A Compositional μ-Calculus Proof System for Statecharts Processes [1]

Francesca Levi

Dipartimento di Informatica, Università di Pisa,
Corso Italia 40, 56100, Pisa, Italy,
levifran@di.unipi.it

Abstract

In this paper we propose a quite general process language called $\mathcal{SP}$, which can be suitably instantiated to model the synchronous language statecharts. The main novelty of the language is an operator of process refinement for representing statecharts hierarchy. Moreover, we define a compositional proof system for checking μ-calculus properties of $\mathcal{SP}$ processes. Such a proof system is sound in general and it is complete for finite-state processes including the one corresponding to statecharts.

Key words: Statecharts, μ-Calculus, Compositional Verification.

1 Introduction

Statecharts [12] are a visual synchronous formalism for the specification of reactive systems, which overcome the limitations of classical state-transition diagrams by introducing notions of hierarchy, concurrency and broadcast communication. The basic idea is to have transitions between basic states labelled by a pair *trigger/action*, where trigger express an enabling condition over input events and action gives the set of produced events. Basic components can be composed in parallel and can be refined. These features permit modular and structured specifications also of complex reactive systems. The language relies on the *synchrony hypothesis* [5]: the reaction of the system to environment inputs is instantaneous. Under this assumption the instantaneous reaction of the system (a step) is defined as a maximal set of parallel transitions *triggered* by the inputs. Due to the synchrony hypothesis transitions simultaneously produce new events, which may enable other transitions at the same time.

[1] Research partially supported by Esprit BRA 8130 LOMAPS

However, as it is well-known, this behaviour leads to some causal paradoxes, such as transitions disabling their own causes and self-enabling transitions. Hence, the definition of a formal semantics for statecharts turned out to be very complex and many different approaches have been suggested over the last few years [14,17,16,31,27,25,13]. Among the various proposals the semantics of [27] provides a very convincing interpretation for solving causality problems. The main idea is to require steps to be maximal sets of parallel transitions, which satisfy in addition some notions of *causality* and *consistency*. Unfortunately, the combination of the synchrony hypothesis with causality and consistency leads to a non-modular semantics [16].

In the reactive systems framework formal verification is an essential task. Propositional μ-calculus [19,29] has received a great success as a logic for verifying concurrent and reactive systems. It contains modal operators for describing actions of a system, and greatest and least fixpoints for expressing safety and liveness properties. These features give a great expressive power to μ-calculus, which indeed subsumes many modal and temporal logics, such as the branching time logics CTL, CTL^* [9]. In the last years, the problem of model-checking for μ-calculus has been extensively studied [11,20,30,34,7,2]. The classical approach consists of algorithms, that compute all the states of the model, which satisfy the formula. Standard approximation techniques are used for computing fixpoints. In contrast in *local model checking* [20,30,34] it is checked if a state satisfies the formula without computing all the states that satisfy the formula. The basic idea is that of using some form of fixpoint induction instead of approximation techniques. This approach has the clear advantage that only a restricted part of the state space is explored. In the framework of CCS-like languages local model checking has been combined with compositional verification [4,21,33,3,10], where the verification of a property for a compound process is reduced to the verification of some derived properties for its components. The role of compositionality for the verification of complex systems is well established. 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.

In this paper we address the problem of compositional verification for statecharts. We introduce a new process language called $\mathcal{SP}$, which can model statecharts according to the semantics of Pnueli and Shalev. Secondly, we define a compositional proof system for checking μ-calculus properties of $\mathcal{SP}$ processes. $\mathcal{SP}$ is a quite general CCS-like language which is parametric in the set of basic actions. In addition to the classical operators of prefixing, choice, parallel composition and recursion, there is an operator of *process refinement*, which naturally captures statecharts hierarchy. Moreover, explicit locations are used in processes as a counterpart of statecharts basic states. In order to define the semantics of $\mathcal{SP}$ as a labelled transition system, two operations over basic actions of *parallel composition* and *conflict* are assumed. The operation of parallel composition realizes the communication among parallel components and it is used in the parallel composition of processes for synchronizing the actions of the two processes. The operation of conflict is typical of this language and relies on the assumption that the performance of actions depends on the environment behaviour. Intuitively, an action a_1 is in conflict with an action a_2 if a_1 is enabled with respect to environment inputs, that do not enable a_2. The main feature of the semantics is that both real actions and *idle* actions of processes are explicitely represented. For this purpose, typed actions $a : \tau$, with $\tau \in \{idle, non - idle\}$ are used. A transition $a : idle$ means that the process is idle on the action a, namely it is idle with respect to the environment inputs, that enable a. Since we are interested in capturing the synchrony of statecharts (maximality of reaction) the semantics is defined so that a process is idle if and only if no real action can be performed because of the environment inputs. This behaviour can be achieved by assuming that a process is idle only on actions that are in conflict with all the possible actions of the process.

These features lead to a semantics for $\mathcal{SP}$ which is very adequate for capturing all the aspects of statecharts. In particular, we show a translation of statecharts into closed $\mathcal{SP}$ processes according to the semantics of Pnueli and Shalev. The idea is that of constructing in a compositional way a process with basic actions representing transitions and locations corresponding to statechart boxes. The refinement and the parallel composition of statecharts are directly captured by the corresponding operators over processes. The main problem is the definition of basic actions and of the corresponding operations of parallel composition and conflict. In fact, it is necessary to consider basic actions, where the input-output behaviour associated to transitions is represented. Unfortunately, since the semantics of Pnueli and Shalev is non-modular, it is not sufficient to observe the input-output behaviour of components. This problem is not surprising as already discussed by many authors [31,17,15,16,22]. One solution is to consider basic actions representing both input-output and the causal relation between inputs and outputs. In the operation of parallel composition an action a resulting from the parallel composition of actions a_1 and a_2 is required to represent the steps given by the union of the steps corresponding to a_1 and a_2. Therefore, the input-output behaviour of action a must be equivalent to the one of these steps, where the internal communication among parallel components is realized. We show that the causal orderings of a_1 and a_2 can be suitably combined in order to obtain an action a which correctly models the steps, where in addition the principle of causality and of consistency are satisfied. On the other hand, the operation of conflict can be simply defined by assuming that an action a_1 is in conflict with a_2 iff it it is enabled with respect to sets of inputs, which prevent the performance of a_2.

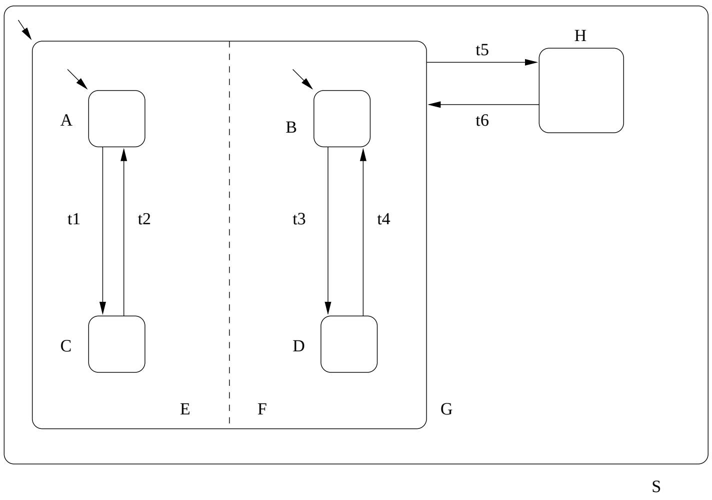

Fig. 1. A statechart

It is important to stress that other semantics of statecharts, such as the ones of [14,25], can be obtained by taking in the translation different basic actions describing transitions and different definitions of parallel composition of actions and of conflict. Moreover, the interest of the language $\mathcal{SP}$ is not limited to statecharts and other classical process calculi, such as SCCS [26] and CBS (Calculus of Broadcast Systems) [28], can be embedded in it.

In the second part of the paper we define a compositional proof system for checking μ-calculus properties of $\mathcal{SP}$ processes. The proof system is defined in the style of [3] and combines together local and compositional reasoning. Rules are of two types:

- logical rules, which work on the structure of the formula. Fixpoint formulas are treated by the method of *tagged* fixpoints of [34];
- compositional rules on the structure of the process for the *next* modalities.

By the combination of these two sets of rules the construction of proofs is completely driven by the syntax of the formula and of the process without reference to the underlying transition system. The proof system is proved to be sound in general and complete for the class of finite-state closed processes, i.e. also for processes modelling statecharts.

The paper is organized as follows. In section 2 statecharts are introduced. Section 3 presents the language $\mathcal{SP}$ and its operational semantics. Section 4 shows the embedding of statecharts in $\mathcal{SP}$ and section 5 shows an example of statechart processes. Propositional μ-calculus is described in section 6. Section 7 and 8 present the compositional proof system. Finally, section 10 contains an example of verification.

2 Introduction to Statecharts

In this section we introduce statecharts and briefly discuss the main features of the semantics proposed in [27].

A statechart specifies a reactive system, which continuously interacts with its environment by receiving and sending signals of a finite set Π of primitive events. We assume $nil \in \Pi$, where nil denotes the null event, and we denote by $\bar{\Pi} = \{\bar{e} \mid e \in \Pi\}$, where $\bar{e}$ denotes the negation of event e. For $e = \bar{e}' \in \bar{\Pi}$, we denote by $\bar{e}$ its negation, i.e. the event e'.

Definition 1 *A* statechart S *over* Π *is defined as follows:*

- S *is* $S = (\{S_1, \dots, S_k\}, T, in, out, \chi, \delta)$ *a* basic statechart *over* Π*, where*
  - $\cdot$ $\{S_1, \dots, S_k\}$ *is a set of boxes, with* $boxes(S) = \{S\} \bigcup_{i \in \{1,k\}} S_i$.
  - $\cdot$ T *is a finite not empty set of transitions.*
  - $\cdot$ $in : T \to \{S_1, \dots, S_k\}$, $out : T \to \{S_1, \dots, S_k\}$.
  - $\cdot$ $\chi : T \to 2^{\Pi \cup \bar{\Pi}} \times 2^{\Pi}$ *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* $e, \bar{e} \notin trigger(t) \cup action(t)$.
  - $\cdot$ $\delta : S \to \{S_1, \dots S_k\}$ *is the default function giving the default substate.*

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

- S *is a* refined statechart $\bigtriangledown(S, \{S_1, \dots, S_k\})$ *over* Π *iff*
  - $\cdot$ S *is a basic non-empty statechart, where* $boxes(S) = \{S\} \bigcup_{i \in \{1,k\}} \{S_i\}$.
  - $\cdot$ $\{S_1, \dots, S_k\}$ *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* $boxes(S) = \{S\} \bigcup_{i \in \{1,k\}} boxes(S_i)$.

- S *is an AND* statechart $S = AND(S_1, S_2)$ *over* Π *iff*
  - $\cdot$ S_1, S_2 *are a non-empty statecharts over* Π*, and* $boxes(S) = \{S\} \bigcup_{i \in \{1,2\}} boxes(S_i)$*, 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* $i \neq j$.

Consider the statechart of figure 1, where transitions are labelled as follows

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

Statecharts E and F are basic statecharts, while G is given by the parallel composition of E and F. Statechart of figure 1 is a refined statechart $\bigtriangledown(S, \{G, H\})$, where S is a basic statechart with boxes G and H. As showed by the picture, state G is refined by AND(E, F), while state H is refined by the empty statechart. Transitions are depicted by arrows and are labelled by a pair $trigger/action$. Intuitively, the trigger express an enabling condition on input events, while action gives the set of produced events. For instance, transition t_1 requires event a from the environment to be enabled and produces event b. Note that the trigger may contain negated events, as the trigger of t_6. The condition $\bar{d}$ has to be interpreted as 'signal d must be absent'. Negated events are essential for expressing priorities and interrupts. The default substate is represented by an arrow without source, such as state A in statechart E. The default state is entered whenever the refined state is entered.

Given $S_1, S_2 \in boxes(S)$, we denote by $LCA(S_1, S_2)$ 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 $S_1, S_2 \in boxes(S'')$, $S' \in boxes(S'')$. Note that, for each pair S_1, S_2, $LCA(S_1, S_2)$ is always defined, since the hierarchy of boxes has a tree-like structure.

A configuration (global state) of a statechart is given by a maximal set of boxes, which are in parallel components.

Definition 2 *Let S be a statechart. A set of states $c \subseteq boxes(S)$ is a* configuration *iff, for each $S_1, S_2 \in c$, $LCA(S_1, S_2)$ is an AND statechart, and c is maximal.*

In the following, we denote by $C(S)$ the set of configurations of statechart S. A configuration $c \in C(S)$ is called the *default configuration* iff, for each $S' inboxes(S)$, such that S' is a basic statechart or a refined statechart, $\delta(S') \in c$. We denote by $\Delta(S)$ the default configuration of S, namely the initial configuration.

2.1 The semantics of Pnueli and Shalev

The semantics of statecharts is based on the synchrony hypothesis: the reaction of the system to inputs is instantaneous. This is indeed an abstraction, which is justified by the assumption that the environment can be described as a discrete process and that the system is infinitely faster than the environment. The main advantage is that the behaviour of a statechart can be defined as the (infinite) sequence of reactions (steps) to enviroment inputs without reference to time. The main problem is to define a suitable notion of step taking the system from a configuration to another. In the approach of Pnueli and Shalev [27], a step satisfies some basic requirements, which can be summarized as the *synchrony hypothesis* (maximality) and the principles of *causality* and *global consistency*.

By synchrony hypothesis all the events generated by transitions taken in a step are simultaneous with inputs. Consider the statechart of figure 1. If the current configuration is $\{S, G, E, F, A, B\}$ and, if the environment produces event a, both transitions t_1 and t_3 performed. In fact, t_1 is enabled with respect to the input and generates event b, which enables t_3. By maximality, since transition t_3 is in parallel with t_1, then it must be taken. The reached configuration is $\{S, G, E, F, C, D\}$. On the other hand, if the environment produces d, then $\{t_5\}$ is the only possible step and the reached configuration is $\{S, H\}$, since state G is left. If the environment produces both a and d, then both $\{t_1, t_3\}$ and $\{t_5\}$ are steps and there is a non deterministic choice. Note that, if the environment does not produce a, b and d, then the empty step is the only step from $\{S, G, E, F, A, B\}$.

The principle of causality requires the existence of a causal ordering between transitions taken in a step. Consider the statechart G of figure 1, where transitions t_1 and t_3 are labelled as a/b and b/a, respectively. Suppose that the current configuration is $\{G, E, F, A, B\}$ and that the environment does not communicate both a and b. One could think that both transitions are performed, since they are parallel and they are enabled with respect to the whole set of generated events $\{a, b\}$. In contrast, in this approach the only possible step is the empty one, since each transition depends on the other.

Global consistency requires transitions of a step to be enabled with respect to the global set of produced events. Consider statechart G, where transitions t_1 and t_3, are labelled as $\bar{a}/b$ and b/a, respectively. If the current configuration is $\{G, E, F, A, B\}$ and if the environment does not produce both a and b, then there is no possible step. The problem is that transition t_1 is enabled with respect to the input and generates event b, which enables transition t_3. By maximality, transition t_3 must be performed and generates instantaneously event a, which disables t_1. Therefore, the performance of t_1 causes the disabling of its own causes. This situation well-known as a causal paradox can be seen in this semantics as a run-time error. Other interpretations of causal paradoxes have been suggested in other semantics.

All these requirements are captured by taking as a step 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 transitions in T are taken.

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

Definition 3 *Let S be a statechart and $I \subseteq \Pi$ be a set of events. We define*

- *For $t_1, t_2 \in T(S)$, t_1 is* consistent *with t_2 iff $t_1 \in T(S_1), t_1 \in T(S_1)$ with $S_1, S_2 \in boxes(S)$, such that $LCA(S_1, S_2)$ is an AND statechart.*
- *For $T \subseteq T(S)$, $con(T) = \{t \in T(S) \mid t \text{ is consistent with } t' \text{ for all } t' \in T\}$.*
- *For $c \in C(S)$, $rel(c) = \{t \in T(S) \mid out(t) \in c\}$.*
- *$trig(I) = \{t \in T(S) \mid \forall e \in trigger(t), e \in I \text{ and } \forall \bar{e} \in trigger(t), e \notin I\}$.*

- *$En_{c,I}(T) = rel(c) \cap con(T) \cap trig(I \cup gen(T))$, where $gen(T) = \bigcup_{t \in T}$ $action(t)$.*

Definition 4 *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 $c \overset{I,O}{\Longrightarrow} c'$ to denote a step from configuration c to configuration c' on input I which produces output O.

Let us briefly explain the definition. Maximality of steps is guaranteed by $En_{c,I}(T) \subseteq T$, which ensures that all enabled transitions are taken. Note that a transition t is enabled iff its source state is in the configuration ($t \in rel(C)$) and if its trigger is satisfied by the set of all produced events ($t \in trig(I)$). The condition of inseparability guarantees the existence of a causal ordering between transitions. Consider the statechart G of figure 1, where transitions t_1 and t_3 are labelled as a/b and b/a, respectively. Suppose that the current configuration is $c = \{G, E, F, A, B\}$ and that the environment does not communicate both a and b. We have $En_{c,\emptyset}(\{t_1, t_3\}) = \{t_1, t_3\}$. However, $\{t_1, t_3\}$ is separable, since $En_{c,\emptyset}(\emptyset) \cap \{t_1, t_3\} = \emptyset$. Thus, the only admissible step is the empty one. Global consistency is achieved by requiring $T \subseteq En_{c,I}(T)$. Consider statechart G, where transitions t_1 and t_3, are labelled as $\bar{a}/b$ and b/a, respectively. Suppose that the current configuration is c and that the environment does not produce both a and b. There is no admissible step, since each set of transitions does not fullfil the requirements. Note that $\{t_1, t_3\}$ is not a step since $En_{c,\emptyset}(\{t_1, t_3\}) = \{t_3\}$, namely t_3 disables t_1.

It is important to stress that empty steps are admissible steps. In contrast, a non-defined reaction denotes an anomalous situation, i.e. a causal paradox.

3 The language $\mathcal{SP}$

In this section we introduce the process language $\mathcal{SP}$. We assume a set Act of basic actions, a set Loc of locations and a set of Var of variables.

Definition 5 *We define processes $\mathcal{SP}$ as follows:*

$$b ::= nil \mid a.t \mid b_1 + b_2$$

$$t ::= \ell :: b \mid rec\, x.t \mid t \bigtriangledown p \mid x$$

$$p ::= \ell :: t \mid \ell :: p_1 \times p_2$$

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, i.e. a tree-like structure. 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. Intuitively, in the statecharts framework 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. This new operator $\bigtriangledown$ is characteristic of $\mathcal{SP}$ and is used to represent statecharts hierarchy.

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.

The labelled transition system semantics of $\mathcal{SP}$ processes is based on two operations over actions: an operation of *parallel composition* $\times : Act \times Act \to 2^{Act}$ and an operation of *conflict* $\# : Act \cup \{*\} \to 2^{Act}$. The parallel composition is used to encode the communication among parallel components. The underlying idea of conflict is that the enabling of actions depends on the environment inputs. So, an action $a' \in \#(a)$ is in conflict with action a if a' is enabled with respect to sets of inputs, that do not enable a. For instance, the action " signal a is absent" is in conflict with the action " signal a is received". The action $\{*\}$ is used to give the semantics of the empty process.

Moreover, we assume that $\times$ is associative, namely, for each $a_1, a_2, a_3 \in Act$, $a_1 \times (a_2 \times a_3) = (a_1 \times a_2) \times a_3$.

Assume $p \in \mathcal{SP}$, we define the set of *current locations*, $loc(p)$ as

- $loc(\ell :: t) = \{\ell\} \cup loc(t)$;
- $loc(\ell :: p_1 \times p_2) = \{\ell\} \cup loc(p_1) \cup loc(p_2)$;
- $loc(\ell :: b) = \{\ell\}$;
- $loc(rec\, x.t) = loc(t)$;
- $loc(t \bigtriangledown p) = loc(t) \cup loc(p)$.

Intuitively, for statecarts processes $loc(p)$ gives the current configuration.

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

Definition 6 *Let* Act *be a set of basic actions. We define* $\mathcal{S}(\text{Act})$ *as the set of* semantic *actions* $a : \tau$*, such that* $a \in \text{Act}$ *and* $\tau \in \{\epsilon, \bar{\epsilon}\}$*. For* $\alpha = a : \tau \in \mathcal{S}(\text{Act})$*,* $type(\alpha) = \tau$*.*

Two types of actions are considered: *non-idle* (ϵ) and idle ($\bar{\epsilon}$). A non-idle action represents a real action of the process. In contrast, an idle action $a : \bar{\epsilon}$ represents the fact that the process is idle on a, i.e. it is idle with respect to the environment inputs, which enable action a. In the statecharts framework non-idle actions will correspond to non-empty steps, while idle actions will correspond to empty steps.

The operation $\times$ can be extended to semantic actions $\mathcal{S}(Act)$ as follows.

Definition 7 *Assume* $\alpha_i = a_i : \tau_i \in \mathcal{S}(\text{Act})$*, for* $i \in \{1, 2\}$*. We define* $\alpha_1 \times \alpha_2$ *as the set* $\{a : \tau \in \mathcal{S}(\text{Act})\}$*, such that* $a \in a_1 \times a_2$ *and* $\tau = \epsilon$ *if* $\tau_i = \epsilon$*, for some* $i \in \{1, 2\}$*, and* $\tau = \bar{\epsilon}$ *otherwise.*

Note that an action obtained by the parallel composition is idle iff both the actions are idle.

The transition relation is obtained by the following rules:

$$\ell_1 :: \ell_2 :: a.t \overset{a:\epsilon}{\mapsto} \ell_1 :: t \qquad\qquad \ell_1 :: \ell_2 :: a.t \overset{b:\bar{\epsilon}}{\mapsto} \ell_1 :: \ell_2 :: a.t \quad b \in \#(a)$$

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

$$\frac{\begin{array}{c}\ell_1 :: \ell_2 :: b_i \overset{\alpha}{\mapsto} p \\ type(\alpha) = \epsilon\end{array}}{\ell_1 :: \ell_2 :: b_1 + b_2 \overset{\alpha}{\mapsto} p}\, choice_1 \qquad \frac{\begin{array}{c}\{\ell_1 :: \ell_2 :: b_i \overset{\alpha}{\mapsto} \ell_1 :: \ell_2 :: b_i\}_{i\in\{1,2\}} \\ type(\alpha) = \bar{\epsilon}\end{array}}{\ell_1 :: \ell_2 :: b_1 + b_2 \overset{\alpha}{\mapsto} \ell_1 :: \ell_2 :: b_1 + b_2}\, choice_2$$

$$\frac{\ell :: t \overset{\alpha}{\mapsto} p' \;\; type(\alpha) = \epsilon}{\ell :: t \bigtriangledown p \overset{\alpha}{\mapsto} p'}\, \bigtriangledown_1 \qquad \frac{p \overset{\alpha}{\mapsto} p' \;\; type(\alpha) = \epsilon}{\ell :: t \bigtriangledown p \overset{\alpha}{\mapsto} \ell :: t \bigtriangledown p'}\, \bigtriangledown_2$$

$$\frac{\begin{array}{c}\ell :: t \overset{\alpha}{\mapsto} \ell :: t \\ p \overset{\alpha}{\mapsto} p \\ type(\alpha) = \bar{\epsilon}\end{array}}{\ell :: t \bigtriangledown p \overset{\alpha}{\mapsto} \ell :: t \bigtriangledown p}\, \bigtriangledown_3$$

$$\frac{\ell :: t[rec\,x.t/x] \overset{\alpha}{\mapsto} p}{\ell :: rec\,x.t \overset{\alpha}{\mapsto} p}\,rec \qquad \frac{\{p_i \overset{\alpha_i}{\mapsto} p_i'\}_{i\in\{1,2\}} \;\; \alpha \in \alpha_1 \times \alpha_2}{\ell :: p_1 \times p_2 \overset{\alpha}{\mapsto} \ell :: p_1' \times p_2'}\,par$$

Let us explain the rules. In order to capture the maximality of reaction of statecharts the underlying idea of the semantics is that idle actions are taken iff no non-idle action is possible because of the behaviour of the environment. Consider a process $\ell_1 :: \ell_2 :: a.t$. If the environment enables action a then $\ell_1 :: \ell_2 :: a.t$ evolves into $\ell_1 :: t$ by performing an action $a : \epsilon$ (non-idle) and location ℓ_2 is left. In contrast, if action a is not enabled by the environment behaviour, then the process is idle. This is represented by transitions labelled by $a' : \bar{\epsilon}$, where a' is in conflict with a. The reached process is $\ell_1 :: \ell_2 :: a.t$. Actions of type idle and non-idle are treated differently in the operators of refinement and of choice. Consider a process $\ell_1 :: \ell_2 :: b_1 + b_2$. Rule $choice_1$ is the classical rule for choice and it is applicable iff the action is non-idle. On the other hand, by rule $choice_2$, b_1 and b_2 must agree on idle actions. This guarantees maximal reaction. In a similar way the semantics of $\ell :: t \bigtriangledown p$ is defined. Idle actions of $\ell :: t \bigtriangledown p$ must be idle actions of both $\ell :: t$ and p as required by $\bigtriangledown_3$. The main idea of process refinement is reflected by the difference between rule $\bigtriangledown_1$ and $\bigtriangledown_2$. When a non-idle action of p is performed, the resulting process is $\ell :: t \bigtriangledown p'$. In contrast, when a non-idle action of $\ell :: t$ is performed the resulting process is p'. This behaviour captures the hierarchy of statecharts boxes. The rule of parallel composition is the standard rule for synchronous languages, i.e. both processes must take an action. The resulting action is obtained by the parallel composition of actions $\times$, which intuitively realizes the communication among parallel components.

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 subterms $\ell :: p_1 \times p_2$ and in process p_1, such that $\ell :: t \bigtriangledown p_1$ is a subterm of p.

Definition 8 *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 $p' \overset{\alpha}{\mapsto} p''$, then $p'' \in \mathcal{R}_p$.*

For a regular process $p \in \mathcal{CSP}$, $\mathcal{R}_p$ is finite.

4 From statecharts to $\mathcal{SP}$ processes

In this section we show as $\mathcal{SP}$ can be instantiated to statecharts. We define a translation α from statecharts into closed processes, such that the semantics of $\alpha(S)$ is equivalent to the Pnueli and Shalev semantics of statechart S.

The idea is that of constructing in a compositional way on the syntax of the statechart a corresponding process where basic actions correspond to transitions and locations to $boxes(S)$. Basic actions are required to describe the input-output behaviour associated to transitions. The main problem is that the semantics of Pnueli and Shalev is not modular, so that the input-output is not sufficient for compositionality. In order to solve this problem we consider actions representing both the pair input-output and the causal ordering between events.

We consider basic actions with the following syntax.

Definition 9 *Let Π 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.* $e, \bar{e} \notin \ell$;
(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* $C[C''/C'] \prec C$ *(transitivity).*

The ordering represents the causality relation between inputs and outputs. Thus, $\{e, e''\} \prec \{e'\}$ means that both e and e'' are necessary for event e' to be produced. In contrast, $\{e\} \prec \{e'\}$ and $\{e''\} \prec \{e'\}$ means that both e and e'' are sufficient for event e' to be produced. By transitivity of the ordering $\{e\} \prec \{e'\}$ and $\{e'\} \prec \{e''\}$ implies $\{e\} \prec \{e''\}$. So, event e is sufficient for generating both e' and e''. The requirement of irreflexivity guarantees that there are not causal cycles. Moreover, the whole set of events is required to be consistent for ensuring global consistency.

Definition 10 *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* $action(\sigma) = \ell \setminus trigger((\ell, \leq))$.

Intuitively, the trigger of σ is the set of events required either to be present or absent in the input for enabling the action, while its action is the set of events produced by the action. Transitions are represented by basic actions with the same input-output.

Definition 11 *Let S be a statechart. For $t \in T(S)$,* $lab(t) = (trigger(t) \cup action(t), \{trigger(t) \prec \{e'\} \mid e' \in action(t) - \{nil, trigger(t)\}\})$.

For instance, a transition with label $a, \bar{c}/d$ will be represented by the action $(\{a, \bar{c}, d\}, \{a, \bar{c}\} \prec \{d\})$, where it is explicitely represented that the presence of a and the absence of c are needed for the enabling of the action and that its performance produces d.

The operation $\times$ of parallel composition is quite complex. Roughly speaking, $\sigma \in \sigma_1 \times \sigma_2$ corresponds to the step given by the union of the steps represented by σ_1 and σ_2. Hence, the input-output of σ must be equivalent to the one of such a step, where the communication among parallel components is realized. Moreover, the conditions of causality and consistency must be checked.

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 12 *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 $(\ell, \prec) \in \Sigma(\Pi)$ and $\prec$ 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.

Consider $\sigma_1 = (\{a, b\}, \{a\} \prec_1 \{b\})$ and $\sigma_2 = (\{b, c\}, \{b\} \prec_2 \{c\})$, which can be seen as actions describing two parallel transitions with labels a/b and b/c, respectively. We have $\sigma_1 \times \sigma_2 = \{\sigma\}$ with $\sigma = (\{a, b, c\}, \{a\} \prec \{b\}, \{b\} \prec \{c\}, \{a\} \prec \{c\})$. The communication due to parallel composition is suitably realized by the transitive closure of orderings. In fact, the input-output of σ, $trigger(\sigma) = \{a\}$ and $action(\sigma) = \{b, c\}$, correctly models the step the step given by the transitions represented by σ_1 and σ_2. in fact, event a is sufficient to trigger both transitions, because of the internal production of event b.

Note that $(\prec_1 \cup \prec_2)^+$ may be reflexive. Consider $\sigma_1 = (\{a, b\}, \{a\} \prec_1 \{b\})$ and $\sigma_2 = (\{a, b\}, \{b\} \prec_2 \{a\})$, where the production of b depends on the receiving of a and the production of a depends on the receiving of b, respectively. The parallel composition of σ_1 and σ_2 cannot be $\sigma = (\{a, b\}, \{a\} \prec \{b\}, \{b\} \prec \{a\})$, because of the causality principle. Actually, $trigger(\sigma) = \emptyset$ implies that both transitions are performed with respect to the empty input. In contrast, by the requirement of irreflexivity, we have $\sigma_1 \times \sigma_2 = \{\sigma_1, \sigma_2\}$ so that either event a or event b is needed from the environment to trigger both transitions.

The operation of conflict is defined as follows.

Definition 13 *For $\sigma \in \Sigma(\Pi)$, 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 set of inputs $I \subseteq \Pi$, I enables σ' only if I does not enable σ.

Consider for instance an action $\sigma = (\{a,b,c\}, \{a,b\} \prec \{c\})$. We have $\#(\sigma) = \{(\{\bar{a},b,c\},\emptyset), (\{\bar{b},a,c\},\emptyset), (\{\bar{a},b,\bar{c}\},\emptyset), (\{\bar{b},a,\bar{c}\},\emptyset), (\{\bar{a},\bar{b},\bar{c}\},\emptyset), (\{\bar{a},\bar{b},c\},\emptyset)\}$. Action σ cannot be performed iff the environment does not produce both a and b. The actions $\#(\sigma)$ represent exactly these conflicting behaviours of the environment.

The translation $\alpha(S)$ is inductively defined over the structure of statechart S.

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

Definition 14 *The translation α is defined 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 $T_i = \{t \in T \mid in(t) = S_i\}$. Let $t_{S_i} = rec\, s_i.S_i :: (\sum_{t\in T_i} lab(t).out(t))$, if $T_i \neq \emptyset$ and $t_{S_i} = S_i :: nil$, 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 $S_j \in free(\alpha(S))$, $\alpha(S) = \alpha(S)[t_{S_j}/S_j]$.*
- *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 $\alpha(S) = S :: \alpha(S_1) \times \alpha(S_2)$.*

Note that, for each statechart S, $\alpha(S)$ is a regular process.

The definitions of $lab(t)$, of $\times$ and $\#$ are defined 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 ($I \leadsto \ell$) 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 15 *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. 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. It must be shown that the condition of inseparability is captured by the definition of $\times$ over $\Sigma(\Pi)$. The complete proof can be found in [24].

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 [14,25].

5 An example

In this section, we explain the translation by considering the statechart of figure 1, where transitions are labelled as

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

For basic statecharts E and F, we have $\alpha(E) = E :: t_1$, $\alpha(F) = F :: t_2$ where

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

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

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, $\alpha(E) \stackrel{\sigma:\bar{\epsilon}}{\mapsto} \alpha(E)$ for $\sigma \in \#(lab(t_1))$, where $\bar{a} \in trigger(\sigma)$. These transitions correspond to the empty steps from $\{E, A\}$ with respect to inputs that do not contain a.

Since G : AND(E, F), then $\alpha(G) = G :: \alpha(E) \times \alpha(F)$. Maximality, causality and consistency of steps are ensured by the definition of $\times$ over actions. By rule *par*, $\alpha(G) \overset{\sigma:\epsilon}{\mapsto} G :: p_1 \times p_2$ where $p_1 = E :: rec\, y.C :: lab(t_2).t_1$, $p_2 = F :: rec\, y.D :: lab(t_4).t_2$ and $\sigma = (\{a, b, c\}, \{a\} \prec \{b\} \prec \{c\})$. In fact, $\sigma \in lab(t_1) \times lab(t_3)$.

This transition corresponds to the step $\{t_1, t_3\}$ from $\{G, E, F, A, B\}$ to the configuration $\{G, E, F, C, D\}$, when the environment produces a. Actually, $trigger(\sigma) = \{a\}$ and $action(\sigma) = \{b, c\}$.

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 $\alpha(E) \overset{(\{\bar{a},b,\bar{d},c\},\emptyset):\bar{\epsilon}}{\mapsto} \alpha(E)$ and $\alpha(F) \overset{(\{b,c\},\{b\}\prec\{c\}):\epsilon}{\mapsto} p_2$, we have

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

This transition corresponds to the non-empty step $\{t_3\}$, when the environment produces b and does not produce both a and d.

Note that there are no steps, such that E takes a transition and F does not take a transition, since transition t_1 triggers transition t_3. In fact, if we combine the only non-empty step of E, $\alpha(E) \overset{lab(t_1):\epsilon}{\mapsto} E :: rec\, y.C :: lab(t_2).t_1$, with an empty step of F, $\alpha(F) \overset{\sigma:\bar{\epsilon}}{\mapsto} \alpha(F)$ for $\sigma \in \#(lab(t_3))$, we have $lab(t_1) : \epsilon \times \sigma : \bar{\epsilon} = \emptyset$ by global consistency. The problem is that $b \in action(lab(t_1))$ and $\bar{b} \in trigger(\sigma)$.

Finally, we have the following empty steps $\alpha(G) \overset{\sigma:\bar{\epsilon}}{\mapsto} \alpha(G)$ for $\sigma \in \#(lab(t_1)) \cap \#(lab(t_3))$, $\sigma \in \{(\{\bar{a}, \bar{b}, d, c\}, \emptyset), (\{\bar{a}, \bar{b}, \bar{d}, c\}, \emptyset), (\{\bar{a}, \bar{b}, d, \bar{c}\}, \emptyset), (\{\bar{a}, \bar{b}, \bar{d}, \bar{c}\}, \emptyset)\}$.

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

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

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

Since $\alpha(G) \overset{\sigma:\epsilon}{\mapsto} G :: p_1 \times p_2$, by rule *rec* and by rule $\bigtriangledown_2$,

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

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.

As it is clear from this example rules $\bigtriangledown_1$ and $\bigtriangledown_2$ for process refinement suitably model the corresponding operator of statecharts, which constructs the hierarchy. 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. Therefore, we have an empty step iff the environment produce neither a, b nor d. This is captured by rule $\bigtriangledown_3$, where the two parts are required to agree on idle actions.

Since $\alpha(G) \overset{\sigma:\bar{\epsilon}}{\mapsto} \alpha(G)$ and $S :: t_4 \overset{\sigma:\bar{\epsilon}}{\mapsto} S :: t_4$, then $S :: t_4 \bigtriangledown \alpha(G) \overset{\sigma:\bar{\epsilon}}{\mapsto} S :: t_4 \bigtriangledown \alpha(G)$, for $\sigma \in \#(lab(t_1)) \cap \#(lab(t_3)) \cap \#(lab(t_5))$.

By rule rec, $\alpha(\bigtriangledown(S, \{G, H\}) \overset{\sigma:\bar{\epsilon}}{\mapsto} \alpha(\bigtriangledown(S, \{G, H\})$.

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

6 The logic

In this section we introduce the logic for expressing temporal properties of $\mathcal{SP}$ processes. We consider formulas of propositional μ-calculus given in a negation free syntax and with tagged fixpoints [34]. The set of formulas of L_μ is defined as

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

where X ranges over propositional variables VAR, P over propositional symbols $PROP$, α over a set of actions $\mathcal{A}$ and $U \subseteq \mathcal{S}$, where $\mathcal{S}$ is a set of states called the *tag*.

Modality $< \alpha >$ is the classical existential next ranging over actions $\mathcal{A}$, while $[\alpha]$ is it dual. We denote by $\nu X.A$ and $\mu X.A$ the formulas $\nu X\{\emptyset\}.A$ and $\mu X\{\emptyset\}.A$, which stand respectively for greatest and least fixpoints of the function represented by A. Moreover, we use the constants $true$ and $false$ for $\nu X.X$ and $\mu X.X$, respectively.

Formulas over VAR, $PROP$, $\mathcal{A}$ and $\mathcal{S}$ are interpreted over a model $\mathcal{M} = ((\mathcal{S}, \mathcal{A}, \mapsto), \rho)$, where $(\mathcal{S}, \mathcal{A}, \mapsto)$ is a labelled transition system and ρ is a valuation assigning subsets of $\mathcal{S}$ to variables VAR and propositional symbols $PROP$.

Given a formula A the set of states of $\mathcal{M}$ satisfying A is defined by $\| A \|_\rho^{\mathcal{M}}$, for short $\| A \|_\rho$:

$$
\begin{aligned}
&\| P \|_\rho = \rho(P) \\
&\| \neg P \|_\rho = \mathcal{S} \setminus \rho(P) \\
&\| X \|_\rho = \rho(X) \\
&\| A_0 \vee A_1 \|_\rho = \| A_0 \|_\rho \cup \| A_1 \|_\rho \\
&\| A_0 \wedge A_1 \|_\rho = \| A_0 \|_\rho \cap \| A_1 \|_\rho \\
&\| \nu X\{U\}.A \|_\rho = \nu V.(\| A \|_{\rho[V/X]} \cup U) \\
&\| \mu X\{U\}.A \|_\rho = \mu V.(\| A \|_{\rho[V/X]} \setminus U) \\
&\| < \alpha > A \|_\rho = \{ s \in \mathcal{S} \mid \exists s' \in \mathcal{S}, s \overset{\alpha}{\mapsto} s', s' \in \| A \|_\rho \} \\
&\| [\alpha] A \|_\rho = \{ s \in \mathcal{S} \mid \forall s' \in \mathcal{S}, s \overset{\alpha}{\mapsto} s' \Longrightarrow s' \in \| A \|_\rho \}
\end{aligned}
$$

where $\mu V.\phi(V)$ and $\nu V.\phi(V)$ denotes the least and greatest fixpoint of the monotonic function $\phi(V)$ on the lattice $(\mathcal{P}(\mathcal{S}), \subseteq)$. For an explanation on the use of tags in fixpoints we refer the reader to [34].

Given a model $\mathcal{M} = ((\mathcal{S}, \mathcal{A}, \mapsto), \rho)$, $s \in \mathcal{S}$ and a formula A, we will say that $s \models_{\mathcal{M}} A$ iff $s \in \| A \|_\rho^{\mathcal{M}}$ for each valuation ρ. Note that for closed formulas valuations are not significant.

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.

Properties of process $p \in \mathcal{CSP}$ over Act and Loc, can be expressed as closed μ-calculus assertions over actions $\mathcal{S}(Act)$ and propositional symbols 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 $\gamma : Loc \rightarrow 2^{\mathcal{CSP}}$, such that $P \in \gamma(p)$, iff $P \in loc(p)$.

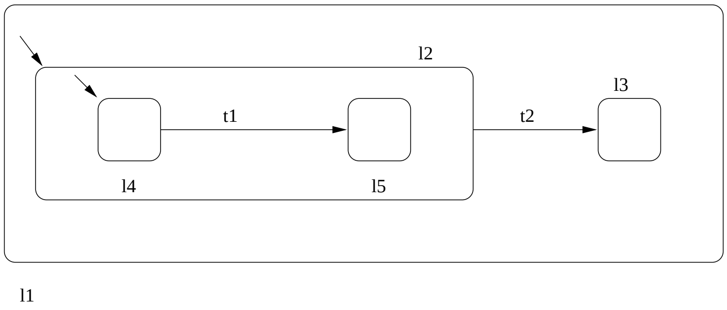

Fig. 2. An example of refinement

7 An introduction to the proof system

In this section 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. There are two types of rules: 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 $p = \ell_2 :: \ell_4 :: a.\ell_5 :: nil$, which models the basic statechart ℓ_2 of figure 2, for $a = lab(t_1)$. Suppose that $c \in \#(a)$. We have that $p \models < a : \epsilon > \ell_5$ and $p \models < c : \bar{\epsilon} > \ell_4$, since $p \overset{a:\epsilon}{\mapsto} \ell_2 :: \ell_5 :: nil$ and $p \overset{c:\bar{\epsilon}}{\mapsto} p$. Since p is a basic process not further decomposable, we directly obtain its transitions from the syntax. Analogously, we can deduce $p \vdash < a : \epsilon > \ell_5$ from $\ell_2 :: \ell_5 :: nil \vdash \ell_5$ and $p \vdash < c : \bar{\epsilon} > \ell_4$ from $p \vdash \ell_4$. Moreover, $\ell_2 :: \ell_5 :: nil \vdash \ell_5$ and $p \vdash \ell_4$ trivially hold, since $\ell_5 \in loc(\ell_2 :: \ell_5 :: nil)$ and $\ell_4 \in loc(p)$, respectively.

Consider the process $p' = \ell_1 :: t \bigtriangledown p$ with $t = \ell_2 :: b.\ell_3 :: nil$, which models the refined statechart of figure 2 for $b = lab(t_2)$. Suppose $c \in \#(a) \cap \#(b)$. We have that $p' \models < a : \epsilon > \ell_2 \wedge \ell_5$, $p' \models < b : \epsilon > \ell_3 \wedge \neg \ell_2$ and $p' \models < c : \bar{\epsilon} > \ell_2 \wedge \ell_4$. In fact, in the labelled transition system, $p' \overset{a:\epsilon}{\mapsto} \ell_1 :: t \bigtriangledown \ell_2 :: \ell_5 :: nil$, $p' \overset{b:\epsilon}{\mapsto} \ell_1 :: \ell_3 :: nil$ and $p' \overset{c:\bar{\epsilon}}{\mapsto} p'$.

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 $p' \overset{b:\epsilon}{\mapsto} \ell_1 :: \ell_3 :: nil$ and $\ell_1 :: \ell_3 :: nil \models \ell_3 \wedge \neg \ell_2$. Since $type(b : \epsilon) = \epsilon$, this transition is obtained by rule $\bigtriangledown_1$, from $\ell_1 :: t \overset{b:\epsilon}{\mapsto} \ell_1 :: \ell_3 :: nil$. Therefore, $\ell_1 :: t \models < b : \epsilon > \ell_3 \wedge \neg \ell_2$.

This reasoning suggests a compositional proof as the following.

$$
\begin{array}{c}
p' \vdash < b : \epsilon > \ell_3 \wedge \neg \ell_2 \\
\hline
\ell_1 :: t \vdash < b : \epsilon > \ell_3 \wedge \neg \ell_2 \\
\hline
\ell_1 :: t \vdash \ell_3 \wedge \neg \ell_2 \\
\hline
\begin{array}{cc}
\ell_1 :: \ell_3 :: nil \vdash \ell_3 & \ell_1 :: \ell_3 :: nil \vdash \neg \ell_2 \\
\hline
\end{array}
\end{array}
$$

On the other hand, consider the formula $< a : \epsilon > \ell_2 \wedge \ell_5$. We have $p' \models < a : \epsilon > \ell_1 \wedge \ell_5$, since $p' \overset{a:\epsilon}{\mapsto} \ell_1 :: t \bigtriangledown \ell_2 :: \ell_5 :: nil$ and $\ell_1 :: t \bigtriangledown \ell_2 :: \ell_5 :: nil \models \ell_1 \wedge \ell_5$. Since $type(a : \epsilon) = \epsilon$, this transition is obtained by rule $\bigtriangledown_2$ from $p \overset{a:\epsilon}{\mapsto} \ell_2 :: \ell_5 :: nil$. However, we cannot prove this property by repeating the same reasoning as before, since the operator $\bigtriangledown$ is not commutative. Therefore, it would not be correct to deduce $\ell_1 :: t \bigtriangledown p \vdash < a : \epsilon > \ell_2 \wedge \ell_5$ from $p \vdash < a : \epsilon > \ell_2 \wedge \ell_5$. These problems can be solved by adopting special formulas, called *extended formulas*, which have been introduced by [3] for treating parallel composition. We use formulas $(A)_{\ell::t\bigtriangledown}$ with the following meaning: $p \models (A)_{\ell::t\bigtriangledown}$ iff $\ell :: t \bigtriangledown p \models A$. So, we can deduce $p' \vdash < a : \epsilon > \ell_2 \wedge \ell_5$ from $p \vdash < a : \epsilon > (\ell_2 \wedge \ell_5)_{\ell_1::t\bigtriangledown}$.

In this way, the problem of checking $< a : \epsilon > \ell_2 \wedge \ell_5$ for process $\ell_1 :: t \bigtriangledown p$ 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.

$$
\begin{array}{cl}
\ell_1 :: t \bigtriangledown p \vdash < a : \epsilon > \ell_2 \wedge \ell_5 & \\
\hline
p \vdash < a : \epsilon > (\ell_2 \wedge \ell_5)_{\ell_1::t\bigtriangledown} & \\
\hline
\ell_2 :: \ell_5 :: nil \vdash (\ell_2 \wedge \ell_5)_{\ell_1::t\bigtriangledown} & shift \\
\hline
\ell_1 :: t \bigtriangledown \ell_2 :: \ell_5 :: nil \vdash \ell_2 \wedge \ell_5 & \\
\hline
\cdots & \\
\hline
\end{array}
$$

Consider the formula $< c : \bar{\epsilon} > \ell_2 \wedge \ell_4$. We have $p' \models < c : \bar{\epsilon} > \ell_2 \wedge \ell_4$, since $p' \overset{c:\bar{\epsilon}}{\mapsto} p'$ and $p' \models \ell_2 \wedge \ell_4$. The transition has been obtained from $\ell_1 :: t \overset{c:\bar{\epsilon}}{\mapsto} \ell_1 :: t$, $p \overset{c:\bar{\epsilon}}{\mapsto} p$ by rule $\bigtriangledown_3$, since $type(c : \bar{\epsilon}) = \bar{\epsilon}$. This construction 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 $p \overset{\alpha}{\mapsto} p''$ with $type(\alpha) = \bar{\epsilon}$ iff

$p = p''$.

$$\cfrac{p' \vdash < c : \bar{\epsilon} > \ell_2 \wedge \ell_4}{\cfrac{\ell_1 :: t \vdash < c : \bar{\epsilon} > true}{\underline{\ell_1 :: t \vdash true}}\, c \in \#(b) \quad \cfrac{p \vdash < c : \bar{\epsilon} > true}{\underline{p \vdash true}}\, c \in \#(a) \quad \cfrac{p \vdash \ell_2 \wedge \ell_4}{\underline{\ldots}}}$$

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 $p_2 = \ell_5 :: \ell_6 :: b.\ell_7 :: nil$, which corresponds to the statechart of figure 3, for $a = lab(t_1)$ and $b = lab(t_2)$. Suppose $c \in a \times b$. We have $p \models < c : \epsilon > \ell_4 \wedge \ell_7$, since $p \overset{c:\epsilon}{\mapsto} \ell_1 :: p_1' \times p_2'$, where $p_1' = \ell_2 :: \ell_4 :: nil$ and $p_2' = \ell_5 :: \ell_7 :: nil$. By definition of the semantics this transition is obtained by rule *par* from $p_1 \overset{a:\epsilon}{\mapsto} p_1'$ and $p_2 \overset{b:\epsilon}{\mapsto} p_2'$.

Our goal is that of avoiding the construction of $p \overset{c:\epsilon}{\mapsto} \ell_1 :: p_1' \times p_2'$. 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 $p_2 \overset{b:\epsilon}{\mapsto} p_2'$. This reasoning suggests the following compositional proof.

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

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 $\ell_1 :: p'' \times p_2' \models \ell_4 \wedge \ell_7$.

For a process $p = \ell_1 :: p_1 \times p_2$, where p_2 is not a basic process, the rules will be 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.

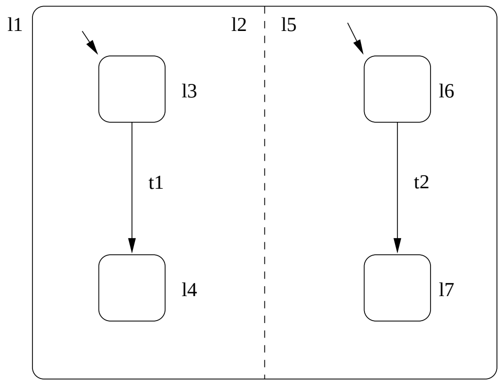

Fig. 3. An example of parallel composition

8 The proof system

In this section we present the proof system. Logical rules, which do not involve process operators, are listed below.

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

$$\frac{p \vdash A_0 \wedge A_1}{p \vdash A_0 \;\; p \vdash A_1} \wedge$$

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

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

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

These rules are quite obvious. The soundness of fixed points rules relies on the following *reduction lemma* [19], which has been exploited by [34] in combination with tagged fixpoints for local model checking.

Lemma 16 *Let E be a set and ψ be a monotonic function on $\mathcal{P}(E)$. For $p \in E$,*

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

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

8.1 Rules for nil

$$\frac{\ell :: \ell_1 :: nil \vdash < a : \bar{\epsilon} > A}{\ell :: \ell_1 :: nil \vdash A} <> nil \; a \in \#(*) \qquad \frac{\ell :: \ell_1 :: nil \vdash [a : \epsilon]A}{} []nil_1$$

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

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

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

8.2 Rules for prefixing

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

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

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

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

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

These rules are quite obvious, since we have $\ell :: \ell_1 :: a.t \overset{\alpha}{\mapsto} p'$ 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 $p' = \ell :: \ell_1 :: a.t$.

8.3 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 $type(\alpha) = \epsilon$.

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

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

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

$$\frac{\ell :: \ell_1 :: b_1 + b_2 \vdash [\alpha]A}{\ell :: \ell_1 :: b_1 + b_2 \vdash A} + []\bar{\epsilon}_1 \quad \frac{\ell :: \ell_1 :: b_1 + b_2 \vdash [\alpha]A}{\ell :: \ell_1 :: b_i \vdash [\alpha]false} + []\bar{\epsilon}_2$$

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

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

8.4 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

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

The corresponding shift rule is:

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

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

$$\frac{\ell :: t \bigtriangledown p \vdash < \alpha > A}{\ell :: t \vdash < \alpha > A} \bigtriangledown <> \epsilon_1 \qquad \frac{\ell :: t \bigtriangledown p \vdash < \alpha > A}{p \vdash < \alpha > (A)_{\ell::t\bigtriangledown}} \bigtriangledown <> \epsilon_2$$

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

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

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

$$\frac{\ell :: t \bigtriangledown p \vdash [\alpha]A}{\ell :: t \bigtriangledown p \vdash A} \bigtriangledown[]\bar{\epsilon}_1 \; \frac{\ell :: t \bigtriangledown p \vdash [\alpha]A}{\ell :: t \vdash [\alpha]false} \bigtriangledown[]\bar{\epsilon}_2$$

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

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

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

8.5 Rules for recursion

Let $\alpha \in \mathcal{S}(Act)$.

$$\frac{\ell :: rec\, x.t \vdash [\alpha]A}{\ell :: t[rec\, x.t/x] \vdash [\alpha]A} rec[] \; \frac{\ell :: rec\, x.t \vdash < \alpha > A}{\ell :: t[rec\, x.t/x] \vdash < \alpha > A} rec <>$$

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:

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

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 decompose 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 $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \overset{\alpha}{\mapsto} \ell :: p_1' \times p_2'$, such that $\ell :: p_1' \times p_2' \models A$. Moreover, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_2 \overset{\alpha}{\mapsto} \ell :: p_1' \times p_2'$ has been obtained by rule *par* from $p_1 \overset{\alpha_1}{\mapsto} p_1'$ and $\ell_1 :: \ell_2 :: b_2 \overset{\alpha_2}{\mapsto} p_2'$ for $\alpha \in \alpha_1 \times \alpha_2$. 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 $choice_1$, $\ell_1 :: \ell_2 :: b_1 + b_2 \overset{\alpha_2}{\mapsto} p_2'$. 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 inspect the type of the two components.

Definition 17 *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 $p \overset{\alpha'}{\mapsto} p'$ is *consistent* with α iff

(1) $\alpha = a : \tau$, $\alpha' = a : \tau'$ and $type^*(\alpha) = \tau'$;
(2) If $\alpha = a : \tau_1 \times \tau_2 \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$, then $p \overset{\alpha'}{\mapsto} p'$ has been been obtained by rule *par* from $p_i \overset{\alpha'_i}{\mapsto} p'_i$, such that $p = \ell :: p_1 \times p_2$, $p' = \ell :: p'_1 \times p'_2$, $\alpha' \in \alpha'_1 \times \alpha'_2$, and $p_i \overset{\alpha'_i}{\mapsto} p'_i$ 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 $\alpha \in \mathcal{E}(Act)$,

- $\|< \alpha > A \| = \{p \in \mathcal{CSP} \mid \exists p' \in \mathcal{CSP}, p \overset{\alpha'}{\mapsto} p', p' \in \| A \|$ and $p \overset{\alpha'}{\mapsto} p'$ is consistent with $\alpha\}$;
- $\| [\alpha]A \| = \{p \in \mathcal{CSP} \mid \forall p' \in \mathcal{CSP}, p \overset{\alpha}{\mapsto} p'$, which is consistent with α, then $p' \in \| A \|\}$.

Intuitively, $p \models < \alpha > A$ with $\alpha \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$ iff there exists $p \overset{\alpha'}{\mapsto} p'$, 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 $p \overset{\alpha'}{\mapsto} p'$ must be obtained by rule *par* applied to premises $\{p_i \overset{\alpha'_i}{\mapsto} p'_i\}_{i \in \{1,2\}}$, 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 $(a : \tau_1) \otimes (a : \tau_2) = \{a : \tau_1 \times \tau_2 \mid a \in a_1 \times a_2\} \subseteq \mathcal{E}(Act)$.

For $\alpha = a : \tau_1 \times \tau_2 \in \mathcal{E}(Act)$, we use the notation $proj_i(\alpha)$ to denote $a : \tau_i \in \mathcal{E}(Act)$.

8.6.1 Expansion rules

Formulas with extended actions are introduced by the following rules.

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

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

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 $\mathcal{E}(\epsilon) = \{\epsilon \times \epsilon, \bar{\epsilon} \times \epsilon, \epsilon \times \bar{\epsilon}\}$ and $\mathcal{E}(\bar{\epsilon}) = \{\bar{\epsilon} \times \bar{\epsilon}\}$.

These rules are based on the consideration that an arrow $\ell :: p_1 \times p_2 \overset{\alpha}{\mapsto} \ell :: p_1' \times p_2'$ is obtained by rule *par* from $\{p_i \overset{\alpha_i}{\mapsto} p_i\}_{i \in \{1,2\}}$, 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 $i \in \{1, 2\}$.

8.6.2 Rules for nil

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

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

The corresponding shift rule is:

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

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

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

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

8.6.3 Rules for prefixing

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

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

$$\frac{\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \vdash [\alpha]A}{\begin{array}{l}\{p_1 \vdash [\alpha_i](A)_{\ell\times\ell_1::t}\}_{i\in\{1,n\}}\\ \{p_1 \vdash [\beta_i](A)_{\ell\times\ell_1::\ell_2::a.t}\}_{i\in\{1,k\}}\end{array}} \times[] \quad \begin{array}{l}\{\alpha_i\}_{i\in\{1,n\}} = \\ \{\alpha' \mid \alpha \in \alpha' \otimes (a:\epsilon)\}\\ \{\beta_i\}_{i\in\{1,k\}} = \\ \{\beta \mid \alpha \in \beta \otimes (b:\bar{\epsilon}), b \in \#(a)\}\end{array}$$

These rules are based on the observation that, if $p_2 = \ell_1 :: \ell_2 :: a.t$, then $p_2 \overset{\alpha_2}{\mapsto} p_2'$ 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 $p_2' = p_2$.

8.6.4 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 $\bar{\epsilon}$. We define extended formulas $(A)_{\times b+}$ with the semantics

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

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 $p_2 = \ell_1 :: \ell_2 :: b_1 + b_2$, $p_2 \overset{\alpha_2}{\mapsto} p_2'$ iff one of the following cases hold:

(1) $\ell_1 :: \ell_2 :: b_i \overset{\alpha_2}{\mapsto} p_2'$, for some $i \in \{1, 2\}$, and $type(\alpha_2) = \epsilon$;
(2) $\ell_1 :: \ell_2 :: b_i \overset{\alpha_2}{\mapsto} \ell_1 :: \ell_2 :: b_i$, 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*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha}{\mapsto} \ell :: p_1' \times p_2'$, 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*, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha}{\mapsto} \ell :: p_1' \times p_2'$, since $\alpha \in \alpha_1 \times \alpha_2$, by the properties of $\times$.

8.6.5 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

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

and

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

The corresponding shift rules are

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

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 $type(proj_2(\alpha)) = \bar{\epsilon}$.

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

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

$$\frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash [\alpha]A}{\ell :: p_1 \times \ell_2 :: t \vdash [\alpha]false} \times [] \bigtriangledown \bar{\epsilon}_2 \quad \frac{\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \vdash [\alpha]A}{\ell :: p_1 \times p \vdash [\alpha]false} \times [] \bigtriangledown \bar{\epsilon}_3$$

8.6.6 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 $rec\, x.t$.

8.6.7 Rules for parallel composition

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

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

The shift rule is

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

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

$$\frac{\ell :: p_1 \times \ell_1 :: (p_3 \times p_4) \vdash [\alpha]A}{\{\ell :: \ell_1 :: (p_1 \times p_3) \times p_4 \vdash [\beta_i](A)_\times\}_{i \in \{1,n\}}} \times \times [] \; left(\alpha) = \{\beta_i\}_{i \in \{1,n\}}$$

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 $left(\alpha)$.

9 Soundness and completeness of the proof system

The proof system is sound for arbitrary $\mathcal{CSP}$ processes. Hence, it can be successfully applied for proving properties of infinite state processes. However, it is complete only for regular processes, which are finite-state.

Theorem 18 *(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 A and is obtained by proving that each rule is backward sound. The soundness of fixpoints rules follows from the reduction lemma 16.

The proof of completeness is based on the definition of an ordering $\prec$ over formulas, which is well-founded for regular processes.

Definition 19 *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 relies on 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 $\{p'_j \vdash true\}_{j\in\{1,k\}}$.

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 20 *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 $p \overset{\alpha_i}{\mapsto} p_i$ that is consistent with α.

The proof of this lemma is shown in appendix A 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 21 *(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 A and is by induction on $\prec$. The main observation is that 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. For formulas $< \alpha > A$ and $[\alpha]A$, lemma 20 is exploited.

10 An example

In this section we show the sketch of a proof for a simple property of the statechart of figure 1. Consider the process $p = S :: rec\, x.(G :: lab(t_5).rec\, y.(H :: lab(t_6).x \bigtriangledown \alpha(H)) \bigtriangledown \alpha(G))$ representing the whole statechart as described in section 5. Let $A \equiv G \wedge ([K]X)$ be an L_μ formula, for $K = \{\alpha \in \mathcal{S}(\Sigma(\Pi)) \mid d \notin trigger(\alpha)\}$. Formula $[K]\phi$ for $K = \{\alpha_1, \ldots, \alpha_n\}$, is a shorthand for $[\alpha_1]\phi \wedge \ldots \wedge [\alpha_n]\phi$.

The formula $\nu X.A$ express the property that always, if the environment does not produce d, state G is entered. Note how modal operators of μ-calculus allows us to express in an elegant way properties involving assumptions on the behaviour of the environment. We have the following proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p \vdash G} \quad \dfrac{\pi_1}{p \vdash [\alpha_1]A_1} \quad \ldots \quad \dfrac{\pi_n}{p \vdash [\alpha_n]A_1}}{\ldots}\wedge}{p \vdash \nu x.A}\nu_1}{}$$

where $A_1 = \nu X\{p\}.A$. Since process p is not in the tag of the fixpoint formula rule ν_1 is applied, i.e. the fixpoint formula is unfolded. Moreover, the formula A_1 is considered in order to record that process p has already been checked.

We will show only proof π_1 for $\alpha_1 = a_1 : \epsilon$ and $a_1 = (\{a, b, c\}, \{a\} \prec \{b\} \prec \{c\})$. The other cases are similar.

1. Proof π_1 is as follows.

$$\dfrac{\dfrac{\dfrac{\pi_{1,1}}{S :: t_4 \vdash [\alpha_1]A_1} \quad \dfrac{\pi_{1,2}}{\alpha(G) \vdash [\alpha_1]A_2}}{S :: t_4 \bigtriangledown \alpha(G) \vdash [\alpha_1]A_1} \bigtriangledown[]\epsilon}{p \vdash [\alpha_1]A_1} rec[]$$

where $A_2 = (A_1)_{S::t_4\bigtriangledown}$ is an extended assertion and $t_4 = G :: lab(t_5).rec\, y.(H :: lab(t_6).t_6 \bigtriangledown \alpha(H))$ and $t_6 = rec\, x.(G :: lab(t_5).rec\, y.(H :: lab(t_6).x \bigtriangledown \alpha(H)) \bigtriangledown \alpha(G))$.

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

$$\dfrac{S :: t_4 \vdash [\alpha_1]A_1}{} []_3$$

Note that rule $[]_3$ can be applied, since $a_1 \neq lab(t_5)$ and $type(\alpha_1) = \epsilon$.

1.2. Since $\alpha(G) = G :: \alpha(E) \times \alpha(F)$, proof $\pi_{1,2}$ is as follows.

$$\dfrac{\alpha(G) \vdash [\alpha_1]A_2}{\dfrac{\alpha(G) \vdash [\alpha_{1,1}]A_2}{\pi_{1,2,1}} \quad \dfrac{\alpha(G) \vdash [\alpha_{1,2}]A_2}{\pi_{1,2,2}} \quad \dfrac{\alpha(G) \vdash [\alpha_{1,3}]A_2}{\pi_{1,2,3}}}\ \mathcal{E}[]$$

where $\alpha_{1,1} = a_1 : \bar{\epsilon} \times \epsilon$, $\alpha_{1,2} = a_2 : \epsilon \times \bar{\epsilon}$, $\alpha_{1,3} = a_3 : \epsilon \times \epsilon$.

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

$$\dfrac{G :: \alpha(E) \times \alpha(F) \vdash [\alpha_{1,1}]A_2}{\dfrac{G :: \alpha(E) \times F :: t'_2 \vdash [\alpha_{1,1}]A_2}{\dfrac{\alpha(E) \vdash [\alpha'_{1,1}]A_2}{}}\times[]}\times[]rec$$

where $t'_2 = B :: lab(t_3).rec\, y.D :: lab(t_4).t_2$ and $t_2 = rec\, x.B :: lab(t_3).rec\, y.$ $D :: lab(t_4).x$. Moreover, $A_3 = (A_2)_{G \times t''_2}$, with $t''_2 = rec\, y.D :: lab(t_4).t_2$ and $\alpha'_{1,1} = (\{a, b\}, \{a\} \prec \{b\}) : \bar{\epsilon}$.

Note that $\alpha'_{1,1} \otimes lab(t_3) : \epsilon = \alpha_{1,1}$.

We have

$$\dfrac{\alpha(E) \vdash [\alpha'_{1,1}]A_3}{\dfrac{E :: t'_1 \vdash [\alpha'_{1,1}]A_3}{}\ []_4}\ rec[]$$

where $t_1 = rec\, x.A :: lab(t_1).rec\, y.C :: lab(t_2).x$, $t'_1 = A :: lab(t_1).rec\, y.C :: lab(t_2).t_1$, $t''_1 = rec\, y.C :: lab(t_2).t_1$.

In fact, $type(\alpha'_{1,1}) = \bar{\epsilon}$ and $(\{a, b\}, \{a\} \prec \{b\}) \notin \#(lab(t_1))$.

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

$$\dfrac{G :: \alpha(E) \times \alpha(F) \vdash [\alpha_{1,2}]A_2}{\dfrac{G :: \alpha(E) \times F :: t'_2 \vdash [\alpha_{1,2}]A_2}{}\ \times[]}\times[]rec$$

In fact, for each $\sigma \in \#(lab(t_3))$, $\bar{b} \in trigger(\sigma)$ so that there exists no action

$\alpha'_{1,2}$ such that $\alpha_{1,2} \in \alpha'_{1,2} \otimes \sigma : \bar{\epsilon}$.

1.2.3. Proof $\pi_{1,2,3}$ is as follows.

$$\dfrac{\dfrac{G :: \alpha(E) \times \alpha(F) \vdash [\alpha_{1,1}]A_2}{G :: \alpha(E) \times F :: t'_2 \vdash [\alpha_{1,3}]A_2}\times[]rec}{\alpha(E) \vdash [\alpha'_{1,3}]A_3}\times[]$$

where $\alpha'_{1,3} = (\{a, b\}, \{a\} \prec \{b\}) : \epsilon$.

Note that $type(proj_2(\alpha_{1,3})) = \epsilon$ and $\alpha_{1,3} = \alpha'_{1,3} \otimes lab(t_3) : \epsilon$.

We have

$$\dfrac{\dfrac{\dfrac{\dfrac{\alpha(E) \vdash [\alpha'_{1,3}]A_3}{E :: t'_1 \vdash [\alpha'_{1,3}]A_3}\,rec[]}{E :: t''_1 \vdash A_3}\,[]_1}{G :: E :: t''_1 \times F :: t''_2 \vdash A_2}\,shift}{S :: t_4 \bigtriangledown G :: E :: t''_1 \times F :: t''_2 \vdash A_1}\,shift$$

where $t_1 = rec\,x.A :: lab(t_1).rec\,y.C :: lab(t_2).x$, $t'_1 = A :: lab(t_1).rec\,y.C :: lab(t_2).t_1$, $t''_1 = rec\,y.C :: lab(t_2).t_1$.

Let $p' = S :: t_4 \bigtriangledown G :: E :: t''_1 \times F :: t''_2$. We have the following proof.

$$\dfrac{\dfrac{p' \vdash \nu X\{p\}.A}{\cdots}\,\nu_1}{\dfrac{p' \vdash G}{} \quad \dfrac{p' \vdash [\alpha_1]A_4}{\pi'_1} \quad \ldots \quad \dfrac{p' \vdash [\alpha_n]A_4}{\pi'_n}}\,\wedge$$

where $A_4 = \nu X\{p, p'\}.A$. Note that rule ν_1 is applied, since the process p' has not been already checked, i.e. it is not in the tag of the fixpoint formula.

Proof π'_1 is analogous to proof π_1 due to the analogy of processes p and p'. By omitting the details, proof π'_1 is as follows.

$$\dfrac{\dfrac{\dfrac{p' \vdash [\alpha_1]A_4}{\cdots}}{p \vdash \nu X\{p, p'\}.A}}{}\,\nu_2$$

In this case rule ν_2 instead of ν_1 is applied and this completes the proof. In a similar way the proofs $p' \vdash [\alpha_i]A_4$, for $i \in \{2, n\}$, are constructed.

Note that the proof is complex, even if the statechart of figure 1 is simple. However, the compositional proof system allows us to reuse subproofs. Moreover, if we replace a component, for instance G, we must redone only the part of the proof concerning $\alpha(G)$.

It is clear that only very simple proofs can be done by hand. However, one can use an interactive proof checker, based for instance on *Coq* [8], for helping the user in the construction of a proof.

11 Conclusions

In this paper we have presented a compositional verification method for statecharts. The result has been achieved in two steps:

(1) A process language $\mathcal{SP}$ has been defined, which can be instantiated to model statecharts according to the semantics of Pnueli and Shalev.
(2) A compositional proof system for checking μ-calculus properties of $\mathcal{SP}$ processes has been defined. Such a proof system is sound in general and it is complete for finite-state processes including the ones representing statecharts.

The process language $\mathcal{SP}$ is parametric in the set of basic actions and in two operations over actions (parallel composition and conflict), which are used in its labelled transition system semantics. Its main features are an operator of refinement and the explicit use of locations. Moreover, both non-idle and idle actions of processes are explicitly represented in the labelled transition system. The assumption is that idle actions are performed iff no non-idle action is possible, i.e. the process is idle only on actions that are in conflict with its possible actions. Such a semantics permits to suitably define in a compositional way processes modelling statecharts, where basic actions describe transitions and locations correspond to statecharts boxes. The main problem is to find adequate basic actions and operations of parallel composition and conflict, which correctly models steps. In fact, the maximality of reaction has to be combined with the principles of causality and consistency. We have shown that these problems can be solved by considering basic actions representing both the input-output of transitions and the causal ordering between input and output.

A process-algebraic version of statecharts has been proposed in [32]. The language has an operator of refinement similar to the one of $\mathcal{SP}$. However, the semantics of the language is significantly different. The main point is that *idle* actions are not explicitely considered so that empty steps of statecarts are not explicitly represented in the labelled transition semantics. This approach leads to a semantics, that is not equivalent to the one of Pnueli and Shalev. We have shown in [22] some examples to point out the necessity of considering idle actions. Intuitively, explicit empty steps are necessary to distinguish empty steps from non-defined reactions, due to causal paradoxes. The same problem arises also in the labelled transition system semantics of [31].

It is important to stress that the language $\mathcal{SP}$ is not ad-hoc for statecharts and other languages such as SCCS and CBS (Calculus of Broadcasting Systems) can be embedded in it. 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 $a?.p \overset{a?}{\mapsto} p$ and $a?.p \overset{b:}{\mapsto} a?.p$, where $a \neq b$. On the other hand, for sending actions the transitions are $a!.p \overset{a!}{\mapsto} p$ and $a!.p \overset{b!}{\mapsto} a!.p$. 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 [28], and $\#(a!)$ (resp. $\#(a?)$)$= \{b :\mid b \in \mathcal{A}, b \neq a\}$, and $\#(*) = \{a :\mid a \in \mathcal{A}\}$.

The compositional proof system for $\mathcal{SP}$ is an adaptation of some ideas proposed in [3] for the compositional verification of a simple CCS-like language. However, the different semantics of our language leads to a more complex proof system, in particular because of the distinction between idle and non-idle actions. Such a proof system provide a sound and complete method for the compositional verification of statecharts.

It is important to stress that the non-modularity of the semantics is a limitation for the application of other methods of compositional verification to statecharts, such as the method of modular model checking based on the assumption-commitment paradigm suggested by [6] for CTL. These difficulties are pointed out also in [15], 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 causal ordering between inputs and outputs is explicitely represented in the axiomatization so that the proof system is quite complex and difficult to apply.

The research presented in this paper could be the basis for future research concerning real-time extensions of statecharts. In [24,23] it is presented an extension of the compositional proof system for treating a real-time (discrete) version of $\mathcal{SP}$. This new language is capable of modelling a real-time version of statecharts with minimal and maximal delay associated to transitions. The logic is an extension of μ-calculus with freeze quantification and clock constraints [1], which is powerful enough to express the classical bounded temporal properties. It would be interesting to study how this approach can cope with a dense timed version of statecharts [18].

Acknowledgement

I wish to thank Andrea Maggiolo-Schettini and Andrea Masini for their comments and suggestions during the preparation of this paper. I also thank the anonymous referees for their helpful suggestions.

References

[1] 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.

[2] H.R. Andersen. Model Checking and Boolean Graphs. *Theoretical Computer Science*, 126:3–30, 1994.

[3] 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.

[4] H.R. Andersen and G. Winskel. Compositional Checking of Satisfaction. *Formal Methods in System Design*, 1(4):144–153, 1992.

[5] G. Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 19:87–152, 1992.

[6] 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.

[7] R. Cleaveland. Tableau-based Model Checking in the Propositional mu-Calculus. *Acta Informatica*, 27:725–747, 1990.

[8] C. Cornes, J. Courant, J. Fillatre, G. Huet, P. Manoury, C. Munoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saibi, and B. Werner. The Coq Proof Assistant Reference Manual-Version 5.10. Technical report, INRIA, Rocquenfort, 1995.

[9] 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.

[10] 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.

[11] 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.

[12] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[13] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions Soft. Eng. Method.*, (5):293–333, 1996.

[14] 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.

[15] J.J.M. Hooman, R. Ramesh, and W.P. de Roever. A Compositional Axiomatization of Statecharts. *Theoretical Computer Science*, 101:289–335, 1992.

[16] 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.

[17] 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.

[18] 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.

[19] D. Kozen. Results on the Propositional mu-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[20] 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.

[21] 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.

[22] F. Levi. A Process Language for Statecharts. In *Analysis and Verification of Multiple-Agent Languages*, volume 1192 of *Lecture Notes in Computer Science*, pages 388–403, 1996.

[23] F. Levi. Compositional Verification of Timed Statecharts. In H. Barringer, editor, *Proc. of the Second International Conference on Temporal Logic*. Applied Logic Series, Kluwer Academic Publishers, 1997. to appear.

[24] F. Levi. *Verification of Temporal and Real-Time Properties of Statecharts*. PhD thesis, TD-6/97,Dipartimento di Informatica, Università di Pisa, 1997.

[25] 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.

[26] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[27] 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.

[28] K.V.S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25, 1995.

[29] V. Pratt. A decidable mu-calculus. In *Proceedings of 22nd. FOCS*, pages 421–427, 1981.

[30] C. Stirling and D. Walker. Local Model Checking in the Modal mu-Calculus. *Theoretical Computer Science*, 89:161–177, 1991.

[31] 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.

[32] 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.

[33] 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.

[34] G. Winskel. A Note on Model Checking the Modal Nu-calculus. *Theoretical Computer Science*, 83:157–187, 1991.

A Appendix

In this appendix we show a sketch of the proofs of theorems 18 and 21. The complete proofs can be found in [24].

Theorem 18 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 $p \vdash A$ then $p \models A$.

PROOF. It is sufficient to show that each rule is backward sound. We show only the main cases for formulas $< \alpha > A$.

- Rule $<>_1$.

Assume $\ell :: t \models A$. By definition, $\ell :: \ell_1 :: a.t \overset{a:\epsilon}{\mapsto} \ell :: t$ so that $\ell :: \ell_1 :: a.t \models < a : \epsilon > A$.

- Rule $+ <> \epsilon$.

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 $+ <> \bar{\epsilon}$.

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 $\ell :: \ell_1 :: b_i \overset{\alpha}{\mapsto} \ell :: \ell_1 :: b_i$, for each $i \in \{1, 2\}$. Since $type(\alpha) = \bar{\epsilon}$, then by rule $choice_2$, $\ell :: \ell_1 :: b_1 + b_2 \overset{\alpha}{\mapsto} \ell :: \ell_1 :: b_1 + b_2$. Since $\ell :: \ell_1 :: b_1 + b_2 \models A$, $\ell :: \ell_1 :: b_1 + b_2 \models < \alpha > A$.

- Rule $\bigtriangledown <> \epsilon_2$.

Assume $p \models < \alpha > (A)_{\ell::t\bigtriangledown}$. Since $\alpha \in \mathcal{S}(Act)$, there exists $p \overset{\alpha}{\mapsto} p'$, such that

$p' \models (A)_{\ell::t\bigtriangledown}$, i.e. $\ell :: t \bigtriangledown p' \models A$. Since $type(\alpha) = \epsilon$, by rule $\bigtriangledown_2$, $\ell :: t \bigtriangledown p \overset{\alpha}{\mapsto} \ell :: t \bigtriangledown p'$. Since $\ell :: t \bigtriangledown p' \models A$, then $\ell :: t \bigtriangledown p \models < \alpha > A$.

- Rule $\bigtriangledown <> \epsilon_1$.

Assume $\ell :: t \models < \alpha > A$. Since $\alpha \in \mathcal{S}(Act)$, there exists $\ell :: t \overset{\alpha}{\mapsto} p'$, where $p' \models A$. Since $type(\alpha) = \epsilon$, by rule $\bigtriangledown_1$, $\ell :: t \bigtriangledown p \overset{\alpha}{\mapsto} p'$. Since $p' \models A$, then $\ell :: t \bigtriangledown p \models < \alpha > A$.

- Rule $\bigtriangledown <> \bar{\epsilon}$.

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 $\ell :: t \overset{\alpha}{\mapsto} \ell :: t$ and $p \overset{\alpha}{\mapsto} p$. By $\bigtriangledown_3$, $\ell :: t \bigtriangledown p \overset{\alpha}{\mapsto} \ell :: t \bigtriangledown p$. Since $\ell :: t \bigtriangledown p \models A$, then $\ell :: t \bigtriangledown p \models < \alpha > A$.

- Rule $\mathcal{E} <>$.

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 $\times <>_1$.

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 $\times <>_2$.

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 $\ell :: p_1' \times \ell_1 :: \ell_2 :: a.t \models A$, $\ell :: p_1 \times \ell_1 :: \ell_2 :: a.t \models < \alpha > A$.

- Rule $\times <> +\epsilon$.

Assume $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \models < \alpha > A$, for some $i \in \{1, 2\}$. By definition, there exists $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha'}{\mapsto} p'$, which is consistent with α and such that $p' \models A$. Moreover, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha'}{\mapsto} p'$ iff by rule *par*, $p_1 \overset{\alpha'_1}{\mapsto} p'_1$ and $\ell_1 :: \ell_2 :: b_i \overset{\alpha'_2}{\mapsto} p'_2$, with $p' = p'_1 \times p'_2$ and $\alpha' \in \alpha'_1 \times \alpha'_2$. Moreover, $\ell_1 :: \ell_2 :: b_i \overset{\alpha'_2}{\mapsto} p'_2$ 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 $choice_1$, $\ell_1 :: \ell_2 :: b_i \overset{\alpha'_2}{\mapsto} p'_2$ implies $\ell_1 :: \ell_2 :: b_1 + b_2 \overset{\alpha'_2}{\mapsto} p'_2$. Therefore, by rule *par*, $p_1 \times \ell_1 :: \ell_2 :: (b_1 + b_2) \overset{\alpha'}{\mapsto} \ell :: p'_1 \times p'_2$, 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 $\times <> +\bar{\epsilon}$.

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 $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha_i}{\mapsto} \ell :: p'_{1,i} \times p'_{2,i}$ such that $\ell :: p'_{1,i} \times p'_{2,i} \models (A)_{\times b_j +}$. Moreover, $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_i \overset{\alpha_i}{\mapsto} \ell :: p'_{1,i} \times p'_{2,i}$ 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 $\ell :: p_1 \times \ell_1 :: \ell_2 :: b_1 + b_2 \models <\alpha> A$.

- Rule $\times <> \bigtriangledown \epsilon_2$.

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 $\ell :: p'_1 \times \ell_2 :: t \bigtriangledown p' \models A$, $\ell :: p_1 \times \ell_2 :: t \bigtriangledown p \models < \alpha > A$.

- Rule $\times\times <>$.

Assume $\ell :: \ell_2 :: (p_1 \times p_2) \times p_3 \models < \beta > (A)_\times$. By definition, there exists $\ell :: \ell_2 :: (p_1 \times p_2) \times p_3 \overset{\beta'}{\mapsto} \ell :: \ell_2 :: (p'_1 \times p'_2) \times p'_3$, 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 $\ell :: \ell_2 :: (p_1 \times p_2) \times p_3 \overset{\beta'}{\mapsto} \ell :: \ell_2 :: (p'_1 \times p'_2) \times p'_3$ iff by rule par, $\ell_2 :: p_1 \times p_2 \overset{\beta'_1}{\mapsto} \ell_2 :: p'_1 \times p'_2$ and $p_3 \overset{\beta'_2}{\mapsto} p'_3$, where $\beta' \in \beta'_1 \times \beta'_2$. Moreover, $\ell_2 :: p_1 \times p_2 \overset{\beta'_1}{\mapsto} \ell_2 :: p'_1 \times p'_2$ iff, by rule par, $p_1 \overset{\beta'_{1,1}}{\mapsto} p'_1$ and $p_2 \overset{\beta'_{1,2}}{\mapsto} p'_2$, 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, $\ell :: p_1 \times \ell_2 :: (p_2 \times p_3) \overset{\beta'}{\mapsto} \ell :: p'_1 \times \ell_2 :: (p'_2 \times p'_3)$. Since $\ell :: \ell_2 :: (p_1 \times p_2) \times p_3 \overset{\beta'}{\mapsto} \ell :: \ell_2 :: (p'_1 \times p'_2) \times p'_3$ is consistent with β, then $\ell :: p_1 \times \ell_2 :: (p_2 \times p_3) \overset{\beta'}{\mapsto} \ell :: p'_1 \times \ell_2 :: (p'_2 \times p'_3)$ 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 soundness of shift rules is obvious.

$\square$

Lemma 20 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 $<> nil$.

- 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, $\ell :: t, p \in \mathcal{R}_p \subseteq \mathcal{S}_p$.

- 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 $p \vdash A$.

- 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 rule $\bigtriangledown_1$ has been applied, then the proof is similar to the first case of choice.
- If rule $\bigtriangledown_2$ has been applied, 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 $p'' \stackrel{\alpha}{\mapsto} p_i$, then by $\bigtriangledown_2$, $p \stackrel{\alpha}{\mapsto} \ell :: p_i$, 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 $p \stackrel{\alpha}{\mapsto} p'$ has been obtained by $\bigtriangledown_3$, the case is analogous to the one of $choice_2$.

- Assume $p = \ell : p_1 \times p_2$.

We have $p \models <\alpha> A$ iff either $\alpha \in \mathcal{S}(Act)$ and there exists $p \stackrel{\alpha}{\mapsto} p'$, where $p' \models A$, or $\alpha \in \mathcal{E}(Act) \setminus \mathcal{S}(Act)$ and there exists $p \stackrel{\alpha'}{\mapsto} p'$, such that it is consistent with α and $p' \models A$.

For $\alpha \in \mathcal{S}(Act)$, $\ell :: p_1 \times p_2 \stackrel{\alpha}{\mapsto} p'$ iff, by rule par, $p_1 \stackrel{\alpha_1}{\mapsto} p'_1$ and $p_2 \stackrel{\alpha_2}{\mapsto} p'_2$, 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 $p \stackrel{\alpha}{\mapsto} p'$ is consistent with β iff $p_1 \stackrel{\alpha_1}{\mapsto} p'_1$ and $p_2 \stackrel{\alpha_2}{\mapsto} p'_2$, with $\alpha \in \alpha_1 \times \alpha_2$. Moreover, for each $i \in \{1,2\}$, $p_i \stackrel{\alpha_i}{\mapsto} p'_i$ 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 $p_2 \stackrel{\alpha_2}{\mapsto} p'_2$ 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 $p_1 \stackrel{\alpha_1}{\mapsto} p'_1$, 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 $p_1 \stackrel{\alpha_1}{\mapsto} p'_1$ 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 $p_1 \stackrel{\alpha'_i}{\mapsto} p''_i$

that is consistent with β_1, then by rule *par*, $p \overset{\alpha_i''}{\mapsto} \ell :: p_i'' \times \ell_1 :: t$ 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 $p_2 = \ell_1 :: \ell_2 :: b_1 + b_2$.

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 $\ell :: p_1' \times \ell_1 :: \ell_2 :: b_2 \models (A)_{\times b_1 +}$. 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 $\ell :: p_{j,2} \times \ell_1 :: \ell_2 :: b'_2 \vdash A$.

- 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 $p_2 = \ell_2 :: p_3 \times p_4$.

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 21 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 16, $p' \models B[\mu X\{U,p'\}.B/X]$. 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 16 $p' \models \nu X\{U\}.B$ 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 20 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.

□