau[]\epsilon_2$$

where

$K_1 = \{\alpha \mid \alpha_{1,1,2} \in \alpha \otimes (b : \bar{\epsilon}), b \in \#(lab(t_4))\}$
$p_1' = B :: D :: [0,1]lab(t_1).rec\ y.E :: lab(t_2).t_1$
$t_1 = B :: rec\ x.D :: [0,1]lab(t_1).rec\ y.E :: lab(t_2).x$
$\omega_{1,1} \equiv D = 0$
$\omega_{1,2} \equiv F = 1.$

In rule $\tau[]\epsilon_2$ we exploit the fact that $\alpha_{1,1,2} \notin (lab(t_1) : \epsilon) \otimes (b : \bar{\epsilon})$, since $a \in action(t_1)$, while $\bar{a} \in trigger(a_1)$.

1.3. Proof $\pi_{1,1,3}$ is analogous to $\pi_{1,1,1}$, since $\bar{a} \in trigger(a_1)$, $a \in trigger(lab(t_4))$ and $\omega_1 \wedge F = 2$ is not satisfiable.

2. Proof $\pi_{1,2}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{A :: p_1 \times p_2, \omega_1 \vdash [\alpha_{1,2}]A_1}{A :: p_1 \times p'_2, \omega_1 \vdash [\alpha_{1,2}]A_1}}{A :: p_1 \times p'_2, \omega_1 \vdash [\alpha_{1,2,1}]A_1}}{\dfrac{A :: p_1 \times p_3, \omega_1 \vdash [\alpha_{1,2,1}](A_1)_{\times +p_4}}{\pi_{1,2,2}} \qquad \dfrac{A :: p_1 \times p_4, \omega_1 \vdash [\alpha_{1,2,1}](A_1)_{\times +p_3}}{\pi_{1,2,3}}}$$

where $\alpha_{1,2,1} = a_1 : \bar{\epsilon} \times \bar{\epsilon} = \mathcal{E}(\alpha_{1,2})$. Since $type(proj_2(\alpha_{1,2,1}))= \bar{\epsilon}$, we have applied rule $\times[] + \bar{\epsilon}$.

2.1. Proof $\pi_{1,2,2}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A :: p_1 \times p_3, \omega_1 \vdash [\alpha_{1,2,1}](A_1)_{\times +p_4}}{p_1, \omega_{1,1} \vdash [K_2]((A_1)_{\times +p_4})_{A \times (p_3, \omega_{1,2})}}\times[]}{p'_1, \omega_{1,1} \vdash [K_2]((A_1)_{\times +p_4})_{A \times (p_3, \omega_{1,2})}}rec[]}{p'_1, \omega_{1,1} - 1 \vdash ((A_1)_{\times +p_4})_{A \times (p_3, \omega_{1,2})}}\tau[]\bar{\epsilon}_1}{A :: p'_1 \times p'_2, \omega_1 - 1 \wedge \omega_{1,2} \vdash A_1}shift}{\pi_{1,2,4}}shift$$

where

$$K_2 = \{\alpha \mid \alpha_{1,2,1} \in \alpha \otimes (b : \epsilon) \text{ for } b \in \#(lab(t_4)) \cap \#(*)\}$$

Let $\omega_2 \equiv \omega_{1,1} - 1 \wedge \omega_{1,2}$. We have $\omega_1 - 1 \equiv D = 1 \wedge F = 1 \equiv \omega_2$.
Proof $\pi_{1,2,4}$ is as follows.

$$\dfrac{\dfrac{\dfrac{A :: p'_1 \times p'_2, \omega_2 \vdash A_1}{A :: p'_1 \times p'_2, \omega_2 \vdash \neg H \wedge [K]A_2}\nu_2}{\dfrac{A :: p'_1 \times p'_2, \omega_2 \vdash \neg H}{prop_2} \qquad \dfrac{A :: p'_1 \times p'_2, \omega_2 \vdash [K]A_2}{\pi_{1,2,5}}}\wedge$$

where $A_2 = \nu X\{(p, \omega_1), (A :: p'_1 \times p'_2, \omega_2)\}.(\neg H \wedge [K]X)$.

Proof $\pi_{1,2,5}$ is analogous to proof π_1. We will discuss this case at the end of the section.

2.2. Proof $\pi_{1,2,3}$ is analogous to $\pi_{1,2,2}$.

3. Proof $\pi_{2,1}$ is as follows.

$$\dfrac{\dfrac{\dfrac{A :: p_1 \times p_2, \omega_1 \vdash [\alpha_{2,1}]A_1}{A :: p_1 \times p'_2, \omega_1 \vdash [\alpha_{2,1}]A_1}\times[]rec}{\left\{\begin{array}{c} A :: p_1 \times p'_2, \omega_1 \vdash [\alpha_{2,1,i}]A_1 \\ \hline \pi_{2,1,i} \end{array}\right\}_{i\in\{1,3\}}}\mathcal{E}[]$$

where $\alpha_{2,1,1} = a_2 : \epsilon \times \epsilon$, $\alpha_{2,1,2} = a_2 : \epsilon \times \bar{\epsilon}$ and $\alpha_{2,1,3} = a_2 : \bar{\epsilon} \times \epsilon$.

3.1. Proof $\pi_{2,1,1}$ is as follows.

$$\dfrac{A :: p_1 \times p'_2, \omega_1 \vdash [\alpha_{2,1,1}]A_1}{\dfrac{\dfrac{A :: p_1 \times p_3, \omega_1 \vdash [\alpha_{2,1,1}]A_1}{p_1, \omega_{1,1} \vdash [K_3](A_1)_{A\times(p'_3,\omega_3)}}\times[]}{\pi_{2,1,4}} \qquad \dfrac{A :: p_1 \times p_4, \omega_1 \vdash [\alpha_{2,1,1}]A_1}{}\tau \times []_2}$$

where

$p'_3 = C :: rec\ y.G :: lab(t_3).t_2$
$\omega_3 \equiv G = 1$
$K_3 = \{\alpha \mid \alpha_{2,1,1} \in \alpha \otimes (lab(t_4) : \epsilon)\}$

We have applied rule $\times[] + \epsilon$, since $type(proj_2(\alpha_{2,1,1})) = \epsilon$.

Proof $\pi_{2,1,4}$ is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{p_1, \omega_{1,1} \vdash [K_3](A_1)_{A\times(p'_3,\omega_3)}}{p'_1, \omega_{1,1} \vdash [K_3](A_1)_{A\times(p'_3,\omega_3)}}rec[]}{p''_1, \omega_4 \vdash (A_1)_{A\times(p'_3,\omega_3)}}\times[]}{A :: p''_1 \times p'_3, \omega_4 \wedge \omega_3 \vdash A_1}shift}{\pi_{2,1,5}}$$

where $\omega_4 \equiv E = 1$ and $p''_1 \equiv B :: rec\ y.E :: lab(t_2).t_1$.

It is quite obvious that $A :: p''_1 \times p'_3, \omega_3 \wedge \omega_4 \vdash A_1$ can be obtained from $A :: p'_1 \wedge p'_2, \omega_2 \vdash A_3$, where $A_3 = \nu X.\{(p, \omega_1), (A :: p''_1 \times p'_3, \omega_3 \wedge \omega_4)\}.(\neg H \wedge [K]X)$. We omit this part of the proof for sake of brevity.

The proof of $A :: p'_1 \wedge p'_2, \omega_2 \vdash A_3$ is analogous to $\pi_{1,2,5}$. We will discuss this case at the end of the section.

3.2. Proof $\pi_{2,1,2}$ is as follows.

$$\dfrac{\dfrac{A :: p_1 \times p_2', \omega_1 \vdash [\alpha_{2,1,2}]A_1}{A :: p_1 \times p_3, \omega_1 \vdash [\alpha_{2,1,2}]false}\times[]}{}\times[] + \bar{\epsilon}_2$$

Actually, $a \in action(\alpha_{1,2,1})$, while $\bar{a} \in trigger(b)$ for each $b \in \#(lab(t_4))$.

3.3. Proof $\pi_{2,1,3}$ is as follows.

$$\dfrac{A :: p_1 \times p_2', \omega_1 \vdash [\alpha_{2,1,3}]A_1}{\dfrac{\dfrac{A :: p_1 \times p_3, \omega_1 \vdash [\alpha_{2,1,3}]A_1}{p_1, \omega_{1,1} \vdash [K_4](A_1)_{A\times(p_3',\omega_3)}}\times[]}{}\pi_{2,1,6} \qquad \dfrac{A :: p_1 \times p_4, \omega_1 \vdash [\alpha_{2,1,3}]A_1}{}\tau \times []_2}$$

where $K_4 = \{\alpha \mid \alpha_{2,1,3} \in \alpha \otimes (lab(t_4) : \epsilon)\}$. We have applied rule $\times[] + \epsilon$, since $type(proj_2(\alpha_{2,1,3})) = \epsilon$.

Proof $\pi_{2,1,6}$ is as follows.

$$\dfrac{\dfrac{p_1, \omega_{1,1} \vdash [K_4](A_1)_{A\times(p_3',\omega_3)}}{p_1', \omega_{1,1} \vdash [K_4](A_1)_{A\times(p_3',\omega_3)}}\tau[]\bar{\epsilon}_1}{}rec[]$$

Actually, for each $b \in \#(*)$, it is not the case that $\alpha_{2,1,3} \in (b : \bar{\epsilon}) \otimes (lab(t_4) : \epsilon)$.

4. The proof $\pi_{2,2}$ can be omitted by observing that $type(\alpha_{2,2}) = \bar{\epsilon}$ and $action(\alpha_{2,2}) \neq \emptyset$.

It remains to show proof $\pi_{1,2,5}$ of $A :: p_1' \times p_2', \omega_2 \vdash [K]A_2$ and proof $\pi_{2,1,5}$ of $A :: p_1' \times p_2', \omega_2 \vdash [K]A_3$.

- For $\pi_{1,2,5}$ we can repeat the same reasoning such as in π_1 and indeed deducing $A :: p_1' \times p_2', \omega_2 \vdash [K]A_2$ from premises $A :: p_1' \times p_2', \omega_2 - 1 \vdash A_2$ and $A :: p_1' \times p_2', \omega_2 \vdash A_5$, where $A_5 = \nu X\{(p, \omega_1), (A :: p_1' \times p_2', \omega_2), (A :: p_1'' \times p_3', \omega_3 \wedge \omega_4)\}.(\neg H \wedge [K]X)$.

  We have $A :: p_1' \times p_2', \omega_2 - 1 \vdash A_2$ by rule *empty*, since $delay(A :: p_1' \times p_2') \equiv D \leq 1 \wedge F \leq 2$ and $\models \forall(\omega_2 - 1 \supset D = 2)$. On the other hand, $A :: p_1' \times p_2', \omega_2 \vdash A_5$ by rule ν_1.

- For $\pi_{1,2,5}$ we can repeat the same reasoning such as in π_1 and indeed deducing $A :: p_1' \times p_2', \omega_2 \vdash [K]A_3$ from premises $A :: p_1' \times p_2', \omega_2 - 1 \vdash A_3$ and $A :: p_1'' \times p_3', \omega_3 \wedge \omega_4 \vdash A_3$.

  We have $A :: p_1' \times p_2', \omega_2 - 1 \vdash A_3$ by rule *empty* and $A :: p_1'' \times p_3', \omega_3 \wedge \omega_4 \vdash A_3$ by rule ν_1.

7.5 Related works

Compositional proof systems for real-time logics are scarse in the literature. In [47, 93] a CSP-like language with synchronous communication via channels is considered. A compositional proof system for checking formulas of linear temporal logic extended with bounded operators as well as with explicit reference to time is proposed. The starting point of this approach is the composition method proposed by [14] for linear temporal logic.

In [19] timed transition modules are used for modelling real-time systems. This model is an extension of timed transition systems that permits to explicitely model the interaction with the environment. The paper presents a compositional proof system for linear temporal logic extended with bounded operators over the discrete time domain. The proof rules proposed in [43] for certain untimed properties are exploited during the verification of some real-time properties.

Conclusions

In this thesis we have given a contribution to the problem of verifying the correctness of statecharts. Propositional μ-calculus turned out to be very adequate for reasoning about statecharts, since it permits to naturally express all the interesting properties characterizing the ongoing interaction with the environment. Local model checking is a very convenient approach for proving μ-calculus properties, since it may possibly avoid to exhaustively explore the whole labelled transition system. However, traditional methods are not compositional and are applicable only after the system has been completely specified. Due to the complexity of reactive systems, compositional verification is essential for a hierarchical and structured development of correct specifications.

As a first step towards compositional verification we have proposed a compositional labelled transition system semantics for statecharts by giving a translation of statecharts into the process language $\mathcal{SP}$. The semantics obtained is proved to be equivalent to the Pnueli and Shalev semantics, which we believe to be the more convincing semantics for statecharts. This method permits to construct in a compositional way the model that is the basis for verification via model checking. However, this approach does not support compositional verification. To this aim we have defined a compositional proof system for checking μ-calculus properties of $\mathcal{SP}$ processes. The proof system combines techniques of local and compositional model checking in the style of [12]. The underlying idea is to reduce the verification problem for a composed process into verification problems for its components by exploiting the algebraic structure of the process.

Since statecharts are not adequate for the specification of real-time systems, where also the quantitative aspect of time must be considered, we have proposed an extension of statecharts and of the language $\mathcal{SP}$ with real-time features by assuming the discrete time domain. The idea is that of associating minimal and maximal delays to transitions in order to specify constructs, such as timeouts as well as watchdogs. We have generalized the Pnueli and Shalev semantics to this case, and we have defined a corresponding semantics for timed $\mathcal{SP}$ processes by adopting a discrete version of timed graphs. In order to express quantitative properties of timed statecharts, we have considered a real-time discrete extension of μ-calculus, called RTL_μ, and we have defined a proof system for checking whether a set of timed states of a discrete timed graph satisfy an RTL_μ formula. This proof system extends the

local model checking approach to real-time. Moreover, it is the basis for extending the compositional technique for untimed processes to timed ones.

This work could be continued in many directions. The considered models of statecharts and of timed statecharts are based on the discrete event approach, namely the environment is modeled as a sequence of discrete events. However, there are applications for which the environment should be more realistically modelled as a continuous process. A first step towards a specification language for this class of systems could be to consider a version of timed statecharts with respect to a dense domain. We have discussed in chapters 5 and 6 the difficulties in treating a dense time domain. Firstly, we believe that the semantics of timed statecharts with respect to a dense time domain needs further investigations. In particular, the role of negated events should be clarified. Moreover, due to the Zeno paradox it is not obvious how to define a dense version of real-time μ-calculus that is sufficiently expressive and that can be treated by local model checking. However, we believe that the study of local model checking methods in this framework could benefit from the proof system proposed for RTL_μ.

A further improvement towards the specification of systems interacting with a continuous environment could be to extend the formalism by allowing both discrete and continuous components. Such continuous components may cause continuous change in the value of some variables typically according to some physical laws. These systems are well-known as *hybrid systems* and in recent years there has been a great interest in their specification and verification. In particular both statecharts [68] and timed graphs [4] have been generalized for this purpose by labelling states with differential equations.

Bibliography

[1] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 1(15):73–132, 1993.

[2] R. Alur. *Techniques for Automatic Verification of Real-time Systems.* PhD thesis, University of Stanford, 1991.

[3] R. Alur, C. Courcoubetis, and D.Dill. Model-Checking for Real-Time Systems. *Information and Computation*, 104(1):2–34, 1993.

[4] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho. Hybrid Automata: An Algorithmic approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag, Berlin, 1993.

[5] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[6] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual Symposium on Principles of Distributed Computing*, pages 139–152. ACM Press, 1991.

[7] R. Alur and T.A. Henzinger. A really temporal logic. In *Proceeding of the 30th Annual Symposium on Foundations of Computer Science*, pages 164–169. IEEE Computer Society Press, 1989.

[8] R. Alur and T.A. Henzinger. Real-time logics: complexity and expressivness. In *Proc. Fifth IEEE Symp. on Logic In Computer Science*, pages 390–401. IEEE Computer Society Press, 1990.

[9] R. Alur and T.A. Henzinger. Logics and models of real time: a survey. In *Proceedings of REX workshop 'Real-time Theory in Practice'*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer-Verlag, Berlin, 1992.

[10] H.R. Andersen. *Verification of Temporal Properties of Concurrent Systems.* PhD thesis, University of Aarhus, Computer Science Department, 1993.

[11] H.R. Andersen. Model Checking and Boolean Graphs. *Theoretical Computer Science*, 126:3–30, 1994.

[12] H.R. Andersen, C. Stirling, and G. Winskel. A Compositional Proof System for the Modal μ-Calculus. In *Proc. Ninth IEEE Symp. on Logic In Computer Science*, pages 144–153, 1994.

[13] H.R. Andersen and G. Winskel. Compositional Checking of Satisfaction. *Formal Methods in System Design*, 1(4):144–153, 1992.

[14] H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal logic specifications. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 51–63, 1984.

[15] G. Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 19:87–152, 1992.

[16] J. Bradfield. A Proof Assistant for Symbolic Model-Checking. In *Proceedings of CAV 92*, volume 663 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1992.

[17] J. Bradfield and C. Stirling. Local Model Checking for Infinite State Space. *Theoretical Computer Science*, 96:157–174, 1992.

[18] P. Caspi, N. Halbwachs, D. Pilaud, and J. Plaice. Lustre, a Declarative Language for Programming Synchronous Systems. In *Proccedings 14-th Symposium of Principles of Programming Languages*, pages 178–188, 1987.

[19] E. Chang, Z. Manna, and A. Pnueli. Compositional Verification of Real-time Systems. In *Proc. Ninth IEEE Symp. on Logic In Computer Science*, pages 458–465, 1994.

[20] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite State Concurrent System using Temporal Logic. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, 1986.

[21] E.M. Clarke, D.E. Long, and K.L. McMillian. Compositional model checking. In *Proc. Fourth IEEE Symp. on Logic In Computer Science*, pages 353–362. IEEE Computer Society Press, 1989.

[22] R. Cleaveland. Tableau-based Model Checking in the Propositional mu-Calculus. *Acta Informatica*, 27:725–747, 1990.

[23] R. Cleaveland and B. Steffen. A linear-time model-checking for alternation free modal μ-calculus. In *Proceedings of CAV 91*, volume 575 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1991.

[24] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. In *Proceedings of CAAP 92*, volume 581 of *Lecture Notes in Computer Science*, pages 145–159. Springer-Verlag, Berlin, 1992.

[25] M. Dam. Compositional Proof Systems for Model Checking Infinite State Processes. In *Proceedings of CONCUR 95*, volume 962 of *Lecture Notes in Computer Science*, pages 12–26. Springer-Verlag, Berlin, 1995.

[26] D. Drusinsky and D. Harel. Using Statecharts for Harware Description and Synthesis. *IEEE Trans. Computer-Aided Design*, 8:798–807, 1989.

[27] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[28] E. Allen Emerson and Chin-Luang Lei. Efficient Model Checking in Fragments of the Propositional mu-Calculus. In *Proc. First IEEE Symp. on Logic In Computer Science*, pages 267–278. IEEE Computer Society Press, 1986.

[29] E.A. Emerson. Real-time and the Mu-Calculus. In *Proceedings of REX workshop 'Real-time Theory in Practice'*, volume 600 of *Lecture Notes in Computer Science*, pages 177–193, 1991.

[30] E.A. Emerson, C.S. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *Proceedings of CAV 93*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396. Springer-Verlag, Berlin, 1993.

[31] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *First Annual Workshop on Computer-Aided Verification*, 1989.

[32] M.J. Fisher and R.E. Ladner. Propositional Dynamic logic of Regular Programs. *Theoretical Computer Science*, 18(2):194–211, 1979.

[33] M.J.C. Gordon. HOL: A proof generating system for higher order logic. In G. Birtwistle and P.A. Subrahmanyam, editors, *VLSI specification, Verification and Synthesis*, pages 73–128. Kluwer, 1988.

[34] O. Grumberg and D. E. Long. Model Checking and Modular Verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.

[35] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[36] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. Statemate: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transations on Software Engineering*, 16:403–414, 1990.

[37] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman. On the Formal Semantics of Statecharts. In *Proc. Second IEEE Symp. on Logic In Computer Science*, pages 54–64. IEEE Computer Society Press, 1987.

[38] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit Clock Temporal Logic. In *Proc. Fifth IEEE Symp. on Logic In Computer Science*, pages 402–413. IEEE Computer Society Press, 1990.

[39] M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of ACM*, 32(1):137–161, 1985.

[40] T.A. Henzinger. Half-order modal logic: how to prove real-time properties. In *Proceedings of the Ninth Annual Symposium on Principles of Distributed Computing*, pages 281–296. ACM Press, 1990.

[41] T.A. Henzinger. *The Temporal Specification and Verification of Real-time Systems*. PhD thesis, University of Stanford, 1991.

[42] T.A. Henzinger, O. Kupferman, and M.Y. Vardi. A Space-Efficient On-the-fly Algorithm for Real-Time Model Checking. In *Proceedings of CONCUR 96*, volume 1119 of *Lecture Notes in Computer Science*, pages 514–529, 1996.

[43] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proc. Eighteenth Annual ACM Symp. on Principles of Programming Languages*, pages 353–363. ACM Press, 1991.

[44] T.A. Henzinger, Z. Manna, and A. Pnueli. Timed Transition Systems. In *Proceedings of REX workshop 'Real-time Theory in Practice'*, volume 600 of *Lecture Notes in Computer Science*, pages 226–251. Springer-Verlag, Berlin, 1992.

[45] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111:193–244, 1994.

[46] J. Hooman. *Specification and Compositional Verification of Real-time Systems*. PhD thesis, Technische Universiteit Eindhoven, 1991.

[47] J. Hooman and J. Widom. A Temporal-Logic Based Compositional Proof System for Real-Time Message Passing. In E. Odijk, M. Rem, and J.C. Syre, editors, *PARLE89: Parallel Architectures and Languages Europe*, volume 366 of *Lecture Notes in Computer Science*, pages 424–441. Springer-Verlag, Berlin, 1989.

[48] J.J.M. Hooman, R. Ramesh, and W.P. de Roever. A Compositional Axiomatization of Statecharts. *Theoretical Computer Science*, 101:289–335, 1992.

[49] C. Huizing and R. Gerth. Semantics of Reactive Systems in Abstract Time. In *Proceedings of REX workshop 'Real-time Theory in Practice'*, volume 600 of *Lecture Notes in Computer Science*, pages 291–314. Springer-Verlag, Berlin, 1992.

[50] C. Huizing, R. Gerth, and W.P. de Roever. Modelling Statecharts in a Fully Abstract Way. In *Proceedings of CAAP 88*, volume 299 of *Lecture Notes in Computer Science*, pages 271–294. Springer-Verlag, Berlin, 1988.

[51] C.B. Jones. Specification and design of (parallel) programs. In *IFIP 9th Word Congress*, Information Processing, pages 321–332. North Holland, 1983.

[52] B. Josko. MCTL-an extension of CTL for modular verification of concurrent systems. In *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 165–187. Springer-Verlag, Berlin, 1987.

[53] B. Josko. Model checking of CTL formulae under liveness assumptions. In *Proceedings of ICALP 87*, volume 267 of *Lecture Notes in Computer Science*, pages 280–289. Springer-Verlag, Berlin, 1987.

[54] B. Josko. Verifying the correctness of AADL modules using model checking. volume 430 of *Lecture Notes in Computer Science*, pages 386–400. Springer-Verlag, Berlin, 1989.

[55] M. Jourdan, F. Maraninchi, and A. Oliveiro. Verifying quantitative real-time properties of synchronous programs. In *Proceedings of CAV 93*, volume 697 of *Lecture Notes in Computer Science*, pages 347–358. Springer-Verlag, Berlin, 1993.

[56] Y. Kesten and A. Pnueli. Timed and Hybrid Statecharts and their Textual Representation. In *Formal Techniques in Real-time and Fault Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, pages 591–620. Springer-Verlag, Berlin, 1992.

[57] D. Kozen. Results on the Propositional mu-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[58] K.G. Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72:265–288, 1990.

[59] K.G. Larsen. Efficient Local Correctness Checking. In *Proceedings of CAV 92*, volume 663 of *Lecture Notes in Computer Science*, pages 30–43. Springer-Verlag, Berlin, 1992.

[60] Kim G. Larsen. Proof System for Hennessy-Milner Logic with Recursion. In *Proc. of CAAP 88*, volume 299 of *Lecture Notes in Computer Science*, pages 215–230. Springer-Verlag, Berlin, 1988.

[61] Kim G. Larsen and Liu Xinxin. Compositionality through an Operational Semantics of Context. In M.S. Paterson, editor, *Proceedings of ICALP 90*, volume 443 of *Lecture Notes in Computer Science*, pages 526–539. Springer-Verlag, Berlin, 1990.

[62] F. Levi. A Process Language for Statecharts. In *Proc. of the Workshop "Analysis and Verification of Multiple-agent Languages"*, volume 1192 of *Lecture Notes in Computer Science*, pages 388–403, 1996.

[63] H.R. Lewis. A logic of concrete time intervals. In *Proc. Fifth IEEE Symp. on Logic In Computer Science*, pages 380–389. IEEE Computer Society Press, 1990.

[64] A. Maggiolo-Schettini and M. Merro. Priorities, interrupts and preemption in Statecharts. In *Proc. of the Workshop Analysis and Verification of Multiple-agent Languages*, 1996. to appear in LNCS.

[65] A. Maggiolo-Schettini and A. Peron. Semantics for Statecharts Based on Graph Rewriting. In *Correct Hardware Design Methodologies*, pages 91–114, Amsterdam, 1992. North-Holland.

[66] A. Maggiolo-Schettini and A. Peron. Retiming Techniques for Statecharts. In *Proceedings of FTRTFT 96*, number 1135 in Lecture Notes in Computer Science, pages 55–71. Springer-Verlag, Berlin, 1996.

[67] A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of Statecharts. In *Proc. of CONCUR 96*, volume 1119 of *Lecture Notes in Computer Science*, pages 687–702. Springer-Verlag, Berlin, 1996.

[68] O. Maler, Z. Manna, and A. Pnueli. From Timed to Hybrid Systems. In *Proceedings of REX workshop 'Real-time Theory in Practice'*, volume 600 of *Lecture Notes in Computer Science*, pages 447–484. Springer-Verlag, Berlin, 1992.

[69] F. Maraninchi. Operational and Compositional Semantics of Synchronous Automaton Composition. In *Proc. of CONCUR 92*, volume 630 of *Lecture Notes in Computer Science*, pages 550–564, 1992.

[70] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[71] S. Owre, J.M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer-Verlag, Berlin, 1992.

[72] A. Peron. *Synchronous and Asynchronous Models for Statecharts*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1993.

[73] A. Peron and A. Maggiolo-Schettini. Transitions as Interrupts: A New Semantics for Timed Statecharts. In *Proceedings of International Symposium on Theoretical Aspects of Computer Software*, number 789 in Lecture Notes in Computer Science, pages 806–821. Springer-Verlag, Berlin, 1994.

[74] D. Pilaud and N. Halbwachs. From a synchronous declarative language to a temporal logic dealing with multiform time. In *Formal Techniques in Real-time and Fault-tolerant Systems*, volume 331 of *Lecture Notes in Computer Science*, pages 99–110. Springer-Verlag, Berlin, 1988.

[75] A. Pnueli. The Temporal Logic of Concurrent Programs. In *Proc. of the 18th annual IEEE Symposium on Foundations of Computer Science*, pages 45–57, 1977.

[76] A. Pnueli and E. Harel. Applications of temporal logic to the specification of real-time systems. In *Formal Techniques in Real-time and Fault-tolerant Systems*, volume 331 of *Lecture Notes in Computer Science*, pages 84–98. Springer-Verlag, Berlin, 1988.

[77] A. Pnueli and M. Shalev. What is in a Step. In *Theoretical Aspects of Computer Science*, volume 526 of *Lecture Notes in Computer Science*, pages 244–464. Springer-Verlag, Berlin, 1991.

[78] K.V.S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25, 1995.

[79] V. Pratt. A decidable mu-calculus. In *Proceedings of 22nd. FOCS*, pages 421–427, 1981.

[80] A.N. Prior. Past, present and future. Oxford University Press, 1967.

[81] O.V. Sokolski and S.A. Smolka. Incremental Model Checking in the Modal Mu-Calculus. In *Proceedings of CAV 94*, volume 818 of *Lecture Notes in Computer Science*, pages 351–363. Springer-Verlag, Berlin, 1994.

[82] O.V. Sokolski and S.A. Smolka. Local Model Checking for Real-Time Systems. In *Proceedings of CAV 95*, volume 939 of *Lecture Notes in Computer Science*, pages 211–224. Springer-Verlag, Berlin, 1995.

[83] C. Stirling. Modal Logic for Communicating Systems. *Theoretical Computer Science*, 49:311–347, 1987.

[84] C. Stirling. Modal and Temporal Logics. In S. Abramsky and D. Gabbay, editors, *Handbook of Logic in Computer Science*, volume 2, pages 477–563. Oxford University Press, 1992.

[85] C. Stirling and D. Walker. Local Model Checking in the Modal mu-Calculus. *Theoretical Computer Science*, 89:161–177, 1991.

[86] R.S. Street and E.A. Emerson. An Automata Theoretic Decision Procedure for the Propositional Mu-Calculus. *Information and Computation*, 81(3):249–264, 1989.

[87] A.C. Uselton and S.A. Smolka. A Compositional Semantics for Statecharts using Labelled Transition Systems. In *Proc. of CONCUR 94*, volume 836 of *Lecture Notes in Computer Science*, pages 2–17, 1994.

[88] A.C. Uselton and S.A. Smolka. A Process Algebraic Semantics for Statecharts. In *Proceedings of IFIP Working Conference on Programming Concepts, Methods and Calculi*, 1994.

[89] G. Winskel. A Note on Model Checking the Modal nu-Calculus. In S. Ronchi della Rocca, editor, *Proceedings of ICALP 89*, volume 372 of *Lecture Notes in Computer Science*, pages 761–772. Springer-Verlag, Berlin, 1989.

[90] G. Winskel. A Compositional Proof System on a Category of Labelled Transition Systems. *Information and Computation*, 87, 1990.

[91] G. Winskel. On the compositional Checking of Validity. In *Proceedings of CONCUR 90*, volume 458 of *Lecture Notes in Computer Science*, pages 481–501. Springer-Verlag, Berlin, 1990.

[92] G. Winskel. A Note on Model Checking the Modal Nu-calculus. *Theoretical Computer Science*, 83:157–187, 1991.

[93] P. Zhou, J. Hooman, and R. Kuiper. Compositional Verification of Real-Time Systems with Explicit Clock Temporal Logic. *Formal Aspects of Computing*, 8(3):294–323, 1996.

Appendix A

Proofs of chapter 3

In this section we show the proof of theorem 3.2.7. We need some preliminaries properties.

Proposition A.0.1 *Let S be a statechart and let $p \in \mathcal{R}_{\alpha(S)}$. Then p has the following syntax:*

- *If S is a basic empty statechart, then $p = S :: S :: nil$;*
- *If S is a basic non-empty statechart $S : (\{S_1, \ldots, S_n\}, T, in, out, \chi, \delta)$, then either $p = S :: rec\ x.S_i :: \sum_{j \in \{1,k\}} a_j.\ t_j$ or $p = S :: S_i :: nil$;*
- *If S is a refined statechart $\bigtriangledown(S, \{S_1, \ldots, S_n\})$, then $p = S :: t'$ where for t' one of the following cases holds:*
  1. *$rec\ x.S_i :: \sum_{j \in \{1,k\}} a_j.ref(t_j, \{(S_1, \alpha(S_i)), \ldots, (S_n, \alpha(S_n))\})\ \bigtriangledown \alpha(S_i)$ with $S :: rec\ x.S_i :: \sum_{j \in \{1,k\}} a_j.t_j \in \mathcal{R}_{\alpha(S)}$;*
  2. *$S_i :: \sum_{j \in \{1,k\}} a_j.ref(t_j[rec\ x.S_i :: \sum_{j \in \{1,k\}} a_j.t_j / x], \{(S_1, \alpha(S_1)), \ldots, (S_n, \alpha(S_n))\}) \bigtriangledown p'$, with $S :: rec\ x.S_i :: \sum_{j \in \{1,k\}} a_j.t_j \in \mathcal{R}_{\alpha(S)}$ and $p' \in \mathcal{R}_{\alpha(S_i)}$;*
  3. *$S_i :: nil \bigtriangledown p'$ with $S :: S_i :: nil \in \mathcal{R}_{\alpha(S)}$ and $p' \in \mathcal{R}_{\alpha(S_i)}$.*
- *If S is an AND statechart $S : AND(S_1, S_2)$, then $p = S :: p_1 \times p_2$ with $p_i \in \mathcal{R}_{\alpha(S_i)}$, for each $i \in \{1, 2\}$.*

Proposition A.0.2 *Let S be a statechart and $p \in \mathcal{R}_{\alpha(S)}$. Then $loc(p) \in C(S)$.*

Propositions A.0.1 and A.0.2 can be trivially proved by induction on the structure of S. Moreover, we need some properties of $\times$ over $\Sigma(\Pi)$.

Lemma A.0.3 *Let $\sigma_1 = (\ell_1, \prec_1) \in \Sigma(\Pi)$ and $\sigma_2 = (\ell_2, \prec_2) \in \Sigma(\Pi)$. The following facts hold:*

1. *For each* $\sigma = (\ell, \prec) \in \sigma_1 \times \sigma_2$,
   - $\bar{e} \in trigger(\sigma) \cap \bar{\Pi}$ *iff* $\bar{e} \in trigger(\sigma_i)$, *for some* $i \in \{1,2\}$.
   - $e \in trigger(\sigma_i) \cap \Pi$, *for some* $i \in \{1,2\}$, *and* $e \notin trigger(\sigma)$, *then* $e \in action(\sigma_j)$, *for* $i \neq j$.
2. *If* $e \in trigger(\sigma_1)$ *and* $e \in action(\sigma_2)$, *then there exists* $\sigma \in \sigma_1 \times \sigma_2$, *such that* $e \notin trigger(\sigma)$.
3. *If* $e \notin trigger(\sigma_i)$, *for each* $i \in \{1,2\}$, *then there exists* $\sigma \in \sigma_1 \times \sigma_2$, *such that* $e \notin trigger(\sigma)$.

In the following, we exploit the property that, for each $p \stackrel{\alpha}{\mapsto} p'$, if $type(\alpha) = \bar{\epsilon}$, then $action(\alpha) = \emptyset$.

Lemma A.0.4 *Let* S *be a statechart over* Π *and* $p \in \mathcal{R}_{\alpha(S)}$. *If* $p \stackrel{\alpha}{\mapsto} p'$, *then* $loc(p) \stackrel{I,O}{\Longrightarrow} loc(p')$, *for each* $I \subseteq \Pi$, *such that* $I \rightsquigarrow trigger(\alpha)$, *and* $O - trigger(\alpha) = action(\alpha)$. *Moreover,* $type(\alpha) = \epsilon$ *implies* $loc(p) = loc(p')$.

Proof. By induction on the structure of S.

1. Let S be a basic empty statechart. By propositions A.0.1 and A.0.2, $p = S :: S :: nil$ and $loc(p) = \{S\}$. We have $p \stackrel{\alpha}{\mapsto} p$, for $\alpha = \sigma : \bar{\epsilon}$ and $\sigma \in \#(*)$ and $En_{loc(p),I}(\emptyset) = \emptyset$, for each $I \subseteq \Pi$, such that $I \rightsquigarrow trigger(\sigma)$.
2. Let $S : (\{S_1, \ldots, S_n\}, T, in, out, \chi, \delta)$ be a basic non-empty statechart. By propositions A.0.1 and A.0.2, either $p = S :: S_i :: nil$ or $p = S :: rec\, x.S_i :: \sum_{j \in \{1,k\}} a_j.t_j$. In both cases, $loc(p) = \{S, S_i\} = c \in C(S)$.
   - If $p = S :: S_i :: nil$, then $T_i = \emptyset$ and $p' = p$, $\alpha = \sigma : \bar{\epsilon}$, for $\sigma \in \#(*)$. We have $En_{c,I}(\emptyset) = \emptyset$, for each $I \subseteq \Pi$, such that $I \rightsquigarrow trigger(\sigma)$.
   - If $p = S :: rec\, x.S_i :: \sum_{j \in \{1,k\}} a_j.t_j$, then $T_i = \{t_1, \ldots, t_k\}$, where $a_j = lab(t_j)$, for each $j \in \{1,k\}$. We have $p \stackrel{\alpha}{\mapsto} p'$ iff one of the following cases holds.
     (a) $\alpha = lab(t_j) : \epsilon$ and $p' = S :: t_j[rec\, x.S_i :: \sum_{j \in \{1,k\}} a_j.t_j / x]$. By definition of $\alpha(S)$, $loc(p') = \{S, in(t_j)\} = c' \in C(S)$. Since $trigger(lab(t_j)) = trigger(t_j)$ and $action(lab(t_j)) = (action(t_j) - \{nil\}) - trigger(t_j)$, for each $I \subseteq \Pi$, such that $I \rightsquigarrow trigger(lab(t_j))$, $En_{c,I}(\{t_j\}) = \{t_j\}$. Actually, $t_j \in trig(I \cup action(t_j))$. Moreover, $t_j \in En_{c,I}(\emptyset)$. Therefore, $\{t_j\}$ is inseparable and is an admissible step from c to c'.
     (b) $\sigma \in \#(lab(t_1)) \cap \ldots \cap \#(lab(t_k))$ and $\alpha = \sigma : \bar{\epsilon}$, $p' = p$. For each $j \in \{1,k\}$, there exists either $e \in trigger(t_j)$ or $\bar{e} \in trigger(t_j)$, such that either $\bar{e} \in trigger(\sigma)$ or $e \in trigger(\sigma)$, respectively. Therefore, if $I \rightsquigarrow trigger(\sigma)$, $e \notin I$ and $e \in I$, respectively. Since $trig(I) \cap rel(c) = \emptyset$, then $En_{c,I}(\emptyset) = \emptyset$.

3. Let S be a refined statechart $\bigtriangledown(S, \{S_1, \ldots, S_n\})$. By propositions A.0.1 and A.0.2, $loc(p) = \{S, c\}$, where $c \in C(S_i)$, for some $i \in \{1, n\}$. We have three cases:

   (a) $p = S :: S_i :: \sum_{j\in\{1,k\}} a_j.ref(t_j[rec\ x.S_i :: \sum_{j\in\{1,k\}} a_j.t_j/x], \{(S_1, \alpha(S_1)), \ldots, (S_n, \alpha(S_n))\}) \bigtriangledown p''$;

   (b) $p = S :: S_i :: nil \bigtriangledown p''$;

   (c) $p = S :: rec\ x.S_i :: \sum_{j\in\{1,k\}} a_j.ref(t_j, \{(S_1, \alpha(S_1)), \ldots, (S_n, \alpha(S_n))\}) \bigtriangledown \alpha(S_i)$;

   where $rec\ x.S_i :: \sum_{j\in\{1,k\}} a_j.t_j \in \mathcal{R}_{\alpha(S)}$ and $p'' \in \mathcal{R}_{\alpha(S_i)}$. We show only the first case, since the others are similar.

   We have $p \overset{\alpha}{\mapsto} p'$ iff either rule $\bigtriangledown_1$, $\bigtriangledown_2$ or $\bigtriangledown_3$ have been applied. Assume $t = S_i :: \sum_{j\in\{1,k\}} a_j.ref(t_j[rec\ x.S_i :: \sum_{j\in\{1,k\}} a_j.t_j/x], \{(S_1, \alpha(S_1)), \ldots, (S_n, \alpha(S_n))\})$ so that $p = S :: t \bigtriangledown p''$.

   - If rule $\bigtriangledown_1$ has been applied, then $t \overset{\alpha}{\mapsto} t'$ and $type(\alpha) = \epsilon$, $p' = S :: t'$. Since $type(\alpha) = \epsilon$, then $t' = ref(t_j[rec\ x.S_i :: \sum_{j\in\{1,k\}} a_j.t_j/x], \{(S_1, \alpha(S_1)), \ldots, (S_n,\ \alpha(S_n))\})$ and $\alpha = a_j : \epsilon$, for some $j \in \{1, k\}$. Since $rec\ x.S_i :: \sum_{j\in\{1,k\}} a_j.t_j = t_1 \in \mathcal{R}_{\alpha(S)}$, then $t_1 \overset{\alpha}{\mapsto} t_1'$ with $t_1' = t_j[rec\ x.S_i :: \sum_{j\in\{1,k\}} a_j.t_j/x]$ and $t_1' \in \mathcal{R}_{\alpha(S)}$. By propositions A.0.1 and A.0.2, since S is basic, $loc(t_1') = \{S, S_h\} \in C(S)$, where $i \neq h$ and $loc(S :: t') = \{S\} \cup \Delta(S_h)$.
     Since $t_1 \overset{\alpha}{\mapsto} t_1'$, then, by induction hypothesis, there exists $T \subseteq T_i$, such that T is an admissible step from $\{S, S_i\}$ to $\{S, S_h\}$, for each $I \subseteq \Pi$, such that $I \rightsquigarrow trigger(\alpha)$ and $gen(T) - trigger(\alpha) = action(\alpha)$. We have that $En_{loc(p),I}(T) = T$ and T is inseparable, since $En_{loc(p),I}(T) = rel(loc(p)) \cap con(T) \cap triggered(I \cup gen(T)) = (rel(\{S, S_i\}) \cup rel(c)) \cap con(T) \cap triggered(I \cup gen(T)) = rel(\{S, S_i\}) \cap con(T) \cap triggered(I \cup gen(T)) = T$. Actually, $t \in rel(c)$ implies $t \notin con(T)$. Therefore T is an admissible step from $loc(p) = \{S, c\}$, for each $I \subseteq \Pi$, such that $I \rightsquigarrow trigger(\alpha)$ and $gen(T) - trigger(\alpha) = action(\alpha)$ and the reached configuration is $\{S, \Delta(S_h)\} = loc(S :: t')$.
   - $p \overset{\alpha}{\mapsto} p'$ has been obtained by the application of $\bigtriangledown_2$. Hence, $p'' \overset{\alpha}{\mapsto} p'''$ and $p' = S :: t \bigtriangledown p'''$, $type(\alpha) = \epsilon$. By induction hypothesis, there exists $T \subseteq T(S_i)$, such that T is an admissible step from c to $c' \in C(S_i)$, such that $c' = loc(p''')$, with respect to $I \rightsquigarrow trigger(\alpha)$ and $gen(T) - trigger(\alpha) = action(\alpha)$. We have that $En_{loc(p),I}(T) = T$ and T is inseparable, since $En_{loc(p),I}(T) = rel(loc(p)) \cap con(T) \cap triggered(I \cup gen(T)) = (rel(\{S, S_i\}) \cup rel(c)) \cap con(T) \cap triggered(I \cup gen(T)) = rel(c) \cap con(T) \cap triggered(I \cup gen(T)) = T$. Hence, T is an admissible step from $\{S, c\}$ to $\{S, c'\} = loc(p')$.

- $p \overset{\alpha}{\mapsto} p'$ has been obtained by the application of $\bigtriangledown_3$. Hence, $p'' \overset{\alpha}{\mapsto} p''$, $t \overset{\alpha}{\mapsto} t$ and $p' = S :: t \bigtriangledown p'' = p$. Moreover, $type(\alpha) = \bar{\epsilon}$. By repeating the same arguments of the previous cases, by induction hypothesis, $\emptyset$ is an admissible step both from $\{S, S_i\}$ and from c, for each $I \subseteq \Pi$, such that $I \rightsquigarrow trigger(\alpha)$. Hence, $En_{loc(p),I}(\emptyset) = \emptyset$.

4. Let $S : AND(S_1, S_2)$ be an AND statechart. By propositions A.0.1 and A.0.2, $p = S :: p_1 \times p_2$, where $p_i \in \mathcal{R}_{\alpha(S_i)}$ and $loc(p_i) = c_i \in C(S_i)$.

   We have $p \overset{\alpha}{\mapsto} p'$ iff, by rule par, $\{p_i \overset{\alpha_i}{\mapsto} p_i'\}_{i\in\{1,2\}}$, $\alpha \in \alpha_1 \times \alpha_2$, $p' = S :: p_1' \times p_2'$.

   By induction hypothesis, for each $i \in \{1, 2\}$ and $I_i \subseteq \Pi$, such that $I_i \rightsquigarrow trigger(\alpha_i)$, there exists $T_i \subseteq T(S_i)$ such that $En_{c_i,I_i}(T_i) = T_i$ and T_i is inseparable. Moreover, $gen(T_i) - trigger(\alpha_i) = action(\alpha_i)$ and $loc(p_i') = c_i' \in C(S_i)$. Let $c' = \{S\} \cup c_1' \cup c_2' \in C(S)$ and $c = \{S\} \cup c_1 \cup c_2 \in C(S)$.

   We will prove that for each $I \subseteq \Pi$, such that $I \rightsquigarrow trigger(\alpha)$,

   (a) $En_{c,I}(\bigcup_{i\in\{1,2\}} T_i) = \bigcup_{i\in\{1,2\}} T_i$;

   (b) $\bigcup_{i\in\{1,2\}} T_i$ is inseparable.

   By definition, $En_{c,I}(\bigcup_{i\in\{1,2\}} T_i) = \bigcup_{i\in\{1,2\}}(rel(c_i) \cap con(T_i) \cap trig(I \bigcup_{i\in\{1,2\}} gen(T_i)))$. Since for each $i \in \{1, 2\}$, $En_{c_i,I_i}(T_i) = rel(c_i) \cap con(T_i) \cap trig(I_i \cup gen(T_i)) = T_i$, we show that $rel(c_i) \cap con(T_i) \cap trig(I_i \cup gen(T_i)) = rel(c_i) \cap con(T_i) \cap trig(I \bigcup_{i\in\{1,2\}} gen(T_i))$.

   In the following, assume $\alpha_i = \sigma_i : \tau_i$, $\alpha = \sigma : \tau$, for each $i \in \{1, 2\}$. Let $t \in rel(c_i) \cap con(T_i) \cap trig(I_i \cup gen(T_i)) = T_i$, for each $I_i \rightsquigarrow trigger(\sigma_i)$. Suppose $t \notin rel(c_i) \cap con(T_i) \cap trig(I \bigcup_{i\in\{1,2\}} gen(T_i))$ for some $I \rightsquigarrow trigger(\sigma)$. One of the following cases holds:

   (a) $e \in trigger(t)$ and $e \in I_i \cup gen(T_i)$ and $e \notin I \bigcup_{i\in\{1,2\}} gen(T_i)$ for each $I_i \rightsquigarrow trigger(\sigma_i)$;

   (b) $\bar{e} \in trigger(t)$ and $e \notin I_i \cup gen(T_i)$ and $e \in I \bigcup_{i\in\{1,2\}} gen(T_i)$ for each $I_i \rightsquigarrow trigger(\sigma_i)$.

   (a) $e \notin I$, $e \in I_i$, $e \notin gen(T_i)$ for all $i \in \{1, 2\}$. Since $e \in I_i$ there are two possibilities; either $e \in trigger(\sigma_i)$ or $e, \bar{e} \notin trigger(\sigma_i)$. In the first case, $e \notin I$ implies $e \notin trigger(\sigma)$. By lemma A.0.3 $e \in action(\sigma_j)$ for $j \neq i$ and $e \in gen(T_j)$. This contradicts $e \notin gen(T_i)$, for each $i \in \{1, 2\}$.

   In the second case, $e, \bar{e} \notin trigger(\sigma_i)$. Consider $I_i' = I_i - e$. We have $I_i' \rightsquigarrow trigger(\sigma_i)$ and $t \in trig(I_i' \cup gen(T_i)) = T_i$. Then $e \in gen(T_i)$, which contradicts $e \notin gen(T_i)$, for each $i \in \{1, 2\}$.

   (b) $\bar{e} \in trigger(t)$ and $e \notin I_i \cup gen(T_i)$ and $e \in I \bigcup_{i\in\{1,k\}} gen(T_i)$. Since $t \in T_i$, $\bar{e} \in trigger(t)$ implies $\bar{e} \in trigger(\sigma_i)$ and, by lemma A.0.3,

$\bar{e} \in trigger(\sigma)$. Therefore, $I \rightsquigarrow trigger(\sigma)$ iff $e \notin I$. Thus $e \in gen(T_j)$ with $j \neq i$ and $e \in \ell_j$, where $\sigma_h = (\ell_h, \prec_h)$ for all $h \in \{1,2\}$. We have $e, \bar{e} \in \ell_1 \cup \ell_2$, and this contradicts the consistency of σ.

Let $t \in rel(c_i) \cap con(T_i) \cap trig(I \bigcup_{i\in\{1,2\}} gen(T_i))$. Suppose $t \notin rel(c_i) \cap con(T_i) \cap trig(I_i \cup gen(T_i)) = T_i$, for each $I_i \rightsquigarrow trigger(\sigma_i)$. For each $I_i \rightsquigarrow trigger(\sigma_i)$, one of the following cases holds:

(a) $e \in trigger(t)$ and $e \in I \bigcup_{i\in\{1,2\}} gen(T_i)$ and $e \notin I_i \cup gen(T_i)$;

(b) $\bar{e} \in trigger(t)$ and $e \notin I \bigcup_{i\in\{1,2\}} gen(T_i)$ and $e \in I_i \cup gen(T_i)$.

(a) $e \in trigger(t)$ and $e \in I \bigcup_{i\in\{1,2\}} gen(T_i)$ and $e \notin I_i \cup gen(T_i)$. Since $e \notin I_i$ there are two cases, either $\bar{e} \in trigger(\sigma_i)$ or $\bar{e}, e \notin trigger(\sigma_i)$.

In the first case, $\bar{e} \in trigger(\sigma_i)$ implies $\bar{e} \in trigger(\sigma)$, so that $e \notin I$ and $e \in gen(T_j)$, for $j \neq i$. We have $e \in \ell_j$, where $\sigma_h = (\ell_h, \prec_h)$ for $h \in \{1,2\}$, and $e, \bar{e} \in \ell_1 \cup \ell_2$. This contradicts the consistency of σ.

In the second case, $\bar{e}, e \notin trigger(\sigma_i)$. Consider $I_i' = I_i \cup e$. We have $I_i' \rightsquigarrow trigger(\sigma_i)$. Thus $t \notin trig(I_i' \cup gen(T_i)) \cap con(T_i) \cap rel(c_i) = T_i$. Therefore, either $e' \in trigger(t)$ and $e' \in I \bigcup_{i\in\{1,2\}} gen(T_i))$ and $e' \notin I_i' \cup gen(T_i)$ or $\bar{e}' \in trigger(t)$ and $e' \notin I \bigcup_{i\in\{1,2\}} gen(T_i))$ and $e' \in I_i' \cup gen(T_i)$. Moreover $e \neq e'$. We repeat the same argument.

(b) $\bar{e} \in trigger(t)$ and $e \notin I \bigcup_{i\in\{1,2\}} gen(T_i)$ and $e \in I_i \cup gen(T_i)$. We have $e \in I_i$ and one of the following cases holds, either $e \in trigger(\sigma_i)$ or $\bar{e}, e \notin trigger(\sigma_i)$. In the first case, $e \in trigger(\sigma_i)$ and $e \notin I$ implies $e \notin trigger(\sigma)$. By lemma A.0.3, $e \in action(\sigma_j)$. We have $e \in gen(T_j)$, which contradicts $e \notin \bigcup_{i\in\{1,2\}} gen(T_i)$. In the second case, consider $I_i' = I_i - e$. We have $I_i' \rightsquigarrow trigger(\sigma_i)$. Thus $t \notin trig(I_i' \cup gen(T_i)) \cap con(T_i) \cap rel(c_i) = T_i$. Therefore, either $e' \in trigger(t)$ and $e' \in I \bigcup_{i\in\{1,2\}} gen(T_i))$ and $e' \notin I_i' \cup gen(T_i)$ or $\bar{e}' \in trigger(t)$ and $e' \notin I \bigcup_{i\in\{1,2\}} gen(T_i))$ and $e' \in I_i' \cup gen(T_i)$. Moreover $e \neq e'$. We repeat the same argument.

Since $trigger(t)$ is finite, $t \in rel(c_i) \cap con(T_i) \cap trig(I_i \cup gen(T_i))$, for each $I_i \rightsquigarrow trigger(\sigma_i)$.

We have proved that $En_{c,I}(\bigcup_{i\in\{1,2\}} T_i) = \bigcup_{i\in\{1,2\}} T_i = T$, for each $I \rightsquigarrow trigger(\sigma)$. It remains to show that T is inseparable.

Assume T is separable, i.e. there exists $T' \subset T$ such that $En_{c,I}(T') \cap (T - T') = \emptyset$. For each $t \in T - T'$, $t \notin trig(I \cup gen(T'))$, while $t \in trig(I \cup gen(T))$. There are two cases:

(a) $e \in trigger(t)$ and $e \notin gen(T')$ and $e \in gen(T)$;

(b) $\bar{e} \in trigger(t)$ and $e \in gen(T')$ and $e \notin gen(T)$

The second case is impossible since $T' \subset T$. In the first case, $e \notin I, e \in gen(T)$ while $e \notin gen(T')$ implies $e \in gen(T - T')$. Since $e \notin I$ and $I \rightsquigarrow trigger(\sigma)$, either $\bar{e} \in trigger(\sigma)$ or $e, \bar{e} \notin trigger(\sigma)$.

Assume $\sigma_i = (\ell_i, \prec_i)$ and $\sigma = (\ell, \prec)$. If $\bar{e} \in trigger(\sigma)$ and $e \in gen(T - T')$, we have $e \in \ell_j$ for some $j \in \{1, 2\}$ and this contradicts the consistency of ℓ. Hence, for each $t \in (T - T')$, there exists $e \in trigger(t)$, such that $e \notin I$, $e \notin gen(T')$, $e \in gen(T - T')$ and $e, \bar{e} \notin trigger(\sigma)$, while $e \in action(\sigma)$.

Let us assume $t_1 \in (T - T')$ and $e_1 \in trigger(t_1)$, which satisfies the previous properties. Since $e_1 \in action(\sigma)$, $C_1 \prec \{e_1\}$ for some $C_1 \in 2^{\Pi \cup \bar{\Pi}}$. By definition of labels, there exists $j \in \{1, 2\}$, such that $C_1 \prec_j \{e_1\}$ and there exists $t_2 \in T_j$ such that $C_1 = trigger(t_2)$ and $e_1 \in action(t_2)$. Since $e_1 \notin gen(T')$ then $t_2 \notin T'$ and $t_2 \in (T - T')$. Therefore there exists $e_2 \in C_1$, such that $e_2, \bar{e}_2 \notin trigger(\sigma)$, $e_2 \notin I$, $e_2 \notin gen(T')$, $e_2 \in gen(T - T')$ and $e_2 \in action(\sigma)$. By repeating the same argument, since T is finite, we have $C_1 \prec \{e_1\}, \ldots, C_n \prec \{e_n\}$, where for each $i \in \{2, n\}$ $e_i \in C_{i-1}$ and $e_j \in C_n$ for some $j \in \{1, n\}$. By transitivity of $\prec$, $\prec$ is reflexive.

We have proved that $T = \bigcup_{i \in \{1,2\}} T_i$ is an admissible step with respect to $I \rightsquigarrow trigger(\alpha)$ and c. By induction hypothesis the configuration reached via this step is c'. We have $c \stackrel{I,O}{\Longrightarrow} c'$, for $O = \bigcup_{i \in \{1,2\}} gen(T_i)$. We show that $O - trigger(\alpha) = action(\alpha)$.

Assume $gen(\bigcup_{i \in \{1,2\}} T_i) - trigger(\sigma) \neq action(\sigma)$. There are two cases:

(a) $e \in gen(\bigcup_{i \in \{1,2\}} T_i) - trigger(\sigma)$ and $e \notin action(\sigma)$;
(b) $e \in action(\sigma)$ and $e \notin gen(\bigcup_{i \in \{1,2\}} T_i) - trigger(\sigma)$.

In the first case, since $e \in gen(\bigcup_{i \in \{1,2\}} T_i)$ then $e \in \ell$. However, $e \notin action(\sigma)$ and $e \notin trigger(\sigma)$ since $e \in gen(\bigcup_{i \in \{1,k\}} T_i) - trigger(\sigma)$. This is a contradiction. In the second case, $e \in action(\sigma)$ and $e \notin gen(\bigcup_{i \in \{1,2\}} T_i) - trigger(\sigma)$. Since $e \in action(\sigma)$ implies $e \in \bigcup_{i \in \{1,2\}} gen(T_i)$, $e \in trigger(\sigma)$. However, $e \in action(\sigma)$ and $e \in trigger(\sigma)$ is a contradiction.

Therefore, $\bigcup_{i \in \{1,2\}} (gen(T_i)) - trigger(\sigma) = action(\sigma)$.

Moreover, $type(\alpha) = \bar{\epsilon}$ iff $type(\alpha_i) = \bar{\epsilon}$, for each $i \in \{1, 2\}$. However, by induction hypothesis, $type(\alpha_i) = \bar{\epsilon}$ implies $c_i = c'_i$. Hence, $type(\alpha) = \bar{\epsilon}$ implies $c = c'$, and this completes the proof.

■

Lemma A.0.5 *Let S be a statechart over Π, $I \subseteq \Pi$, $c_N \in C(S)$, such that $\{c_i \stackrel{I_i,O_i}{\Longrightarrow} c_{i+1}\}_{i \in \{1,N\}}$, where $c_1 = \Delta(S)$.*

There exist $\{p_i \in \mathcal{R}_{\alpha(S)}\}_{i \in \{1,N\}}$, such that $\{p_i \stackrel{\alpha_i}{\Longrightarrow} p_{i+1}\}_{i \in \{1,N\}}$, $p_1 = \alpha(S)$, and $loc(p_i) = c_i$, $I_i \rightsquigarrow trigger(\alpha_i)$ and $O_i - trigger(\alpha_i) = action(\alpha_i)$, for each $i \in \{1, N\}$. Moreover, if $c_i = c_{i+1}$, then $type(\alpha_i) = \bar{\epsilon}$.

Proof. By induction on the structure of S.

1. Let S be a basic empty statechart. Then $c_i = \{S\}$, for each $i \in \{1, N\}$ and $O_i = \emptyset$, $I_i \subseteq \Pi$. By proposition A.0.1, $\alpha(S) = S :: S :: nil$ and $\mathcal{R}_{\alpha(S)} = \{\alpha(S)\}$. We have $\alpha(S) \stackrel{\sigma:\bar{\epsilon}}{\mapsto} \alpha(S)$, for each $\sigma \in \#(*)$. By definition, for each $I_i \subseteq \Pi$, there exists $\sigma \in \#(*)$, such that $I_i \rightsquigarrow trigger(\sigma)$.

2. Let $S : (\{S_1, \ldots, S_k\}, T, in, out, \chi, \delta)$ be a basic non-empty statechart. The proof is by induction on N. If $N = 1$, then, by propositions A.0.1 and A.0.2, $\alpha(S) = S :: rec\ x.\delta(S).t$ and $loc(\alpha(S)) = \Delta(S)$ by definition.

   If $N > 1$, then by induction hypothesis, there exist $\{p_i \in \mathcal{R}_{\alpha(S)}\}_{i\in\{1,N-1\}}$, such that $\{p_i \stackrel{\alpha_i}{\Longrightarrow} p_{i+1}\}_{i\in\{1,N-1\}}$ and $loc(p_i) = c_i$.

   Suppose $c_{N-1} = \{S, S_i\}$, for $i \in \{1, k\}$. If $c \stackrel{I,O}{\Longrightarrow} c'$, there exists $T \subseteq T(S)$, such that $En_{c_{N-1},I_{N-1}}(T) = T$ and T is inseparable. There are two cases, either $T = \emptyset$ or $T \neq \emptyset$.

   (a) If $T = \emptyset$, then $c_N = c_{N-1}$, $O_{N-1} = \emptyset$. Since $En_{c_{N-1},I_{N-1}}(\emptyset) = rel(c_{N-1}) \cap con(\emptyset) \cap trig(I_{N-1}) = \emptyset$, either $t \notin trig(I_{N-1})$, for each $t \in T_i$, or $T_i = \emptyset$.
   If $T_i = \emptyset$, then, by propositions A.0.1 and A.0.2, $p_{N-1} = S :: S_i :: nil$. By definition, there exists $\sigma \in \#(*)$ such that $p_{N-1} \stackrel{\sigma:\bar{\epsilon}}{\mapsto} p_{N-1}$ and $I_{N-1} \rightsquigarrow trigger(\sigma)$.
   If $T_i = \{t_1, \ldots, t_n\}$ and $trig(I_{N-1}) \cap rel(c_{N-1}) = \emptyset$, for each $i \in \{1, n\}$ one of the following cases holds:

   - $e_i \in \Pi, e_i \notin I_{N-1}$ and $e_i \in trigger(t_i)$;
   - $e_i \in \bar{\Pi}, \bar{e}_i \in I_{N-1}$ and $e_i \in trigger(t_i)$.

   Hence, there exists $\sigma \in \#(lab(t_i))$, for each $i \in \{1, n\}$, such that $I_{N-1} \rightsquigarrow trigger(\sigma)$.
   By propositions A.0.2 and A.0.1, $p_{N-1} = S :: t_{N-1}$, where $t_{N-1} = rec\ x.S_i :: \sum_{j\in\{1,n\}} lab(t_j).t'_j$. Since $S_i :: lab(t_j).t'_j[t_{N-1}/x] \stackrel{\sigma:\bar{\epsilon}}{\mapsto} S_i :: lab(t_j).t'_j[t_{N-1}/x]$, for $j \in \{1, n\}$, then by rules $choice_2$ and rec_2, $p_{N-1} \stackrel{\sigma:\bar{\epsilon}}{\mapsto} p_{N-1}$.

   (b) If $T = \{t\}$, then $c_N = \{S, in(t)\}$ and $O_{N-1} = action(t)$. We have $En_{c,I_{N-1}}(\{t\}) = \{t\}$ and $I_{N-1} \rightsquigarrow trigger(lab(t))$. By propositions A.0.1 and A.0.2, $p_{N-1} = S :: t_{N-1}$, where $t_{N-1} = rec\ x.S_i :: \sum_{t\in T_i} lab(t).t_{in(t)}$.
   We have $S_i :: lab(t).t_{in(t)}[t_{N-1}/x] \stackrel{lab(t):\epsilon}{\mapsto} S_i :: t_{in(t)}[t_{N-1}/x]$.
   By rule rec_1, $p_{N-1} \stackrel{lab(t):\epsilon}{\mapsto} S :: S_i :: t_{in(t)}[t_{N-1}/x] = p_N$. By definition of $\alpha(S)$, $loc(p_N) = \{S, in(t)\}$ and $gen(t) - trigger(lab(t)) = action(lab(t))$.

3. Let S be a refined statechart, $\bigtriangledown(S, \{S_1, \ldots, S_k\})$. The proof is by induction on N. If $N = 1$, then $c_N = \Delta(S)$ and $loc(\alpha(S)) = \Delta(S)$, by definition.

If $N > 1$, then , by induction hypothesis, there exists $\{p_i \in \mathcal{R}_{\alpha(S)}\}_{i\in\{1,N-1\}}$, such that $p_i \overset{\alpha_i}{\mapsto} p_{i+1}$, and $loc(p_i) = c_i$, for each $i \in \{1, N-1\}$. Since $c_{N-1} \overset{I_{N-1},O_{N-1}}{\Longrightarrow} c_N$, there exists $T \subseteq T(\bigtriangledown(S, \{S_1, \ldots, S_k\}))$, such that $En_{c_{N-1},I_{N-1}}(T) = T$, $O_{N-1} = gen(T)$, and T is inseparable. Assume $c_{N-1} = \{S, c\}$, with $c \in C(S_i)$. By propositions A.0.1 and A.0.2, since $loc(p_{N-1}) = c_{N-1}$, $p_{N-1} = S :: t_{N-1} \bigtriangledown p'_{N-1}$, where $loc(t_{N-1}) = \{S_i\}$ and $loc(p'_{N-1}) = c$. There are two cases, either $T = \emptyset$ or $T \neq \emptyset$.

(a) If $T = \emptyset$, then $c_N = c_{N-1}$. We have $En_{c_{N-1},I_{N-1}}(\emptyset) = \emptyset$ iff $En_{c,I_{N-1}}(\emptyset) = \emptyset$ and $En_{S\cup S_i,I_{N-1}}(\emptyset) = \emptyset$. By induction hypothesis, there exist $S :: t_{N-1} \overset{\alpha_{N-1}}{\mapsto} S :: t_{N-1}$ and $p'_{N-1} \overset{\alpha_{N-1}}{\mapsto} p'_{N-1}$, where $I_{N-1} \rightsquigarrow trigger(\alpha_{N-1})$ and $type(\alpha_{N-1}) = \bar{\epsilon}$. Since $type(\alpha) = \bar{\epsilon}$, by rule $\bigtriangledown_3$, $p_{N-1} \overset{\alpha}{\mapsto} p_{N-1}$.

(b) If $T \neq \emptyset$, either $T \subseteq T(S_i)$ or $T \subseteq T(S)$. Suppose $T \subseteq T(S_i)$, we have $En_{c,I_{N-1}}(T) = T$ and T is inseparable. By induction hypothesis, there exists $p'_{N-1} \overset{\alpha_{N-1}}{\mapsto} p'_N$ and $I_{N-1} \rightsquigarrow trigger(\alpha_{N-1})$, $gen(T) - trigger(\alpha_{N-1}) = action(\alpha_{N-1})$, $loc(p'_N) = \{c'\}$, where $c_N = \{S, c'\}$ with $c' \in C(S_i)$. Moreover, $type(\alpha_{N-1}) = \epsilon$.
Since $type(\alpha) = \epsilon$, by rule $\bigtriangledown_2$, $p_{N-1} \overset{\alpha}{\mapsto} S :: t_{N-1} \bigtriangledown p'_N = p_N$ and $loc(p_N) = c_N$. The other case is similar.

4. Let S : AND(S_1, S_2). The proof is by induction on N. If $N = 1$, then $c_N = \{S\} \cup \Delta(S_1) \cup \Delta(S_2)$. We have $\alpha(S) = S :: \alpha(S_1) \times \alpha(S_2)$. By induction hypothesis, $loc(\alpha_i) = \Delta(S_i)$, so that $loc(\alpha(S)) = c_N$.

If $N > 1$, then, by induction hypothesis, there exists $\{p_i \in \mathcal{R}_{\alpha(S)}\}_{i\in\{1,N-1\}}$, such that $p_i \overset{\alpha_i}{\mapsto} p_{i+1}$ and $loc(p_i) = c_i$, where $c_i = S \cup c_{i,1} \cup c_{i,2}$ and $c_{i,j} \in C(S_j)$ for each $j \in \{1, 2\}$. By propositions A.0.1 and A.0.2, since $loc(p_{N-1}) = c_{N-1}$, $p_{N-1} = S :: p_{N-1,1} \times p_{N-1,2}$ and $loc(p_{N-1,j}) = c_{N-1,j}$, for each $j \in \{1, 2\}$.

Since $c_{N-1} \overset{I_{N-1},O_{N-1}}{\Longrightarrow} c_N$, there exists $T \subseteq T(S)$, such that $En_{c_{N-1},I_{N-1}}(T) = T$ and T is not separable. By definition $T = T_1 \cup T_2$, with $T_j \subseteq T(S_j)$.

There are several cases, either $T_1 = T_2 = \emptyset$ or $T_1 = \emptyset$ and $T_2 \neq \emptyset$ or $T_2 = \emptyset$ and $T_1 \neq \emptyset$ or $T_1 \neq \emptyset$ and $T_2 \neq \emptyset$.

- Suppose $T_1 = T_2 = \emptyset$.

Therefore, $c_N = c_{N-1}$ and $O_{N-1} = \emptyset$. Since $En_{c_{N-1},I_{N-1}}(\emptyset) = \emptyset$, then for each $j \in \{1, 2\}$, $En_{c_{N-1,j},I_{N-1}}(\emptyset) = \emptyset$. By induction hypothesis, for each $j \in \{1, 2\}$, $p_{N-1,j} \overset{\alpha_{N-1,j}}{\mapsto} p_{N-1,j}$ and $I_{N-1} \rightsquigarrow trigger(\alpha_{N-1,j})$, $type(\alpha_{N-1,j}) = \bar{\epsilon}$. We have two cases:

(a) If $\alpha_{N-1,1} \times \alpha_{N-1,2} \neq \emptyset$, by rule par, $p_{N-1} \overset{\alpha_{N-1}}{\mapsto} p_{N-1}$, where $\alpha_{N-1} \in \alpha_{N-1,1} \times \alpha_{N-1,2}$. Moreover, $I_{N-1} \rightsquigarrow trigger(\alpha_{N-1})$. Since $type(\alpha_{N-1,i}) =$

$\bar{\epsilon}$, then $action(\alpha_{N-1,i}) = \emptyset$, so that by lemma A.0.3, $trigger(\alpha_{N-1}) = trigger(\alpha_{N-1,1}) \cup trigger(\alpha_{N-1,2})$.

(b) If $\alpha_{N-1,1} \times \alpha_{N-1,2} = \emptyset$, $e \in trigger(\alpha_{N-1,1})$, $\bar{e} \in trigger(\alpha_{N-1,2})$. However, $I_{N-1} \rightsquigarrow trigger(\alpha_{N-1,j})$, for each $j \in \{1,2\}$, so that $e \in I_{N-1}$ and $e \notin I_{N-1}$, which is a contradiction.

- Suppose $T_1 = \emptyset$, $T_2 \neq \emptyset$.

Therefore, $En_{c_{N-1},I_{N-1}}(T_2) = rel(c_{N-1}) \cap con(T_2) \cap trig(I_{N-1} \cup gen(T_2)) = T_2$ and $c_N = \{S\} \cup c_{N,2} \cup c_{N-1,1}$. We have $rel(c_{N-1,1}) \cap trig(I_{N-1} \cup gen(T_2)) = \emptyset$ and $En_{c_{N-1,2},I_{N-1}}(T_2) = rel(c_{N-1,2}) \cap trig(I_{N-1} \cup gen(T_2)) \cap con(T_2) = T_2$. Moreover, for each $T' \subset T_2$ $En_{c_{N-1,2},I}(T') \cap T_2 - T' \neq \emptyset$, since $En_{c_{N-1},I_{N-1}}(T') \cap T_2 - T' \neq \emptyset$. Therefore, T_2 is an admissible step from $c_{N-1,2}$ to $c_{N,2}$. By induction hypothesis $p_{N-1,2} \overset{\alpha_{N-1,2}}{\mapsto} p_{N,2}$ and $I_{N-1} \rightsquigarrow trigger(\alpha_{N_1,2})$, $loc(p_{N,2}) = c_{N,2}$, $type(\alpha_{N-1,2}) = \epsilon$.

Consider $I' = I_{N-1} \cup gen(T_2)$. Since $rel(c_{N-1,1}) \cap trig(I_{N-1} \cup gen(T_2)) = \emptyset$, $En_{c_{N-1,1},I'}(\emptyset) = \emptyset$. Therefore, $\emptyset$ is an admissible step from $c_{N-1,1}$ w.r.t. I'. By induction hypothesis, $p_{N-1,1} \overset{\alpha_{N-1,1}}{\mapsto} p_{N-1,1}$ and $I' \rightsquigarrow trigger(\alpha_{N-1,1})$, $type(\alpha_{N-1,1}) = \bar{\epsilon}$. We have two cases. In the following, let $\alpha_{N-1} = \sigma : \tau$, $\alpha_{N-1,i} = \sigma_i : \tau_i$ and $\sigma_i = (\ell_i, \prec_i)$, $\sigma = (\ell, \prec)$.

(a) If $\alpha_{N-1,1} \times \alpha_{N-1,2} \neq \emptyset$, by applying par, $p_{N-1} \overset{\alpha_{N-1}}{\mapsto} p_N$, for $\alpha_{N-1} \in \alpha_{N-1,1} \times \alpha_{N-1,2}$ and $p_N = S :: p_{N,1} \times p_{N,2}$. We have $loc(p_N) = c_N$. We show that there exists $\alpha_{N_1} \in \alpha_{N-1,1} \times \alpha_{N-1,2}$, such that $I_{N-1} \rightsquigarrow trigger(\alpha_{N-1})$.

Suppose $I_{N-1} \not\rightsquigarrow trigger(\sigma)$, for all $\sigma \in \sigma_1 \times \sigma_2$. Then, for each $\sigma \in \sigma_1 \times \sigma_2$ one of the following cases holds:

i. $e \in trigger(\sigma)$ and $e \notin I - N - 1$.

ii. $\bar{e} \in trigger(\sigma)$ and $e \in I_{N-1}$.

In the second case by lemma A.0.3, $\bar{e} \in trigger(\sigma_i)$ for some $i \in \{1,2\}$. However, $I - N - 1 \subseteq I'$, so that $e \in I'$. Therefore, either $I' \not\rightsquigarrow trigger(\sigma_1)$ or $I - N - 1 \not\rightsquigarrow trigger(\sigma_2)$.

In the first case, since $type(\alpha_{N-1,1}) = \bar{\epsilon}$, then $\prec_1 = \emptyset$ and $\prec = \prec_2$. Therefore, by lemma A.0.3, either $e \in trigger(\sigma_1)$ or $e \in trigger(\sigma_2)$. If $e \in trigger(\sigma_2)$ then $I_{N-1} \not\rightsquigarrow trigger(\sigma_2)$. If $e \in trigger(\sigma_1)$, since $I' \rightsquigarrow trigger(\sigma_1)$, then $e \in gen(T_2)$. Since $gen(T_2) - trigger(\sigma_2) = action(\sigma_2)$, $e \in action(\sigma_2)$, i.e. there exists $C \in 2^{\Pi \cup \bar{\Pi}}$, such that $C \prec_2 \{e\}$. However, $\prec = \prec_2$ implies $C \prec \{e\}$ and this contradicts $e \in trigger(\sigma)$.

(b) If $\alpha_{N-1,_1} \times \alpha_{N-1,2} = \emptyset$, there exist $e, \bar{e}$, such that either $e \in \ell_1$ and $\bar{e} \in \ell_2$ or vice-versa. Let $e \in \ell_1$ and $\bar{e} \in \ell_2$. Since $action(\sigma_1) = \emptyset$, $e \in trigger(\sigma_1)$. Then, $I' \rightsquigarrow trigger(\sigma_1)$ and $e \in I'$. If $e \in I_{N-1}$, then

$I - N - 1 \not\rightsquigarrow trigger(\sigma_2)$, since $\bar{e} \in trigger(\sigma_2)$. If $e \in gen(T_2)$, $gen(T_2) - trigger(\sigma_2) = action(\sigma_2)$. However, $e \notin trigger(\sigma_2), e \notin action(\sigma_2)$, since $\bar{e} \in \ell_2$. This contradicts $e \in gen(T_2)$. Let $\bar{e} \in \ell_1$ and $e \in \ell_2$. Since $I' \rightsquigarrow trigger(\sigma_1)$, $e \notin I'$, i.e. $e \notin I_{N-1}$ and $e \notin gen(T_2)$. However, $e \in \ell_2$ implies that either $e \in trigger(\sigma_2)$ or $e \in action(\sigma_2)$. In the first case, this contradicts $I_{N-1} \rightsquigarrow trigger(\sigma_2)$, while in the second case $gen(T_2) - trigger(\sigma_2) = action(\sigma_2)$ implies $e \in gen(T_2)$.

There exists $\alpha_{N-1} \in \alpha_{N-1,1} \times \alpha_{N-1,2}$, such that $I_{N-1} \rightsquigarrow trigger(\alpha_{N-1})$. By induction hypothesis $O_{N-1} = gen(T_2)$ and $gen(T_2) - trigger(\alpha_{N-1}) = action(\alpha_{N-1})$. Since $type(\alpha_{N-1,2}) = \epsilon$, then $type(\alpha_{N-1}) = \epsilon$, and this completes the case.

- Suppose $T_1 \neq \emptyset$, $T_2 \neq \emptyset$.

Therefore, $En_{c_{n-1},I_{N-1}}(T_2 \cup T_1) = rel(c_{N-1}) \cap con(T_2 \cup T_1) \cap trig(I_{N-1} \cup gen(T_2) \cup gen(T_1)) = T_2 \cup T_1$ and $c_N = \{S\} \cup c_{N,1} \cup c_{N,2}$. We have $rel(c_{N-1,1}) \cap con(T_1) \cap trig(I_{N-1} \cup gen(T_2) \cup gen(T_1)) = T_1$ and $rel(c_{N-1,2}) \cap trig(I_{N-1} \cup gen(T_2) \cup gen(T_1)) \cap con(T_2) = T_2$.

Consider $I' = I_{N-1} \cup gen(T_1) \cup gen(T_2)$. We have $En_{c_{N-1,j},I'}(T_j) = rel(c_j) \cap con(T_j) \cap trig(I_{N-1} \cup gen(T_2) \cup gen(T_1)) = T_j$, for each $j \in \{1,2\}$. Moreover, for each $T'_j \subset T_j$, $En_{c_{N-1,j},I'}(T'_j) \cap (T_j - T'_j) \neq \emptyset$, since $En_{c_{N_1},I'}(T'_j) = T_j$. Therefore, T_j is an admissible step from $c_{N-1,j}$ with input I', for each $j \in \{1,2\}$.

By induction hypothesis, there exists $p_{N-1,j} \overset{\alpha_{N-1,j}}{\mapsto} p_{N,j}$ such that $I' \rightsquigarrow trigger(\alpha_{N_1,j})$ and $gen(T_j) - trigger(\alpha_{N-1,j}) = action(\alpha_{N-1,j})$, $loc(p_{N,j}) = c_{N,j}$, for each $j \in \{1,2\}$.

In the following, assume $\alpha_{N-1} = \sigma : \tau$, $\alpha_{N-1,j} = \sigma_j : \tau_j$. We have two cases, either $\alpha_{N-1,1} \times \alpha_{N-1,2} \neq \emptyset$ or $\alpha_{N-1,1} \times \alpha_{N-1,2} = \emptyset$.

Suppose $\sigma_1 \times \sigma_2 = \emptyset$, for each σ_1, σ_2, where $\sigma_j = (\ell_j, \prec_j)$. Either $\bar{e} \in trigger(\sigma_1)$ and $e \in \ell_2$, or vice-versa. Since $I' \rightsquigarrow trigger(\sigma_1), trigger(\sigma_2)$, then $e \in action(\sigma_2)$. This implies $e \in gen(T_2)$ and $e \in I'$. Therefore $I' \not\rightsquigarrow trigger(\sigma_1)$, which is a contradiction.

Suppose $\sigma_1 \times \sigma_2 \neq \emptyset$. By applying rule *par*, $p_{N-1} \overset{\alpha_{N-1}}{\mapsto} p_N$, with $\alpha_{N-1} \in \alpha_{N-1,1} \times \alpha_{N-1,2}$, and $p_N = S :: p_{N,1} \times p_{N,2}$. Suppose $I_{N-1} \not\rightsquigarrow trigger(\alpha_{N-1})$, for each $\alpha_{N-1} \in \alpha_{N-1,1} \times \alpha_{N-1,2}$ and for all $\alpha_{N-1,1}$, $\alpha_{N-1,2}$, such that $p_{N-1,i} \overset{\alpha_{N-1,i}}{\mapsto} p_{N,i}$, with $I' \rightsquigarrow trigger(\alpha_{N-1,i})$ and $gen(T_i) - trigger(\alpha_{N-1,i}) = action(\alpha_{N-1,i})$.

Note that $I' \rightsquigarrow trigger(\sigma)$, for each $\sigma \in \sigma_1 \times \sigma_2$ and for each σ_1 and σ_2. In fact, if $I' \not\rightsquigarrow trigger(\sigma)$, then either $\bar{e} \in trigger(\sigma)$ and $e \in I'$ or $e \in trigger(\sigma)$ and $e \notin I'$. In the first case, by lemma A.0.3, $\bar{e} \in trigger(\sigma_j)$ for some $j \in \{1,2\}$. Therefore $I' \not\rightsquigarrow trigger(\sigma_j)$, for some $j \in \{1,2\}$, which is a contradiction. If $e \in trigger(\sigma)$ and $e \notin I'$, there are two cases. Either $e \in trigger(\sigma_j)$ for some

$j \in \{1,2\}$, which contradicts $I' \rightsquigarrow trigger(\sigma_j)$, or $e \notin trigger(\sigma_j)$, for each $j \in \{1,2\}$. However, in this case, $e \in action(\sigma_j)$, for some $j \in \{1,2\}$ and, by induction hypothesis, $e \in gen(T_1 \cup T_2)$. We have $e \in I'$, which contradicts the hypothesis.

Since $I_{N-1} \not\rightsquigarrow trigger(\sigma)$, for each σ, there are two possibilities:

(a) $\bar{e} \in trigger(\sigma)$ and $e \in I_{N-1}$, $e \notin I'$;

(b) $e \in trigger(\sigma)$ and $e \notin I_{N-1}$, $e \in I'$.

In the first case, since $\bar{e} \in trigger(\sigma)$, then $\bar{e} \in trigger(\sigma_j)$, for some $j \in \{1,2\}$, by lemma A.0.3. This contradicts $I' \rightsquigarrow trigger(\sigma_j)$, for all $j \in \{1,2\}$, since $e \notin I'$ and $I_{N-1} \subseteq I'$.

Hence, for each σ the first case holds. We will show that this leads to a contradiction.

For each σ there are two cases, either $e \notin trigger(\sigma_j)$, for each $j \in \{1,2\}$ or $e \in trigger(\sigma_j)$, for some $j \in \{1,2\}$. In the first case, by lemma A.0.3, there exists $\sigma' \in \sigma_1 \times \sigma_2$, where $e \notin trigger(\sigma')$. Moreover $I' - \{e\} \rightsquigarrow trigger(\sigma')$, since $I' \rightsquigarrow trigger(\sigma')$ and σ' is consistent. Therefore, there exists $e' \in trigger(\sigma')$ where $e' \notin I_{N-1}$ and $e' \neq e$. We repeat the same argument with respect to $I' - \{e\}$.

If $e \in trigger(\sigma_j)$, for some $j \in \{1,2\}$, we can distinguish two cases, either $e \in trigger(\sigma_j), gen(T_j)$ for some $j \in \{1,2\}$ or $e \in trigger(\sigma_j), e \in gen(T_h)$ and $e \notin trigger(\sigma_h)$, $e \notin gen(T_j)$ for $h \neq j$.

Suppose $e \in trigger(\sigma_j), e \in gen(T_h)$ and $e \notin trigger(\sigma_h)$, $e \notin gen(T_j)$ for $h \neq j \in \{1,2\}$. By lemma A.0.3, there exists $\sigma' \in \sigma_1 \times \sigma_2$, where $e \notin trigger(\sigma')$ and $I' - \{e\} \rightsquigarrow trigger(\sigma')$, since $I' \rightsquigarrow trigger(\sigma')$. There exists $e' \in trigger(\sigma')$, with $e \neq e'$ and $e' \notin I_{N-1}$ while $e' \in I' - \{e\}$, i.e. $e' \in gen(T) - \{e\}$. We repeat the same argument with respect to $I' - \{e\}$.

Suppose $e \in trigger(\sigma_j), gen(T_j)$, for some $j \in \{1,2\}$. We will use the following property. If $e \in trigger(\sigma_j), gen(T_j)$, one of the following cases holds:

1) there exists $T'_j \subset T_j$, where $e \in trigger(t)$, for each $t \in T'_j$ and $e \notin gen(T_j - T'_j)$;

2) there exists $p_{N-1,j} \overset{\alpha'_{N-1,j}}{\mapsto} p_{N,j}$, where $\alpha'_{N-1,j} = \sigma'_j : \tau'_j$ and $e \notin trigger(\sigma'_j)$, $I' \rightsquigarrow trigger(\sigma'_j)$, $gen(T_j) - trigger(\sigma'_j) = action(\sigma'_j)$.

Actually, we have that T_j is an admissible step with respect to I'. If T_j is an admissible step with respect to $I' - \{e\}$, then, by induction hypothesis, there exists $p_{N-1,j} \overset{\alpha'_{N-1,j}}{\mapsto} p_{N,j}$, where $\alpha'_{n-1,j} = \sigma'_j : \tau'_j$ and $e \notin trigger(\sigma'_j)$, $I' - \{e\} \rightsquigarrow trigger(\sigma'_j)$ and $gen(T_j) - trigger(\sigma'_j) = action(\sigma'_j)$. Moreover, $I' \rightsquigarrow trigger(\sigma'_j)$.

In fact, suppose $I' \not\rightsquigarrow trigger(\sigma'_j)$, then either $e' \in trigger(\sigma'_j)$ and $e' \notin I'$ or $\bar{e}' \in trigger(\sigma'_j)$ and $e' \in I'$. In the first case, $e' \notin I'$ and $e' \in I' - \{e\}$ is a contradiction. In the second case $e' \in I'$ and $e' \notin I' - \{e\}$ imply $e = e'$. However, since $e \in gen(T_j)$, $e \in action(\sigma'_j)$. This contradicts the consistency of σ'_j. Therefore $I' \rightsquigarrow trigger(\sigma'_j)$.

If T_j is not an admissible step with respect to $I' - \{e\}$, then either $En_{c_{N-1,j},I'-\{e\}}(T_j) \neq T_j$ or T_j is separable. However, $e \in gen(T_j)$ implies $trig(I' \cup gen(T_j)) = trig(I' - \{e\} \cup gen(T_j))$. Therefore, $En_{c_{N-1,j},I'-\{e\}}(T_j) = T_j$. If T_j is separable, there exists $T'_j \subset T_j$, such that $En_{c_{N-1,j},I'-\{e\}}(T_j - T'_j) \cap T'_j = \emptyset$. However, $En_{c_{N-1,j},I'}(T_j - T'_j) \cap T'_j \neq \emptyset$, so that, for each $t \in T'_j$, $e \in trigger(t)$ and $e \notin gen(T_j - T'_j)$.

We use the previous property for the case of $e \in trigger(\sigma_1), e \in gen(T_1)$ (or viceversa $e \in trigger(\sigma_2), e \in gen(T_2)$). One of the following cases holds:

(a) $e \in trigger(\sigma_1), gen(T_1)$ and $e \notin trigger(\sigma_2), gen(T_2)$;

(b) $e \in trigger(\sigma_1), gen(T_1)$ and $e \notin trigger(\sigma_2)$ $e \in gen(T_2)$;

(c) $e \in trigger(\sigma_1), gen(T_1)$ and $e \in trigger(\sigma_2)$ $e \in gen(T_2)$;

(d) $e \in trigger(\sigma_1), gen(T_1)$ and $e \in trigger(\sigma_2)$ $e \notin gen(T_2)$.

(a) Suppose $e \in trigger(\sigma_1), gen(T_1)$ and $e \notin trigger(\sigma_2), gen(T_2)$. If case 2) holds, then consider $\sigma'_1 \times \sigma_2$. Since $e \notin \ell_2$, for each $\sigma' \in \sigma'_1 \times \sigma_2$, $e \notin trigger(\sigma')$. Moreover, $I' - \{e\} \rightsquigarrow trigger(\sigma')$. There exists $e' \neq e$, such that $e' \in trigger(\sigma')$ and $e' \notin I$, while $e' \in I' - \{e\}$. We repeat the same argument with respect to $I' - \{e\}$. If case 1) holds, since $e \notin gen(T_2)$, $En_{c_{N-1},I_{N-1}}(T - T'_1) \cap T'_1 = \emptyset$. This contradicts the fact that T is inseparable with respect to I_{N-1}.

(b) Suppose $e \in trigger(\sigma_1), gen(T_1)$ and $e \notin trigger(\sigma_2)$, $e \in gen(T_2)$. Since $e \notin trigger(\sigma_2)$, then, by lemma A.0.3, there exists $\sigma' \in \sigma_1 \times \sigma_2$, where $e \notin trigger(\sigma')$ and $I' - \{e\} \rightsquigarrow trigger(\sigma')$ since $I' \rightsquigarrow trigger(\sigma')$. There exists $e' \neq e$ such that $e' \in trigger(\sigma')$ and $e' \notin I$ while $e' \in gen(T)$. We repeat the same argument.

(c) Suppose $e \in trigger(\sigma_1), gen(T_1)$ and $e \in trigger(\sigma_2)$ $e \in gen(T_2)$. If, for each $i \in \{1, 2\}$, case 2) holds, then there exists $\sigma' \in \sigma'_1 \times \sigma'_2$, such that $e \notin trigger(\sigma')$ and $I' - \{e\} \rightsquigarrow trigger(\sigma')$. There exists $e' \neq e$, such that $e' \in trigger(\sigma')$ and $e' \notin I$, while $e' \in gen(T) - \{e\}$. We repeat the same argument with respect to $I' - \{e\}$. If, for each $i \in \{1, 2\}$, case 1) holds, then we have $En_{c_{N-1},I_{N-1}}(T - (T'_1 \cup T'_2)) \cap (T'_1 \cup T'_2) = \emptyset$, since $e \notin I_{N-1} \cup gen(T - (T'_1 \cup T'_2))$. This contradicts the fact that T is inseparable.

If for 1 case 2) holds and for 2 case 1) holds, consider $\sigma'_1 \times \sigma_2$. Since $e \notin trigger(\sigma'_1)$, by lemma A.0.3, there exists $\sigma' \in \sigma'_1 \times \sigma_2$, where $e \notin$

$trigger(\sigma')$. Moreover $I' - \{e\} \rightsquigarrow trigger(\sigma')$. There exists $e' \neq e$ such that $e' \in trigger(\sigma')$ and $e' \notin I_{N-1}$, while $e' \in I' - \{e\}$. We repeat the same argument with respect to $I' - \{e\}$.

(d) Suppose $e \in trigger(\sigma_1), gen(T_1)$ and $e \in trigger(\sigma_2)$ $e \notin gen(T_2)$. If case 2) holds, then consider $\sigma_1' \times \sigma_2$. Since $e \notin trigger(\sigma_1')$, by lemma A.0.3, there exists $\sigma' \in \sigma_1' \times \sigma_2$, where $e \notin trigger(\sigma')$. Moreover $I' - \{e\} \rightsquigarrow trigger(\sigma')$. There exists $e' \neq e$, such that $e' \in trigger(\sigma')$ and $e' \notin I_{N-1}$, while $e' \in gen(T) - \{e\}$. We repeat the same argument with respect to $I' - \{e\}$.

If case 1) holds, then $En_{c_{N-1}, I_{N-1}}(T - T_1') \cap T_1' = \emptyset$, since $e \notin gen(T_2)$. This contradicts the fact that T is inseparable.

Since $gen(T)$ is finite, the previous reasoning leads to a contradiction.

We have shown that there exists $p_{N-1} \overset{\alpha_{N-1}}{\mapsto} p_N$, such that $loc(p_N) = c_N$ and $I_{N-1} \rightsquigarrow trigger(\alpha_{N-1})$. Moreover, since $type(\alpha_{N-1,i}) = \epsilon$, then $type(\alpha_{N-1}) = \epsilon$. It is obvious that $gen(T_1 \cup T_2) - trigger(\alpha_{N-1}) = action(\alpha_{N-1})$ as shown in lemma A.0.4.

■

Theorem 3.2.7 follows from lemma A.0.4 and lemma A.0.5.

Appendix B

Proofs of chapter 4

B.1 Soundness of the proof system

In this appendix we show the soundness of the proof system of chapter 4. We introduce some preliminaries properties.

We say that $p[]$ is a context over Act and Loc iff it it generated by the grammar of $\mathcal{SP}$ with the additional clause $\mathcal{SP} ::= []$.

For a context $p[]$ over Act and Loc and for $p' \in \mathcal{CSP}$ over Act and Loc, we denote by $p[p']$ the process obtained by replacing $[]$ with p'.

Proposition B.1.1 *For each context $p[]$ over Act and Loc and for each closed L_μ formula A over Act and Loc:*

- $p[\ell :: rec\ x.t] \models A$ *iff* $p[\ell :: t[rec\ x.t/x] \models A$;
- $p[\ell :: \ell_1 :: b_1 + b_2] \models A$ *iff* $p[\ell :: \ell_1 :: b_2 + b_1] \models A$.

Theorem 4.4.1 Assume $p \in \mathcal{CSP}$ over Act and Loc. For a closed L_μ formula A over Act and Loc, $p \vdash A$ implies $p \models A$.

Proof. We show that each rule is backward sound. It is obvious that we must show the soundness for arbitrary formulas A, possibly extended, over propositional symbols Loc and actions $\mathcal{E}(Act)$. The soundness of logical rules has been proved by [12]. Therefore, we consider only the significant cases for formulas $< \alpha > A$ and $[\alpha]A$, where $\alpha \in \mathcal{E}(Act)$.

- Rule $<> nil$.

$$\frac{\ell :: \ell_1 :: nil \vdash < a : \bar{\epsilon} > A}{\ell :: \ell_1 :: nil \vdash A}\ a \in \#(*)$$

Assume $\ell :: \ell_1 :: nil \models A$. Since $a \in \#(*)$, $\ell :: \ell_1 :: nil \overset{a:\epsilon}{\mapsto} \ell :: \ell_1 :: nil$. Therefore, $\ell :: \ell_1 :: nil \models < a : \epsilon > A$.

- The cases of $[]nil_1$, $[]nil_2$ and $[]nil_3$ are similar.

- Rule $<>_1$.

$$\frac{\ell :: \ell_1 :: a.t \vdash < a : \epsilon > A}{\ell :: t \vdash A}$$

Assume $\ell :: t \models A$. By definition, $\ell :: \ell_1 :: a.t \overset{a:\epsilon}{\mapsto} \ell :: t$. Since $\ell :: t \models A$, then $\ell :: \ell_1 :: a.t \models < a : \epsilon > A$.

- The case of $<>_2$ is similar to the one of $<> nil$.

- Rule $[]_1$.

$$\frac{\ell :: \ell_1 :: a.t \vdash [a : \epsilon]A}{\ell :: t \vdash A}$$

Assume $\ell :: t \models A$. By definition, $\ell :: \ell_1 :: a.t \overset{a:\epsilon}{\mapsto} p_1$ iff $p_1 = \ell_1 :: t$. Therefore, $\ell :: \ell_2 :: a.t \models [a : \epsilon]A$.

- The cases of $[]_2$, $[]_3$, $[]_4$ are similar to the previous ones.

- Rule $+ <> \epsilon$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash < \alpha > A}{\ell :: \ell_1 :: b_i \vdash < \alpha > A}\, type(\alpha) = \epsilon$$

Assume $\ell :: \ell_1 :: b_i \models < \alpha > A$. Since $\alpha \in \mathcal{S}(Act)$, there exists $\ell :: \ell_1 :: b_i \overset{\alpha}{\mapsto} p'$, such that $p' \models A$. Since $type(\alpha) = \epsilon$, by rule $choice_1$, $\ell :: \ell_1 :: b_1 + b_2 \overset{\alpha}{\mapsto} p'$. Since $p' \models A$, then $\ell :: \ell_1 :: b_1 + b_2 \models < \alpha > A$.

- Rule $+[]\epsilon$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash [\alpha]A}{\left\{\ \ell :: \ell_1 :: b_i \vdash [\alpha]A\ \right\}_{i \in \{1,2\}}}\, type(\alpha) = \epsilon$$

Assume $\ell :: \ell_1 :: b_i \models [\alpha]A$, for each $i \in \{1, 2\}$. For each $i \in \{1, 2\}$, if $\ell :: \ell_i :: b_i \overset{\alpha}{\mapsto} p'$, then $p' \models A$. Since $type(\alpha) = \epsilon$, $\ell :: \ell_1 :: b_1 + b_2 \overset{\alpha}{\mapsto} p'$ iff by rule $choice_1$, $\ell :: \ell_1 :: b_i \overset{\alpha}{\mapsto} p'$ for some $i \in \{1, 2\}$. Hence, $\ell :: \ell_1 :: b_1 + b_2 \models [\alpha]A$.

- Rule $+ <> \bar{\epsilon}$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash < \alpha > A}{\left\{ \ \ell :: \ell_1 :: b_i \vdash < \alpha > true \ \right\}_{i \in \{1,2\}} \quad \ell :: \ell_1 :: b_1 + b_2 \vdash A} \, type(\alpha) = \bar{\epsilon}$$

Assume $\ell :: \ell_1 :: b_1 \models < \alpha > true$, $\ell :: \ell_1 :: b_2 \models < \alpha > true$ and $\ell :: \ell_1 :: b_1 + b_2 \models A$. Since $\alpha \in \mathcal{S}(Act)$, there exists $\ell :: \ell_1 :: b_i \stackrel{\alpha}{\mapsto} \ell :: \ell_1 :: b_i$, for each $i \in \{1, 2\}$. Since $type(\alpha) = \bar{\epsilon}$, then by rule $choice_2$, $\ell :: \ell_1 :: b_1 + b_2 \stackrel{\alpha}{\mapsto} \ell :: \ell_1 :: b_1 + b_2$. Since $\ell :: \ell_1 :: b_1 + b_2 \models A$, $\ell :: \ell_1 :: b_1 + b_2 \models < \alpha > A$.

- Rule $+[]\bar{\epsilon}_1$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash [\alpha]A}{\ell :: \ell_1 :: b_1 + b_2 \vdash A} \, type(\alpha) = \bar{\epsilon}$$

Assume $\ell :: \ell_1 :: b_1 + b_2 \models A$. Since $\alpha \in \mathcal{S}(Act)$ and $type(\alpha) = \bar{\epsilon}$, $\ell :: \ell_1 :: b_1 + b_2 \stackrel{\alpha}{\mapsto} p'$, iff by rule $choice_2$, $\ell :: \ell_1 :: b_i \stackrel{\alpha}{\mapsto} \ell :: \ell_1 :: b_i$ and $p' = \ell :: \ell_1 :: b_1 + b_2$. Since $\ell :: \ell_1 :: b_1 + b_2 \models A$, then $\ell :: \ell_1 :: b_1 + b_2 \models [\alpha]A$.

.

- Rule $+[]\bar{\epsilon}_2$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash [\alpha]A}{\ell :: \ell_1 :: b_i \vdash [\alpha]false} \, type(\alpha) = \bar{\epsilon}$$

Assume $\ell :: \ell_1 :: b_i \models [\alpha]false$, for some $i \in \{1, 2\}$. Since $\alpha \in \mathcal{S}(Act)$, there are no arrows $\ell :: \ell_1 :: b_i \stackrel{\alpha}{\mapsto} p'$. Since $type(\alpha) = \bar{\epsilon}$, then rule $choice_2$ is not applicable and $\ell :: \ell_1 :: b_1 + b_2 \stackrel{\alpha}{\not\mapsto}$. Therefore, $\ell :: \ell_1 :: b_1 + b_2 \models [\alpha]A$.

- Rule $\bigtriangledown <> \epsilon_2$.

$$\frac{\ell :: t \bigtriangledown p \vdash < \alpha > A}{p \vdash < \alpha > (A)_{\ell::t\bigtriangledown}} \, type(\alpha) = \epsilon$$

Assume $p \models < \alpha > (A)_{\ell::t\bigtriangledown}$. Since $\alpha \in \mathcal{S}(Act)$, there exists $p \stackrel{\alpha}{\mapsto} p'$, such that $p' \models (A)_{\ell::t\bigtriangledown}$, i.e. $\ell :: t \bigtriangledown p' \models A$. Since $type(\alpha) = \epsilon$, by rule $\bigtriangledown_2$, $\ell :: t \bigtriangledown p \stackrel{\alpha}{\mapsto} \ell :: t \bigtriangledown p'$. Since $\ell :: t \bigtriangledown p' \models A$, then $\ell :: t \bigtriangledown p \models < \alpha > A$.

- Rule $\bigtriangledown <> \epsilon_1$.

$$\frac{\ell :: t \bigtriangledown p \vdash < \alpha > A}{\ell :: t \vdash < \alpha > A} \, type(\alpha) = \epsilon$$

Assume $\ell :: t \models\!< \alpha > A$. Since $\alpha \in \mathcal{S}(Act)$, there exists $\ell :: t \stackrel{\alpha}{\mapsto} p'$, where $p' \models A$. Since $type(\alpha) = \epsilon$, by rule $\bigtriangledown_1$, $\ell :: t \bigtriangledown p \stackrel{\alpha}{\mapsto} p'$. Since $p' \models A$, then $\ell :: t \bigtriangledown p \models\!< \alpha > A$.

- Rule $\bigtriangledown[]\epsilon$.

$$\frac{\ell :: t \bigtriangledown p \vdash [\alpha]A}{\ell :: t \vdash [\alpha]A \quad p \vdash [\alpha](A)_{\ell::t\bigtriangledown}}\, type(\alpha) = \epsilon$$

Assume $\ell :: t \models [\alpha]A$ and $p \models [\alpha](A)_{\ell::t\bigtriangledown}$. Since $\alpha \in \mathcal{S}(Act)$, if $\ell :: t \stackrel{\alpha}{\mapsto} p'$, $p' \models A$ and, if $p \stackrel{\alpha}{\mapsto} p'$, $p' \models (A)_{\ell::t\bigtriangledown}$, i.e. $\ell :: t \bigtriangledown p' \models A$.

Since $type(\alpha) = \epsilon$, then $\ell :: t \bigtriangledown p \stackrel{\alpha}{\mapsto} p'$ iff either $\ell :: t \stackrel{\alpha}{\mapsto} p'$ $(\bigtriangledown_1)$ or $p \stackrel{\alpha}{\mapsto} p''$ and $p' = \ell :: t \bigtriangledown p''$ $(\bigtriangledown_2)$. Therefore, $\ell :: t \bigtriangledown p \models [\alpha]A$.

- Rule $\bigtriangledown <>\bar{\epsilon}$.

$$\frac{\ell :: t \bigtriangledown p \vdash\!< \alpha > A}{\ell :: t \vdash\!< \alpha > true \quad p \vdash\!< \alpha > true \quad \ell :: t \bigtriangledown p \vdash A}\, type(\alpha) = \bar{\epsilon}$$

Assume $\ell :: t \models\!< \alpha > true$ and $p \models\!< \alpha > true$ and $\ell :: t \bigtriangledown p \models A$. Since $\alpha \in \mathcal{S}(Act)$, there exist $\ell :: t \stackrel{\alpha}{\mapsto} \ell :: t$ and $p \stackrel{\alpha}{\mapsto} p$. By $\bigtriangledown_3$, $\ell :: t \bigtriangledown p \stackrel{\alpha}{\mapsto} \ell :: t \bigtriangledown p$. Since $\ell :: t \bigtriangledown p \models A$, then $\ell :: t \bigtriangledown p \models\!< \alpha > A$.

- The cases of $\bigtriangledown[]\bar{\epsilon}_i$, for $i \in \{1, 3\}$ are similar to that of $+[]\bar{\epsilon}_1$ and $+[]\bar{\epsilon}_2$.

- Rule $rec <>$.

$$\frac{\ell :: rec\, x.t \vdash\!< \alpha > A}{\ell :: t[rec\, x.t/x] \vdash\!< \alpha > A}\, \alpha \in \mathcal{S}(Act)$$

Assume $\ell :: t[rec\, x.t/x] \models\!< \alpha > A$. There are two cases, either $type(\alpha) = \epsilon$ or $type(\alpha) = \bar{\epsilon}$.

In the first case, there exists $\ell :: t[rec\, x.t] \stackrel{\alpha}{\mapsto} p'$, where $p' \models A$. Since $type(\alpha) = \epsilon$, then, by rule rec_1, $\ell :: rec\, x.t \stackrel{\alpha}{\mapsto} p'$. Since $p' \models A$, then $\ell :: rec\, x.t \models\!< \alpha > A$.

In the second case, there exists $\ell :: t[rec\, x.t/x] \stackrel{\alpha}{\mapsto} \ell :: t[rec\, x.t/x]$, where $\ell :: t[rec\, x.t/x] \models A$. Since $type(\alpha) = \bar{\epsilon}$, by rule rec_2, $\ell :: rec\, x.t \stackrel{\alpha}{\mapsto} \ell :: rec\, x.t$. By proposition B.1.1, $\ell :: t[rec\, x.t/x] \models A$ implies $\ell :: rec\, x.t \models A$. Hence, $\ell :: rec\, x.t \models\!< \alpha > A$.

- The case of $rec[]$ is analogous to the previous one.

- Rule $\mathcal{E} <>$.

$$\frac{\ell :: p_1 \times p_2 \vdash\!< \alpha > A}{\ell :: p_1 \times p_2 \vdash\!< \beta > A}\, \beta \in \mathcal{E}(\alpha)$$

Assume $\ell :: p_1 \times p_2 \models < \beta > A$. There exists $\ell :: p_1 \times p_2 \overset{\beta'}{\mapsto} p$, which is consistent with β and $p \models A$. Since $\beta \in \mathcal{E}(\alpha)$, $type^*(\beta) = type(\alpha)$, so that $type(\alpha) = type(\beta')$ and $\alpha = \beta'$. Since $p \models A$, $\ell :: p_1 \times p_2 \models < \alpha > A$.

- Rule $\mathcal{E}[]$.

$$\frac{\ell :: p_1 \times p_2 \vdash [\alpha]A}{\left\{\ \ell :: p_1 \times p_2 \vdash [\beta_i]A\ \right\}_{i\in\{1,n\}}}\ \mathcal{E}(\alpha) = \{\beta_1, \ldots, \beta_n\}$$

Assume $\ell :: p_1 \times p_2 \models [\beta_i]A$, for each $i \in \{1, n\}$. By definition, for each $\ell :: p_1 \times p_2 \overset{\beta'_i}{\mapsto} p$, which is is consistent with β_i, $p \models A$. Hence, $type^*(\beta_i) = type(\alpha)$. We have $\ell :: p_1 \times p_2 \overset{\alpha}{\mapsto} p$, iff $p_i \overset{\alpha_i}{\mapsto} p'_i$ and $p = \ell :: p'_1 \times p'_2$ and $\alpha \in \alpha_1 \times \alpha_2$. Moreover, if $type(\alpha) = \bar{\epsilon}$, then $type(\alpha_i) = \bar{\epsilon}$, for each $i \in \{1, 2\}$, and, if $type(\alpha) = \epsilon$, then $type(\alpha_i) = \epsilon$, for some $i \in \{1, 2\}$. Since $\{\beta_i\}_{\in\{1,n\}} = \mathcal{E}(\alpha)$, then, $\ell :: p_1 \times p_2 \overset{\alpha}{\mapsto} p$ iff it is consistent with β_i, for some $i \in \{1, n\}$. Therefore, $\alpha = \beta'_i$, for some $i \in \{1, n\}$. Since $p \models A$, $\ell :: p_1 \times p_2 \models [\alpha]A$.

- Rule $\times <>_1$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \vdash < \alpha > A}{p_1 \vdash < \beta > (A)_{\ell\times\ell_1::t}}\ \alpha \in \beta \otimes (a : \epsilon)$$

Assume $p_1 \models < \beta > (A)_{\ell\times\ell_1::t}$. There exists $p_1 \overset{\beta'}{\mapsto} p'_1$, such that is it consistent with β and $p'_1 \models (A)_{\ell\times\ell_1::t}$, namely $\ell :: p'_1 \times \ell_1 :: t \models A$.

Since $\alpha \in \beta \otimes (a : \epsilon)$, then there exists $\alpha' \in \beta' \times (a : \epsilon)$, such that $type^*(\alpha) = type(\alpha')$. Since $\ell_1 :: \ell_2 :: a.t \overset{a:\epsilon}{\mapsto} \ell_1 :: t$, by rule *par*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \overset{\alpha'}{\mapsto} \ell :: p'_1 \times \ell_1 :: t$, which is consistent with α. Since $\ell :: p'_1 \times \ell_1 :: t \models A$, $\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \models < \alpha > A$.

- Rule $\times <>_2$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \vdash < \alpha > A}{p_1 \vdash < \beta > (A)_{\ell\times\ell_1::\ell_2::a.t}}\ \alpha \in \beta \otimes (b : \bar{\epsilon}), b \in \#(a)$$

Assume $p_1 \models < \beta > (A)_{\ell\times\ell_1::\ell_2::a.t}$. There exists $p_1 \overset{\beta'}{\mapsto} p'_1$, such that it is consistent with β and $p'_1 \models (A)_{\ell\times\ell_1::\ell_2::a.t}$, namely $\ell :: p'_1 \times \ell_1 :: \ell_2 :: a.t \models A$.

Since $\alpha \in \beta \otimes (b : \bar{\epsilon})$, then there exists $\alpha' \in \beta' \times (b : \bar{\epsilon})$, such that $type^*(\alpha) = type(\alpha')$. Since $\ell_1 :: \ell_2 :: a.t \overset{b:\bar{\epsilon}}{\mapsto} \ell_1 :: \ell_2 :: a.t$, by rule *par*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \overset{\alpha'}{\mapsto} \ell :: p'_1 \times \ell_1 :: \ell_2 :: a.t$, which is consistent with α.

Since $\ell :: p'_1 \times \ell_1 :: \ell_2 :: a.t \models A$, $\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \models < \alpha > A$.

- The cases of $\times[]$ and of *nil* are similar to the previous ones.

- Rule $\times <> +\epsilon$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash < \alpha > A}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash < \alpha > A} \, type(proj_2(\alpha)) = \epsilon$$

Assume $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \models < \alpha > A$, for some $i \in \{1, 2\}$. By definition, there exists $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha'}{\mapsto} p'$, which is consistent with α and such that $p' \models A$. Moreover, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha'}{\mapsto} p'$ iff by rule *par*, $p_1 \stackrel{\alpha'_1}{\mapsto} p'_1$ and $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha'_2}{\mapsto} p'_2$, with $p' = p'_1 \times p'_2$ and $\alpha' \in \alpha'_1 \times \alpha'_2$. Moreover, $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha'_2}{\mapsto} p'_2$ is consistent with $\alpha_2 = a_2 : \tau_2$, for $\alpha = a : \tau_1 \times \tau_2$ and $\alpha'_2 = a_2 : \tau'_2$. Since $type(proj_2(\alpha)) = \epsilon$, $type(\alpha'_2) = \epsilon$ and $\alpha'_2 = \alpha_2$. By rule $choice_1$, $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha'_2}{\mapsto} p'_2$ implies $\ell_1 :: \ell_2 :: b_1 + b_2 \stackrel{\alpha'_2}{\mapsto} p'_2$. Therefore, by rule *par*, $p_1 \times \ell_1 :: \ell_2 :: (b_1 + b_2) \stackrel{\alpha'}{\mapsto} \ell :: p'_1 \times p'_2$, which is consistent with α. Since $\ell :: p'_1 \times p'_2 \models A$, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \models < \alpha > A$.

- Rule $\times[] + \epsilon$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash [\alpha]A}{\left\{ \ \ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash [\alpha]A \ \right\}_{i \in \{1,2\}}} \, type(proj_2(\alpha)) = \epsilon$$

Assume $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 \models [\alpha]A$ and $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \models [\alpha]A$. By definition, for each $i \in \{1, 2\}$, if $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha'}{\mapsto} \ell :: p'_1 \times p'_2$ that is consistent with α, $\ell :: p'_1 \times p'_2 \models A$.

Moreover, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha'}{\mapsto} \ell :: p'_1 \times p'_2$ iff $p_1 \stackrel{\alpha'_1}{\mapsto} p'_1$ and $\ell_1 :: \ell_2 :: b_1 \stackrel{\alpha'_2}{\mapsto} p'_2$, where $\alpha' \in \alpha'_1 \times \alpha'_2$ and $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha'_2}{\mapsto} \ell_1 :: t'_2$ is consistent with $\alpha_2 = a_2 : \tau_2$, where $\alpha = a : \tau_1 \times \tau_2$ and $\alpha'_2 = a_2 : \tau'_2$.

Since $type(proj_2(\alpha)) = \epsilon$, $\alpha_2 = \alpha'_2$ and $type(\alpha'_2) = \epsilon$. Therefore, $\ell_1 :: \ell_2 :: b_1 + b_2 \stackrel{\alpha'_2}{\mapsto} p'_2$ iff by rule $choice_1$, for some $i \in \{1, 2\}$, $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha'_2}{\mapsto} p'_2$. Therefore, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \models [\alpha]A$.

- Rule $\times <> +\bar{\epsilon}$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash < \alpha > A}{\left\{ \ \ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash < \alpha > (A)_{\times b_j +} \ \right\}_{i,j \in \{1,2\}, i \neq j}} \, type(proj_2(\alpha)) = \bar{\epsilon}$$

Assume $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 \models < \alpha > (A)_{\times b_2 +}$ and $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \models < \alpha > (A)_{\times b_1 +}$. For each $i \in \{1, 2\}$, there exists $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha_i}{\mapsto} \ell :: p'_{1,i} \times p'_{2,i}$ such that $\ell :: p'_{1,i} \times p'_{2,i} \models (A)_{\times b_j +}$. Moreover, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha_i}{\mapsto} \ell :: p'_{1,i} \times p'_{2,i}$ is consistent with α, for each $i \in \{1, 2\}$.

For each $i \in \{1,2\}$, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha_i}{\mapsto} \ell :: p'_{1,i} \times p'_{2,i}$ has been obtained by rule *par* applied to $p_1 \overset{\alpha'_i}{\mapsto} p'_{1,i}$ and $\ell_1 :: \ell_2 :: b_i \overset{\alpha''_i}{\mapsto} p'_{2,i}$ and $\alpha_i \in \alpha'_i \times \alpha''_i$. Moreover, $p_1 \overset{\alpha'_i}{\mapsto} p'_{1,i}$ is consistent with $a'_i : \tau_1$ and $\ell_1 :: \ell_2 :: b_i \overset{\alpha''_i}{\mapsto} p'_{2,i}$ is consistent with $a''_i : \tau_2$, where $\alpha = a : \tau_1 \times \tau_2$ and $\alpha''_i = a''_i : \tau''_i$, $\alpha'_i = a'_i : \tau'_i$.

Since $type(proj_2(\alpha)) = \bar{\epsilon}$, then $type(\alpha''_i) = \bar{\epsilon}$ and $p'_{i,2} = \ell_1 :: \ell_2 :: b_i$. By the properties of $\times$, $\alpha''_1 = \alpha''_2$. Therefore, by rule $choice_2$, $\ell_1 :: \ell_2 :: b_1 + b_2 \overset{\alpha''_i}{\mapsto} \ell_1 :: \ell_2 :: b_1 + b_2$. By rule *par*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \overset{\alpha_i}{\mapsto} \ell :: p_{1,i} \times \ell_1 :: \ell_2 :: b_1 + b_2$, which is consistent with α. Since $\ell :: p_{1,1} \times \ell_1 :: \ell_2 :: b_1 \models (A)_{\times b_2 +}$ and $\ell :: p'_{1,2} \times \ell_1 :: \ell_2 :: b_2 \models (A)_{\times b_1 +}$, then $\ell :: p'_{1,1} \times \ell_1 :: \ell_2 :: b_1 + b_2 \models A$ and $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \models\!< \alpha > A$.

- The cases of $\times[] + \bar{\epsilon}_i$, for $i \in \{1,2\}$, are similar to the previous one.

- Rule $\times <> \bigtriangledown \epsilon_2$.

$$\frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash\!< \alpha > A}{\ell :: p_1 \times p \vdash (A)_{\times \ell_2 :: t}} \; type(proj_2(\alpha)) = \epsilon$$

Assume $\ell :: p_1 \times p \models\!< \alpha > (A)_{\times \ell_2 :: t \bigtriangledown}$. There exists $\ell :: p_1 \times p \overset{\alpha'}{\mapsto} \ell :: p'_1 \times p'$, which is consistent with α and such that $\ell :: p'_1 \times p' \models (A)_{\times \ell_2 :: t \bigtriangledown}$, i.e. $\ell :: p'_1 \times \ell_2 :: t \bigtriangledown p' \models A$.

We have $\ell :: p_1 \times p \overset{\alpha'}{\mapsto} \ell :: p'_1 \times p'$ iff by rule *par*, $p_1 \overset{\alpha'_1}{\mapsto} p'_1$, $p \overset{\alpha'_2}{\mapsto} p'$ and $\alpha' \in \alpha'_1 \times \alpha'_2$. Moreover, $p_1 \overset{\alpha'_1}{\mapsto} p'_1$ is consistent with $\alpha_1 = a_1 : \tau_1$ and $p \overset{\alpha'_2}{\mapsto} p'$ is consistent with $\alpha_2 = a_2 : \tau_2$, for $\alpha = a : \tau_1 \times \tau_2$ and $\alpha'_i = a_i : \tau'_i$.

Since $type(proj_2(\alpha)) = \epsilon$, then $\alpha_2 = \alpha'_2$ and $type(\alpha'_2) = \epsilon$. By rule $\bigtriangledown_2$, $\ell_2 :: t \bigtriangledown p \overset{\alpha'_2}{\mapsto} \ell_2 :: t \bigtriangledown p'$.

By applying rule *par*, $\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \overset{\alpha'}{\mapsto} \ell :: p'_1 \times \ell_2 :: t \bigtriangledown p'$, which is consistent with α. Since $\ell :: p'_1 \times \ell_2 :: t \bigtriangledown p' \models A$, $\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \models\!< \alpha > A$.

- The case of rule $\times[] \bigtriangledown \epsilon_1$ is similar to the previous one. Rules $\times <> \bigtriangledown \bar{\epsilon}$, $\times[] \bigtriangledown \bar{\epsilon}_i$, for $i \in \{1,3\}$, are similar to the one of $\times <> + \bar{\epsilon}$, $\times[] + \bar{\epsilon}_i$, respectively.

- Rule $\times <> rec$.

$$\frac{\ell :: p_1 \times \ell_2 :: rec\ x.t_2 \vdash\!< \alpha > A}{\ell :: p_1 \times \ell_2 :: t_2[rec\ x.t_2/x] \vdash\!< \alpha > A}$$

Assume $\ell :: p_1 \times \ell_2 :: t_2[rec\ x.t_2/x] \models\!< \alpha > A$. There exists $\ell :: p_1 \times \ell_2 :: t_2[rec\ x.t_2/x] \overset{\alpha'}{\mapsto} \ell :: p'_1 \times p'_2$, which is consistent with α and such that $\ell :: p'_1 \times p'_2 \models A$. By definition $\ell :: p_1 \times \ell_2 :: t_2[rec\ x.t_2/x] \overset{\alpha'}{\mapsto} \ell :: p'_1 \times p'_2$ has been obtained by rule *par* applied to $p_1 \overset{\alpha'_1}{\mapsto} p'_1$ and $\ell_2 :: t_2[rec\ x.t_2/x] \overset{\alpha'_2}{\mapsto} p'_2$, where $\alpha' \in \alpha'_1 \times \alpha'_2$. Moreover,

$\ell_2 :: t_2[rec\ x.t_2/x] \stackrel{\alpha'_2}{\mapsto} p'_2$ is consistent with $\alpha_2 = a_2 : \tau_2$, where $\alpha = a : \tau_1 \times \tau_2$ and $\alpha'_2 = \alpha_2$. There are two cases, either $type(\alpha'_2) = \epsilon$ or $type(\alpha'_2) = \bar{\epsilon}$.

If $type(\alpha'_2) = \epsilon$, then, by rule rec_1, $\ell_2 :: t_2[rec\ x.t_2/x] \stackrel{\alpha'_2}{\mapsto} p'_2$ implies $\ell_2 :: rec\ x.t_2 \stackrel{\alpha'_2}{\mapsto} p'_2$ and by rule par, $\ell :: p_1 \times \ell_2 :: rec\ x.t_2 \stackrel{\alpha'}{\mapsto} \ell :: p'_1 \times p'_2$, which is consistent with α. Since $\ell :: p'_1 \times p'_2 \models A$, $\ell :: p_1 \times \ell_2 :: rec\ x.t_2 \models < \alpha > A$.

In the other case, $p'_2 = \ell_2 :: t_2[rec\ x.t_2/x]$ and $\ell_2 :: t_2[rec\ x.t_2/x] \stackrel{\alpha'_2}{\mapsto} \ell_2 :: t_2[rec\ x.t_2/x]$ implies by rule rec_2, $\ell_2 :: rec\ x.t_2 \stackrel{\alpha'_2}{\mapsto} \ell_2 :: rec\ x.t_2$. By rule par, $\ell :: p_1 \times \ell_2 :: rec\ x.t_2 \stackrel{\alpha'}{\mapsto} \ell :: p'_1 \times \ell_2 :: rec\ x.t_2$, which is consistent with α.

By proposition B.1.1, $\ell_2 :: p'_1 \times \ell_2 :: t[rec\ x.t_2/x] \models A$ implies $\ell_2 :: p'_1 \times \ell_2 :: rec\ x.t_2 \models A$ and $\ell :: p_1 \times \ell_2 :: rec\ x.t_2 \vdash < \alpha > A$.

- Rule $\times\times <>$.

$$\frac{\ell :: p_1 \times \ell_2 :: (p_2 \times p_3) \vdash < \alpha > A}{\ell :: \ell_2 :: (p_1 \times p_2) \times p_3 \vdash < \beta > (A)_\times}\ \beta \in left(\alpha)$$

Assume $\ell :: \ell_2 :: (p_1 \times p_2) \times p_3 \models < \beta > (A)_\times$. By definition, there exists $\ell :: \ell_2 :: (p_1 \times p_2) \times p_3 \stackrel{\beta'}{\mapsto} \ell :: \ell_2 :: (p'_1 \times p'_2) \times p'_3$, which is consistent with β and such that $\ell :: \ell_2 :: (p'_1 \times p'_2) \times p'_3 \models (A)_\times$, i.e. $\ell :: p'_1 \times \ell_2 :: (p'_2 \times p'_3) \models A$.

We have $\ell :: \ell_2 :: (p_1 \times p_2) \times p_3 \stackrel{\beta'}{\mapsto} \ell :: \ell_2 :: (p'_1 \times p'_2) \times p'_3$ iff by rule par, $\ell_2 :: p_1 \times p_2 \stackrel{\beta'_1}{\mapsto} \ell_2 :: p'_1 \times p'_2$ and $p_3 \stackrel{\beta'_2}{\mapsto} p'_3$, where $\beta' \in \beta'_1 \times \beta'_2$. Moreover, $\ell_2 :: p_1 \times p_2 \stackrel{\beta'_1}{\mapsto} \ell_2 :: p'_1 \times p'_2$ iff, by rule par, $p_1 \stackrel{\beta'_{1,1}}{\mapsto} p'_1$ and $p_2 \stackrel{\beta'_{1,2}}{\mapsto} p'_2$, where $\beta'_1 \in \beta'_{1,1} \times \beta'_{1,2}$.

By associativity of $\times$, $\beta' \in \beta'_{1,1} \times (\beta'_{1,2} \times \beta'_2)$. By rule par, $\ell :: p_1 \times \ell_2 :: (p_2 \times p_3) \stackrel{\beta'}{\mapsto} \ell :: p'_1 \times \ell_2 :: (p'_2 \times p'_3)$. Since $\ell :: \ell_2 :: (p_1 \times p_2) \times p_3 \stackrel{\beta'}{\mapsto} \ell :: \ell_2 :: (p'_1 \times p'_2) \times p'_3$ is consistent with β, then $\ell :: p_1 \times \ell_2 :: (p_2 \times p_3) \stackrel{\beta'}{\mapsto} \ell :: p'_1 \times \ell_2 :: (p'_2 \times p'_3)$ is consistent with α. Actually, $\beta \in left(\alpha)$. Since $\ell :: p'_1 \times \ell_2 :: (p'_2 \times p'_3) \models A$, then $\ell :: p_1 \times \ell_2 :: (p_2 \times p_3) \models < \alpha > A$.

- The cases of $\times[]rec$ and $\times \times []$ are similar to the previous ones.
- The soundness of shift rules is obvious.

■

B.2 Completeness of the proof system

In this appendix we show the completeness of the proof system of chapter 4 for regular processes.

Lemma 4.4.4 Assume $p \in \mathcal{CSP}$ over Act and Loc and A a closed formula (possibly extended) over Act and Loc. For $\alpha \in \mathcal{E}(Act)$, if $p \models < \alpha > A$ (resp. $p \models [\alpha]A$), then

1. there exist sets $\{p_i \in \mathcal{S}_p\}_{i\in\{1,n\}}$ and $\{p'_j \in \mathcal{S}_p\}_{j\in\{1,k\}}$, such that $\{p_i \models A\}_{i\in\{1,n\}}$;
2. $p \vdash < \alpha > A$ (resp. $p \vdash [\alpha]A$)) can be derived from $\{p_i \vdash A\}_{i\in\{1,n\}}$ and $\{p'_j \vdash true\}_{j\in\{1,k\}}$.

Moreover, for each $i \in \{1, n\}$, there exists $p \overset{\alpha_i}{\mapsto} p_i$ that is consistent with α.

Proof. The proof is by induction on $m(p)$. We show only the significant cases.

- Assume $p = \ell :: \ell_1 :: nil$.

$p \models < \alpha > A$ iff $\alpha \in \mathcal{S}(Act)$ and $\alpha = b : \bar{\epsilon}$ with $b \in \#(*)$ and $p \models A$. Moreover, we obtain $p \vdash < \alpha > A$ from $p \vdash A$ by rule $<> nil$. The case of $[\alpha]A$ is analogous.

- Assume $p = \ell :: \ell_1 :: a.t$.

$p \models < \alpha > A$ iff $\alpha \in \mathcal{S}(Act)$ and either $\alpha = a : \epsilon$ and $\ell :: t \models A$ or $\alpha = b : \bar{\epsilon}$ with $b \in \#(a)$ and $p \models A$.

We have $p \vdash < \alpha > A$ from $\ell :: t \vdash A$ and $p \vdash A$, by applying rules $<>_1$, $<>_2$ respectively. Moreover, $\ell :: t, p \in \mathcal{R}_p \subseteq \mathcal{S}_p$. The case of $[\alpha]A$ is analogous.

- Assume $p = \ell :: \ell_1 :: b_1 + b_2$. $p \models < \alpha > A$ iff $\alpha \in \mathcal{S}(Act)$ and there exists $p \overset{\alpha}{\mapsto} p'$, such that $p' \models A$. By definition $p \overset{\alpha}{\mapsto} p'$ has been obtained either by rule $choice_1$ or $choice_2$.
  - If $p \overset{\alpha}{\mapsto} p'$ has been obtained by $choice_1$, $\ell :: \ell_1 :: b_i \overset{\alpha}{\mapsto} p'$ and $type(\alpha) = \epsilon$, for some $i \in \{1, 2\}$. Therefore, $\ell :: \ell_1 :: b_i \models < \alpha > A$.

    Since $m(\ell :: \ell_1 :: b_i) < m(\ell :: \ell_1 :: b_1 + b_2)$, by induction hypothesis, for some $i \in \{1, 2\}$, there exist $\{p_j \in \mathcal{S}_{\ell::\ell_1::b_i}\}_{j\in\{1,n\}}$ and $\{p'_j \in \mathcal{S}_{\ell::\ell_1::b_i}\}_{j\in\{1,k\}}$, such that $p_j \models A$, for $j \in \{1, n\}$. Moreover, $\ell :: \ell_1 :: b_i \vdash < \alpha > A$ can be derived from $\{p_j \vdash A\}_{j\in\{1,n\}}$ and $\{p'_j \vdash true\}_{j\in\{1,k\}}$.
    1. It is obvious that $p_j, p'_j \in \mathcal{S}_p$, for each $j \in \{1, n\}$, $j \in \{1, k\}$.
    2. Since $type(\alpha) = \epsilon$, $p \vdash < \alpha > A$ can be obtained from $\{p_j \vdash A\}_{j\in\{1,n\}}$ and $\{p'_j \vdash true\}_{j\in\{1,k\}}$, by applying rule $+ <> \epsilon$ to $\ell :: \ell_1 :: b_i \vdash < \alpha > A$.

    Moreover, since $\ell :: \ell_1 :: b_i \overset{\alpha}{\mapsto} p_j$ and $type(\alpha) = \epsilon$, then by $choice_1$, $p \overset{\alpha}{\mapsto} p_j$ for each $j \in \{1, n\}$.
  - If $p \overset{\alpha}{\mapsto} p'$ has been obtained by rule $choice_2$, then $\ell :: \ell_1 :: b_1 \overset{\alpha}{\mapsto} \ell :: \ell_1 :: b_1$ and $\ell :: \ell_1 :: b_2 \overset{\alpha}{\mapsto} \ell :: \ell_1 :: b_2$ and $type(\alpha) = \bar{\epsilon}$, $p' = p$. Hence, $p \models A$ and $\ell :: \ell_1 :: b_i \models < \alpha > true$, for each $i \in \{1, 2\}$. Since $m(\ell :: \ell_1 :: b_i) < m(\ell :: \ell_1 :: b_1 + b_2)$, by induction hypothesis, for each $i \in \{1, 2\}$, there exist $\{p_{i,j} \in \mathcal{S}_{\ell::\ell_1::b_i}\}_{j\in\{1,n_i\}}$ such that, for each $i \in \{1, 2\}$, $\ell :: \ell_1 :: b_i \vdash < \alpha > true$, can be obtained from $\{p_{i,j} \vdash true\}_{j\in\{1,n_i\}}$.

1. It is obvious that $p, p_{i,j} \in \mathcal{S}_p$, for each $i \in \{1,2\}$ and for each $j \in \{1, n_i\}$.
2. Since $type(\alpha) = \bar{\epsilon}$, $p \vdash < \alpha > A$ can be obtained from $p \vdash A$ and $\{p_{i,j} \vdash true\}_{i\in\{1,2\},j\in\{1,n_i\}}$, by applying rule $+ <> \bar{\epsilon}$ to $\{\ell :: \ell_1 :: b_i \vdash < \alpha > true\}_{i\in\{1,2\}}$ and $p \vdash A$.

$p \models [\alpha]A$ iff $\alpha \in \mathcal{S}(Act)$ and for each $p \overset{\alpha}{\mapsto} p'$, $p' \models A$. There are two cases.

– Suppose $type(\alpha) = \epsilon$. We have $p \overset{\alpha}{\mapsto} p'$ iff, by $choice_1$, either $\ell :: \ell_1 :: b_1 \overset{\alpha}{\mapsto} p'$ or $\ell :: \ell_1 :: b_2 \overset{\alpha}{\mapsto} p'$ and $type(\alpha) = \epsilon$. Therefore, both $\ell :: \ell_1 :: b_1 \models [\alpha]A$ and $\ell :: \ell_1 :: b_2 \models [\alpha]A$.

Since $m(\ell :: \ell_1 :: b_i) < m(\ell :: \ell_1 :: b_1 + b_2)$, by induction hypothesis, for each $i \in \{1,2\}$, there exist $\{p_{i,j} \in \mathcal{S}_{\ell::\ell_1::b_i}\}_{j\in\{1,n_i\}}$ and $\{p'_{i,j} \in \mathcal{S}_{\ell::\ell_1::b_i}\}_{j\in\{1,k_i\}}$, such that $p_{i,j} \models A$, for $j \in \{1, n_i\}$. Moreover, $\ell :: \ell_1 :: b_i \vdash [\alpha]A$ can be derived from $\{p_{i,j} \vdash A\}_{j\in\{1,n_i\}}$ and $\{p'_{i,j} \vdash true\}_{j\in\{1,k_i\}}$.

1. It is obvious that $p_{i,j}, p'_{i,j} \in \mathcal{S}_p$, for each $j \in \{1, n_i\}$, $j \in \{1, k_i\}$ and $i \in \{1,2\}$.
2. Since $type(\alpha) = \epsilon$, $p \vdash [\alpha]A$ can be obtained from premises $\{p_{i,j} \vdash A\}_{j\in\{1,n_i\},i\in\{1,2\}}$ and $\{p'_{i,j} \vdash true\}_{j\in\{1,k_i\},i\in\{1,2\}}$, by applying rule $+[]\epsilon$ to $\{\ell :: \ell_1 :: b_i \vdash [\alpha]A\}_{i\in\{1,2\}}$.

Moreover, since $type(alpha) = \epsilon$ and $\ell :: \ell_1 :: b_i \overset{\alpha}{\mapsto} p_{i,j}$, then by $choice_1$, $p \overset{\alpha}{\mapsto} p_{i,j}$ for each $j \in \{1, n_i\}$ and $i \in \{1,2\}$.

– Suppose $type(\alpha) = \bar{\epsilon}$. We have $p \overset{\alpha}{\mapsto} p'$ iff, by rule $choice_2$, $\ell :: \ell_1 :: b_i \overset{\alpha}{\mapsto} \ell :: \ell_1 :: b_i$ for each $i \in \{1,2\}$ and $p' = p$.

There are two cases. Either $\ell :: \ell_1 :: b_i \overset{\alpha}{\not\mapsto}$, for some $i \in \{1,2\}$, or $\ell :: \ell_1 :: b_i \overset{\alpha}{\mapsto} \ell :: \ell_1 :: b_i$ for each $i \in \{1,2\}$.

In the first case, $\ell :: \ell_1 :: b_i \models [\alpha]false$. Since $m(\ell :: \ell_1 :: b_i) < m(\ell :: \ell_1 :: b_1 + b_2)$, by induction hypothesis, $\ell :: \ell_1 :: b_i \vdash [\alpha]false$. We obtain $p \vdash [\alpha]false$ from $\ell :: \ell_1 :: b_i \vdash [\alpha]false$ by applying rule $+[]\bar{\epsilon}_2$

In the second case, $p \models A$ and $p \vdash < \alpha > A$ can be obtained from $p \vdash A$ by rule $+[]\bar{\epsilon}_1$.

- Assume $p = \ell :: rec\, x.t$. $p \models < \alpha > A$ iff $\alpha \in \mathcal{S}(Act)$ and there exists $p \overset{\alpha}{\mapsto} p'$, where $p' \models A$. By definition $p \overset{\alpha}{\mapsto} p'$ has been obtained either by rule rec_1 or by rule rec_2.

– If $p \overset{\alpha}{\mapsto} p'$ has been obtained by rec_1, then $\ell :: t[rec\, x.t/x] \overset{\alpha}{\mapsto} p'$ and $type(\alpha) = \epsilon$. Therefore, $\ell :: t[rec\, x.t/x] \models < \alpha > A$.

Since $m(\ell :: t[rec\, x.t/x]) < m(\ell :: rec\, x.t)$, by induction hypothesis, there exist $\{p_i \in \mathcal{S}_{\ell::t[rec\, x.t/x]}\}_{i\in\{1,n\}}$ and $\{p'_j \in \mathcal{S}_{\ell::t[rec\, x.t/x]}\}_{j\in\{1,k\}}$, such that $p_i \models A$, for each $i \in \{1, n\}$. Moreover, $\ell :: t[rec\, x.t/x] \vdash < \alpha > A$ can be obtained from $\{p_i \vdash A\}_{i\in\{1,n\}}$ and $\{p'_j \vdash true\}_{j\in\{1,k\}}$.

1. It is obvious that $p_i, p'_j \in \mathcal{S}_p$, for each $i \in \{1,n\}$ and for each $j \in \{1,k\}$.
2. $\ell :: rec\ x.t \vdash < \alpha > A$ can be obtained from $\{p_i \vdash A\}_{i\in\{1,n\}}$ and $\{p'_j \vdash true\}_{i\in\{1,k\}}$, by applying rule $rec <>$ to $\ell :: t[rec\ x.t/x] \vdash < \alpha > A$.

Moreover, since $type(\alpha) = \epsilon$ and $\ell :: t[rec\ x.t/x] \overset{\alpha}{\mapsto} p_i$, then by rec_1, $p \overset{\alpha}{\mapsto} p_i$ for each $i \in \{1,n\}$.

– If $p \overset{\alpha}{\mapsto} p'$ has been obtained by rec_2 then $\ell :: t[rec\ x.t/x] \overset{\alpha}{\mapsto} \ell :: t[rec\ x.t/x]$ and $type(\alpha) = \epsilon$, $p = p'$. Therefore, $\ell :: rec\ x.t \models A$ and by proposition B.1.1, $\ell :: t[rec\ x.t/x] \models A$ and $\ell :: t[rec\ x.t/x] \models < \alpha > A$. Since $m(\ell :: t[rec\ x.t/x]) < m(\ell :: rec\ x.t)$, by induction hypothesis, there exist $\{p_i \in \mathcal{S}_{\ell::t[rec\ x.t/x]}\}_{i\in\{1,n\}}$ and $\{p'_j \in \mathcal{S}_{\ell::t[rec\ x.t/x]}\}_{j\in\{1,k\}}$, such that $p_i \models A$, for each $i \in \{1,n\}$. Moreover, $\ell :: t[rec\ x.t/x] \vdash < \alpha > A$ can be obtained from $\{p_i \vdash A\}_{i\in\{1,n\}}$ and $\{p'_j \vdash true\}_{i\in\{1,k\}}$. The proof is analogous to the previous case.

The case of $[\alpha]A$ is similar.

- Assume $p = \ell :: t \bigtriangledown p''$. $p \models < \alpha > A$ iff $\alpha \in \mathcal{S}(Act)$ and there exists $p \overset{\alpha}{\mapsto} p'$, where $p' \models A$. By definition $p \overset{\alpha}{\mapsto} p'$ has been obtained by the application either of rule $\bigtriangledown_1$, $\bigtriangledown_2$ or $\bigtriangledown_3$.

  – If $p \overset{\alpha}{\mapsto} p'$ has been obtained by $\bigtriangledown_1$, then $\ell :: t \overset{\alpha}{\mapsto} p'$, where $type(\alpha) = \epsilon$ and $\ell :: t \models < \alpha > A$. Since $m(\ell :: t) < m(p)$, by induction hypothesis, there exist $\{p_i \in \mathcal{S}_{\ell::t}\}_{i\in\{1,n\}}$ and $\{p'_j \in \mathcal{S}_{\ell::t}\}_{j\in\{1,k\}}$, such that $p_i \models A$, for each $i \in \{1,n\}$. Moreover, $\ell :: t \vdash < \alpha > A$ can be obtained from $\{p_i \vdash A\}_{i\in\{1,n\}}$ and $\{p'_j \vdash true\}_{j\in\{1,k\}}$.

    1. It is obvious that $p_i, p'_j \in \mathcal{S}_p$, for each $i \in \{1,n\}$ and for each $j \in \{1,k\}$.
    2. Since $type(\alpha) = \epsilon$, $\ell :: t \bigtriangledown p'' \vdash < \alpha > A$ can be derived from $\{p_i \vdash A\}_{i\in\{1,n\}}$ and $\{p'_j \vdash true\}_{j\in\{1,k\}}$, by applying rule $\bigtriangledown <> \epsilon_1$ to $\ell :: t \vdash < \alpha > A$.

    Moreover, since $type(\alpha) = \epsilon$ and $\ell :: t \overset{\alpha}{\mapsto} p_i$, then by $\bigtriangledown_1$, $p \overset{\alpha}{\mapsto} p_i$ for each $i \in \{1,n\}$.

  – If $p \overset{\alpha}{\mapsto} p'$ has been obtained by $\bigtriangledown_2$, then $p'' \overset{\alpha}{\mapsto} p'''$, where $type(\alpha) = \epsilon$ and $p' = \ell :: t \bigtriangledown p'''$. Since $\ell :: t \bigtriangledown p''' \models A$, then $p''' \models (A)_{\ell::t\bigtriangledown}$ and $p'' \models < \alpha > (A)_{\ell::t\bigtriangledown}$. Since $m(p'') < m(p)$, by induction hypothesis, there exist $\{p_i \in \mathcal{S}_{p''}\}_{i\in\{1,n\}}$ and $\{p'_j \in \mathcal{S}_{p''}\}_{j\in\{1,k\}}$, such that $p_i \models (A)_{\ell::t\bigtriangledown}$, for each $i \in \{1,n\}$. Moreover, $p'' \vdash < \alpha > (A)_{\ell::t\bigtriangledown}$ can be obtained from $\{p_i \vdash (A)_{\ell::t\bigtriangledown}\}_{i\in\{1,n\}}$ and $\{p'_j \vdash true\}_{j\in\{1,k\}}$.

    1. Since $p_i \in \mathcal{S}_{p''}$ and $p_i \models (A)_{\ell::t\bigtriangledown}$, then $\ell :: t \bigtriangledown p_i \models A$, for each $i \in \{1,n\}$. Since $type(\alpha) = \epsilon$ and $p'' \overset{\alpha}{\mapsto} p_i$, then by $\bigtriangledown_2$, $p \overset{\alpha}{\mapsto} \ell :: p_i$,

for each $i \in \{1,n\}$. Therefore, $\ell :: t \bigtriangledown p_i \in \mathcal{R}_p \subseteq \mathcal{S}_p$. It is obvious that $p'_j \in \mathcal{S}_p$, for each $j \in \{1,k\}$.

2. Since $type(\alpha) = \epsilon$, $\ell :: t \bigtriangledown p'' \vdash < \alpha > A$ can be obtained from $\{p_i \vdash (A)_{\ell::t\bigtriangledown}\}_{i\in\{1,n\}}$ and $\{p'_j \vdash true\}_{j\in\{1,k\}}$ by applying rule $\bigtriangledown <> \bar{\epsilon}_2$ to $p'' \vdash < \alpha > (A)_{\ell::t\bigtriangledown}$. Moreover, $\{p_i \vdash (A)_{\ell::t\bigtriangledown}\}_{i\in\{1,n\}}$, are obtained by shift rule from $\{\ell :: t \bigtriangledown p_i \vdash A\}_{i\in\{1,n\}}$.

– If $p \overset{\alpha}{\mapsto} p'$ has been obtained by $\bigtriangledown_3$, the case is analogous to the one of $choice_2$.

The case of $[\alpha]A$ is similar.

- Assume $p = \ell : p_1 \times p_2$.

We have $p \models < \alpha > A$ iff either $\alpha \in \mathcal{S}(Act)$ and there exists $p \overset{\alpha}{\mapsto} p'$, where $p' \models A$, or $\alpha \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$ and there exists $p \overset{\alpha'}{\mapsto} p'$, such that it is consistent with α and $p' \models A$.

For $\alpha \in \mathcal{S}(Act)$, $\ell :: p_1 \times p_2 \overset{\alpha}{\mapsto} p'$ iff, by rule par, $p_1 \overset{\alpha_1}{\mapsto} p'_1$ and $p_2 \overset{\alpha_2}{\mapsto} p'_2$, with $\alpha \in \alpha_1 \times \alpha_2$ and $p' = \ell :: p'_1 \times p'_2$. Assume $\alpha_i = a_i : \tau_i$, $\alpha = a : \tau$. We have $p \models < \beta > A$, for $\beta = a : \tau_1 \times \tau_2 \in \mathcal{E}(Act)$ iff $\beta \in \mathcal{E}(\alpha)$. We obtain $p \vdash < \alpha > A$ from $p \vdash < \beta > A$ by applying rule $\mathcal{E}$ $<>$.

Hence, it is sufficient to show the case of $< \beta > A$ with $\beta \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$. A similar reasoning applies also to the case of $[\alpha]A$. The proof is by cases on p_2.

Note that $p \overset{\alpha}{\mapsto} p'$ is consistent with β iff $p_1 \overset{\alpha_1}{\mapsto} p'_1$ and $p_2 \overset{\alpha_2}{\mapsto} p'_2$, with $\alpha \in \alpha_1 \times \alpha_2$. Moreover, for each $i \in \{1,2\}$, $p_i \overset{\alpha_i}{\mapsto} p'_i$ is consistent with $\beta_i = a_i : \tau_i$, where $\alpha_i = a_i : \tau'_i$ and $\beta = a : \tau_1 \times \tau_2$, $\alpha = a : \tau$ and $type(\alpha) = type^*(\beta)$. Therefore, $\beta \in \beta_1 \otimes \beta_2$ and by definition of extended types, $\beta_2 = \alpha_2$.

- Suppose $p_2 = \ell_1 :: \ell_2 :: a.t$.

  We have $p_2 \overset{\alpha_2}{\mapsto} p'_2$ iff either $\alpha_2 = a : \epsilon$ and $p'_2 = \ell_1 :: t$ or $\alpha_2 = b : \bar{\epsilon}$ with $b \in \#(a)$ and $p'_2 = p_2$.

  – If $\alpha_2 = a : \epsilon$, then $p_1 \overset{\alpha_1}{\mapsto} p'_1$, such that $\ell :: p'_1 \times \ell_1 :: t \models A$, i.e. $p_1 \models < \alpha_1 > (A)_{\ell\times\ell_1::t}$. There exists $\beta_1 = a_1 : \tau_1 \in \mathcal{E}(Act)$ and $\beta \in \beta_1 \otimes \alpha_2$ and $p_1 \overset{\alpha_1}{\mapsto} p'_1$ is consistent with β_1. Therefore, $p_1 \models < \beta_1 > (A)_{\ell\times\ell_1::t}$.
  Since $m(p_1) < m(p)$, by induction hypothesis, there exist $\{p''_i \in \mathcal{S}_{p_1}\}_{i\in\{1,n\}}$ and $\{p'''_j \in \mathcal{S}_{p_1}\}_{j\in\{1,k\}}$, such that $p''_i \models (A)_{\ell\times\ell_1::t}$, for each $i \in \{1,n\}$. Moreover, $p_1 \vdash < \beta_1 > (A)_{\ell\times\ell_1::t}$ can be obtained from $\{p''_i \vdash (A)_{\ell\times\ell_1::t}\}_{i\in\{1,n\}}$ and $\{p'''_j \vdash true\}_{j\in\{1,k\}}$.

  1. Since $p''_i \models (A)_{\ell\times\ell_1::t}$, then $\ell :: p''_i \times \ell_1 :: t \models A$. Moreover, since $p_1 \overset{\alpha'_1}{\mapsto} p''_i$ that is consistent with β_1, then by rule par, $p \overset{\alpha''_1}{\mapsto} \ell :: p''_i \times \ell_1 :: t$ that is consistent with β. Therefore, $\ell :: p''_i \times \ell_1 :: t \in \mathcal{S}_p$, for each $i \in \{1,n\}$, and $p'''_j \in \mathcal{S}_p$, for each $j \in \{1,k\}$.

2. We have $\ell :: p_1 \times p_2 \vdash < \beta > A$ from $p_1 \vdash < \beta_1 > (A)_{\ell\times\ell_1::t}$ by applying rule $\times <>_1$. Moreover, for each $i \in \{1,n\}$, we have $p_i'' \vdash (A)_{\ell\times\ell_1::t}$ from $\ell :: p_i'' \times \ell_1 :: t \vdash A$ by shift rule. Therefore, $p \vdash < \beta > A$ can be obtained from $\{\ell :: p_i'' \times \ell_1 :: t \vdash A\}_{i\in\{1,n\}}$ and $\{p_j''' \vdash true\}_{j\in\{1,k\}}$.

– If $\alpha_2 = b : \bar{\epsilon}$, the proof is similar to the previous one.

- Suppose $p_2 = \ell_1 :: \ell_2 :: nil$. The proof is similar to the previous case.

- Suppose $p_2 = \ell_1 :: \ell_2 :: b_1 + b_2$.

  By definition $p_2 \overset{\alpha_2}{\mapsto} p_2'$ has been obtained either by rule $choice_1$ or by rule $choice_2$.

  – If $p_2 \overset{\alpha_2}{\mapsto} p_2'$ has been obtained by rule $choice_1$, then $\ell_1 :: \ell_2 :: b_i \overset{\alpha_2}{\mapsto} p_2'$, where $type(\alpha_2) = \epsilon$, for some $i \in \{1,2\}$. By rule par, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha}{\mapsto} \ell :: p_1' \times p_2'$.

    Since $\ell :: p_1' \times p_2' \models A$, then $\ell :: p_1 \times \ell :: \ell_2 :: b_i \models\!< \alpha > A$ and $\ell :: p_1 \times \ell :: \ell_2 :: b_i \models\!< \beta > A$. Since $m(\ell :: p_1 \times \ell :: \ell_2 :: b_i) < m(p)$, by induction hypothesis, there exist $\{p_j' \in \mathcal{S}_{\ell::p_1\times\ell_1::\ell_2::b_i}\}_{j\in\{1,n\}}$ and $\{p_j''' \in \mathcal{S}_{\ell::p_1\times\ell_1::\ell_2::b_i}\}_{j\in\{1,k\}}$, such that $\{p_j'' \models A\}_{j\in\{1,n\}}$. Moreover, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash < \beta > A$ can be obtained from $\{p_j'' \vdash A\}_{j\in\{1,n\}}$ and $\{p_j''' \vdash true\}_{j\in\{1,k\}}$.

    1. It is obvious that $p_j'', p_j''' \in \mathcal{S}_p$, for each $j \in \{1,n\}$ and for each $j \in \{1,k\}$.
    2. Since $type(\alpha_2) = \epsilon$, then $type(proj_2(\beta)) = \epsilon$, and we obtain $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash < \beta > A$ by applying rule $\times <> +\epsilon$ to $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash < \beta > A$.

    Moreover, since $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha_j}{\mapsto} p_j''$ that is consistent with β and $type(proj_2(\beta)) = \epsilon$, it is obvious that $p \overset{\alpha_j}{\mapsto} p_j''$, for each $j \in \{1,n\}$.

  – If $p_2 \overset{\alpha_2}{\mapsto} p_2'$ has been obtained by rule $choice_2$, then $\ell_1 :: \ell_2 :: b_1 \overset{\alpha_2}{\mapsto} \ell_1 :: \ell_2 :: b_1$ and $\ell_1 :: \ell_2 :: b_2 \overset{\alpha_2}{\mapsto} \ell_1 :: \ell_2 :: b_2$, where $type(\alpha_2) = \bar{\epsilon}$ and $p_2' = p_2$. By rule par, for each $i \in \{1,2\}$, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha}{\mapsto} \ell :: p_1' \times \ell_1 :: \ell_2 :: b_i$. Since $p_2' = p_2$, $\ell :: p_1' \times \ell_1 :: \ell_2 :: b_1 + b_2 \models A$ and $\ell :: p_1' \times \ell_1 :: \ell_2 :: b_1 \models (A)_{\times b_2+}$ and $\ell :: p_1' \times \ell_1 :: \ell_2 :: b_2 \models (A)_{\times b_1+}$, by proposition B.1.1. Moreover, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 \models\!< \alpha > (A)_{\times b_2+}$ and $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \models\!< \alpha > (A)_{\times b_1+}$ and $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 \models\!< \beta > (A)_{\times b_2+}$ and $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \models\!< \beta > (A)_{\times b_1+}$.

    Since $m(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i) < m(p)$, by induction hypothesis, for each $i \in \{1,2\}$, there exist $\{p_{j,i}'' \in \mathcal{S}_{\ell::p_1\times\ell_1::\ell_2::b_i}\}_{j\in\{1,n_i\}}$, $\{p_{j,i}''' \in \mathcal{S}_{\ell::p_1\times\ell_1::\ell_2::b_i}\}_{j\in\{1,k_i\}}$ such that $p_{j,1}'' \models (A)_{\times b_2+}$ and $p_{j,2}'' \models (A)_{\times b_1+}$. Moreover, for each $i \in \{1,2\}$, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash < \beta > (A)_{\times b_j+}$ can be obtained from $\{p_{j,i}'' \vdash (A)_{\times b_j+}\}_{j\in\{1,n_i\}}$ and $\{p_{j,i}''' \vdash true\}_{j\in\{1,k_i\}}$.

1. Since $p''_{j,2} \models (A)_{\times b_1+}$ and $p''_{j,1} \models (A)_{\times b_2+}$, $p''_{j,2} = \ell :: p_{j,2} \times \ell_1 :: \ell_2 :: b'_2$ and $p''_{j,1} = \ell :: p_{j,1} \times \ell_1 :: \ell_2 :: b'_1$. Hence, $\ell :: p_{j,1} \times \ell_1 :: \ell_2 :: b'_1 + b_2 \models A$ and $\ell :: p_{j,2} \times \ell_1 :: \ell_2 :: b_1 + b'_2 \models A$. It is quite obvious that $\ell :: p_{j,1} \times \ell_1 :: \ell_2 :: b'_1 + b_2, \ell :: p_{j,2} \times \ell_1 :: \ell_2 :: b_1 + b'_2 \in \mathcal{S}_p$ and $p'''_{i,j} \in \mathcal{S}_p$.
2. Since $type(\alpha_2) = \bar{\epsilon}$, then $type(proj_2(\beta)) = \bar{\epsilon}$ and $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \vdash < \beta > A$ can be obtained from $\{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \vdash < \beta > (A)_{\times b_j}\}_{i,j \in \{1,2\}, i \neq j}$ by applying rule $\times <> +\bar{\epsilon}$. By shift rule premises $p''_{j,1} \vdash (A)_{\times b_2+}$ and $p''_{j,2} \vdash (A)_{\times b_1+}$ are obtained from $\ell :: p_{j,1} \times \ell_1 :: \ell_2 :: b'_1 + b_2 \vdash A$ and $\ell :: p_{j,2} \times \ell_1 :: \ell_2 :: b'_2 \vdash A$.

The case of $[\alpha]A$ is analogous.

- Suppose $p_2 = \ell_1 :: t \bigtriangledown p''$. By definition $p_2 \overset{\alpha_2}{\mapsto} p'_2$ has been obtained either by rule $\bigtriangledown_1$, $\bigtriangledown_2$ or $\bigtriangledown_3$. The cases of $\bigtriangledown_1$ and $\bigtriangledown_3$ are similar to those of $choice_1$ and $choice_2$, respectively.

  Assume $p_2 \overset{\alpha_2}{\mapsto} p'_2$ has been obtained by rule $\bigtriangledown_2$. By definition, $p'' \overset{\alpha_2}{\mapsto} p'''$, where $type(\alpha_2) = \epsilon$ and $p'_2 = \ell :: t \bigtriangledown p'''$. By rule par, $\ell :: p_1 \times p'' \overset{\alpha}{\longmapsto} \ell :: p'_1 \times p'''$.

  Since $\ell :: p'_1 \times p'_2 \models A$, then $\ell :: p'_1 \times p''' \models (A)_{\times \ell_1 :: t\bigtriangledown}$. Moreover, $\ell :: p_1 \times p'' \models\!< \alpha > (A)_{\times \ell_1 :: t\bigtriangledown}$ and $\ell :: p_1 \times p'' \models\!< \beta > (A)_{\times \ell_1 :: t\bigtriangledown}$. Since $m(\ell :: p_1 \times p'') < m(p)$, by induction hypothesis, there exist $\{p''_j \in \mathcal{S}_{\ell :: p_1 \times p''}\}_{j \in \{1,n\}}$ and $\{p'''_j \in \mathcal{S}_{\ell :: p_1 \times p''}\}_{j \in \{1,k\}}$, such that $\{p''_j \models (A)_{\times \ell_1 :: t\bigtriangledown}\}_{j \in \{1,n\}}$. Moreover, $\ell :: p_1 \times p'' \vdash < \beta > (A)_{\times \ell_1 :: t\bigtriangledown}$ can be obtained from $\{p''_j \vdash (A)_{\times \ell_1 :: t\bigtriangledown}\}_{j \in \{1,n\}}$ and $\{p'''_j \vdash true\}_{j \in \{1,k\}}$.

  1. For each $j \in \{1, k\}$, $p'''_j \in \mathcal{S}_p$. Since $p''_j \models (A)_{\times \ell_1 :: t\bigtriangledown}$, then $p''_j = \ell :: p_{1,j} \times p_{2,j}$ and $\ell :: p_{1,j} \times \ell_1 :: t \bigtriangledown p_{2,j} \models A$. Since $\ell :: p'_1 \times p'_2 \overset{\alpha_j}{\mapsto} p''_j$ that is consistent with β and $type(proj_2(\beta)) = \epsilon$, then $p \overset{\alpha_j}{\mapsto} \ell :: p_{1,j} \times \ell_1 :: t \bigtriangledown p_{2,j}$. Therefore, $\ell :: p_{1,j} \times \ell_1 :: t \bigtriangledown p_{2,j} \in \mathcal{S}_p$.
  2. Since $type(\alpha_2) = \epsilon$, then $type(proj_2(\beta)) = \epsilon$, and we obtain $\ell :: p_1 \times \ell :: t \bigtriangledown p'' \vdash < \beta > A$ by applying rule $\times <> \bigtriangledown\epsilon_1$ to $\ell :: p_1 \times p'' \vdash < \beta > (A)_{\times \ell_1 :: t\bigtriangledown}$. Moreover, premises $\{p''_j \vdash (A)_{\times \ell_1 :: t\bigtriangledown}\}_{j \in \{1,n\}}$ are obtained by shift rule from $\{\ell :: p_{1,j} \times \ell_1 :: t \bigtriangledown p_{2,j} \vdash A\}_{i \in \{1,n\}}$.

- Suppose $p_2 = \ell_2 :: rec\, x.t$.

  By definition $p_2 \overset{\alpha_2}{\mapsto} p'_2$ iff either rule rec_1 or rec_2 has been applied.

  - If $p_2 \overset{\alpha_2}{\mapsto} p'_2$ has been obtained by rule rec_1, then $\ell_2 :: t[rec\, x.t/x] \overset{\alpha_2}{\mapsto} p'_2$ and $type(\alpha_2) = \epsilon$. By rule par, $\ell :: p_1 \times \ell_2 :: t[rec\, x.t/x] \overset{\alpha}{\mapsto} \ell :: p'_1 \times p'_2$. Since $\ell :: p'_1 \times p'_2 \models A$, then $\ell :: p_1 \times \ell_2 :: t[rec\, x.t/x] \models\!< \alpha > A$ and $\ell :: p_1 \times \ell_2 :: t[rec\, x.t/x] \models\!< \beta > A$.

Since $m(\ell :: p_1 \times \ell_2 :: t[rec\ x.t/x]) < m(p)$, by induction hypothesis, there exist $\{p_i'' \in \mathcal{S}_{\ell::p_1\times\ell_2::t[rec\ x.t/x]}\}_{i\in\{1,n\}}$ and $\{p_j''' \in \mathcal{S}_{\ell::p_1\times\ell_2::t[rec\ x.t/x]}\}_{j\in\{1,k\}}$, such that $p_i'' \models A$, for each $i \in \{1,n\}$. Moreover, $\ell :: p_1 \bigtriangledown \ell_2 :: t[rec\ x.t/x] \vdash < \beta > A$ can be obtained from $\{p_i'' \vdash A\}_{i\in\{1,n\}}$ and $\{p_j''' \vdash true\}_{j\in\{1,k\}}$.

1. It is obvious that $p_i'', p_j''' \in \mathcal{S}_p$, for each $i \in \{1,n\}$, and for each $j \in \{1,k\}$.
2. We obtain $\ell :: p_1 \times p_2 \vdash < \beta > A$ by applying rule $\times <> rec$ to $\ell :: p_1 \times \ell_2 :: t[rec\ x.t/x] \vdash < \beta > A$.

Moreover, since $\ell :: p_1 \bigtriangledown \ell_2 :: t[rec\ x.t/x] \overset{\alpha_i}{\mapsto} p_i''$ that is consistent with β and $type(proj_2(\beta)) = \epsilon$, it is obvious that $\ell :: p_1 \bigtriangledown \ell_2 :: rec\ x.t \overset{\alpha_i}{\mapsto} p_i''$, for each $i \in \{1,n\}$.

– If $p_2 \overset{\alpha_2}{\mapsto} p_2'$ has been obtained by rule rec_2, then $\ell_2 :: t[rec\ x.t/x] \overset{\alpha_2}{\mapsto} \ell_2 :: t[rec\ x.t/x]$, $type(\alpha_2) = \bar{\epsilon}$ and $p_2' = \ell_2 :: rec\ x.t$. Since $\ell :: p_1' \times p_2' \models A$, then by proposition B.1.1, $\ell :: p_1' \times \ell_2 :: t[rec\ x.t/x] \models A$. Moreover, $\ell :: p_1 \times \ell_2 :: t[rec\ x.t/x] \models < \alpha > A$, $\ell :: p_1 \times \ell_2 :: t[rec\ x.t/x] \models < \beta > A$. The proof is as in the previous case.

- Suppose $p_2 = \ell_2 :: p_3 \times p_4$.

We have $p_2 \overset{\alpha_2}{\mapsto} p_2'$ iff $p_3 \overset{\alpha_2'}{\mapsto} p_3'$ and $p_4 \overset{\alpha_2''}{\mapsto} p_4'$, where $\alpha_2 \in \alpha_2' \times \alpha_2''$ and $p_2' = \ell_2 :: p_3' \times p_4'$. Since $\ell :: p_1' \times \ell_2 :: (p_3' \times p_4') \models A$, then $\ell :: \ell_2 :: (p_1' \times p_3') \times p_4' \models (A)_\times$.

We have $\alpha \in (\alpha_1 \times \alpha_2') \times \alpha_2''$ by associativity of $\times$. By rule par, $\ell :: \ell_2 :: (p_1 \times p_3) \times p_4 \overset{\alpha}{\mapsto} \ell :: \ell_2 :: (p_1' \times p_3') \times p_4'$.

Moreover, $\ell :: \ell_2 :: (p_1 \times p_3) \times p_4 \overset{\alpha}{\mapsto} \ell :: \ell_2 :: (p_1' \times p_3') \times p_4'$ is consistent with β', for $\beta' \in left(\beta)$, such that $\beta' = a : (\tau_1 \times \tau_2) \times \tau_3$, where $\tau_3 = type(\alpha_2'')$, $type(\alpha_2') = \tau_2$ and $\tau_1 = type(\alpha_1)$.

Therefore, $\ell :: \ell_2 :: (p_1 \times p_3) \times p_4 \models < \beta' > (A)_\times$. Since $m(\ell :: \ell_2 :: (p_1 \times p_3) \times p_4) < m(p)$, by induction hypothesis there exist $\{p_i'' \in \mathcal{S}_{\ell::\ell_2::(p_1\times p_3)\times p_4)}\}_{i\in\{1,n\}}$ and $\{p_j''' \in \mathcal{S}_{\ell::\ell_2::(p_1\times p_3)\times p_4)}\}_{j\in\{1,k\}}$, such that $p_i'' \models (A)_\times$, for each $i \in \{1,n\}$. Moreover, $\ell :: \ell_2 :: p_1 \times p_3 \times p_4 \vdash < \beta' > (A)_\times$ can be obtained from from $\{p_i'' \vdash (A)_\times\}_{i\in\{1,n\}}$ and $\{p_j''' \vdash true\}_{j\in\{1,k\}}$.

1. Since $p_i'' \in \mathcal{S}_{\ell::\ell_2::(p_1\times p_3)\times p_4)}$ and $p_i'' \models (A)_\times$, then $p_i'' = \ell :: \ell_2 :: (p_{i,1}'' \times p_{i,3}'') \times p_{i,4}''$, and $\ell :: p_{i,1}'' \times \ell_2 :: (p_{i,3}'' \times p_{i,4}'') \models A$. By repeating a reasoning similar to the one of the other cases, $\ell :: p_{i,1}'' \times \ell_2 :: (p_{i,3}'' \times p_{i,4}'') \in \mathcal{S}_p$. Moreover, $p_j''' \in \mathcal{S}_p$, for each $j \in \{1,k\}$.
2. Since $\beta' \in left(\beta)$, we obtain $\ell :: p_1 \times p_2 \vdash < \beta > A$ by applying rule $\times\times <>$ to $\ell :: \ell_2 :: (p_1 \times p_3) \times p_4 \vdash < \beta' > (A)_\times$. Moreover, $\{p_i'' \vdash (A)_\times\}_{i\in\{1,n\}}$ are obtained by shift rule from $\{\ell :: p_{i,1}'' \times \ell_2 :: (p_{i,3}'' \times p_{i,4}'') \vdash A\}_{i\in\{1,n\}}$.

■

Theorem 4.4.5 Assume $p \in \mathcal{CSP}$ be a regular process over Act and Loc. For a closed L_μ formula A over Act and Loc and with tags in $\mathcal{S}_p$, if $p \models A$ then $p \vdash A$.

Proof. We show that for each $p' \in \mathcal{S}_p$, if $p' \models A$, then $p' \vdash A$. The proof is by well-founded induction on $\prec$.

- $A = P$ $(resp. \neg P)$. Since $p' \models A$ then $P \in loc(p')$(resp. $P \notin loc(p')$). In both cases $p' \vdash A$ by $prop_1$ (resp. $prop_2$).

- $A = A_0 \wedge A_1$. Since $p' \models A$, then $p' \models A_i$, for each $i \in \{0,1\}$. Moreover, $A_i \prec A$ and, by induction hypothesis, $p' \vdash A_i$, for each $i \in \{0,1\}$. The claim trivially follows by rule $\wedge$.

- $A = A_0 \vee A_1$. Since $p' \models A$, then $p' \models A_i$, for some $i \in \{0,1\}$. Moreover, $A_i \prec A$ and, by induction hypothesis, $p' \vdash A_i$, for some $i \in \{0,1\}$. The claim trivially follows by rule $\vee$.

- $A = \mu X\{U\}.B$. Since $p' \models \mu X\{U\}.B$, then by the semantics of least fixpoints $p' \notin U$ and by lemma 2.1.3, $p' \models B[\mu X\{U, p'\}.B/X]$. Since $B[\mu X\{U, p'\}.B/X] \prec \mu X\{U\}.B$, by induction hypothesis, $p' \vdash B[\mu X\{U, p'\}.B/X]$ and by rule μ, $p' \vdash \mu X\{U\}.B$.

- $A = \nu X\{U\}.B$. By lemma 2.1.3 $p' \models \nu X\{U\}.B$ implies that either $p' \in U$ or $p' \models B[\nu X\{U, p'\}.B/X]$. In the first case $p' \vdash \nu X\{U\}.B$ by rule ν_2. In the second case, since $B[\nu X\{U, p'\}.B/X] \prec \nu X\{U\}.B$, by induction hypothesis, $p' \vdash B[\nu X\{U, p'\}.B/X]$ and by rule ν_1, $p' \vdash \nu X\{U\}.B$.

- $A =< \alpha > B$ $(A = [\alpha]B)$. Since $p' \models< \alpha > B$ then by lemma 4.4.4 it follows that there exist $\{p_i \in \mathcal{S}_{p'}\}_{i\in\{1,n\}}$ and $\{p'_i \in \mathcal{S}_{p'}\}_{i\in\{1,k\}}$, such that $\{p_i \models B\}_{i\in\{1,n\}}$. Moreover, $p' \vdash< \alpha > B$ can be obtained from $\{p_i \vdash B\}_{i\in\{1,n\}}$ and $\{p'_i \vdash true\}_{i\in\{1,k\}}$. Since $B \prec< \alpha > B$ and by definition $\{p_i \in \mathcal{S}_p\}_{i\in\{1,n\}}$ and $\{p'_i \in \mathcal{S}_p\}_{i\in\{1,k\}}$, by induction hypothesis proofs $\{p_i \vdash B\}_{i\in\{1,n\}}$ and $\{p'_i \vdash true\}_{i\in\{1,k\}}$ indeed exist.

■

Appendix C

Proofs of chapter 6

C.1 Soundness of the proof system

Lemma 6.3.1 Let ψ be a monotonic function on the lattice $\mathcal{P}(D)$. For $P \in \mathcal{P}(D)$,

1. $P \subseteq \psi(\mu V.(\psi(V) \setminus P)) \Rightarrow P \subseteq \mu V.(\psi(V))$;
2. $P \subseteq \psi(\nu V.(\psi(V) \cup P)) \Rightarrow P \subseteq \nu V.(\psi(V))$.

Proof.

1. If $P \subseteq \psi(\mu V.(\psi(V) \setminus P))$ then $P \subseteq \mu V.(\psi(V))$. Consider the function $\psi'(V) \equiv \psi(V) \setminus P$. We have $\psi'(\mu V.\psi(V)) = \psi(\mu V.\psi(V)) \setminus P = \mu V.\psi(V) \setminus P$. Therefore, $\psi'(\mu V.\psi(V)) \subseteq \mu V.\psi(V)$, i.e. $\mu V.\psi(V)$ is a prefixed point of ψ'. Since $\mu V.(\psi(V) \setminus P)$ is the least fixed point of this function, $\mu V.(\psi(V) \setminus P) \subseteq \mu V.\psi(V)$. Since ψ is monotonic, $\psi(\mu V.(\psi(V) \setminus P)) \subseteq \psi(\mu V.\psi(V)) = \mu V.\psi(V)$. Since $P \subseteq \psi(\mu V.(\psi(V) \setminus P))$, then $P \subseteq \mu V.\psi(V)$.
2. If $P \subseteq \psi(\nu V.(\psi(V) \cup P))$ then $P \subseteq \nu V.(\psi(V))$. Since $\nu V.(\psi(V) \cup P)$ is a fixed point of the function $\psi'(V) \equiv \psi(V) \cup P$, we have $\nu V.(\psi(V) \cup P) = P \cup \psi(\nu V.(\psi(V) \cup P))$ and since $P \subseteq \psi(\nu V.(\psi(V) \cup P))$, $\nu V.(\psi(V) \cup P) = \psi(\nu V.(\psi(V) \cup P))$. Hence, $\nu V.(\psi(V) \cup P)$ is a fixed point of ψ, i.e. a postfixed point. Therefore $\nu V.(\psi(V) \cup P) \subseteq \nu V.\psi(V)$, as $\nu V.\psi(V)$ is the greatest fixed point and by monotonicity $\psi(\nu V.(\psi(V) \cup P)) \subseteq \psi(\nu V.\psi(V))$. Since $P \subseteq \psi(\nu V.(\psi(V) \cup P))$, then $P \subseteq \psi(\nu V.\psi(V)) = \nu V.\psi(V)$.

■

Theorem 6.3.2 Let $\mathcal{G} = ((S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph, γ be a corresponding valuation for propositional symbols $PROP$ and A be a closed RTL_μ formula over $\mathcal{G}$, $PROP$ and C. For $s \in S$ and $\omega \in \Omega(clock(s) \cup C)$, if $s, \omega \vdash A$ then $s, \omega \models A$.

Proof. We show that each rule is backward sound.

- The soundness of rules $prop_1, prop_2, , taut, disj, thin, \wedge, \vee$ is trivial.

- $$\frac{s, \omega \vdash x.A}{s, reset(\omega, (clock(s) \cup C) \setminus \{x\}, x) \vdash A}\, reset$$

  Suppose $s, reset(\omega, (clock(s) \cup C) \setminus \{x\}, x) \models A$. By proposition 6.2.2, if $\pi \models \omega$, then $pro_{(clock(s) \cup C) \setminus \{x\}}(\pi) \cup \{x := 0\} \models reset(\omega, (clock(s) \cup C) \setminus \{x\}, x)$. Moreover, $pro_{(clock(s) \cup C) \setminus \{x\}}(\pi) \cup \{x := 0\} = \pi[x := 0]$, since $x \in C$. Hence, $s, \omega \models x.A$.

- $$\frac{s, \omega_1 \vdash < \alpha > A}{s', \omega_2 \vdash A} <> \quad \begin{array}{l} \models \forall(\omega_1 \supset \omega) \text{ for } s \stackrel{\alpha,\omega}{\longmapsto} s' \\ \omega_2 \equiv reset(\omega_1, C_1, C_2) - 1 \\ C_1 = (clock(s') \cap clock(s)) \cup C \\ C_2 = clock(s') \setminus clock(s) \\ \models \forall(\omega_2 \supset delay(s')) \end{array}$$

  Suppose that $s', \omega_2 \models A$. Since $\models \forall(\omega_1 \supset \omega)$, then for each $\pi \models \omega_1$, $\pi \models \omega$. By proposition 6.2.2, $reset(\pi, C_1, C_2) \models reset(\omega_1, C_1, C_2)$ and $reset(\pi, C_1, C_2) + 1 \models reset(\omega_1, C_1, C_2) - 1 \equiv \omega_2$. Since $\models \forall(\omega_2 \supset delay(s'))$, then for each $\pi \models \omega_1$, there exists $(s, \pi) \stackrel{\alpha}{\mapsto} (s', reset(\pi, C_1, C_2)+1)$, such that $(s', reset(\pi, C_1, C_2) +1) \models A$. Hence, $s, \omega \models < \alpha > A$.

- $$\frac{s, \omega \vdash [\alpha]A}{\{s_i, \omega_i \vdash A\}_{i \in \{1,n\}}}\, [] \quad \begin{array}{l} \{s \stackrel{\alpha,\omega_i'}{\longmapsto} s_i \mid\not\models \forall(\omega \wedge \omega_i' \supset false)\}_{i \in \{1,n\}} \\ \omega_i \equiv reset(\omega_i' \wedge \omega, C_{1,i}, C_{2,i}) - 1 \\ C_{1,i} = (clock(s_i) \cap clock(s)) \cup C \\ C_{2,i} = clock(s_i) \setminus clock(s) \end{array}$$

  Suppose that for each $i \in \{1, n\}$, $s_i, \omega_i \models A$. Let $\{s \stackrel{\alpha,\omega_i'}{\mapsto} s_i\}_{i \in \{1,n\}}$ be the set of transitions, such that $\omega \wedge \omega_i'$ is satisfiable. For each π, such that $\pi \models \omega$, there exists $(s, \pi) \stackrel{\alpha}{\mapsto} (s', \pi')$ iff for some $i \in \{1, n\}$, $\pi \models \omega_i'$ and $\pi' = reset(\pi, C_{1,i}, C_{2,i}) + 1$, $s' = s_i$, $\pi' \models delay(s_i)$.

  By proposition 6.2.2, if $\pi \models \omega_i' \wedge \omega$, then $reset(\pi, C_{1,i}, C_{2,i}) \models reset(\omega_i' \wedge \omega, C_{1,i}, C_{2,i})$, and $reset(\pi, C_{1,i}, C_{2,i}) + 1 \models reset(\omega_i' \wedge \omega, C_{1,i}, C_{2,i}) - 1 \equiv \omega_i$. Therefore, $s, \omega \models [\alpha]A$.

- Rule ν_1. By definition of the semantics.

- Rule ν_2. Suppose $s, \omega \models A[\nu X\{U \cup \{(s, \omega)\}\}.A/X]$. It is obvious that $(s, \omega) \in \| A[\nu X\{U \cup \{(s, \omega)\}\}.A/X] \cup U \|_{\mathcal{M}(\mathcal{G},\gamma,C)}$. By lemma 6.3.1 $s, \omega \models \nu X\{U\}.A$.

- Rule μ. Suppose $s, \omega \models A[\mu X\{U \cup \{(s, \omega)\}\}.A/X]$. Since $\| (s, \omega) \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \cap \| U \|_{\mathcal{M}(\mathcal{G},\gamma,C)} = \emptyset$, then $(s, \omega) \in \| A[\nu X\{U \cup \{(s, \omega)\}\}.A/X] \setminus U \|_{\mathcal{M}(\mathcal{G},\gamma,C)}$. Hence, by lemma 6.3.1 $s, \omega \models \mu X\{U\}.A$.

■

C.2 Completeness of the proof system

Lemma 6.3.4 Let $\mathcal{G} = (S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph, γ be a corresponding valuation over $PROP$ and A be a fixpoint-quantifier free RTL_μ formula over $\mathcal{G}$, $PROP$ and C with free formula variables $X_1, \ldots, X_n$. If $\rho(X_i) \in \mathcal{R}_{(\mathcal{G},A)}$, for each $i \in \{1, n\}$, then $\| A \|_{\mathcal{M}(\mathcal{G},\gamma,C),\rho} \in \mathcal{R}_{(\mathcal{G},A)}$.

Proof.

The proof is by induction on the structure of A. We use $\| - \|_\rho$ for $\| - \|_{\mathcal{M}(\mathcal{G},\gamma,C),\rho}$.

- $A = X$ is a logical variable. Then $\| X \|_\rho = \{(s, \pi) \mid s \in \rho(X)\}$. Since $\rho(X) \in \mathcal{R}_{(\mathcal{G},A)}$, then $\| A \|_\rho \in \mathcal{R}_{(\mathcal{G},A)}$.

- $A = P$ is a propositional symbol. Then $\| A \|_\rho = \{(s, \pi) \mid s \in \gamma(P)\}$. Hence, $\| A \|_\rho = \{(s, \pi) \mid s \in \gamma(P), \pi \models true\} \in \mathcal{R}_{(\mathcal{G},A)}$.

- $A = \neg P$ where P is a propositional symbol. Then $\| A \|_\rho = \{(s, \pi) \mid s \in S \setminus \gamma(P)\}$. Hence, $\| A \|_\rho \in \mathcal{R}_{(\mathcal{G},A)}$.

- $A = \omega \in \Omega(C)$. By definition ω is a $(\mathcal{G}, A)$-constraint and $\| A \|_\rho = \{(s, \pi) \mid s \in S, \pi \models \omega \wedge delay(s)\}$. Since $\omega \wedge delay(s)$ is a $(\mathcal{G}, A)$-constraint, then $\| A \|_\rho \in \mathcal{R}_{(\mathcal{G},A)}$.

- $A = A_1 \wedge A_2$. Hence $\| A \|_\rho = \| A_1 \|_\rho \cap \| A_2 \|_\rho$. Since by induction hypothesis, for each $i \in \{1, 2\}$, $\| A_i \|_\rho = \mathcal{R}_i \in \mathcal{R}_{(\mathcal{G},A_i)}$, then $\| A_1 \|_\rho \cap \| A_2 \|_\rho = \mathcal{R}_1 \cap \mathcal{R}_2 \in \mathcal{R}_{(\mathcal{G},A)}$, since the intersection of regions is trivially a region.

- $A = A_1 \vee A_2$. Hence $\| A \|_\rho = \| A_1 \|_\rho \cup \| A_2 \|_\rho$. Since by induction hypothesis, for each $i \in \{1, 2\}$, $\| A_i \|_\rho = \mathcal{R}_i \in \mathcal{R}_{(\mathcal{G},A_i)}$, $\| A \|_\rho = \mathcal{R}_1 \cup \mathcal{R}_2 \in \mathcal{R}_{(\mathcal{G},A)}$, since the union of regions is trivially a region.

- $A = x.A_1$. By definition $\| x.A_1 \|_\rho = \{(s, \pi) \mid (s, \pi[x := 0]) \in \| A_1 \|_\rho\}$. By induction hypothesis, $\| A_1 \|_\rho = \mathcal{R} \in \mathcal{R}_{(\mathcal{G},A_1)}$. Hence, for each $s \in S$, $(s, \pi) \in \mathcal{R}$ iff $\pi \models \omega_s$, for some $(\mathcal{G}, A)$-constraint $\omega_s \in \Omega(clock(s) \cup C)$. We have that $\pi[x := 0] \models \omega_s$ iff $\pi \models \omega_s[x := 0]$. Therefore, $(s, \pi) \in \| A \|_\rho$ iff $\pi \models \omega_s[x := 0]$ and $\omega_s[x := 0] \in \Omega(C \cup clock(s))$ is a $(\mathcal{G}, A)$-constraint. Hence, $\| A \|_\rho \in \mathcal{R}_{\mathcal{G},A}$.

- $A = < \alpha > A_1$. By definition $\| A \|_\rho = \| < \alpha > \| (\| A_1 \|_\rho)$. By induction hypothesis, $\| A_1 \|_\rho = \mathcal{R} \in \mathcal{R}_{(\mathcal{G},A_1)}$. Moreover, $\| < \alpha > \| (\mathcal{R}) = \{(s, \pi) \mid (s, \pi) \overset{\alpha}{\mapsto} (s', \pi'), (s', \pi') \in \mathcal{R}\}$. Since $\mathcal{R}$ is a $(\mathcal{G}, A_1)$-region, then, for each $s \in S$, there exists $\omega_s \in \omega(clock(s) \cup C)$, such that ω_s is a $(\mathcal{G}, A_1)$-constraint and $\pi \models \omega_s$ iff $(s, \pi) \in \mathcal{R}$. It is obvious that $\models \forall(\omega_s \supset delay(s))$.

  For each $s \in S$, let $\{s \overset{\alpha,\omega_j}{\longmapsto} s_j\}_{j \in \{1,k_s\}}$ be the set of transitions from s. For each $j \in \{1, k_s\}$, consider the clock constraint $\omega'_j \equiv \omega_j \wedge (\omega_{s_j} + 1)[C_{2,j} := 0]$, where $C_{2,j} \equiv clock(s_j) \setminus clock(s)$.

  We show that, for each $s \in S$, $(s, \pi) \in \| < \alpha > \| (\mathcal{R})$ iff $\pi \models \bigvee_{j \in \{1,k_s\}} \omega'_j \equiv \omega'_s$.

1. $\Rightarrow$ Suppose $(s,\pi) \in \| < \alpha > \| (\mathcal{R})$, i.e. there exists $(s,\pi) \stackrel{\alpha}{\mapsto} (s',\pi')$, with $(s',\pi') \in \mathcal{R}$. By definition, $\pi \models \omega_j$, for some $j \in \{1, k_s\}$ and $s' = s_j$, $\pi' = reset(\pi, C_{1,j}, C_{2,j}) + 1$, where $C_{1,j} = C \cup (clock(s) \cap clock(s_j))$, and $\pi' \models delay(s_j)$. Moreover, $\pi' \models \omega_{s_j}$.

   Since $reset(\pi, C_{1,j}, C_{2,j}) + 1 \models \omega_{s_j}$, then $reset(\pi, C_{1,j}, C_{2,j}) \models \omega_{s_j} + 1$, by proposition 6.2.2. Moreover, $reset(\pi, C_{1,j}, C_{2,j}) = pro_{C_{1,j}}(\pi) \cup \{C_{2,j} := 0\}$. Hence, $\pi \models (\omega_{s_j} + 1)[C_{2,j} := 0]$, since $C_{2,j} \cap C_{1,j} = \emptyset$. Since, $\pi \models \omega_j$, then $\pi \models \omega'_j$.

2. $\Leftarrow$ Suppose $\pi \models \bigvee_{j \in \{1,k_s\}} \omega'_j$. Then there exists $j \in \{1, k_s\}$, such that $\pi \models \omega'_j \equiv \omega_j \wedge (\omega_{s_j} + 1)[C_{2,j} := 0]$. By proposition 6.2.2, $reset(\pi, C_{1,j}, C_{2,j}) \models reset((\omega_{s_j} + 1)[C_{2,j} := 0], C_{1,j}, C_{2,j})$ so that $reset(\pi, C_{1,j}, C_{2,j}) \models \omega_{s_j} + 1$. By proposition 6.2.2, $reset(\pi, C_{1,j}, C_{2,j}) + 1 \models \omega_{s_j}$. Since, $\pi \models \omega_j$ and $reset(\pi, C_{1,j}, C_{2,j}) + 1 \models \omega_{s_j}$ and $\models \forall(\omega_{s_j} \supset delay(s_j))$, then $(s,\pi) \stackrel{\alpha}{\mapsto} (s_j, reset(\pi, C_{1,j}, C_{2,j}) + 1)$ and $(s_j, reset(\pi, C_{1,j}, C_{2,j}) + 1) \in \mathcal{R}$.

Therefore, $(s,\pi) \in \| < \alpha > \| (\mathcal{R})$ iff $\pi \models \omega'_s$, where $\omega'_s = \bigvee_{j \in \{1,k_s\}} \omega'_j$. Since $\omega_j \in \Omega(clock(s))$ and $\omega_{s_j} \in \Omega(C \cup clock(s_j))$ are $(\mathcal{G}, A)$-constraints, then $\omega_j \wedge (\omega_{s_j} + 1)[C_{2,j} := 0] \in \Omega(clock(s) \cup C)$ is a $(\mathcal{G}, A)$-constraint. Hence, $\| A \|_\rho \in \mathcal{R}_{(\mathcal{G},A)}$.

- $A = [\alpha]A_1$. Note that $\| A \|_\rho = TS \setminus \| < \alpha > \|_\rho (TS \setminus \| A_1 \|_\rho)$. By induction hypothesis, $\| A_1 \|_\rho \in \mathcal{R}_{(\mathcal{G},A_1)}$. Moreover, $TS \setminus \mathcal{R} \in \mathcal{R}_{(\mathcal{G},A)}$ if $\mathcal{R}$ is a region. By applying the previous result for $<>$, $\| A \|_\rho \in \mathcal{R}_{(\mathcal{G},A)}$.

■

Lemma 6.3.6 Let ψ be a monotonic function on the lattice $\mathcal{P}(D)$. For $P \in \mathcal{P}(D)$,

$$P \subseteq \nu V.\psi(V) \Rightarrow P \subseteq \psi(\nu V.(\psi(V) \cup P)).$$

Proof. Assume $P \subseteq \nu V.\psi(V)$. Then $P \cup \nu V.\psi(V) = \nu V.\psi(V) = \psi(\nu V.\psi(V))$. Hence, $\nu V.\psi(V)$ is a postfixed point of $\psi'(V) = \psi(V) \cup P$. Since $\nu V.(\psi(V) \cup P)$ is the greatest fixed point of this function, then $\nu V.\psi(V) \subseteq \nu V.(\psi(V) \cup P)$. Since ψ is monotonic, $\nu V.\psi(V) = \psi(\nu V.\psi(V)) \subseteq \psi(\nu V.(\psi(V) \cup P))$. Hence, $P \subseteq \nu V.\psi(V)$ implies $P \subseteq \psi(\nu V.(\psi(V) \cup P))$. ■

Lemma 6.3.7 Let ψ be a monotonic function on the lattice $\mathcal{P}(D)$ such that $\mu V.\psi(V) = \psi^k(\emptyset)$ for some integer k. For $P \in \mathcal{P}(D)$, there exists $\{P_i\}_{i \in \{1,n\}}$, such that $P = \bigcup_{i \in \{1,n\}} P_i$, and, for each $i \in \{1, n\}$,

$$P_i \subseteq \mu V.\psi(V) \Rightarrow P_i \subseteq \psi(\mu V.(\psi(V) \setminus P_i)).$$

Proof. Since $P \subseteq \mu V.\psi(V) = \psi^k(\emptyset)$ then there exists $\{k_1, \ldots, k_n\}$ such that $k_i \leq k$ and $k_i < k_{i+1}$, for each $i \in \{1, n\}$, and $\psi^{k_i}(\emptyset) \cap P = P'_i$ and for each $k' < k_i$, $\psi^{k'}(\emptyset) \cap P \subset P'_i$. Since ψ is monotonic, $P'_{i-1} \subset P'_i$.

Let $P_i = (P'_i \setminus P'_{i-1})$. We prove that $P_i \subseteq \psi(\mu V.(\psi(V) \setminus P_i))$ for each $i \in \{1, n\}$. Let $\psi_i(V) = \psi(V) \setminus P_i$. Since $P_i \notin \psi^{k'}(\emptyset)$, for each $i \in \{1, n\}$ and for each $k' < k_i$, then $\psi^{k'}(\emptyset) = \psi_i^{k'}(\emptyset)$ for each $k' < k_i$. Hence, $\psi^{k_i-1}(\emptyset) = \psi_i^{k_i-1}(\emptyset)$. Since $P_i \subseteq \psi(\psi^{k_i-1}(\emptyset)) = \psi^{k_i}(\emptyset)$ then $P_i \subseteq \psi(\psi_i^{k_i-1}(\emptyset))$.

Since $\psi_i^{k_i-1}(\emptyset) \subseteq \mu V.(\psi(V) \setminus P_i)$, by monotonicity $\psi(\psi_i^{k_i-1}(\emptyset)) \subseteq \psi(\mu V.(\psi(V) \setminus P_i))$. Hence, $P_i \subseteq \psi(\psi_i^{k_i-1}(\emptyset))$ implies $P_i \subseteq \psi(\mu V.(\psi(V) \setminus P_i))$. ■

Theorem 6.3.9 Let $\mathcal{G} = (S, s_0, Act, C', \mapsto, delay, clock)$ be a discrete timed graph, γ be a corresponding valuation over $PROP$ and A be a closed RTL_μ formula over $\mathcal{G}$, $PROP$ and C. For $s \in S$ and $\omega \in \Omega(clock(s) \cup C)$, if $s, \omega \models A$ then $s, \omega \vdash A$.

Proof. If $s, \omega \models A$ there are two cases:

1. $\omega \wedge delay(s)$ is not satisfiable. By rule *empty*, $s, \omega \vdash A$;

2. $\omega \wedge delay(s)$ is satisfiable. It is sufficient to consider the case of $\models \forall(\omega \supset delay(s))$. The proof is by induction on $\prec$, that is well-founded for finite-state timed graphs.

   - $A \equiv P$. Therefore, $s, \omega \models A$ iff $s \in \gamma(P)$. We have $s, \omega \vdash P$ by $prop_1$.
   - $A \equiv \neg P$. Therefore, $s, \omega \models A$ iff $s \notin \gamma(P)$. Hence, $s, \omega \vdash P$ by $prop_2$.
   - $A \equiv \omega'$. Therefore, $s, \omega \models A$ iff $\models \forall(\omega \supset \omega')$. We have $s, \omega \vdash A$ by applying rules *taut*.
   - $A \equiv A_1 \wedge A_2$. Therefore, $s, \omega \models A_1$ and $s, \omega \models A_2$. Since $A_i \prec A$, by induction hypothesis, both $s, \omega \vdash A_1$ and $s, \omega \vdash A_2$. By rule $\wedge$, $s, \omega \vdash A$.
   - $A \equiv A_1 \vee A_2$. Therefore, either $s, \omega \models A_1$ or $s, \omega \models A_2$. Since $A_i \prec A$, induction hypothesis, either $s, \omega \vdash A_1$ or $s, \omega \vdash A_2$ and by rule $\vee$, $s, \omega \vdash A$.
   - $A \equiv x.A_1$. By definition $s, \omega \models x.A_1$ iff, for each $\pi \models \omega$, $(s, \pi[x := 0]) \models A_1$. Since ω is satisfiable, by proposition 6.2.2, for each $\pi' \models reset(\omega, (C \cup clock(s)) \setminus \{x\}, x)$, there exists $\pi \models \omega$, such that $\pi' = pro_{(C \cup clock(s)) \setminus \{x\}}(\pi) \cup \{x := 0\}$. Since $x \in C$, then $\pi[x := 0] = \pi'$. Hence, $(s, \pi') \models A_1$, for each $\pi' \models reset(\omega, (C \cup clock(s)) \setminus \{x\}, x)$, i.e. $s, reset(\omega, (C \cup clock(s)) \setminus \{x\}, x) \models A_1$. Since $A_1 \prec A$, by induction hypothesis, $s, reset(\omega, (C \cup clock(s)) \setminus \{x\}, x) \vdash A_1$ and by rule $reset$, $s, \omega \vdash A$.
   - $A \equiv < \alpha > A_1$. By definition for each $\pi \models \omega$, there exists $(s, \pi) \stackrel{\alpha}{\mapsto} (s', \pi')$, such that $(s', \pi') \models A_1$.

     Let $\{s \stackrel{\alpha, \omega_j}{\mapsto} s_j\}_{j \in \{1,n\}}$ be the set of transitions such that, for each $\pi \models \omega$, $\pi \models \omega_j$, $(s_j, reset(\pi, C_{1,j}, C_{2,j}) + 1) \models A_1$ and $reset(\pi, C_{1,j}, C_{2,j}) + 1 \models$

$delay(s_j)$, for some $j \in \{1,n\}$, where $C_{1,j} = C \cup (clock(s) \cap clock(s_j))$ and $C_{2,j} = clock(s_j) \cup clock(s)$.

By lemma 6.3.5, $\| A_1 \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \in \mathcal{R}_{(\mathcal{G},A_1)}$. Therefore, for each $j \in \{1,n\}$, there exists $\omega_{s_j} \in \Omega(clock(s_j) \cup C)$, such that $(s_j,\pi_j) \in \| A_1 \|_{\mathcal{M}(\mathcal{G},\gamma,C)}$ iff $\pi_j \models \omega_{s_j}$. It is obvious that $\models \forall(\omega_{s_j} \supset delay(s_j))$.

For each $\pi \models \omega$, there exists $j \in \{1,n\}$, such that $\pi \models \omega_j$ and $reset(\pi, C_{1,j}, C_{2,j}) + 1 \models \omega_{s_j}$. By proposition 6.2.2, $reset(\pi, C_{1,j}, C_{2,j}) \models \omega_{s_j} + 1$ and $\pi \models (\omega_{s_j} + 1)[C_{2,j} := 0]$.

Let $\omega'_j \equiv \omega_j \wedge \omega \wedge (\omega_{s_j} + 1)[C_{2,j} := 0]$. Since, for each $\pi \models \omega$, $\pi \models \omega_j$, for some $j \in \{1,n\}$, and $\pi \models (\omega_{s_j} + 1)[C_{2,j} := 0]$, then $\omega'_1 \vee \ldots \vee \omega'_n \equiv \omega$. Therefore, by rule $disj$, $s, \omega \vdash < \alpha > A_1$ can be obtained from $\{s, \omega'_j \vdash < \alpha > A_1\}_{j \in \{1,n\}}$.

For each $j \in \{1,n\}$, consider $reset(\omega'_j, C_{1,j}, C_{2,j}) - 1$. By proposition 6.2.2, if $\pi' \models reset(\omega'_j, C_{1,j}, C_{2,j}) - 1$, then $\pi' - 1 \models reset(\omega'_j, C_{1,j}, C_{2,j})$. Since ω'_j is satisfiable, then, by proposition 6.2.2, there exists π, such that $\pi \models \omega'_j$ and $\pi' = reset(\pi, C_{1,j}, C_{2,j})$. If $\pi \models \omega'_j$, then $\pi \models \omega_j \wedge (\omega_{s_j} + 1)[C_{2,j} := 0]$. It is obvious that $\pi' \models \omega_{s_j}$. Therefore, $\models \forall(reset(\omega'_j, C_{1,j}, C_{2,j}) - 1 \supset \omega_{s_j})$, so that $s_j, reset(\omega'_j, C_{1,j}, C_{2,j}) - 1 \models A_1$, for each $j \in \{1,n\}$. Since $A_1 \prec A$, by induction hypothesis $s_j, reset(\omega'_j, C_{1,j}, C_{2,j}) - 1 \vdash A_1$.

Since $\models \forall(\omega'_j \supset \omega_j)$ and $\models \forall(reset(\omega'_j, C_{1,j}, C_{2,j}) - 1 \supset delay(s_j))$, since $\models \forall(reset(\omega'_j, C_{1,j}, C_{2,j}) - 1 \supset \omega_{s_j}))$ and $\models \forall(\omega_{s_j} \supset delay(s_j))$, rule $<>$ is applicable and we obtain $s, \omega'_j \vdash < \alpha > A_1$ from $s_j, reset(\omega'_j, C_{1,j}, C_{2,j}) - 1 \vdash A_1$, for each $j \in \{1,n\}$.

- $A = [\alpha]A_1$. By definition $s, \omega \models A$ iff for each $\pi \models \omega$, if $(s,\pi) \overset{\alpha}{\mapsto} (s',\pi')$, then $(s',\pi') \models A_1$.

  Let $\{s \overset{\alpha,\omega_i}{\mapsto} s_i\}_{i \in \{1,n\}}$ be the transitions, such that $\omega'_i \wedge \omega$ is satisfiable. For each $\pi \models \omega$ and for each $i \in \{1,n\}$, if $\pi \models \omega_i$ and $reset(\pi, C_{1,i}, C_{2,i}) + 1 \models delay(s_i)$, then $(s_i, reset(\pi, C_{1,i}, C_{2,i}) + 1) \models A_1$, where $C_{1,i} = (clock(s) \cap clock(s_i) \cup C$ and $C_{2,i} = clock(s_i) \setminus clock(s)$.

  Let $\omega'_i \equiv reset(\omega_i \wedge \omega, C_{1,i}, C_{2,i}) - 1$, for each $i \in \{1,n\}$. We show that, for each $i \in \{1,n\}$, $s_i, \omega'_i \vdash A_1$.

  Suppose $\models \forall(\omega'_i \supset delay(s_i))$. If $\pi \models \omega'_i$, then $\pi \models delay(s_i)$ and $\pi \models reset(\omega_j \wedge \omega, C_{1,i}, C_{2,i}) - 1$. By proposition 6.2.2, $\pi - 1 \models reset(\omega_i \wedge \omega, C_{1,i}, C_{2,i})$. Since $\omega_i \wedge \omega$ is satisfiable, then, by proposition 6.2.2, there exists π', such that $\pi' \models \omega_i \wedge \omega$ and $reset(\pi', C_{1,i}, C_{2,i}) = \pi - 1$, i.e. $\pi = reset(\pi', C_{1,i}, C_{2,i}) + 1$. Therefore, $(s,\pi') \overset{\alpha}{\mapsto} (s_i,\pi)$, so that $(s_i,\pi) \models A_1$. Hence, $s_i, \omega'_i \models A_1$ for each $i \in \{1,n\}$. Since $A_1 \prec A$, by induction hypothesis, $s_i, \omega'_i \vdash A_1$.

  If $\not\models \forall(\omega'_i \supset delay(s_i))$, then $\omega'_i \equiv \omega'_{i,1} \vee \omega'_{i,2}$, where $\models \forall(\omega'_{i,1} \wedge delay(s_i) \supset false)$ and $\models \forall(\omega'_{i,2} \supset delay(s_i))$. It is obvious that both $s_i, \omega'_{i,1} \vdash A_1$, by rule $empty$ and $s_i, \omega'_{i,2} \vdash A_1$, by repeating the same reasoning as before.

We have $s_i, \omega_i' \vdash A_1$ by applying rule $disj$.

Finally, we obtain $s, \omega \vdash [\alpha]A_1$ from $\{s_i, \omega_i' \vdash A_1\}_{i\in\{1,n\}}$, by rule [] .

- $A = \nu X\{U\}.B$. If $\| (s,\omega) \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \subseteq \| U \|_{\mathcal{M}(\mathcal{G},\gamma,C)}$, then by ν_1, $s, \omega \vdash \nu X\{U\}.B$.

  Otherwise, by lemma 6.3.5, $\| \nu X\{U\}.B \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \in \mathcal{R}_{(\mathcal{G},A)}$, namely there exists a $(\mathcal{G}, A)$-constraint $\omega' \in \Omega(clock(s) \cup C)$, such that $\models \forall(\omega \supset \omega')$ and $s, \omega' \models A$. We have $s, \omega \vdash A$ from $s, \omega' \vdash A$ by rule $thin$.

  By lemma 6.3.6, $(s, \omega') \in \| B[\nu X\{U \cup \{(s,\omega')\}.B/X] \cup U \|_{\mathcal{M}(\mathcal{G},\gamma,C)}$. There are two cases:

  (a) If $\| (s,\omega') \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \cap \| U \|_{\mathcal{M}(\mathcal{G},\gamma,C)} = \emptyset$, then $s, \omega' \models B[\nu X\{U \cup \{(s,\omega')\}.B/X]$. Moreover, since $B[\nu X\{U \cup \{(s,\omega')\}.B/X] \prec A$, by induction hypothesis $s, \omega' \vdash B[\nu X\{U \cup \{(s,\omega')\}.B/X]$. We have $s, \omega' \vdash A$ by applying rule ν_2.

  (b) If $\| (s,\omega') \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \cap \| U \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \neq \emptyset$, then there exists $\{(s,\omega_i) \in U\}_{i\in\{1,n\}}$, such that $\omega' \wedge \omega_i$ is satisfiable, for each $i \in \{1, n\}$. Consider $\omega_1' \equiv \omega' \wedge (\bigvee_{i\in\{1,n\}} \omega_i)$ and $\omega_2' \equiv \omega' \wedge \neg(\bigvee_{i\in\{1,n\}} \omega_i)$. It is obvious that $\omega' \equiv \omega_1' \vee \omega_2'$. Hence, $s, \omega' \vdash A$ can be obtained by rule $disj$ from $s, \omega_1' \vdash A$ and $s, \omega_2' \vdash A$. Moreover, $\models \forall(\omega_1' \supset \bigvee_{i\in\{1,n\}} \omega_i)$. Hence, by rule ν_1, $s, \omega_1' \vdash A$. On the other hand, $\| (s,\omega_2') \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \cap \| U \|_{\mathcal{M}(\mathcal{G},\gamma,C)} = \emptyset$. Therefore, $s, \omega_2' \models B[\nu X\{U \cup \{(s,\omega_2')\}.B/X]$ and, since $B[\nu X\{U \cup \{(s,\omega_2')\}.B/X] \prec A$, by induction hypothesis $s, \omega_2' \vdash B[\nu X\{U \cup \{(s,\omega_2')\}.B/X]$. We have $s, \omega_2' \vdash A$ by applying rule ν_2.

- $A = \mu X\{U\}.B$. By lemma 6.3.7 and by lemma 6.3.5, there exist $(\mathcal{G}, A)$-constraints $\omega_1, \ldots, \omega_k$, such that $\omega \equiv \bigvee_{i\in\{1,k\}} \omega_i$ and $s, \omega_i \models B[\mu X.\{U \cup \{(s,\omega_i)\}B/X]$.

  Moreover, as it can be seen from the semantics of tagged least fixpoints $\| (s,\omega_i) \|_{\mathcal{M}(\mathcal{G},\gamma,C)} \not\subseteq \| U \|_{\mathcal{M}(\mathcal{G},\gamma,C)}$. Therefore, $B[\mu X.\{U \cup \{(s,\omega_i)\}B/X] \prec A$ and by induction hypothesis, $s, \omega_i \vdash B[\mu X.\{U \cup \{(s,\omega_i)\}B/X]$, for each $i \in \{1, k\}$. By applying rules $disj$ and rule μ, we have $s, \omega \vdash \mu X\{U\}.B$.

■

Appendix D

Proofs of chapter 7

D.1 Soundness of the proof system

Proposition D.1.1 *For each context $p[]$ over Act and Loc, for each closed RTL_μ formula A over Act, Loc and C, and for each $\omega \in \omega(C \cup Loc)$):*

1. $p[rec\ x.t], \omega \models A$ *iff* $p[t[rec\ x.t/x]], \omega \models A$;
2. $p[\ell :: \ell_1 :: b_1 + b_2], \omega \models A$ *iff* $p[\ell :: \ell_1 :: b_2 + b_1], \omega \models A$.

Theorem 7.3.1 Assume $p \in \mathcal{CTSP}$ over Act and Loc. For a closed RTL_μ formula A over Act, Loc and C and for $\omega \in \Omega(C \cup loc(p))$, if $p, \omega \vdash A$, then $p, \omega \models A$.

Proof. It is sufficient to show that each rule for $< \alpha > A$ and $[\alpha]A$ is backward sound. The soundness of the other rules has been proved in theorem 6.3.2. We show only the cases, that are significantly different from the untimed ones (theorem 4.4.1).

- Rule $<> nil$.

$$\frac{\ell :: \ell_1 :: nil, \omega_1 \vdash < a : \bar{\epsilon} > A}{\ell :: \ell_1 :: nil, \omega_1 - 1 \vdash A} \quad a \in \#(*)$$

Assume $\ell :: \ell_1 :: nil, \omega_1 - 1 \models A$. Since $a \in \#(*)$, then $\ell :: \ell_1 :: nil \overset{a:\bar{\epsilon},true}{\longmapsto} \ell :: \ell_1 :: nil$. Since $delay(\ell :: \ell_1 :: nil) \equiv true$, then, rule $<>$ is applicable and, by theorem 6.3.2, $\ell :: \ell_1 :: nil, \omega_1 \models < a : \bar{\epsilon} > A$.

- Rule $<> \epsilon$.

$$\frac{\ell :: \ell_1 :: a.t, \omega_1 \vdash < a : \epsilon > A}{\ell :: t, \omega_2 \vdash A} \quad \begin{array}{l} \omega_2 \equiv reset(\omega_1, C_1, C_2) - 1 \\ C_1 = \{\ell\} \cup C \\ C_2 = loc(t) \\ \models \forall(\omega_2 \supset delay(\ell :: t)) \end{array}$$

Assume $\ell :: t, \omega_2 \models A$. By definition $\ell :: \ell_1 :: a.t \overset{a:\epsilon,true}{\longmapsto} \ell :: t$. Since $clock(\ell :: \ell_1 :: a.t) \cap clock(\ell :: t) = \{\ell\}$ and $clock(\ell :: t) \setminus clock(\ell :: \ell_1 :: a.t) = loc(t)$, then rule $<>$ is applicable and by theorem 6.3.2, $\ell :: \ell_1 :: a.t, \omega_1 \models < a : \epsilon > A$.

- Rule $<> \bar{\epsilon}$.

$$\frac{\ell :: \ell_1 :: a.t, \omega_1 \vdash < a_1 : \bar{\epsilon} > A}{\ell :: \ell_1 :: a.t, \omega_1 - 1 \vdash A} \quad a_1 \in \#(a)$$

Assume $\ell :: \ell_1 :: a.t, \omega_1 - 1 \models A$. By definition $\ell :: \ell_1 :: a.t \overset{a_1:\bar{\epsilon},true}{\longmapsto} \ell :: \ell_1 :: a.t$. Since $delay(\ell :: \ell_1 :: a.t) \equiv true$, then rule $<>$ is applicable and by theorem 6.3.2, $\ell :: \ell_1 :: a.t, \omega_1 \models < a_1 : \bar{\epsilon} > A$.

- Rule $\tau <> \epsilon$.

$$\frac{\ell :: \ell_1 :: [m, M]a.t, \omega_1 \vdash < a : \epsilon > A}{\ell :: t, \omega_2 \vdash A} \quad \begin{array}{l} \models \forall(\omega_1 \supset m \leq \ell_1 \leq M) \\ \omega_2 \equiv reset(\omega_1, C_1, C_2) - 1 \\ C_1 = \{\ell\} \cup C \\ C_2 = loc(t) \\ \models \forall(\omega_2 \supset delay(\ell :: t)) \end{array}$$

Assume $\ell :: t, \omega_2 \models A$. By definition $\ell :: \ell_1 :: [m, M]a.t \overset{a:\epsilon,m\leq\ell_1\leq M}{\longmapsto} \ell :: t$. Since $clock(\ell :: \ell_1 :: [m, M]a.t) \cap clock(\ell :: t) = \{\ell\}$ and $clock(\ell :: t) \setminus clock(\ell :: \ell_1 :: [m, M]a.t) = loc(t)$, then rule $<>$ is applicable and by theorem 6.3.2, $\ell :: \ell_1 :: [m, M]a.t, \omega_1 \models < a : \epsilon > A$.

- Rule $\tau <> \bar{\epsilon}$.

$$\frac{\ell :: \ell_1 :: [m, M]a.t, \omega_1 \vdash < a_1 : \bar{\epsilon} > A}{\ell :: \ell_1 :: [m, M]a.t, \omega_1 - 1 \vdash A} \quad \begin{array}{l} a_1 \in \#(*) \\ \models \forall(\omega_1 - 1 \supset \ell_1 \leq M) \end{array}$$

Assume $\ell :: \ell_1 :: [m, M]a.t, \omega_1 - 1 \models A$. By definition $\ell :: \ell_1 :: [m, M]a.t \overset{a_1:\bar{\epsilon},true}{\longmapsto} \ell :: \ell_1 :: [m, M]a.t$. Since $delay(\ell :: \ell_1 :: [m, M]a.t) \equiv \ell_1 \leq M$, then rule $<>$ is applicable and by theorem 6.3.2, $\ell :: \ell_1 :: [m, M]a.t, \omega_1 \models < a_1 : \bar{\epsilon} > A$.

- Rule $+ <> \epsilon$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\ell :: \ell_1 :: b_1, \omega_1 \vdash < \alpha > A} \quad type(\alpha) = \epsilon$$

Assume $\ell :: \ell_1 :: b_1, \omega_1 \models < \alpha > A$. By definition, for each $\pi \models \omega_1$, there exists $\ell :: \ell_1 :: b_1 \stackrel{\alpha,\omega}{\longmapsto} p'$, such that $\pi \models \omega$ and $(p', \pi' + 1) \models A$, $\pi' + 1 \models delay(p')$, for $\pi' = reset(\pi, C_1, C_2)$, where $C_1 = (loc(\ell :: \ell_1 :: b_1) \cap loc(p')) \cup C$ and $C_2 = loc(p') \setminus loc(\ell :: \ell_1 :: b_1)$. By rule $choice_1$, $\ell :: \ell_1 :: b_1 + b_2 \stackrel{\alpha,\omega}{\longmapsto} p'$. Since the process is well-formed, $loc(\ell :: \ell_1 :: b_1 + b_2) = loc(\ell :: \ell_1 :: b_1)$, so that $reset(\pi, (loc(\ell :: \ell_1 :: b_1 + b_2) \cap loc(p')) \cup C, loc(p') \setminus loc(\ell :: \ell_1 :: b_1 + b_2)) = reset(\pi, C_1, C_2)$. Therefore, for each $\pi \models \omega_1$, $(\ell :: \ell_1 :: b_1 + b_2, \pi) \stackrel{\alpha}{\mapsto} (p', \pi' + 1)$ and $(p', \pi' + 1) \models A$. By definition, $\ell :: \ell_1 :: b_1 + b_2, \omega_1 \models < \alpha > A$.

- Rule $+ <> \bar{\epsilon}$.

$$\frac{\ell :: \ell_1 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\begin{array}{l}\{\ell :: \ell_1 :: b_i, \omega_1 \vdash < \alpha > true\}_{i \in \{1,2\}} \\ \ell :: \ell_1 :: b_1 + b_2, \omega_1 - 1 \vdash A\end{array}} \quad type(\alpha) = \bar{\epsilon}$$

Assume

$$\{\ell :: \ell_1 :: b_i, \omega_1 \models < \alpha > true\}_{i \in \{1,2\}} \quad \ell :: \ell_1 :: b_1 + b_2, \omega_1 - 1 \models A.$$

For each $\pi \models \omega_1$ and for each $i \in \{1, 2\}$, there exists $\ell :: \ell_1 :: b_i \stackrel{\alpha,true}{\longmapsto} \ell :: \ell_1 :: b_i$, such that $\pi + 1 \models delay(\ell : \ell_1 :: b_i)$.

By rule $choice_2$, $\ell :: \ell_1 :: b_1 + b_2 \stackrel{\alpha,true}{\longmapsto} \ell :: \ell_1 :: b_1 + b_2$ and $\pi + 1 \models delay(\ell :: \ell_1 :: b_1 + b_2)$. Hence, for each $\pi \models \omega_1$, $(\ell :: \ell_1 :: b_1 + b_2, \pi) \stackrel{\alpha}{\mapsto} (\ell :: \ell_1 :: b_1 + b_2, \pi + 1)$. By proposition 6.2.2, $\pi \models \omega_1$ implies $\pi + 1 \models \omega_1 - 1$. Since, $\ell :: \ell_1 :: b_1 + b_2, \omega_1 - 1 \models A$, then $\ell :: \ell_1 :: b_1 + b_2, \omega_1 \models < \alpha > A$.

- Rule $\bigtriangledown <> \epsilon_1$. Let $\omega_t \equiv pro_{C \cup loc(\ell::t)}(\omega_1)$.

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: t, \omega_t \vdash < \alpha > A} \quad type(\alpha) = \epsilon$$

Assume $\ell :: t, \omega_t \models < \alpha > A$. By definition, for each $\pi \models \omega_t$, there exists $\ell :: t \stackrel{\alpha,\omega}{\longmapsto} p'$, such that $\pi \models \omega$ and $(p', \pi' + 1) \models A$, $\pi' + 1 \models delay(p')$, where $\pi' = reset(\pi, (loc(\ell :: t) \cap loc(p')) \cup C, loc(p') \setminus loc(\ell :: t))$.

By rule $\bigtriangledown_1$, $\ell :: t \bigtriangledown p \stackrel{\alpha,\omega}{\longmapsto} p'$. Since the process is well-formed, $Loc(p) \cap Loc(\ell :: t) = main(t)$ and $main(t) \notin Loc(p')$. Therefore, $reset(\pi, (loc(\ell :: t \bigtriangledown p) \cap loc(p')) \cup C, loc(p') \setminus loc(\ell :: t \bigtriangledown p)) = \pi'$. For each $\pi \models \omega_1$, by proposition 6.2.2, $pro_{C \cup loc(\ell::t)}(\pi) \models \omega_t$ so that $\pi \models \omega$ and $(\ell :: t \bigtriangledown p, \pi) \stackrel{\alpha}{\mapsto} (p', \pi' + 1)$, such that $(p', \pi' + 1) \models A$. By definition $\ell :: t \bigtriangledown p, \omega_1 \models < \alpha > A$.

- Rule $\bigtriangledown <> \epsilon_2$. Let $\omega_2 = pro_{loc(\ell::t)}(\omega_1) - 1$ and $\omega_p \equiv pro_{C \cup (loc(p) \setminus main(t))}(\omega_1)$.

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{p, \omega_p \vdash < \alpha > (A)_{(\ell::t, \omega_2)\bigtriangledown}} \quad \begin{array}{l} type(\alpha) = \epsilon \\ \models \forall(\omega_2 \supset delay(\ell :: t)) \end{array}$$

Assume $p, \omega_p \models < \alpha > (A)_{(\ell::t,\omega_2)\bigtriangledown}$. By definition, for each $\pi_1 \models \omega_p$, there exists $p \overset{\alpha,\omega}{\longmapsto} p'$, such that

1. $\pi_1 \models \omega$;
2. $(p', \pi_1' + 1) \models (A)_{(\ell::t,\omega_2)\bigtriangledown}$, where $\pi_1' = reset(\pi_1, C_1, C_2)$ for $C_1 = (loc(p) \cap loc(p')) \cup C)$ and $C_2 = loc(p') \setminus loc(p)$. Therefore, $(\ell :: t \bigtriangledown p', \pi_1' \cup \pi_2') \models A$, for each $\pi_2' \models \omega_2$;
3. $\pi_1' + 1 \models delay(p')$.

By rule $\bigtriangledown_2$, $\ell :: t \bigtriangledown p \overset{\alpha,\omega}{\longmapsto} \ell :: t \bigtriangledown p'$. By proposition 6.2.2, for each $\pi \models \omega_1$, $pro_{loc(p)\cup C}(\pi) \models \omega_p$ and $pro_{loc(\ell::t)}(\pi) \models pro_{loc(\ell::t)}(\omega_1)$ so that $\pi \models \omega$.

For each $\pi \models \omega_1$, let $\pi' = reset(\pi, C_1', C_2')$, for $C_1' = (loc(\ell :: t \bigtriangledown p) \cap loc(\ell :: t \bigtriangledown p') \cup C)$ and $C_2' = loc(\ell :: t \bigtriangledown p') \setminus loc(\ell :: t \bigtriangledown p)$. Since the process is well-formed $Loc(p) \cap Loc(\ell :: t) = main(t)$ and $main(t) \in loc(p) \cap loc(p')$. By proposition 7.2.1, $\pi' = \pi_1' \cup \pi_2'$, where $\pi_2' = pro_{loc(\ell::t)}(\pi)$ and $\pi_1' = reset(\pi_1, C_1, C_2)$ for $\pi_1 = pro_{C\cup loc(p)}(\pi)$. Moreover, $\pi' + 1 = (\pi_1' + 1) \cup (\pi_2' + 1)$.

By proposition 6.2.2, $\pi_2' + 1 \models pro_{loc(\ell::t)}(\omega_1) - 1 \equiv \omega_2$. Since $\pi_1' + 1 \models delay(p')$ and $\pi_2' + 1 \models \omega_2$ and $\models \forall(\omega_2 \supset delay(\ell :: t))$, then $\pi' + 1 \models delay(\ell :: t \bigtriangledown p')$.

Therefore, there exists $(\ell :: t \bigtriangledown p, \pi) \overset{\alpha}{\mapsto} (\ell :: t \bigtriangledown p', \pi' + 1)$, where $(\ell :: t \bigtriangledown p', \pi' + 1) \models A$. Hence, $\ell :: t \bigtriangledown p, \omega_1 \models < \alpha > A$.

- Rule $\bigtriangledown <> \bar{\epsilon}$. Let $\omega_t \equiv pro_{loc(\ell::t)\cup C}(\omega_1)$ and $\omega_p \equiv pro_{(loc(p)\setminus main(t))\cup C}(\omega_1)$.

$$\frac{\ell :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\begin{array}{l}\ell :: t, \omega_t \vdash < \alpha > true \\ p, \omega_p \vdash < \alpha > true \\ \ell :: t \bigtriangledown p, \omega_1 - 1 \vdash A\end{array}} \bigtriangledown <> \bar{\epsilon} \qquad type(\alpha) = \bar{\epsilon}$$

Assume

$\ell :: t, \omega_t \models < \alpha > true$
$p, \omega_p \models < \alpha > true$
$\ell :: t \bigtriangledown p, \omega_1 - 1 \models A$.

For each $\pi_1 \models \omega_t$, there exists $\ell :: t \overset{\alpha,true}{\longmapsto} \ell :: t$ and for each $\pi_2 \models \omega_p$, there exists $p \overset{\alpha,true}{\longmapsto} p$. Moreover, $\pi_1 + 1 \models delay(\ell :: t)$ and $\pi_2 + 1 \models delay(p)$.

By rule $\bigtriangledown_3$, $\ell :: t \bigtriangledown p \overset{\alpha,true}{\longmapsto} \ell :: t \bigtriangledown p$. By proposition 6.2.2, for each $\pi \models \omega_1$, $\pi_1 = pro_{loc(p)\cup C}(\pi) \models \omega_p$ and $\pi_2 = pro_{loc(\ell::t)\cup C}(\pi) \models \omega_t$. Since, $\pi_1 + 1 \models delay(\ell :: t)$ and $\pi_2 + 1 \models delay(p)$ and $\pi + 1 = (\pi_1 + 1) \cup (\pi_2 + 1)$ by proposition 7.2.1, then $\pi + 1 \models delay(\ell :: t \bigtriangledown p)$.

Therefore, for each $\pi \models \omega_1$, $(\ell :: \ell_1 :: t \bigtriangledown p, \pi) \overset{\alpha}{\mapsto} (\ell :: t \bigtriangledown p, \pi + 1)$. By proposition 6.2.2, $\pi \models \omega_1$ implies $\pi + 1 \models \omega_1 - 1$. Since, $\ell :: t \bigtriangledown p, \omega_1 - 1 \models A$, then $\ell :: t \bigtriangledown p, \omega_1 \models < \alpha > A$.

- Rules for recursion are analogous to the ones for the untimed case by exploiting proposition D.1.1.

- Rule $\times <> nil$. Let $p_2 = \ell_1 :: \ell_2 :: nil$ and $\omega_2 \equiv pro_{\{\ell_1,\ell_2\}}(\omega_1) - 1$.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, pro_{loc(p_1)\cup C}(\omega_1) \vdash < \beta > (A)_{\ell\times(p_2,\omega_2)}} \quad \begin{array}{l}\alpha \in \beta \otimes (b : \bar{\epsilon}) \\ b \in \#(*)\end{array}$$

Assume $p_1, pro_{loc(p_1)\cup C}(\omega_1) \models < \beta > (A)_{\ell\times(p_2,\omega_2)}$. By definition, for each $\pi_1 \models pro_{loc(p_1)\cup C}(\omega_1)$ there exists $p_1 \stackrel{\beta,\omega}{\longmapsto} p_1'$, such that

1. $\pi_1 \models \omega$;
2. $(p_1', \pi_1' + 1) \models (A)_{\ell\times(p_2,\omega_2)}$, where $\pi_1' = reset(\pi_1, C_1, C_2)$, where $C_1 = C \cup (loc(p_1) \cap loc(p_1'))$ and $C_2 = loc(p_1') \setminus loc(p_1)$;
3. $\pi_1' + 1 \models delay(p_1')$.

By definition, $\ell_1 :: \ell_2 :: nil \stackrel{b:\bar{\epsilon},true}{\longmapsto} \ell_1 :: \ell_2 :: nil$. Since $\alpha \in \beta \otimes (b : \bar{\epsilon})$, then by rule *par*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: nil \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times \ell_1 :: \ell_2 :: nil$.

For each $\pi \models \omega_1$, $pro_{loc(p_1)\cup C}(\pi) \models pro_{loc(p_1)\cup C}(\omega_1)$ by proposition 6.2.2 so that $\pi \models \omega$. Since the process is well-formed $Loc(p_1) \cap Loc(p_2) = \emptyset$ and $Loc(p_1') \cap Loc(p_2) = \emptyset$. By proposition 7.2.1, $\pi' = reset(\pi, C_1', C_2') = reset(\pi, C_1, C_2) \cup pro_{\{\ell_1,\ell_2\}}(\pi) = \pi_1' \cup \pi_2'$, where $C_1' = C \cup ((loc(p_1) \cup \{\ell_1, \ell_2\}) \cap (loc(p_1') \cup \{\ell_1, \ell_2\}))$ and $C_2' = (loc(p_1') \cup \{\ell_1, \ell_2\} \setminus (loc(p_1) \cup \{\ell_1, \ell_2\})$ and $\pi_1' = reset(\pi_1, C_1, C_2)$ for $\pi_1 = pro_{loc(p_1)\cup C}(\pi)$.

Moreover, by proposition 6.2.2, $\pi_2' \models pro_{\{\ell_1,\ell_2\}}(\omega_1)$ and $\pi_2' + 1 \models pro_{\{\ell_1,\ell_2\}}(\omega_1) - 1 \equiv \omega_2$. Since $(p_1', \pi_1' + 1) \models (A)_{\ell\times(p_2,\omega_2)}$ and $\pi' + 1 = (\pi_1' + 1) \cup (\pi_2' + 1)$, then $(\ell :: p_1' \times \ell_1 :: \ell_2 :: nil, \pi' + 1) \models A$. Moreover, since $delay(\ell_1 :: \ell_2 :: nil) \equiv true$ and $\pi_1' + 1 \models delay(p_1')$, $\pi' + 1 \models delay(\ell :: p_1' \times p_2)$. Therefore, for each $\pi \models \omega_1$, $(\ell :: p_1 \times p_2, \pi) \stackrel{\alpha}{\mapsto} (\ell :: p_1' \times p_2, \pi' + 1)$, such that $(\ell :: p_1' \times p_2, \pi' + 1) \models A$. By definition, $\ell :: p_1 \times \ell_1 :: \ell_2 :: nil, \omega_1 \models < \alpha > A$.

- Rule $\times <>_1$. Let $p_2 = \ell_1 :: \ell_2 :: a.t$ and $\omega_2 = reset(\omega_1, \{\ell_1\}, loc(t)) - 1$.

$$\frac{\ell :: p_1 \times p_2, \omega_1 \vdash < \alpha > A}{p_1, pro_{C\cup loc(p_1)}(\omega_1) \vdash < \beta > (A)_{\ell\times(\ell_1::t,\omega_2)}} \quad \begin{array}{l}\alpha \in \beta \otimes (a : \epsilon) \\ \models \forall(\omega_2 \supset delay(\ell_1 :: t))\end{array}$$

Assume $p_1, pro_{loc(p_1)\cup C}(\omega_1) \models < \beta > (A)_{\ell\times(\ell_1::t,\omega_2)}$. By definition, for each $\pi_1 \models pro_{loc(p_1)\cup C}(\omega_1)$ there exists $p_1 \stackrel{\beta,\omega}{\longmapsto} p_1'$, such that

1. $\pi_1 \models \omega$;

2. $(p_1', \pi_1'+1) \models (A)_{\ell\times(\ell_1::t,\omega_2)}$, where $\pi_1' = reset(\pi_1, C_1, C_2)$, for $C_1 = C \cup (loc(p_1) \cap loc(p_1'))$ and $C_2 = loc(p_1') \setminus loc(p_1)$;

3. $\pi_1' + 1 \models delay(p_1')$.

By definition, $\ell :: \ell_2 :: a.t \stackrel{a:\epsilon,true}{\longmapsto} \ell_1 :: t$. Since $\alpha \in \beta \otimes (a:\epsilon)$, then by rule *par*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times \ell_1 :: t$.

For each $\pi \models \omega_1$, $pro_{loc(p_1)\cup C}(\pi) \models pro_{loc(p_1)\cup C}(\omega_1)$ so that $\pi \models \omega$. Since the process is well-formed by proposition 7.2.1, $\pi' = reset(\pi, C_1', C_2') = reset(\pi, C_1, C_2) \cup reset(\pi, C_1'', C_2'') = \pi_1' \cup \pi_2'$, where $C_1' = C \cup ((loc(\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t) \cap loc(\ell :: p_1' \times \ell_1 :: t))$ and $C_2' = loc(\ell :: p_1' \times \ell_1 :: t)' \setminus loc(\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t)$ and $C_1'' = loc(\ell_1 :: \ell_2 :: a.t) \cap loc(\ell_1 :: t)) = \{\ell_1\}$, $C_2'' = loc(\ell_1 :: t) \setminus loc(\ell_1 :: \ell_2 :: a.t) = loc(t)$.

Moreover, by proposition 6.2.2, $\pi_2' \models reset(\omega_1, C_1'', C_2'')$ and $\pi_2' + 1 \models \omega_2$. Since $(p_1', \pi_1'+1) \models (A)_{\ell\times(\ell_1::t,\omega_2)}$ and $\pi' = (\pi_1'+1) \cup (\pi_2'+1)$, then $(\ell :: p_1' \times \ell_1 :: t, \pi'+1) \models A$. Moreover, since $\pi_2' + 1 \models \omega_2$ and $\models \forall(\omega_2 \supset delay(\ell_1 :: t))$ and $\pi_1' + 1 \models delay(p_1')$, then $\pi' + 1 \models delay(\ell :: p_1' \times \ell_1 :: t)$. By definition, $\ell :: p_1 \times p_2, \omega_1 \models < \alpha > A$.

- The case of $\times <>_2$ is analogous to the one of $\times <> nil$.
- The cases of $\tau <>_i$ are analogous to the ones of $<>_i$. However, in this case, $delay(\ell_1 :: \ell_2 :: [m, M]a.t) \equiv \ell_2 \leq M$ and the action a can be performed iff $m \leq \ell_2 \leq M$. Hence, these additional conditions are considered.
- Rule $\times <> +\epsilon$.

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \vdash < \alpha > A} \quad type(proj_2(\alpha)) = \epsilon$$

Assume $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega_1 \models < \alpha > A$. By definition, for each $\pi \models \omega_1$, there exists $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times p_2'$, such that

1. $\pi \models \omega$;

2. $\ell :: p_1' \times p_2', \pi'+1 \models A$, where $\pi' = reset(\pi, C_1, C_2)$ for $C_1 = C \cup (loc(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i) \cap loc(\ell :: p_1' \times p_2'))$ and $C_2 = loc(\ell :: p_1' \times p_2') \setminus loc(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i)$;

3. $\pi' + 1 \models delay(\ell :: p_1' \times p_2')$.

By rule *par*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times p_2'$ iff $p_1 \stackrel{\alpha_1,\omega_1}{\longmapsto} p_1'$ and $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha_2,\omega_2}{\longmapsto} p_2'$ with $\alpha \in \alpha_1 \otimes \alpha_2$ and $\omega \equiv \omega_1 \wedge \omega_2$. Since $type(proj_2(\alpha)) = \epsilon$, then $type(\alpha_2) = \epsilon$ and by rule $choice_1$, $\ell_1 :: \ell_2 :: b_1 + b_2 \stackrel{\alpha_2,\omega_2}{\longmapsto} p_2'$. By rule *par*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \stackrel{\alpha,\omega}{\longmapsto} \ell :: p_1' \times p_2'$.

Since the process is well-formed, $loc(\ell_1 :: \ell_2 :: b_1 + b_2) = loc(\ell_1 :: \ell_2 :: b_i)$ so that $reset(\pi, C_1, C_2) = reset(\pi, C_1', C_2')$, where $C_1' = C \cup (loc(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2) \cap loc(\ell :: p_1' \times p_2'))$ and $C_2' = loc(\ell :: p_1' \times p_2') \setminus loc(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2)$. Therefore, for each $\pi \models \omega_1$, $(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \pi) \stackrel{\alpha}{\mapsto} (\ell :: p_1' \times p_2', \pi' + 1)$ such that $(\ell :: p_1' \times p_2', \pi' + 1) \models A$. By definition, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega_1 \models < \alpha > A$.

- Rule $\times <> \bigtriangledown\epsilon_2$. Let $\omega_p \equiv pro_{C\cup(loc(\ell::p_1\times p)\setminus main(t))}(\omega_1)$ and $\omega_2 \equiv pro_{loc(\ell_1::t)}(\omega_1) -1$.

$$\frac{\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \vdash < \alpha > A}{\ell :: p_1 \times p, \omega_p \vdash < \alpha > (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}} \quad \begin{array}{l} type(proj_2(\alpha)) = \epsilon \\ \models \forall(\omega_2 \supset delay(\ell_1 :: t)) \end{array}$$

Assume $\ell :: p_1 \times p, \omega_p \models < \alpha > (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}$. By definition, for each $\pi_1 \models \omega_p$ there exists $\ell :: p_1 \times p \stackrel{\alpha,\omega}{\longmapsto} \ell :: p'_1 \times p'$, such that

1. $\pi_1 \models \omega$;

2. $(\ell :: p'_1 \times p', \pi'_1 + 1) \models (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}$, where $\pi'_1 = reset(\pi_1, C_1, C_2)$ and $C_1 = C \cup (loc(\ell :: p_1 \times p) \cap loc(\ell :: p'_1 \times p'))$ and $C_2 = loc(\ell :: p'_1 \times p') \setminus (loc(\ell :: p_1 \times p)$;

3. $\pi'_1 + 1 \models delay(\ell :: p'_1 \times p')$.

By rule par, $\ell :: p_1 \times p \stackrel{\alpha,\omega}{\longmapsto} \ell :: p'_1 \times p'$ iff $p_1 \stackrel{\alpha_1,\omega_1}{\longmapsto} p'_1$ and $p \stackrel{\alpha_2,\omega_2}{\longmapsto} p'$ with $\alpha \in \alpha_1 \otimes \alpha_2$ and $\omega \equiv \omega_1 \wedge \omega_2$. Since $type(proj_2(\alpha)) = \epsilon$, then $type(\alpha_2) = \epsilon$ and by rule $\bigtriangledown_2$, $\ell_1 :: t \bigtriangledown p \stackrel{\alpha_2,\omega_2}{\longmapsto} \ell_1 :: t \bigtriangledown p'$. Moreover, by rule par, $\ell :: p_1 \times \ell_1 :: t \bigtriangledown p \stackrel{\alpha,\omega}{\longmapsto} \ell :: p'_1 \times \ell_1 :: t \bigtriangledown p'$.

For each $\pi \models \omega_1$, we have

1. Since $pro_{C\cup loc(\ell::p_1\times p)}(\pi) \models \omega_p$, by proposition 6.2.2, $\pi_1 = pro_{C\cup loc(\ell::p_1\times p)}(\pi) \models \omega$ so that $\pi \models \omega$;

2. Since the process is well-formed, by proposition 7.2.1, $\pi' = reset(\pi, C'_1, C'_2) = reset(\pi, C_1, C_2) \cup \pi'_2 \equiv \pi'_1 \cup \pi'_2$, where $\pi'_2 = pro_{loc(\ell_1::t)}(\pi)$ and $C'_1 = C \cup (loc(\ell :: p_1 \times (\ell_1 :: t \bigtriangledown p)) \cap loc(\ell :: p'_1 \times (\ell_1 :: t \bigtriangledown p'))$ and $C'_2 = loc(\ell :: p'_1 \times (\ell_1 :: t \bigtriangledown p')) \setminus loc(\ell :: p_1 \times (\ell_1 :: t \bigtriangledown p))$ and $\pi'_1 = reset(pro_{loc(\ell::p_1\times p)}(\pi), C_1, C_2)$.

3. $\pi' + 1 \models delay(\ell :: p'_1 \times (\ell_1 :: t \bigtriangledown p'))$, since $\pi'_1 + 1 \models delay(\ell :: p'_1 \times p')$ and $\pi'_2 + 1 \models pro_{loc(\ell_1::t)}(\omega_1) - 1 \equiv \omega_2$, by proposition 6.2.2, and $\models \forall(\omega_2 \supset delay(\ell_1 :: t))$.

Therefore, $(\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \pi) \stackrel{\alpha}{\mapsto} (\ell :: p'_1 \times \ell_1 :: t \bigtriangledown p', \pi' + 1)$. Since $(\ell :: p'_1 \times p', \pi'_1 + 1) \models (A)_{\times(\ell_1::t,\omega_2)\bigtriangledown}$ and $\pi' + 1 = (\pi'_1 + 1) \cup (\pi'_2 + 1)$, then $(\ell :: p'_1 \times \ell_1 :: t \bigtriangledown p', \pi' + 1) \models A$. Therefore, $\ell :: p_1 \times \ell_1 :: t \bigtriangledown p, \omega_1 \models < \alpha > A$.

- The cases of recursion and parallel composition are analogous to the untimed ones.

■

D.2 Completeness of the proof system

Lemma 7.3.2 Assume $p \in \mathcal{CTSP}$ over Act and Loc. For a closed RTL_μ formula A over Act, Loc and C and for $\omega \in \Omega(C \cup loc(p))$, if $p, \omega \models < \alpha > A$ (resp. $p, \omega \models [\alpha]A$), with $\alpha \in \mathcal{E}(Act)$, then

1. there exist sets $\{p_i \in \mathcal{S}_p\}_{i\in\{1,n\}}$ and $\{p'_j \in \mathcal{S}_p\}_{j\in\{1,k\}}$, such that $\{p_i, \omega_i \models A\}_{i\in\{1,n\}}$ and $\{p'_j, \omega'_j \models true\}_{j\in\{1,k\}}$;
2. $p, \omega \vdash < \alpha > A$ (resp. $p, \omega \vdash [\alpha]A$)) can be derived from $\{p_i, \omega_i \vdash A\}_{i\in\{1,n\}}$ and $\{p'_j, \omega'_j \vdash true\}_{j\in\{1,k\}}$.

Moreover, for each $i \in \{1, n\}$ and $\pi \models \omega$, there exists $(p, \pi) \overset{\alpha_i}{\mapsto} (p_i, \pi')$, where $\pi' \models \omega_i$, such that it is consistent with α.

Proof. The proof is by induction on $m(p)$. We show only the relevant cases for $< \alpha > A$, which differ significantly from the untimed ones (lemma 4.4.4). We exploit the following property. As shown in theorem 6.3.9, if $p, \omega \models < \alpha > A$, then there exists $\omega \equiv \bigvee_{i\in\{1,n\}} \omega_i$, such that $p, \omega_i \models < \alpha > A$. Moreover, for each $i \in \{1, n\}$, there exists $p \overset{\alpha,\omega'_i}{\longmapsto} p'_i$ such that rule $<>$ is applicable, i.e.

1. $\models \forall(\omega_i \supset \omega'_i)$;
2. $p'_i, \omega_{2,i} \models A$, where $\omega_{2,i} \equiv reset(\omega_i, C_{1,i}, C_{2,i}) - 1$, where $C_{1,i} = C \cup (clock(p'_i) \cap clock(p))$, $C_{2,i} = clock(p'_i) \setminus clock(p)$;
3. $\models \forall(\omega_{2,i} \supset delay(p'_i))$.

We can assume without loss of generality $n = 1$ and $\omega'_1 \equiv \omega'$, $\omega_{2,1} \equiv \omega_2$, $C_{j,1} = C_j$, for each $j \in \{1, 2\}$. Consider $p, \omega \models < \alpha > A$.

- Assume $p = \ell_1 :: \ell_2 :: nil$.

By definition $\alpha = b : \bar{\epsilon}$, for $b \in \#(*)$ and $p' = p$, $\omega' \equiv true$. Therefore, $\omega_2 \equiv \omega - 1 \equiv reset(\omega, C_1, C_2) - 1$. We have $p, \omega - 1 \models A$ and $p, \omega \vdash < \alpha > A$ is derivable from $p, \omega - 1 \vdash A$, by rule $<> nil$.

- Assume $p = \ell_1 :: \ell_2 :: a.t$.

There are two cases, either $\alpha = a : \epsilon$ and $p' = \ell_1 :: t$ or $\alpha = b : \bar{\epsilon}$, for $b \in \#(a)$ and $p' = p$. In both cases $\omega' \equiv true$.

In the first case, $\omega_2 \equiv reset(\omega, C_1, C_2) - 1 \equiv reset(\omega, \{\ell_1\} \cup C, loc(t)) - 1$, since the process is well-formed. We have $\ell_1 :: t, reset(\omega, \{\ell_1\} \cup C, loc(t)) - 1 \models A$ and $p, \omega \vdash < \alpha > A$ is derivable from $\ell_1 :: t, reset(\omega, \{\ell_1\} \cup C, loc(t)) - 1 \vdash A$, by rule $<> \epsilon$.

In the second case, $\omega_2 \equiv reset(\omega, C_1, C_2) - 1 \equiv \omega - 1$. We have $p, \omega - 1 \models A$ and $p, \omega \vdash < \alpha > A$ is derivable from $p, \omega - 1 \vdash A$, by rule $<> \bar{\epsilon}$.

- Assume $p = \ell_1 :: \ell_2 :: [m, M]a.t$.

There are two cases, either $\alpha = a : \epsilon$ and $p' = \ell_1 :: t$ and $\omega' \equiv m \leq \ell_2 \leq M$ or $\alpha = b : \bar{\epsilon}$, for $b \in \#(a)$ and $p' = p$, $\omega' \equiv true$.

In the first case, $\omega_2 \equiv reset(\omega, C_1, C_2) - 1 \equiv reset(\omega, \{\ell_1\} \cup C, loc(t)) - 1$, since the process is well-formed. Hence, $\ell_1 :: t, reset(\omega, \{\ell_1\} \cup C, loc(t)) - 1 \models A$. Since $\models \forall(reset(\omega, \{\ell_1\} \cup C, loc(t)) - 1 \supset m \leq \ell_2 \leq M)$, $p, \omega \vdash < \alpha > A$ is derivable from $\ell_1 :: t, reset(\omega, \{\ell_1\} \cup C, loc(t)) - 1 \vdash A$, by rule $\tau <> \epsilon$.

In the second case, $\omega_2 \equiv reset(\omega, C_1, C_2) - 1 \equiv \omega - 1$. Therefore, $p, \omega - 1 \models A$. Since $\models \forall(\omega - 1 \supset delay(p))$ and $delay(p) \equiv \ell_2 \leq M$, $p, \omega \vdash < \alpha > A$ is derivable from $p, \omega - 1 \vdash A$, by rule $\tau <> \bar{\epsilon}$.

- $p = \ell_1 :: \ell_2 :: b_1 + b_2$.

  There are two cases: either $type(\alpha) = \epsilon$ or $type(\alpha) = \bar{\epsilon}$.

  - If $type(\alpha) = \epsilon$, then $p \overset{\alpha,\omega'}{\longmapsto} p'$ iff, by rule $choice_1$, $\ell_1 :: \ell_2 :: b_i \overset{\alpha,\omega'}{\longmapsto} p'$, for some $i \in \{1, 2\}$.

    Since the process is well-formed, $reset(\omega, C_1', C_2') \equiv reset(\omega, C_1, C_2)$, for $C_1' = C \cup (loc(\ell_1 :: \ell_2 :: b_i) \cap loc(p'))$ and $C_2' = loc(p') \setminus loc(\ell_1 :: \ell_2 :: b_i)$. Since $\omega_2 \equiv reset(\omega, C_1, C_2) - 1$, then $p', \omega_2 \models A$. Moreover, the side condition of rule $<>$ is true with respect to $\ell_1 :: \ell_2 :: b_i \overset{\alpha,\omega'}{\longmapsto} p'$, by the soundness of $<>$ (theorem 6.3.2), $\ell_1 :: \ell_2 :: b_i, \omega \models < \alpha > A$.

    Since $m(\ell_1 :: \ell_2 :: b_i) \leq m(\ell_1 :: \ell_2 :: b_1 + b_2)$, by induction hypothesis there exist $\{p_j \in \mathcal{S}_{\ell_1::\ell_2::b_i}\}_{j\in\{1,n\}}$ and $\{p_j' \in \mathcal{S}_{\ell_1::\ell_2::b_i}\}_{j\in\{1,k\}}$ such that

    1. $\{p_j, \omega_j' \models A\}_{j\in\{1,n\}}$ and $\{p_j', \omega_j'' \models true\}_{j\in\{1,k\}}$;
    2. $\ell_1 :: \ell_2 :: b_i, \omega \vdash < \alpha > A$ can be obtained from $\{p_j, \omega_j' \vdash A\}_{j\in\{1,n\}}$ and $\{p_j', \omega_j'' \vdash true\}_{j\in\{1,k\}}$.

    By rule $+ <> \epsilon$, $\ell_1 :: \ell_2 :: b_1 + b_2, \omega \vdash < \alpha > A$ can be obtained from $\ell_1 :: \ell_2 :: b_i, \omega \vdash < \alpha > A$. Moreover, $p_j, p_j' \in \mathcal{S}_{\ell_1::\ell_2::b_1+b_2}$, for each $j \in \{1, n\}, j \in \{1, k\}$. By repeating a reasoning analogous to the one of the untimed case, for each $j \in \{1, n\}$ and $\pi \models \omega$, there exists $(p, \pi) \overset{\alpha}{\mapsto} (p_j, \pi')$, where $\pi' \models \omega_j'$.

  - If $type(\alpha) = \bar{\epsilon}$, then $p \overset{\alpha,\omega'}{\longmapsto} p'$ iff, by rule $choice_2$, $\ell_1 :: \ell_2 :: b_i \overset{\alpha,true}{\longmapsto} \ell_1 :: \ell_2 :: b_i$, for each $i \in \{1, 2\}$, and $p' = p$, $\omega' \equiv true$.

    Since $\omega_2 \equiv reset(\omega, C_1, C_2) - 1 \equiv \omega - 1$, then $p, \omega - 1 \models A$. Moreover, $\ell_1 :: \ell_2 :: b_i, \omega \models < \alpha > true$, for each $i \in \{1, 2\}$. Since $m(\ell_1 :: \ell_2 :: b_i) \leq m(\ell_1 :: \ell_2 :: b_1 + b_2)$, by induction hypothesis there exist $\{p_{i,j} \in \mathcal{S}_{\ell_1::\ell_2::b_i}\}_{j\in\{1,n_i\}i\in\{1,2\}}$, such that

    1. $\{p_{i,j}, \omega_{i,j}' \models true\}_{j\in\{1,n_i\},i\in\{1,2\}}$;

2. $\ell_1 :: \ell_2 :: b_i, \omega \vdash < \alpha > true$ can be obtained from $\{p_{i,j}, \omega'_{i,j} \vdash A\}_{j \in \{1,n_i\}}$, for each $i \in \{1,2\}$.

By rule $+ <> \bar{\epsilon}$, $p, \omega \vdash < \alpha > A$ can be obtained from $p, \omega - 1 \vdash A$ and $\{\ell_1 :: \ell_2 :: b_i, \omega \vdash < \alpha > true\}_{i \in \{1,2\}}$. Moreover, $p_{i,j} \in \mathcal{S}_{\ell_1 :: \ell_2 :: b_1 + b_2}$.

- $p = \ell :: t \bigtriangledown p_1$.

There are two cases: either $type(\alpha) = \epsilon$ or $type(\alpha) = \bar{\epsilon}$.

– If $type(\alpha) = \epsilon$, then $p \overset{\alpha,\omega'}{\longmapsto} p'$ has been obtained either by rule $\bigtriangledown_1$ or by rule $\bigtriangledown_2$.

1. We have $\ell :: t \overset{\alpha,\omega'}{\longmapsto} p'$ and $\bigtriangledown_1$ has been applied. Since the process is well-formed, $Loc(p') \subseteq Loc(\ell :: t)$ and $Loc(p') \cap Loc(p_1) = \emptyset$. Therefore, $reset(\omega, C'_1, C'_2) \equiv reset(\omega, C_1, C_2)$, where $C'_1 = C \cup (loc(p') \cap loc(\ell :: t))$ and $C'_2 = loc(p') \setminus loc(\ell :: t)$. Moreover, $reset(\omega, C'_1, C'_2) \equiv reset(pro_{\ell :: t \cup C}(\omega), C'_1, C'_2)$.
Since $p', \omega_2 \models A$, then $p', reset(pro_{\ell :: t \cup C}(\omega), C'_1, C'_2) - 1 \models A$ and rule $<>$ is applicable with respect to $\ell :: t \overset{\alpha,\omega'}{\longmapsto} p'$. Actually,
   (a) $\models \forall(\omega \supset \omega')$ implies $\models \forall(pro_{\ell :: t \cup C}(\omega) \supset \omega')$, since $\omega' \in \Omega(loc(\ell :: t))$;
   (b) $\models \forall(\omega_2 \supset delay(p'))$.

   By the soundness of $<>$, $\ell :: t, pro_{loc(\ell :: t) \cup C}(\omega) \models < \alpha > A$. Since $m(\ell :: t) < m(\ell :: t \bigtriangledown p)$, by induction hypothesis there exist $\{p_i \in \mathcal{S}_{\ell :: t}\}_{i \in \{1,n\}}$ and $\{p'_j \in \mathcal{S}_{\ell :: t}\}_{j \in \{1,k\}}$ such that
   (a) $\{p_i, \omega'_i \models A\}_{i \in \{1,n\}}$ and $\{p'_j, \omega''_j \models true\}_{j \in \{1,k\}}$;
   (b) $\ell :: t, pro_{C \cup loc(\ell :: t)}(\omega) \vdash < \alpha > A$ can be obtained from $\{p_i, \omega'_i \vdash A\}_{i \in \{1,n\}}$ and $\{p'_j, \omega''_j \vdash true\}_{j \in \{1,k\}}$.

   We have $\ell :: t \bigtriangledown p, \omega \vdash < \alpha > A$ from $\ell :: t, pro_{C \cup loc(\ell :: t)}(\omega) \vdash < \alpha > A$, by rule $\bigtriangledown <> \epsilon_1$. Moreover, $p_i, p'_j \in \mathcal{S}_{\ell :: t \bigtriangledown p_1}$ and by repeating a reasoning analogous to that of the untimed case, for each $i \in \{1, n\}$ and $\pi \models \omega$, there exists $(p, \pi) \overset{\alpha}{\mapsto} (p_i, \pi')$, where $\pi' \models \omega_i$.

2. We have $p_1 \overset{\alpha,\omega'}{\longmapsto} p'_1$, $p' = \ell :: t \bigtriangledown p'_1$ and $\bigtriangledown_2$ has been applied.
Let $C_{1,1} = C \cup ((loc(p'_1) \cap loc(p_1)) \setminus main(t))$ and $C_{1,2} = loc(p'_1) \setminus loc(p_1)$, while $C_{2,1} = loc(\ell :: t)$ and $C_{2,2} = \emptyset$. Since the process is well-formed and ω does not contain $x + n\mathcal{R}y + n$ for $x, y \in loc(p)$, by proposition 7.2.2, $reset(\omega, C_1, C_2) \equiv reset(\omega, C_{1,1}, C_{1,2}) \wedge reset(\omega, C_{2,1}, C_{2,2})$. Moreover, $reset(\omega, C_{2,1}, C_{2,2}) \equiv pro_{loc(\ell :: t)}(\omega)$ and $reset(\omega, C_{1,1}, C_{1,2}) \equiv reset(\omega_p, C'_{1,1}, C_{1,2})$, where $C'_{1,1} = C \cup (loc(p'_1) \cap loc(p_1)))$ and $\omega_p \equiv pro_{C \cup (loc(p_1) \setminus main(t))}(\omega_1)$.

Since $\omega_2 \equiv reset(\omega, C_1, C_2) - 1$ and $\ell :: t \bigtriangledown p_1', \omega_2 \models A$, then $p_1', reset(\omega_p, C_{1,1}', C_{1,2}) - 1 \models (A)_{(\ell::t, pro_{loc(\ell::t)}(\omega)-1)\bigtriangledown}$. Moreover, rule $<>$ is applicable with respect to $p_1 \stackrel{\alpha,\omega'}{\longmapsto} p_1'$. Actually,

(a) $\models \forall(\omega \supset \omega')$ implies $\models \forall(\omega_p \supset \omega')$, since $\omega' \in \Omega(loc(p_1) \setminus main(t))$;

(b) $\models \forall(\omega_2 \supset delay(p'))$ implies $\models \forall(reset(\omega_p, C_{1,1}', C_{1,2}) - 1 \supset delay(p_1'))$.

By the soundness of this rule, $p_1, \omega_p \models < \alpha > (A)_{(\ell::t, pro_{loc(\ell::t)}(\omega)-1)\bigtriangledown}$. Since $m(p_1) < m(\ell :: t \bigtriangledown p_1)$, by induction hypothesis there exist $\{p_{1,i} \in \mathcal{S}_{p_1}\}_{i\in\{1,n\}}$ and $\{p_{1,j}' \in \mathcal{S}_{p_1}\}_{j\in\{1,k\}}$ such that

(a) $\{p_{1,i}, \omega_i' \models (A)_{(\ell::t, pro_{loc(\ell::t)}(\omega)-1)\bigtriangledown}\}_{i\in\{1,n\}}$, $\{p_{1,j}', \omega_j'' \models true\}_{j\in\{1,k\}}$;

(b) $p_1, \omega_p \vdash < \alpha > (A)_{(\ell::t, pro_{loc(\ell::t)}(\omega)-1)}$ can be obtained from $\{p_{1,i}, \omega_i' \vdash (A)_{(\ell::t, pro_{loc(\ell::t)}(\omega)-1)\bigtriangledown}\}_{i\in\{1,n\}}$ and $\{p_{1,j}', \omega_j'' \vdash true\}_{j\in\{1,k\}}$.

Since $\models \forall(\omega_2 \supset delay(p'))$, then $\models \forall(pro_{\ell::t}(\omega) - 1 \supset delay(\ell :: t))$. We obtain $\ell :: t \bigtriangledown p_1, \omega \vdash < \alpha > A$ from $p_1, \omega_p \vdash < \alpha > (A)_{(\ell::t, pro_{loc(\ell::t)}(\omega)-1)\bigtriangledown}$, by rule $\bigtriangledown <> \epsilon_2$.

Since $p_{1,i}, \omega_i' \models (A)_{(\ell::t, pro_{loc(\ell::t)}(\omega)-1)\bigtriangledown}$, then $\ell :: t \bigtriangledown p_{1,i}, \omega_i' \wedge pro_{loc(\ell::t)}(\omega) - 1 \models A$. By repeating a reasoning analogous to the one for the untimed case, $\ell :: t \bigtriangledown p_{1,i}, p_{1,j}' \in \mathcal{S}_{\ell::t\bigtriangledown p_1}$. Moreover, $\{p_{1,i}, \omega_i' \vdash (A)_{(\ell::t, pro_{loc(\ell::t)}(\omega)-1)\bigtriangledown}\}_{i\in\{1,n\}}$ are obtained by shift rule from $\{\ell :: t \bigtriangledown p_{1,i}, \omega_i' \wedge pro_{loc(\ell::t)}(\omega) - 1 \vdash A\}_{i\in\{1,n\}}$.

– If $type(\alpha) = \bar{\epsilon}$ the proof is analogous to the corresponding case for choice.

- $p = \ell :: p_1 \times p_2$.

If $p, \omega \models < \beta > A$ with $\beta \in \mathcal{E}(Act)$, then there exists $p \stackrel{\alpha,\omega'}{\longmapsto} p'$ such that is consistent with β. We omit the details regarding the structure of β that are analogous to the ones for the untimed case. By rule *par*, $p \stackrel{\alpha,\omega'}{\longmapsto} p'$ iff $p_i \stackrel{\alpha_i,\omega_i'}{\longmapsto} p_i'$, for each $i \in \{1, 2\}$, and $\alpha \in \alpha_1 \otimes \alpha_2$, $\omega' \equiv \omega_1' \wedge \omega_2'$ and $p' = \ell :: p_1' \times p_2'$.

The proof is by cases on p_2.

- Suppose $p_2 = \ell_1 :: \ell_2 :: nil$. We have $p_2' = p_2$ and $\omega_2' \equiv true$ and $\alpha_2 = a : \bar{\epsilon}$ for $a \in \#(*)$. Since the process is well-formed, by proposition 7.2.2, $reset(\omega, C_1, C_2) \equiv reset(\omega, C_1', C_2') \wedge pro_{\{\ell_1,\ell_2\}}(\omega)$, where $C_1' = C \cup (loc(p_1) \cap loc(p_1'))$ and $C_2' = loc(p_1') \setminus loc(p_2')$. Moreover, $reset(pro_{C\cup loc(p_1)}(\omega), C_1', C_2') \equiv reset(\omega, C_1', C_2')$. Let $\omega_3 \equiv pro_{\{\ell_1,\ell_2\}}(\omega) - 1$.

  Since $\omega_2 \equiv reset(\omega, C_1, C_2) - 1$ and $p', \omega_2 \models A$, then $p_1', reset(pro_{C\cup loc(p_1)}(\omega), C_1', C_2') - 1 \models (A)_{\ell\times(p_2,\omega_3)}$. Moreover, rule $<>$ is applicable with respect to $p_1 \stackrel{\alpha_1,\omega'}{\longmapsto} p_1'$. Actually,

  1. $\models \forall(\omega \supset \omega')$ implies $\models \forall(pro_{loc(p_1)\cup C}(\omega) \supset \omega')$, since $\omega' \in \Omega(loc(p_1))$;

2. $\models \forall(\omega_2 \supset delay(\ell :: p_1' \times p_2))$ implies $\models \forall(reset(pro_{C \cup loc(p_1)}(\omega), C_1', C_2') - 1 \supset delay(p_1'))$.

By the soundness of this rule, $p_1, pro_{loc(p_1) \cup C}(\omega) \models < \alpha_1 > (A)_{\ell \times (p_2, \omega_3)}$.

Since $m(p_1) < m(\ell :: p_1 \times p_2)$, by induction hypothesis, there exist $\{p_{1,i} \in \mathcal{S}_{p_1}\}_{i \in \{1,n\}}$ and $\{p_{1,j}' \in \mathcal{S}_{p_1}\}_{j \in \{1,k\}}$ such that

1. $\{p_{1,i}, \omega_{1,i}' \models (A)_{\ell \times (p_2, \omega_3)}\}_{i \in \{1,n\}}$ and $\{p_{1,j}', \omega_{1,j}'' \models true\}_{j \in \{1,k\}}$;
2. $p_1, pro_{C \cup loc(p_1)}(\omega) \vdash < \alpha_1 > (A)_{\ell \times (p_2, \omega_3)}$ can be obtained from $\{p_{1,i}, \omega_{1,i}' \vdash (A)_{\ell \times (p_2, \omega_3)}\}_{i \in \{1,n\}}$ and $\{p_{1,j}', \omega_{1,j}'' \vdash true\}_{j \in \{1,k\}}$.

We obtain $\ell :: p_1 \times p_2, \omega \vdash < \alpha > A$ from $p_1, pro_{loc(p_1) \cup C}(\omega) \vdash < \alpha_1 > (A)_{\ell \times (p_2, \omega_3)}$ by applying rule $\times <> nil$, since $\alpha \in \alpha_1 \otimes \alpha_2$.

Since $p_{1,i}, \omega_{1,i}' \models (A)_{\ell \times (p_2, \omega_3)}$, then $\ell :: p_{1,i} \times p_2, \omega_{1,i}' \wedge \omega_3 \models A$. Moreover, premises $\{p_{1,i}, \omega_{1,i}' \vdash (A)_{\ell \times (p_2, \omega_3)}\}_{i \in \{1,n\}}$ are obtained from $\{\ell :: p_{1,i} \times p_2, \omega_{1,i}' \wedge \omega_3 \vdash A\}_{i \in \{1,n\}}$, by shift rule. It is obvious that $p_{1,j}', \ell :: p_{1,i} \times \ell_1 :: t \in \mathcal{S}_{\ell :: p_1 \times p_2}$.

- Suppose $p_2 = \ell_1 :: \ell_2 :: [m, M]a.t$. There are two cases: either $p_2' = p_2$ and $\omega_2 \equiv true$ and $\alpha_2 = b : \bar{\epsilon}$ for $b \in \#(*)$ or $p_2' = \ell_1 :: t$ and $\omega_2 \equiv m \leq \ell_2 \leq M$, $\alpha_2 = a : \epsilon$.

  - Suppose $\alpha_2 = b : \bar{\epsilon}$. Since the process is well-formed, by proposition 7.2.2, $reset(\omega, C_1, C_2) \equiv reset(\omega, C_1', C_2') \wedge pro_{loc(p_2)}(\omega)$, where $C_1' = C \cup (loc(p_1) \cap loc(p_1'))$ and $C_2' = loc(p_1') \backslash loc(p_2')$. Moreover, $reset(\omega, C_1', C_2') \equiv reset(pro_{C \cup loc(p_1)}(\omega), C_1', C_2')$. Let $\omega_1 \equiv reset(pro_{C \cup loc(p_1)}(\omega), C_1', C_2') - 1$ and $\omega_3 \equiv pro_{loc(p_2)}(\omega) - 1$.
    Since $p', reset(\omega, C_1, C_2) - 1 \models A$, then $p_1', \omega_1 \models (A)_{\ell \times (p_2, \omega_3)}$. Moreover, rule $<>$ is applicable with respect to $p_1 \overset{\alpha_1, \omega'}{\longmapsto} p_1'$. Actually,
    1. $\models \forall(\omega \supset \omega')$ implies $\models \forall(pro_{loc(p_1) \cup C}(\omega) \supset \omega')$;
    2. $\models \forall(\omega_2 \supset delay(\ell :: p_1' \times p_2))$ implies $\models \forall(\omega_1 \supset delay(p_1'))$.

    By the soundness of this rule, $p_1, pro_{loc(p_1) \cup C}(\omega) \models < \alpha_1 > (A)_{\ell \times (p_2, \omega_3)}$.
    Since $m(p_1) < m(\ell :: p_1 \times p_2)$, by induction hypothesis, there exist $\{p_{1,i} \in \mathcal{S}_{p_1}\}_{i \in \{1,n\}}$ and $\{p_{1,j}' \in \mathcal{S}_{p_1}\}_{j \in \{1,k\}}$ such that
    1. $\{p_{1,i}, \omega_{1,i}' \models (A)_{\ell \times (p_2, \omega_3)}\}_{i \in \{1,n\}}$ and $\{p_{1,j}', \omega_{1,j}'' \models true\}_{j \in \{1,k\}}$;
    2. $p_1, pro_{C \cup loc(p_1)}(\omega) \vdash < \alpha_1 > (A)_{\ell \times (p_2, \omega_3)}$ can be obtained from $\{p_{1,i}, \omega_{1,i}' \vdash (A)_{\ell \times (p_2, \omega_3)}\}_{i \in \{1,n\}}$ and $\{p_{1,j}', \omega_{1,j}'' \vdash true\}_{j \in \{1,k\}}$.

    Since $\models \forall(\omega_2 \supset delay(\ell :: p_1' \times p_2))$, then $\models \forall(\omega_3 \supset delay(p_2))$ and $delay(p_2) \equiv \ell_2 \leq M$. Therefore, we obtain $\ell :: p_1 \times p_2, \omega \vdash < \alpha > A$ from $p_1, pro_{loc(p_1) \cup C}(\omega) \vdash < \alpha_1 > (A)_{\ell \times (p_2, \omega_3)}$, since $\alpha \in \alpha_1 \otimes \alpha_2$, by applying $\tau \times <>_2$.

Since $p_{1,i}, \omega'_{1,i} \models (A)_{\ell\times(p_2,\omega_3)}$, then $\ell :: p_{1,i} \times p_2, \omega'_{1,i} \wedge \omega_3 \models A$. Moreover, premises $\{p_{1,i}, \omega'_{1,i} \vdash (A)_{\ell\times(p_2,\omega_3)}\}_{i\in\{1,n\}}$ are obtained from $\{\ell :: p_{1,i} \times p_2, \omega'_{1,i} \wedge \omega_3 \vdash A\}_{i\in\{1,n\}}$, by shift rule. It is obvious that $p'_{1,j}, \ell :: p_{1,i} \times p_2 \in \mathcal{S}_{\ell::p_1\times p_2}$.

– Suppose $\alpha_2 = a : \epsilon$. Since the process is well-formed, by proposition 7.2.2, $reset(\omega, C_1, C_2) \equiv reset(\omega, C'_1, C'_2) \wedge reset(\omega, \{\ell_1\}, loc(t))$, where $C'_1 = C \cup (loc(p_1) \cap loc(p'_1))$ and $C'_2 = loc(p'_1) \setminus loc(p'_2)$. Moreover, $reset(\omega, C'_1, C'_2) \equiv reset(pro_{C\cup loc(p_1)}(\omega), C'_1, C'_2)$.
Let $\omega_1 \equiv reset(pro_{C\cup loc(p_1)}(\omega), C'_1, C'_2) - 1$ and $\omega_3 \equiv reset(\omega, \{\ell_1\}, loc(t)) - 1$.
Since $p', reset(\omega, C_1, C_2) - 1 \models A$, then $p'_1, \omega_1 \models (A)_{\ell\times(\ell_1::t,\omega_3)}$. Moreover, rule $<>$ is applicable with respect to $p_1 \overset{\alpha_1,\omega'_1}{\longmapsto} p'_1$. Actually,

1. $\models \forall(\omega \supset \omega')$ implies $\models \forall(pro_{loc(p_1)\cup C}(\omega) \supset \omega'_1)$;
2. $\models \forall(\omega_2 \supset delay(\ell :: p'_1 \times p'_2))$ implies $\models \forall(\omega_1 \supset delay(p'_1))$.

By the soundness of this rule, $p_1, pro_{loc(p_1)\cup C}(\omega) \models < \alpha_1 > (A)_{\ell\times(\ell_1::t,\omega_3)}$.
Since $m(p_1) < m(\ell :: p_1 \times p_2)$, by induction hypothesis, there exist $\{p_{1,i} \in \mathcal{S}_{p_1}\}_{i\in\{1,n\}}$ and $\{p'_{1,j} \in \mathcal{S}_{p_1}\}_{j\in\{1,k\}}$ such that

1. $\{p_{1,i}, \omega'_{1,i} \models (A)_{\ell\times(\ell_1::t,\omega_3)}\}_{i\in\{1,n\}}$ and $\{p'_{1,j}, \omega''_{1,j} \models true\}_{j\in\{1,k\}}$;
2. $p_1, pro_{C\cup loc(p_1)}(\omega) \vdash < \alpha_1 > (A)_{\ell\times(\ell_1::t,\omega_3)}$ can be obtained from $\{p_{1,i}, \omega'_{1,i} \vdash (A)_{\ell\times(\ell_1::t,\omega_3)}\}_{i\in\{1,n\}}$ and $\{p'_{1,j}, \omega''_{1,j} \vdash true\}_{j\in\{1,k\}}$.

Since $\models \forall(\omega_2 \supset delay(\ell :: p'_1 \times p'_2))$, then $\models \forall(\omega_3 \supset delay(\ell_1 :: t))$ and $\models \forall(\omega \supset \omega')$ implies $\models \forall(\omega \supset \omega'_2)$, where $\omega'_2 \equiv m \leq \ell_2 \leq M$.
We obtain $\ell :: p_1 \times p_2, \omega \vdash < \alpha > A$ from $p_1, pro_{loc(p_1)\cup C}(\omega) \vdash < \alpha_1 > (A)_{\ell\times(\ell_2::t,\omega_3)}$, since $\alpha \in \alpha_1 \otimes \alpha_2$, by applying rule $\tau\times <>_1$.
Since $p_{1,i}, \omega'_{1,i} \models (A)_{\ell\times(\ell_1::t,\omega_3)}$, then $\ell :: p_{1,i} \times \ell_1 :: t, \omega'_{1,i} \wedge \omega_3 \models A$. Moreover, premises $\{p_{1,i}, \omega'_{1,i} \vdash (A)_{\ell\times(\ell_1::t,\omega_3)}\}_{i\in\{1,n\}}$ are obtained from $\{\ell :: p_{1,i} \times \ell_1 :: t, \omega'_{1,i} \wedge \omega_3 \vdash A\}_{i\in\{1,n\}}$, by shift rule. It is obvious that $p'_{1,j}, \ell :: p_{1,i} \times \ell_1 :: t \in \mathcal{S}_{\ell::p_1\times p_2}$.

- Suppose $p_2 = \ell_1 :: \ell_2 :: a.t$. The proof is similar to the previous one.

- Suppose $p_2 = \ell_1 :: \ell_2 :: b_1 + b_2$. There are two cases: either $type(proj_2(\alpha)) = \epsilon$ or $type(proj_2(\alpha)) = \bar{\epsilon}$.

  – $type(proj_2(\alpha)) = \epsilon$. Therefore, $type(\alpha_2) = \epsilon$ and $p_2 \overset{\alpha_2,\omega'_2}{\longmapsto} p'_2$ iff $\ell_1 :: \ell_2 :: b_i \overset{\alpha_2,\omega_2}{\longmapsto} p'_2$, for some $i \in \{1, 2\}$. By rule par, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha,\omega'}{\longmapsto} \ell :: p'_1 \times p'_2$.
  Since the process is well-formed, then $reset(\omega, C_1, C_2) \equiv reset(\omega, C'_1, C'_2)$, where $C'_1 = C \cup (loc(\ell_1 :: \ell_2 :: b_i) \cap loc(p'_2))$ and $C'_2 = loc(p'_2) \setminus loc(\ell_1 :: \ell_2 :: b_i)$.

Since $\omega_2 \equiv reset(\omega, C_1, C_2) - 1$ and $p', \omega_2 \models A$, then $\ell :: p'_1 \times p'_2, reset(\omega, C'_1, C'_2) - 1 \models A$ and rule $<>$ is applicable with respect to $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \stackrel{\alpha,\omega'}{\longmapsto} \ell :: p'_1 \times p'_2$. By the soundness of this rule, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega \models < \alpha > A$.

Since $m(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i) < m(\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2)$, by induction hypothesis there exist $\{\ell :: p_{1,j} \times p_{2,j} \in \mathcal{S}_{\ell::p_1 \times \ell_1::\ell_2::b_i}\}_{j\in\{1,n\}}$ and $\{\ell :: p'_{1,j} \times p'_{2,j} \in \mathcal{S}_{\ell::p_1\times\ell_1::\ell_2::b_i}\}_{j\in\{1,k\}}$ such that

1. $\{\ell :: p_{1,j} \times p_{2,j}, \omega'_{1,j} \models A\}_{j\in\{1,n\}}$ and $\{\ell :: p'_{1,j} \times p'_{2,j}, \omega''_{1,j} \models true\}_{j\in\{1,k\}}$;
2. $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega \vdash < \alpha > A$ can be obtained from $\{\ell :: p_{1,j} \times p_{2,j}, \omega'_{1,j} \vdash A\}_{j\in\{1,n\}}$ and $\{\ell :: p'_{1,j} \times p'_{2,j}, \omega''_{1,j} \vdash true\}_{j\in\{1,k\}}$.

We obtain $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2, \omega \vdash < \alpha > A$ from $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega \vdash < \alpha > A$, by rule $\times <> +\epsilon$. Moreover, $\ell :: p_{1,j} \times p_{2,j}, \ell :: p'_{1,j} \times p'_{2,j} \in \mathcal{S}_{\ell::p_1\times p_2}$.

– $type(proj_2(\alpha)) = \bar{\epsilon}$. Therefore, $type(\alpha_2) = \bar{\epsilon}$ and $p_2 \stackrel{\alpha_2,\omega'_2}{\longmapsto} p'_2$ iff $\ell_1 :: \ell_2 :: b_i \stackrel{\alpha_2,true}{\longmapsto} \ell_1 :: \ell_2 :: b_i$, for each $i \in \{1, 2\}$, and $\omega'_2 \equiv true$, $p'_2 = p_2$.

By rule par, for each $i \in \{1, 2\}$, $\ell :: p_1 \times \ell_1 : \ell_2 :: b_i \stackrel{\alpha,\omega'}{\longmapsto} \ell :: p'_1 \times \ell_1 : \ell_2 :: b_i$. Since the process is well-formed, $reset(\omega, C_1, C_2) \equiv reset(\omega, C_{1,1}, C_{1,2}) \equiv reset(\omega, C_{2,1}, C_{2,2})$, where $C_{i,1} = C \cup (loc(\ell :: p_1 \times \ell_1 : \ell_2 :: b_i) \cap loc(\ell :: p'_1 \times \ell_1 : \ell_2 :: b_i))$ and $C_{i,2} = loc(\ell :: p'_1 \times \ell_1 : \ell_2 :: b_i) \setminus loc(\ell :: p_1 \times \ell_1 : \ell_2 :: b_i)$. Since $p', reset(\omega, C_1, C_2) - 1 \models A$ and $p' = \ell :: p'_1 \times \ell_1 : \ell_2 :: b_1 + b_2$, then $\ell :: p'_1 \times \ell_1 : \ell_2 :: b_i, \omega_2 \models (A)_{\times+b_j}$, for $i \neq j$.

Moreover, for each $i \in \{1, 2\}$, rule $<>$ is applicable with respect to $\ell :: p_1 \times \ell_1 : \ell_2 :: b_i \stackrel{\alpha,\omega'}{\longmapsto} \ell :: p'_1 \times \ell_1 : \ell_2 :: b_i$. Actually, $\models \forall(\omega_2 \supset delay(\ell :: p'_1 \times \ell_1 : \ell_2 :: b_1 + b_2))$ implies $\models \forall(\omega_2 \supset delay(\ell :: p'_1 \times \ell_1 : \ell_2 :: b_i))$.

By the soundness of this rule, $\ell :: p_1 \times \ell_1 : \ell_2 :: b_i, \omega \models < \alpha > (A)_{\times+b_j}$, for each $i \in \{1, 2\}$. Since $m(\ell :: p_1 \times \ell_1 : \ell_2 :: b_i) < m(\ell :: p_1 \times \ell_1 : \ell_2 :: b_1 + b_2))$, by induction hypothesis there exist $\{\ell :: p_{1,j,i} \times p_{2,j,i} \in \mathcal{S}_{\ell::p_1\times\ell_1::\ell_2::b_i}\}_{j\in\{1,n_i\},i\in\{1,2\}}$, $\{\ell :: p'_{1,j,i} \times p'_{2,j,i} \in \mathcal{S}_{\ell::p_1\times\ell_1::\ell_2::b_i}\}_{j\in\{1,k_i\},i\in\{1,2\}}$ such that

1. $\{\ell :: p_{1,j,i} \times p_{2,j,i}, \omega'_{1,j,i} \models (A)_{\times+b_k}\}_{j\in\{1,n_i\},i\in\{1,2\},k\neq i}$ and $\{\ell :: p'_{1,j,i} \times p'_{2,j,i}, \omega''_{1,j,i} \models true\}_{j\in\{1,k_i\},i\in\{1,2\}}$;
2. $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i, \omega \vdash < \alpha > A$ can be obtained from $\{\ell :: p_{1,j,i} \times p_{2,j,i}, \omega'_{1,j,i} \vdash (A)_{\times+b_k}\}_{j\in\{1,n_i\}}$ and $\{\ell :: p'_{1,j,i} \times p'_{2,j,i}, \omega''_{1,j,i} \vdash true\}_{j\in\{1,k_i\}}$, for each $i \in \{1, 2\}$.

We obtain $\ell :: p_1 \times \ell_1 : \ell_2 :: b_1 + b_2, \omega \vdash < \alpha > A$ from $\{\ell :: p_1 \times \ell_1 : \ell_2 :: b_i, \omega \vdash < \alpha > (A)_{\times+b_j}\}_{i,j\in\{1,2\},i\neq j}$, by rule $\times <> +\bar{\epsilon}$. Moreover, $\ell :: p'_{1,j,i} \times p'_{2,j,i} \in \mathcal{S}_{\ell::p_1\times p_2}$ for each $j \in \{1, k_i\}$ and premises $\{\ell :: p_{1,j,i} \times p_{2,j,i}, \omega'_{1,j,i} \vdash (A)_{\times+b_k}\}_{j\in\{1,n_i\}}$ are obtained by shift rule from $\{\ell :: p_{1,j,i} \times p''_{2,j,i}, \omega'_{1,j,i} \vdash A\}_{j\in\{1,n_i\}}$, where $\ell :: p_{1,j,i} \times p''_{2,j,i} \in \mathcal{S}_{\ell::p_1\times p_2}$.

- Suppose $p_2 = \ell_1 :: t \bigtriangledown p_3$. There are two cases: either $type(proj_2(\alpha)) = \epsilon$ or $type(proj_2(\alpha)) = \bar{\epsilon}$.

  - $type(proj_2(\alpha)) = \epsilon$. Therefore, $type(\alpha_2) = \epsilon$ and $p_2 \overset{\alpha_2,\omega_2'}{\longmapsto} p_2'$ iff either $\ell_1 :: t \overset{\alpha_2,\omega_2'}{\longmapsto} p_2'$ (rule $\bigtriangledown_1$) or $p_3 \overset{\alpha_2,\omega_2'}{\longmapsto} p_3'$ and $p_2' = \ell_1 :: t \bigtriangledown p_3'$ (rule $\bigtriangledown_2$). The first case is similar to the corresponding case for choice. In the second case, by rule *par*, $\ell :: p_1 \times p_3 \overset{\alpha,\omega'}{\longmapsto} \ell :: p_1' \times p_3'$.
    Since the process is well-formed by proposition 7.2.2, $reset(\omega, C_1, C_2) \equiv reset(\omega, C_1', C_2') \wedge pro_{loc(\ell_1::t)}(\omega)$, where $C_1' = C \cup ((loc(\ell :: p_1' \times p_3') \cap loc(\ell :: p_1 \times p_3)) \setminus main(t))$, $C_2' = loc(\ell :: p_1' \times p_3') \setminus loc(\ell :: p_1 \times p_3)$. Moreover, $reset(\omega, C_1', C_2') \equiv reset(\omega_p, C_1'', C_2')$, where $C_1'' = C_1' \cup main(t)$ and $\omega_p \equiv pro_{C \cup (loc(\ell::p_1 \times p_3) \setminus main(t))}(\omega)$.
    Let $\omega_1 \equiv reset(\omega_p, C_1'', C_2') - 1$ and $\omega_3 \equiv pro_{loc(\ell_1::t)}(\omega) - 1$.
    Since $\omega_2 \equiv reset(\omega, C_1, C_2) - 1$ and $p', \omega_2 \models A$, i.e. $\ell :: p_1' \times \ell_1 :: t \bigtriangledown p_3', \omega_2 \models A$, and $\omega_2 \equiv \omega_1 \wedge \omega_3$, then $\ell :: p_1' \times p_3', \omega_1 \models (A)_{\times(\ell_1::t,\omega_3)\bigtriangledown}$.
    Moreover, rule $<>$ is applicable with respect to $\ell :: p_1 \times p_3 \overset{\alpha,\omega'}{\longmapsto} \ell :: p_1' \times p_3'$. Actually,
    1. $\models \forall(\omega \supset \omega')$ implies $\models \forall(\omega_p \supset \omega')$;
    2. $\models \forall(\omega_2 \supset delay(\ell :: p_1' \times \ell_2 :: t \bigtriangledown p_3'))$ implies $\models \forall(\omega_1 \supset delay(\ell :: p_1' \times p_3'))$.

    By the soundness of this rule, $\ell :: p_1 \times p_3, \omega_1 \models < \alpha > (A)_{\times(\ell_1::t,\omega_3)\bigtriangledown}$.
    Since $m(\ell :: p_1 \times p_3) < m(\ell :: p_1 \times \ell_1 :: t \bigtriangledown p_3)$, by induction hypothesis there exist $\{\ell :: p_{1,i} \times p_{3,i} \in \mathcal{S}_{\ell::p_1 \times p_3}\}_{i \in \{1,n\}}$ and $\{\ell :: p_{1,j}' \times p_{3,j}' \in \mathcal{S}_{\ell::p_1 \times p_3}\}_{j \in \{1,k\}}$ such that
    1. $\{\ell :: p_{1,i} \times p_{3,i}, \omega_i \models (A)_{\times(\ell_1::t,\omega_3)\bigtriangledown}\}_{i \in \{1,n\}}$ and $\{\ell :: p_{1,j}' \times p_{3,j}', \omega_j' \models true\}_{j \in \{1,k\}}$;
    2. $\ell :: p_1 \times p_3, \omega_p \vdash < \alpha > (A)_{\times(\ell_1::t,\omega_3)\bigtriangledown}$ can be obtained from $\{\ell :: p_{1,i} \times p_{3,i}, \omega_i \vdash (A)_{\times(\ell_1::t,\omega_3)\bigtriangledown}\}_{i \in \{1,n\}}$ and $\{\ell :: p_{1,j}' \times p_{3,j}', \omega_j' \vdash true\}_{j \in \{1,k\}}$.

    Since $\models \forall(\omega_2 \supset delay(\ell :: p_1' \times \ell_1 :: t \bigtriangledown p_3'))$ then $\models \forall(\omega_3 \supset delay(\ell_1 :: t))$. We obtain $\ell :: p_1 \times \ell_1 :: t \bigtriangledown p_3, \omega \vdash < \alpha > A$ from $\ell :: p_1 \times p_3, \omega_p \vdash < \alpha > (A)_{\times(\ell_1::t,\omega_3)\bigtriangledown}$, by rule $\times <> \bigtriangledown\epsilon_2$.
    Since $\ell :: p_{1,i} \times p_{3,i}, \omega_i \models (A)_{\times(\ell_1::t,\omega_3)\bigtriangledown}$, then $\ell :: p_{1,i} \times \ell_1 :: t \bigtriangledown p_{3,i}, \omega_i \wedge \omega_3 \models A$. Moreover, premises $\{\ell :: p_{1,i} \times p_{3,i}, \omega_i \vdash (A)_{\times(\ell_1::t,\omega_3)\bigtriangledown}\}_{i \in \{1,n\}}$ are obtained by shift rule from $\{\ell :: p_{1,i} \times \ell_1 :: t \bigtriangledown p_{3,i}, \omega_i \wedge \omega_3 \vdash A\}_{i \in \{1,n\}}$. It is obvious that $\ell :: p_{1,i} \times \ell_1 :: t \bigtriangledown p_{3,i}, \ell :: p_{1,j}' \times p_{3,j}' \in \mathcal{S}_{\ell::p_1 \times p_2}$.
  - $type(proj_2(\alpha)) = \bar{\epsilon}$. It is analogous to the corresponding one for choice.

- The cases of $p_2 = \ell_1 :: rec\ x.t$ and $p_2 = \ell_1 :: p_3 \times p_4$ are straightforward.

■