Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

A Temporal Semantics for Basic Process Algebra

Alessandro Fantechi, Stefania Gnesi, Valeria Perticaroli

Nota Interna B4-33
Agosto 1992

A Temporal Semantics for Basic Process Algebra

A. Fantechi, S. Gnesi - I.E.I.-C.N.R. Pisa, Italy
V. Perticaroli - S.I.P. Bologna, Italy

August 24, 1992

Abstract

In this paper we present a denotational semantics for Basic Process Algebra. The denotational domain is a new logic obtained by enriching the branching temporal logic CTL with a fixed point operator and a branching sequential composition operator, named *chop branching*. The temporal semantics so obtained is proved to be fully abstract with respect to the bisimulation semantics defined on BPA terms.

1 Introduction

In the last years there have been several attempts at defining new logics or at using existing ones to specify properties of reactive and concurrent systems. The gain of associating suitable logics, such as modal or temporal logics, to communicating systems is the possibility of using deductive methods to prove properties. To provide modularity in the specification and verification of concurrent systems the *compositional* denotation by logic assertions of concurrent systems, specified by a process algebra, becomes an important research issue.

Sound and complete *proof systems* for the satisfiability relation between concurrent systems and formulae [14,18,19] are compositional if they deduce that a system of processes satisfies or implements a given logical assertion from the knowledge of the assertions satisfied by its components, closely following the syntactic structure.

Compositionality can also be guaranteed by defining a *compositional semantics* for a process algebra, which associates to each process a logic formula expressing the properties of its execution sequences, e.g. a *temporal semantics* (a particular denotational semantics [16]).

It is well known that a denotational semantics $\mathcal{M}$ is a homomorphism from the term algebra to the interpretation algebra and therefore preserves compositionality: when the interpretation algebra is a temporal logic $\mathcal{L}$, we have:

$$\forall p1, p2, .., pn \in \mathcal{P} : \mathcal{M}(\mathcal{O}(p1, p2, .., pn)) = f(\mathcal{M}(p1), \mathcal{M}(p2), .., \mathcal{M}(pn)),$$

where $\mathcal{P}$ is the process language and f is an operator (or a composition of operators) of $\mathcal{L}$.

If we want to define a *temporal semantics* for $\mathcal{P}$, when $\mathcal{P}$ is a process language with an associated operational semantics, it is necessary to establish a link between the temporal semantics provided and the original operational one: the temporal semantics should be *fully abstract* with respect to the operational semantics. A denotational semantics, $\mathcal{M}$, is fully abstract with respect to the operational semantics $\mathcal{O}$, modulo an equivalence relation ($\approx$), if and only if :

$$\forall p, q \in \mathcal{P}.\ \mathcal{M}[p] = \mathcal{M}[q] \Rightarrow \mathcal{O}[p] \approx \mathcal{O}[q]$$

and

$$\forall\ context\ \mathcal{C}.\ \mathcal{O}[\mathcal{C}(p)] \approx \mathcal{O}[\mathcal{C}(q)] \Rightarrow \mathcal{M}[p] = \mathcal{M}[q].$$

The purpose of this paper is to present a temporal semantics of Basic Process Algebra$_{\delta,\ rec}$ (BPA$_{\delta,rec}$ in the following), a simplified version of the Algebra of Communicating Processes, which has as primitive the process constant δ and the operators of sequential composition, alternative composition and recursion [5,6]. On BPA$_{\delta,rec}$ terms a bisimulation equivalence relation is defined; according to this, our aim is to give the temporal semantics of BPA$_{\delta,rec}$ fully abstract with respect to bisimulation equivalence. The logic domain in which we define the temporal semantics for BPA$_{\delta,rec}$ will be the branching temporal logic CTL [8,9] enriched with a maximal fixed point operator and with a new logic operator, the *chop branching*, an extension of the chop operator defined in [3]. From now on we call this logic $\nu CTL + C_Q$.

This extension is needed because in [10] we have proved that νCTL has not sufficient expressive power to give the meaning to a non-idempotent composition operator, as the sequential composition operator of BPA$_{\delta,rec}$ is.

The paper is organized as follows: Section 2 describes BPA$_{\delta,rec}$; Section 3 presents its operational semantics; then, in Section 4, we define the logic $\nu CTL + C_Q$ which is shown to be expressive for BPA$_{\delta,rec}$ with respect to bisimulation equivalence in Section 5.

2 Basic Process Algebra$_{\delta,\ rec}$

Signature:

sorts $\mathcal{A}$—set of atomic actions
$\mathcal{P}$—set of processes; $\mathcal{A} \subseteq \mathcal{P}$
$\mathcal{V}ar$—infinite numerable set of process variables; $\mathcal{V}ar \subseteq \mathcal{P}$
functions $+ : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$—alternative composition
$\cdot : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$—sequential composition
constants $\delta \in \mathcal{A}$—deadlock

Basic Process Algebra$_{\delta,\ rec}$—BPA$_{\delta,rec}$—starts from a collection $\mathcal{A} = \{a, b, c, \ldots\}$ of given objects, called atomic actions. These actions usually have no duration and form the basic building blocks of our systems.

The two compositional operators we consider are "$\cdot$", denoting sequential composition, and "$+$" for alternative composition. If x and y are two processes, then $x \cdot y$ is the process that starts the execution of y after the completion of x and $x + y$ is the process that chooses either x or y and executes the chosen process.

A vital element in the present set-up of process algebra is the process δ, signifying "deadlock". The process $a \cdot b$ performs its two steps and then stops, silently and happily; but the process $a \cdot b \cdot \delta$ deadlocks after the actions a and b: it wants to do a proper action but it cannot.

Central to theories of concurrency is the solving of recursive equations or equivalently, the finding of fixed points. In this formal system a recursion expression is a syntactic construct of type $\langle X_1 \mid \mathcal{E} \rangle$, where X_1 is the recursion variable and $\mathcal{E} = \{X_i = T_i(X_1, \ldots, X_n) : i = 1, \ldots, n\}$ is a finite set of recursion equations. The $T_i(X_1, \ldots, X_n)$ are process expressions in the signature of $\mathrm{BPA}_{\delta,rec}$, possibly containing occurrences of recursion variables $X_1, \ldots, X_n$.

Table 1 gives an axiomatic characterization of $\mathrm{BPA}_{\delta,rec}$, where $x, y, z \in \mathcal{P}$, $X_i \in \mathcal{V}ar$ for $i = 1, \ldots, n$ and $\mathcal{E}$ is a finite set of recursion equations.

Basic Process Algebra$_{\delta,rec}$	
$x + y = y + x$	(A1)
$(x + y) + z = x + (y + z)$	(A2)
$x + x = x$	(A3)
$(x + y) \cdot z = x \cdot z + y \cdot z$	(A4)
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	(A5)
$x + \delta = x$	(A6)
$\delta \cdot x = \delta$	(A7)
$\dfrac{x_i = \langle X_i \mid \mathcal{E} \rangle, i = 1, \ldots, n}{x_1 = T_1(x_1, \ldots, x_n)}$	(R1)
$\dfrac{x_i = T_i(x_1, \ldots, x_n), i = 1, \ldots, n}{x_1 = \langle X_1 \mid \mathcal{E} \rangle}$ $T_i(X_1, \ldots, X_n)$ is "guarded"	(R2)
$\dfrac{\mathcal{E} = \mathcal{E}'}{\langle X_1 \mid \mathcal{E} \rangle = \langle X_1 \mid \mathcal{E}' \rangle}$	(R3)

Table 1

This axiom system is the core of a variety of more extensive process axiomatisations including for instance axioms for parallel operators on processes or communication functions.

Since time has a direction, sequential composition is not commutative and because the time to which choices take place is important, there is not the other distributive law

$$z \cdot (x + y) = z \cdot x + z \cdot y.$$

In this formal system we suppose that recursion is guarded, i.e. any occurrence of X_j $(j = 1, \ldots, n)$ in $T_i(X_1, \ldots, X_n)$ is preceded - guarded - by an atom from the action

alphabet; more precisely, every occurrence of X_j is in a subexpression of the form $a \cdot t'$ for some atom a and expression t'.

By a process we mean an element of some algebra satisfying the axioms of $\mathrm{BPA}_{\delta,rec}$; the x, y, z in Table 1 range over processes. This algebra is a process algebra for $\mathrm{BPA}_{\delta,rec}$; another process algebra for $\mathrm{BPA}_{\delta,rec}$ is the graph model consisting of finitely branching process graphs modulo bisimulation [1,4].

3 Operational semantics of $\mathrm{BPA}_{\delta,\, rec}$

The scope of this paper is to define a temporal semantics for $\mathrm{BPA}_{\delta,rec}$ and to prove its fully abstractness w.r.t the bisimulation equivalence. We need therefore to give an operational characterization of the semantics of $\mathrm{BPA}_{\delta,rec}$. For the sake of simplicity, we derive from the signature of $\mathrm{BPA}_{\delta,rec}$ a language of behaviour expressions defining processes of the process algebra, with the syntax given in Table 2, in which we explicit recursion through the use of a *rec* operator; B, B_1, B_2 are generic behaviour expressions, X is a recursion variable and B in $recX.B$ is a guarded expression containing X.

Name	*Syntax*
deadlock, inaction	δ
action	a
successful termination	ϵ
alternative composition	$B_1 + B_2$
sequential composition	$B_1 \cdot B_2$
variable	X
recursion	$recX.B$

Table 2

The operational semantics for this language is given in two steps: first a Labelled Transition System - LTS in the following - is associated to a process, then an equivalence relation is defined on the transition systems [11].

A LTS is a triple $(\mathcal{P}, \sum, \{R_x, x \in \sum\})$ such that:

- $\mathcal{P}$ is a set of processes,
- $\sum$ is a set of actions,
- $R_x \subseteq \mathcal{P} \times \mathcal{P}$.

We will use the notation $p - g \rightarrow q$ to mean that $(p, q) \in R_g$ and we will say that p is able to perform action g and transform in q. We will also use p, q (with indexes) as ranging over processes.

In Table 3 the operational semantics of $\mathrm{BPA}_{\delta,rec}$ is shown, where the transitions of processes are labelled by the atomic actions (marked as a) and by the successful termination action $\surd$, not user definable; therefore, we have $\sum = \mathcal{A} \cup \{\surd\}$. The notation $[x_1/y_1, \ldots, x_n/y_n]$ means that y_i is replaced by x_i for $i = 1, \ldots, n$.

Operator	*Operational semantics*	*Informal meaning*
deadlock, inaction		it denotes a process which cannot perform any action
successful termination	$\epsilon - \surd \rightarrow \delta$	it models the process able to emit the successful termination signal $\surd$
action	$a - a \rightarrow \epsilon$	it models a process which can perform the action a
alternative composition	$\dfrac{B_1 - a\surd \rightarrow B_1'}{B_1 + B_2 - a\surd \rightarrow B_1'}$ $\dfrac{B_2 - a\surd \rightarrow B_2'}{B_1 + B_2 - a\surd \rightarrow B_2'}$	the actions of the process are the set of possible actions of B_1 and B_2
sequential composition	$\dfrac{B_1 - a\surd \rightarrow B_1'}{B_1 \cdot B_2 - a\surd \rightarrow B_1' \cdot B_2}$ $\dfrac{B_1 - \surd \rightarrow \delta \quad B_2 - a\surd \rightarrow B_2'}{B_1 \cdot B_2 - a\surd \rightarrow B_2'}$ $\dfrac{B_1 - a \rightarrow \delta}{B_1 \cdot B_2 - a \rightarrow \delta}$	this operator allows sequential process composition
recursion	$\dfrac{B[recX.B/X] - a\surd \rightarrow B'}{recX.B - a\surd \rightarrow B'}$	the actions of the process are those of the behaviour expression B replacing the recursion variable X with $recX.B$

Table 3

On LTSs several equivalence relations are defined; among them we consider the *bisimulation equivalence* [15] and so we will refer to the operational semantics of $\mathrm{BPA}_{\delta,rec}$ as bisimulation semantics.

The operational semantics of $\mathrm{BPA}_{\delta,rec}$ is not in contrast with the semantics induced by the axioms and by the inference rules of Table 1 since the LTSs derived by the application of the operational semantics rules can be seen as elements of the set of finitely branching process graphs.

4 A Branching Temporal Logic for $\mathrm{BPA}_{\delta,\,rec}$

Since we are interested in giving the temporal semantics to $\mathrm{BPA}_{\delta,rec}$ and the behaviour expressions in the above language can be modelled by finite or infinite, discrete in time, sequences of actions, we will define a branching time temporal logic having as models finite or infinite structures.

We choose a standard branching time logic, CTL, with the addition of a maximal fixed point constructor and a new logic operator, the *chop branching*, to define the branching temporal semantics of our language.

The atomic formulae of temporal logic are borrowed from the action set $\mathcal{A} \cup \{\surd\} \cup \{\mathit{false}, \mathit{true}\}$ (we will call them action predicates). The operators that we will use are the usual first order logic connectives:

$$\neg, \vee, \wedge;$$

together with the usual temporal operators:

$$AX, EX, AX!, EX!,$$

a maximal fixed point constructor:

$$\nu x.\Phi(x),$$

and the new logic operator chop branching:

$$C_Q.$$

In order to define the semantics of temporal logic formulae, we need to introduce the concept of model. A model is a 4-uple (S, s_0, R, L), defining a Kripke structure M, where:

- S is a non-empty set of states;
- $s_0 \in S$ is the initial state or "root" of S;
- R is a relation on S defining the structure of the model; the properties of R are:

  (i) $\forall s \in S$, $(s, s) \notin R$ and $(s, s_0) \notin R$;

  (ii) $\forall s, s' \in S$, $(s, s') \in R \Rightarrow (\forall s'' \in S - \{s\},\ (s'', s') \notin R)$;

  (iii) $\forall s \in S - \{s_0\}, \exists s_1, \ldots, s_n \in S$ such that $(s_0, s_1), (s_1, s_2), \ldots, (s_{n-1}, s_n), (s_n, s) \in R$;

- $L : S \longrightarrow 2^{\text{Prop}}$ is a mapping from S to the powerset of Prop (the set of atomic propositions, Prop $= \mathcal{A} \cup \{\surd\} \cup \{true, false\}$). We have:

  (i) $\forall s \in S$, $true \in L(s)$ and $false \notin L(s)$;

  (ii) $\forall s \in S, \forall a, b \in \mathcal{A} \cup \{\surd\}$, if $a \neq b$ and $a \in L(s)$ then $b \notin L(s)$. (This means that only one action can be performed in a transition, and implies that $(a \wedge b) \equiv false$ if $a \neq b$);

  (iii) $\forall s \in S - \{s_0\}$, $\exists a \in \mathcal{A} \cup \{\surd\}$, such that $a \in L(s)$.

Now we can reach the state s' from s via the occurrence of an action a if and only if $(s, s') \in R$ and $a \in L(s')$.

The formulae of temporal logic are interpreted on the states of a Kripke structure M; by $s \models_M \Phi$ we indicate that a state s in a model M satisfies a formula Φ.

Informally, $EX\,\Phi$ means that *if next states exist* then *a next state exists* which satisfies Φ. The formula $EX!\,\Phi$ means that *a next state exists* which satisfies Φ. Moreover, $AX\,\Phi$ means that *if next states exist* then *all next states* satisfy Φ; the formula $AX!\,\Phi$ means that *a next state exists* and *all next states* satisfy Φ. Also, we have that every state satisfies *true* and no state satisfies *false*.

Note that by $\Phi^k(true)$ we denote the temporal formula $\Phi(\Phi(\ldots\Phi(true)\ldots))$ k-times. The maximal fixed point constructor $\nu x.\Phi(x)$ denotes the maximal solution to $x = \Phi(x)$, which exists if the function $\Phi(x)$ is monotonic and continuous. The monotonicity requirement can be ensured by the appearance of the temporal formula x in Φ under an even number of negations [2] and continuity derives from the continuity of all logical operators.

The chop branching operator is an extension, in branching temporal logic, of the chop operator introduced in [3] for linear temporal logic. The formula $\Phi\, C_Q\, \Psi$, where Q is an atomic proposition, holds, for a state s of a model M, if the submodel of M whose root is s can be decomposed in a model M_0 whose root satisfies Φ and in a family of models $\{M_i\}_{i \in I}$ whose roots are states satisfying the formula Ψ.

Now we show formally, by means of the satisfaction relation, the meaning of the $\nu CTL + C_Q$ operators.

- propositional connectives

$s \models_M Q$ iff $Q \in L(s)$
$s \models_M \neg\Phi$ iff not $(s \models_M \Phi)$
$s \models_M \Phi \vee \Psi$ iff $(s \models_M \Phi)$ or $(s \models_M \Psi)$
$s \models_M \Phi \wedge \Psi$ iff $(s \models_M \Phi)$ and $(s \models_M \Psi)$

- branching temporal operators

$s \models_M AX\,\Phi$ iff $R(s) = \{\}$ or for all $s' \in R(s)$, $s' \models_M \Phi$
$s \models_M EX\,\Phi$ iff $R(s) = \{\}$ or exists $s' \in R(s) : s' \models_M \Phi$
$s \models_M AX!\,\Phi$ iff $R(s) \neq \{\}$ and for all $s' \in R(s)$, $s' \models_M \Phi$
$s \models_M EX!\,\Phi$ iff $R(s) \neq \{\}$ and exists $s' \in R(s) : s' \models_M \Phi$

- maximal fixed point constructor

$s \models_M \nu x.\Phi(x)$ iff $s \models_M \wedge_{n\geq 0}\Phi^n(true)$ where $\Phi^0(x) = x$ and $\Phi^{n+1}(x) = \Phi(\Phi^n(x))$

- chop branching

$s \models_M \Phi\, C_Q\, \Psi$ iff there exist a possibly empty set of indexes I, a submodel

$$\mathrm{M}_0 = (S_0 \cup \{s_i\}_{i\in I} \cup \{f_i\}_{i\in I}, s, R_0, L_0)$$

and a family of submodels, of cardinality equal to the number of elements of I,

$$\{\mathrm{M}_i\}_{i\in I} = \{(S_i, s_i, R_i, L_i)\}_{i\in I}$$

such that:

- $S_0 \cup \{s_i\}_{i\in I} \subseteq S$;
- $\{f_i\}_{i\in I} \cap S = \{\}$;
- $\neg\exists s' \in S_0 \cup \{s_i\}_{i\in I} : (s_i, s') \in R_0$ for all $i \in I$;
- $R_0 = \{(s', s'') \in R : s', s'' \in S_0 \cup \{s_i\}_{i\in I}\} \cup \{(s_i, f_i), \text{ for } i \in I\}$;
- $L(s') \neq Q$ if $s' \in S_0 \cup \{s_i\}_{i\in I}$;
- $L_0(s') = \begin{cases} L(s') & \text{if } s' \in S_0 \cup \{s_i\}_{i\in I} \\ Q & \text{if } s' = f_i \text{ for } i \in I \end{cases}$
- $\{s_i\} = (S_0 \cup \{s_i\}_{i\in I} \cup \{f_i\}_{i\in I}) \cap S_i$, for all $i \in I$;
- $S_i \subseteq S$ for all $i \in I$;
- $\cap_{i\in I} S_i = \{\}$;
- $R_i = \{(s', s'') \in R : s', s'' \in S_i\}$ for all $i \in I$;
- $L_i(s') = L(s')$ for $s' \in S_i$ and for $i \in I$;
- $\forall s' \in S$ such that s' is reachable from s by means of R (that is there exist $s_1, \ldots, s_n \in S$ such that $(s, s_1), (s_1, s_2), \ldots, (s_{n-1}, s_n), (s_n, s') \in R$) then or $s' \in S_0 \cup \{s_i\}_{i\in I}$ or $s' \in S_i$ for some $i \in I$;
- $s \models_{M_0} \Phi \ \wedge_{i\in I} \ s_i \models_{M_i} \Psi$.

Adequacy

As we are interested in reasoning in a temporal logic $\mathcal{L}$ on terms of a process language $\mathcal{P}$, with an associated operational semantics, we need to associate to processes in $\mathcal{P}$ the

sets of properties (formulae in $\mathcal{L}$) which they satisfy. This requires to define the satisfaction relation $\models \subseteq \mathcal{P} \times \mathcal{T}$, written $p \models \Phi$ and read "p satisfies the property Φ". The definition of this relation induces an equivalence $\equiv_{\mathcal{L}}$ between processes which enjoies the same properties. Formally, we define:

$$p \equiv_{\mathcal{L}} q \text{ if and only if } \mathcal{F}(p) = \mathcal{F}(q)$$

where:

$$\mathcal{F}(p) = \{\Phi : \Phi \in \mathcal{L} \wedge p \models \Phi\}$$

Now, if $\approx \subseteq \mathcal{P} \times \mathcal{P}$ is the equivalence defined on the operational semantics of the process language $\mathcal{P}$, it is possible to define a relation between $\approx$ and $\equiv_{\mathcal{L}}$:

A logic $\mathcal{L}$ is *adequate* [17] w.r.t. an equivalence ($\approx$) defined on a given process language $\mathcal{P}$, if for every pair of processes p and q:

$$p \equiv_{\mathcal{L}} q \text{ if and only if } p \approx q.$$

In [7] it is shown that CTL is adequate with respect to strong bisimulation on Kripke structures; in this case adequacy is not lost when enriching the logic with new operators, as the maximal fixed point and the chop operator. We can extend this result to strong bisimulation on $\text{BPA}_{\delta,rec}$ by relating the models of the logic and Labelled Transitions Systems: they differ only because in the former labels are associated to states while in the latter they are associated to arcs. We can transform each other shifting the information from states to arcs or viceversa [13]. Therefore, $\nu CTL + C_Q$ is adequate w.r.t. strong bisimulation on $\text{BPA}_{\delta,rec}$ terms.

5 Expressiveness of $\nu CTL + C_Q$ for $\text{BPA}_{\delta,\, rec}$

Now, we can define a stronger relation between a process language and a logic, based on the notion of fully abstractness: informally a logic $\mathcal{T}$ is fully expressive (a refinement of the *expressiveness* relation given in [17]) w.r.t. an equivalence relation $\approx$, defined on a process language $\mathcal{P}$, if and only if it is possible to define a semantics $\mathcal{M}$ for $\mathcal{P}$ such that the denotation domain of $\mathcal{M}$ is $\mathcal{L}$, and $\mathcal{M}$ is fully abstract with respect to the operational semantics defining $\approx$.

A logic $\mathcal{L}$ is *fully expressive* w.r.t. an equivalence relation ($\approx$) defined on a process language $\mathcal{P}$, if it is adequate and for every process $p \in \mathcal{P}$ there exists a *compositional logic semantics* $\mathcal{M}(p) \in \mathcal{L}$ such that:

$$\mathcal{M}(p) \equiv_{\mathcal{L}} \mathcal{M}(q) \text{ if and only if } p \approx q$$

Theorem 5.1 *$\nu CTL + C_Q$ is fully expressive for $\text{BPA}_{\delta,\text{rec}}$ with respect to the bisimulation equivalence.*

Proof (by construction): The theorem is proved giving a denotational semantics to $\mathrm{BPA}_{\delta,rec}$ where the domain of denotations is $\nu CTL + C_Q$ in order to produce a semantics fully abstract with respect to bisimulation equivalence. The branching temporal semantics of $\mathrm{BPA}_{\delta,rec}$ is shown below. It is defined by means of the semantics function $\mathcal{M}$ and the auxiliary functions $\mathcal{S}$, $\mathcal{L}$.

Hence the type of the semantic functions is:

$$\begin{aligned}\mathcal{M} &: \mathrm{BPA}_{\delta,rec} \longrightarrow \text{logic formulae}\\ \mathcal{S} &: \mathrm{BPA}_{\delta,rec} \longrightarrow \text{logic formulae}\\ \mathcal{T} &: \mathrm{BPA}_{\delta,rec} \longrightarrow \text{logic formulae}\end{aligned}$$

For all $B \neq recX.B$, $B \neq X$, $B \neq B_1 \cdot B_2$ in $\mathrm{BPA}_{\delta,rec}$ we have:

$$\mathcal{M}(B) = \mathcal{S}(B) \wedge AX\, \mathcal{T}(B)$$

moreover we define:

$$\begin{aligned}&\mathcal{M}(recX.B) = \nu x.\, \mathcal{M}(B)\\ &\mathcal{M}(X) = x\\ &\mathcal{M}(B_1 \cdot B_2) = \mathcal{M}(B_1)\, C_{\surd}\, \mathcal{M}(B_2)\end{aligned}$$

- deadlock, inaction

$\mathcal{S}(\delta) = \mathit{true}$
$\mathcal{T}(\delta) = \mathit{false}$

The meaning of an inaction behaviour is

$$\mathcal{M}(\delta) = \mathit{true} \wedge AX\, \mathit{false} = AX\, \mathit{false};$$

the states satisfying this logic formula are states without successors because δ is a completely inactive process and so it does not admit any type of transition.

- successful termination

$\mathcal{S}(\epsilon) = EX!(\surd \wedge \mathcal{M}(\delta))$
$\mathcal{T}(\epsilon) = \surd \wedge \mathcal{M}(\delta)$

The meaning of successful termination is:

$$\mathcal{M}(\epsilon) = EX!(\surd \wedge \mathcal{M}(\delta)) \wedge AX\,(\surd \wedge \mathcal{M}(\delta)).$$

The meaning $\mathcal{M}(\epsilon)$ is true for states which admit successors verifying $\mathcal{M}(\delta)$ and to which we arrive through the execution of the special action $\surd$. The logic formula $AX\,(\surd \wedge \mathcal{M}(\delta))$ guarantees that different successors cannot exist.

- action

$\mathcal{S}(a) = EX!(a \wedge \mathcal{M}(\epsilon))$
$\mathcal{T}(a) = a \wedge \mathcal{M}(\epsilon)$

The meaning of the action operator is:

$$\mathcal{M}(a) = EX!(a \wedge \mathcal{M}(\epsilon)) \wedge AX(a \wedge \mathcal{M}(\epsilon)).$$

The meaning $\mathcal{M}(a)$ is true for states which admit successors verifying $\mathcal{M}(\epsilon)$ and to which we arrive through the execution of the action a. The logic formula $AX(a \wedge \mathcal{M}(\epsilon))$ guarantees that different successors cannot exist.

- alternative composition

$\mathcal{S}(B_1 + B_2) = \mathcal{S}(B_1) \wedge \mathcal{S}(B_2)$
$\mathcal{T}(B_1 + B_2) = \mathcal{T}(B_1) \vee \mathcal{T}(B_2)$

The meaning of the alternative composition behaviour is:

$$\mathcal{M}(B_1 + B_2) = \mathcal{S}(B_1) \wedge \mathcal{S}(B_2) \wedge AX(\mathcal{T}(B_1) \vee \mathcal{T}(B_2)).$$

The meaning of the choice operator is true for states admitting both behaviours described by B_1 and B_2.

- sequential composition

$\mathcal{M}(B_1 \cdot B_2) = \mathcal{M}(B_1) C_{\surd} \mathcal{M}(B_2)$
$\mathcal{S}(B_1 \cdot B_2) = \mathcal{S}(B_1) C_{\surd} \mathcal{M}(B_2)$
$\mathcal{T}(B_1 \cdot B_2) = \mathcal{T}(B_1) C_{\surd} \mathcal{M}(B_2)$

The meaning of the behaviour expression $B_1 \cdot B_2$ is true for states verifying the logic formula $\mathcal{M}(B_1) C_{\surd} \mathcal{M}(B_2)$. The $C_{\surd}$ operator simulates the behaviour of the sequential composition construct.

- recursion

$\mathcal{M}(recX.B) = \nu x.\mathcal{M}(B)$
$\mathcal{M}(X) = x$
$\mathcal{S}(recX.B) = \mathcal{S}(B)[\nu x.\mathcal{M}(B)/x]$
$\mathcal{T}(recX.B) = \mathcal{T}(B)[\nu x.\mathcal{M}(B)/x]$

The meaning of $recX.B$ is the maximal solution of the equation $x = \mathcal{M}(B)$.

The semantics $\mathcal{M}$ is fully abstract with respect to bisimulation semantics of $\mathrm{BPA}_{\delta,rec}$. Since we have a compositional constraint, the fully abstractness relation reduces to: $\mathcal{M}(p) \equiv_{\mathcal{L}} \mathcal{M}(q)$ *if and only if* $p \approx q$.

For the if-direction of the proof we show that the $\mathcal{M}$ function respects the laws of bisimulation on $\mathrm{BPA}_{\delta,rec}$. For the other implication we show the correspondence between the transitions of a generic process and its $\mathcal{M}$-formula using inductive arguments. For the complete proof see the Appendix.

6 Conclusions

We have given a temporal semantics for $\mathrm{BPA}_{\delta,rec}$ fully abstract w.r.t. bisimulation equivalence. The advantages of the temporal semantics approach when providing the semantics of concurrent programs can be summed up in the following observations:

(i) verifying that a program satisfies a given property (expressed by a temporal logic formula) is reduced to verifying that the formula expressing the temporal meaning of the program logically implies the given property formula;

(ii) verifying the equivalence of programs is reduced to verifying logical equivalence.

These observations make it possible to unify these two verification problems, as they amount to checking the validity of a logic formula.

The temporal semantics of $\mathrm{BPA}_{\delta,rec}$ has required the definition of a new logic operator, able to model sequential composition. The obtained logic is obviously more expressive than temporal or modal logics as νCTL and μ calculus; in fact, the addition of the chop operator permits to express context-free processes. The interest in this logic is not limited to the temporal semantics presented in this paper, but it provides a general specification language able to express the behaviour of processes defined by means of sequential composition.

It is therefore our intention to study more deeply the logic $\nu CTL + C_Q$, in particular its expressive power and the applicability to it of algorithms to decide bisimulation equivalences on context-free processes [1,12].

References

[1] J. C. M. Baeten, J. A. Bergstra, J. W. Klop: *Decidibility of bisimulation equivalence for processes generating context-free languages.* Technical Report CS-R8632, CWI, September 1987.

[2] B. Banieqbal, H. Barringer: *Temporal Logic Fixed Point Calculus* in: Proc. Colloquium on Temporal Logic and Specification, Altrincham, England, 1987, Lecture Notes in Computer Science, vol.398, 62-74, 1989.

[3] H. Barringer, R. Kuiper, A. Pnueli: *Now You May Compose Temporal Logic Specifications.* Proceedings 16th ACM Symposium on the Theory of Computing, 1984.

[4] J. A. Bergstra, J. W. Klop: *Algebra of communicating processes with abstraction.* Theoretical Computer Science vol. 37 (1), pp. 77-121, 1985.

[5] J. A. Bergstra, J. W. Klop: *Process algebra: specification and verification in bisimulation semantics.* In: CWI Monograph 4, Proceedings of the CWI Symposium Mathematics and Computer Science II (eds. M. Hazewinkel, J. K. Lenstra & L.G.L.T. Meertens), North-Holland, pp. 61-94, Amsterdam 1986.

[6] J. A. Bergstra & J. W. Klop: *Process theory based on bisimulation semantics.* Springer Lecture Notes in Computer Science vol. 354, pp. 50-122, 1988.

[7] M. C. Browne, E. M. Clarke, O. Grümberg: *Characterizing Finite Kripke Structures in Propositional Temporal Logic.* Theoretical Computer Science vol. 59, pp. 115-131, 1988.

[8] E. M. Clarke, E. A. Emerson: *Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic.* Proceedings of the IBM Workshop on Logic of Programs, Lecture Notes in Computer Science vol. 131, Springer-Verlag, pp. 52-71, New-York 1981.

[9] E. A. Emerson, J. Y. Halpern: *"Sometimes" and "Not Never" Rivisited: On Branching versus Linear Time Temporal Logic.* Journal of ACM vol. 33, pp.151-178, January 1986.

[10] A. Fantechi, S. Gnesi, G. Ristori: *Compositionality and Bisimulation: a negative result* Information Processing Letters, vol. 39, pp. 109-114, 1991.

[11] J. F. Groote, F. W. Vaandrager: *Structured Operational Semantics and Bisimulation as a Congruence.* Proceedings 16th ICALP, Lecture Notes in Computer Science, vol. 372, pp.423-438, 1989.

[12] H. Hüttel, C. Stirling: *Actions Speak Louder than Words: Proving Bisimilarity of Context-Free Processes.* Proceedings LICS 91, Computer Society Press, 1991.

[13] B. Jonsson, H. Khan, J. Parrow: *Implementing a Model Checking Algorithm by Adapting Existing Automated Tools.* Springer Lecture Notes in Computer Science vol. 407, pp. 179-188, 1990.

[14] K. G. Larsen: *Proof Systems for Satisfiability in Hennessy-Milner Logic with Recursion.* , Theoretical Computer Science vol. 72, pp. 265-288, North-Holland 1990.

[15] D. M. R. Park: *Concurrency and automata on infinite sequences.* Proceedings 5th GI Conference, Springer Lecture Notes in Computer Science vol. 104, 1981.

[16] A. Pnueli, *The Temporal Semantics of Concurrent Programs* Theoretical Computer Science, vol.13, 1981, pp.45-60.

[17] A. Pnueli: *Linear and Branching Structures in the Semantics and Logic of Reactive Systems.* Proceedings of 12th ICALP, Lecture Notes in Computer Science vol. 194, July 1985.

[18] C. Stirling, *A Complete Compositional Modal Proof System for a Subset of CCS* Proceedings of 12th ICALP, Lecture Notes in Computer Science vol. 194, July 1985, pp. 475-486.

[19] G. Winskel, *Compositional Checking of Validity of Finite State Processes* Workshop on Concurrency and Compositionality, San Miniato, March 1990.

Appendix

Proof of Theorem 4.1

Lemma A.1 $p \overset{a}{\not\longrightarrow} \forall a$ *iff* $\mathcal{S}(p) \equiv true$.
Proof.

By structural induction; *only if*:

(i) $p = \delta$
$\delta \overset{a}{\not\longrightarrow} \forall a$ and $\mathcal{S}(\delta) = true$;

(ii) $p = p' + p''$
$p' + p'' \overset{a}{\not\longrightarrow} \forall a$ implies $p' \overset{a}{\not\longrightarrow} \forall a$ and $p'' \overset{a}{\not\longrightarrow} \forall a$. We have, for the induction hypothesis: $\mathcal{S}(p') \equiv true$ and $\mathcal{S}(p'') \equiv true$ and then $\mathcal{S}(p' + p'') = \mathcal{S}(p') \wedge \mathcal{S}(p'') \equiv true$;

(iii) $p = p' \cdot p''$
$p' \cdot p'' \overset{a}{\not\longrightarrow} \forall a$ implies $p' \overset{a}{\not\longrightarrow} \forall a$. For the induction hypothesis $\mathcal{S}(p') \equiv true$ and then $\mathcal{S}(p' \cdot p'') = \mathcal{S}(p')\, C_{\surd}\, \mathcal{M}(p'') \equiv true\, C_{\surd}\, \mathcal{M}(p'') \equiv true$;

(iv) $p = recX.q(X)$
$recX.q(X) \overset{a}{\not\longrightarrow} \forall a$ implies $q(recX.q(X)) \overset{a}{\not\longrightarrow} \forall a$. For the induction hypothesis $\mathcal{S}(q(recX.q(X))) \equiv true$ and so $\mathcal{S}(recX.q(X)) = \mathcal{S}(q(recX.q(X))) \equiv true$.

if:

(i) $\mathcal{S}(\delta) = true$
$\delta \overset{a}{\not\longrightarrow} \forall a$;

(ii) $\mathcal{S}(p' + p'') = \mathcal{S}(p') \wedge \mathcal{S}(p'') \equiv true$
then we have $\mathcal{S}(p') \equiv true$ ed $\mathcal{S}(p'') \equiv true$ and for the induction hypothesis $p' \overset{a}{\not\longrightarrow} \forall a$ and $p'' \overset{a}{\not\longrightarrow} \forall a$. Therefore $p' + p'' \overset{a}{\not\longrightarrow} \forall a$;

(iii) $\mathcal{S}(p' \cdot p'') = \mathcal{S}(p')\, C_{\surd}\, \mathcal{M}(p'') \equiv true$
for the definition of the operator $C_{\surd}$ we have that $\mathcal{S}(p') \equiv true$. Then for the induction hypotesis $p' \overset{a}{\not\longrightarrow} \forall a$ and so $p' \cdot p'' \overset{a}{\not\longrightarrow} \forall a$;

(iv) $\mathcal{S}(recX.q(X)) \equiv true$
since $\mathcal{S}(q(recX.q(X))) = \mathcal{S}(recX.q(X))$ we have that $\mathcal{S}(q(recX.q(X))) \equiv true$ and for induction hypothesis $q(recX.q(X)) \overset{a}{\not\longrightarrow} \forall a$. Therefore $recX.q(X) \overset{a}{\not\longrightarrow} \forall a$. ∎

Lemma A.2 $p \overset{a}{\not\longrightarrow} \forall a$ *iff* $\mathcal{T}(p) \equiv false$.
Proof.

Is analogous to that of Lemma A.1.

Lemma A.3 $p \overset{a}{\longrightarrow} p'$ *iff* $\mathcal{S}(p) \equiv EX!(a \wedge \mathcal{M}(p')) \wedge s$.
Proof.

By structural induction; *only if*:

(i) $p = \epsilon$
$\epsilon \overset{\surd}{\longrightarrow} \delta$ and $\mathcal{S}(\epsilon) = EX!(\surd \wedge \mathcal{M}(\delta))$;

(ii) $p = a$
$a \overset{a}{\longrightarrow} \epsilon$ and $\mathcal{S}(a) = EX!(a \wedge \mathcal{M}(\epsilon))$;

(iii) $p = p_1 + p_2$
if $p_1 + p_2 \overset{a}{\longrightarrow} p'$ then $p_1 \overset{a}{\longrightarrow} p'$ or $p_2 \overset{a}{\longrightarrow} p'$; for the induction hypothesis we have:
$\mathcal{S}(p_1) \equiv EX!(a \wedge \mathcal{M}(p')) \wedge s'$ or
$\mathcal{S}(p_2) \equiv EX!(a \wedge \mathcal{M}(p')) \wedge s''$.
Therefore $\mathcal{S}(p_1 + p_2) = \mathcal{S}(p_1) \wedge \mathcal{S}(p_2) \equiv EX!(a \wedge \mathcal{M}(p')) \wedge s$;

(iv) $p = p_1 \cdot p_2$
$p_1 \cdot p_2 \overset{a}{\longrightarrow} p'$ implies $p_1 \overset{a}{\longrightarrow} p_1'$ with $p' = p_1' \cdot p_2$; for the induction hypothesis we have that $\mathcal{S}(p_1) \equiv EX!(a \wedge \mathcal{M}(p_1')) \wedge s'$. Therefore $\mathcal{S}(p_1 \cdot p_2) = \mathcal{S}(p_1)\, C_{\surd}\, \mathcal{M}(p_2) = (EX!(a \wedge \mathcal{M}(p_1')) \wedge s')\, C_{\surd}\, \mathcal{M}(p_2) \equiv (EX!(a \wedge \mathcal{M}(p_1'))\, C_{\surd}\, \mathcal{M}(p_2)) \wedge (s'\, C_{\surd}\, \mathcal{M}(p_2)) = EX!(a \wedge \mathcal{M}(p_1' \cdot p_2)) \wedge s = EX!(a \wedge \mathcal{M}(p')) \wedge s$;

(v) $p = recX.q(X)$
$recX.q(X) \overset{a}{\longrightarrow} p'$ if and only if $q(recX.q(X)) \overset{a}{\longrightarrow} p'$. Then for the induction hypothesis $\mathcal{S}(q(recX.q(X))) = EX!(a \wedge \mathcal{M}(p')) \wedge s$ and so $\mathcal{S}(recX.q(X)) = \mathcal{S}(q(recX.q(X))) = EX!(a \wedge \mathcal{M}(p')) \wedge s$.

if:

(i) $p = \epsilon$
$\mathcal{S}(\epsilon) = EX!(\surd \wedge \mathcal{M}(\delta))$ and $\epsilon \overset{\surd}{\longrightarrow} \delta$;

(ii) $p = a$
$\mathcal{S}(a) = EX!(a \wedge \mathcal{M}(\epsilon))$ and $a \overset{a}{\longrightarrow} \epsilon$;

(iii) $p = p_1 + p_2$
$\mathcal{S}(p_1 + p_2) = \mathcal{S}(p_1) \wedge \mathcal{S}(p_2) \equiv EX!(a \wedge \mathcal{M}(p')) \wedge s$; this implies that:
$\mathcal{S}(p_1) \equiv EX!(a \wedge \mathcal{M}(p')) \wedge s_1$ or
$\mathcal{S}(p_2) \equiv EX!(a \wedge \mathcal{M}(p')) \wedge s_2$.
For the induction hypothesis we have $p_1 \overset{a}{\longrightarrow} p'$ or $p_2 \overset{a}{\longrightarrow} p'$ and so $p_1 + p_2 \overset{a}{\longrightarrow} p'$;

(iv) $p = p_1 \cdot p_2$
$\mathcal{S}(p_1 \cdot p_2) \equiv EX!(a \wedge \mathcal{M}(p')) \wedge s = (EX!(a \wedge \mathcal{M}(\epsilon))\, C_{\surd}\, \mathcal{M}(p')) \wedge s = (\mathcal{S}(a)\, C_{\surd}\, \mathcal{M}(p')) \wedge s = \mathcal{S}(a \cdot p') \wedge s$. This implies that $p_1 \cdot p_2 = a \cdot p' + p''$ where $\mathcal{S}(p'') = s$; since $a \cdot p' + p'' \xrightarrow{a} p'$ we have that $p_1 \cdot p_2 \xrightarrow{a} p'$;

(v) $p = recX.q(X)$
$\mathcal{S}(recX.q(X)) = \mathcal{S}(q(recX.q(X))) \equiv EX!(a \wedge \mathcal{M}(p')) \wedge s$; for the induction hypothesis we have that $q(recX.q(X)) \xrightarrow{a} p'$ and so $recX.q(X) \xrightarrow{a} p'$. ∎

Lemma A.4 *$p \xrightarrow{a} p'$ iff $\mathcal{T}(p) \equiv (a \wedge \mathcal{M}(p')) \vee t$.*
Proof.

Is analogous to that of Lemma A.3.

Lemma A.5 *If $p \xrightarrow{a_i} p_i$ $(i = 1, \ldots, n)$ are the only transitions of p then:*

$$\mathcal{S}(p) \equiv \wedge_{i=1,\ldots,n}\, EX!(a_i \wedge \mathcal{M}(p_i)).$$

Proof.

By contradiction.
From the Lemma A.3 we have $\mathcal{S}(p) \equiv \wedge_{i=1,\ldots,n}\, EX!(a_i \wedge \mathcal{M}(p_i)) \wedge s$; if the thesis not hold then we have $s \equiv EX!(b \wedge \mathcal{M}(q)) \wedge s'$ and for all $i = 1, \ldots, n$ $a_i \neq b$ or $\mathcal{M}(p_i) \neq \mathcal{M}(q)$. But for the Lemma A.3 $p \xrightarrow{b} q$ so we have a contradiction. ∎

Lemma A.6 *If $p \xrightarrow{a_i} p_i$ $(i = 1, \ldots, n)$ are the only transitions of p then:*

$$\mathcal{T}(p) \equiv \vee_{i=1,\ldots,n}\, (a_i \wedge \mathcal{M}(p_i)).$$

Proof.

Is analogous to that of Lemma A.5.

Lemma A.7 *If $\mathcal{S}(p) \equiv \wedge_{i=1,\ldots,n}\, EX!(a_i \wedge \mathcal{M}(p_i))$ then $\exists q_1, \ldots, q_m$ such that:*

- $\forall i \in \{1, \ldots, n\}\ \exists j \in \{1, \ldots, m\} : p \xrightarrow{a_i} q_j$ *and* $\mathcal{M}(p_i) \equiv \mathcal{M}(q_j)$;
- $\forall j \in \{1, \ldots, m\}\ \exists i \in \{1, \ldots, n\} : p \xrightarrow{a_i} q_j$ *and* $\mathcal{M}(p_i) \equiv \mathcal{M}(q_j)$.

Proof.

Since $\mathcal{S}(p) \not\equiv$ *true*, we have, for the Lemma A.1, that $\exists q_1, \ldots, q_m, b_1, \ldots, b_m$ such that the transitions of p are $p \xrightarrow{b_j} q_j$ $(j = 1, \ldots, m)$; for the Lemma A.5 we have that $\mathcal{S}(p) \equiv \wedge_{j=1,\ldots,m}\, EX!(b_j \wedge \mathcal{M}(q_j))$. Hence $\wedge_{i=1,\ldots,n}\, EX!(a_i \wedge \mathcal{M}(p_i)) \equiv \wedge_{i=1,\ldots,m}\, EX!(b_j \wedge \mathcal{M}(q_j))$ and this implies the thesis of the Lemma. ∎

Lemma A.8 *If $\mathcal{T}(p) \equiv \vee_{i=1,\ldots,n}\, (a_i \wedge \mathcal{M}(p_i))$ then $\exists q_1, \ldots, q_m$ such that:*

- $\forall i \in \{1,\ldots,n\}\ \exists j \in \{1,\ldots,m\} : p \xrightarrow{a_i} q_j$ *and* $\mathcal{M}(p_i) \equiv \mathcal{M}(q_j)$;
- $\forall j \in \{1,\ldots,m\}\ \exists i \in \{1,\ldots,n\} : p \xrightarrow{a_i} q_j$ *and* $\mathcal{M}(p_i) \equiv \mathcal{M}(p_j)$.

Proof.

Analogous to that of the Lemma A.7.

Lemma A.9 $p \overset{a}{\not\longrightarrow} \forall a$ *iff* $\mathcal{M}(p) \equiv AX\, false$.
Proof.

By structural induction:

(i) $p \neq recX.q(X),\ p \neq p_1 \cdot p_2$
$\mathcal{M}(p) = \mathcal{S}(p) \wedge AX\, \mathcal{T}(p)$ and the result follows immediatly by the Lemmas A.1 and A.2;

(ii) $p = p_1 \cdot p_2$; *only if*:
$p_1 \cdot p_2 \overset{a}{\not\longrightarrow} \forall a$ implies $p_1 \overset{a}{\not\longrightarrow} \forall a$. For the induction hypothesis $\mathcal{M}(p) \equiv AX\, false$ and so $\mathcal{M}(p) = \mathcal{M}(p_1)\, C_{\surd}\, \mathcal{M}(p_2) \equiv AX\, false\, C_{\surd}\, \mathcal{M}(p_2) \equiv AX\, false$.
If:
$\mathcal{M}(p_1 \cdot p_2) = \mathcal{M}(p_1)\, C_{\surd}\, \mathcal{M}(p_2) \equiv AX\, false$
for the definition of the operator $C_{\surd}$ we have that $\mathcal{M}(p_1) \equiv AX\, false$. Therefore for the induction hypothesis $p_1 \overset{a}{\not\longrightarrow} \forall a$ and so we have that $p_1 \cdot p_2 \overset{a}{\not\longrightarrow} \forall a$;

(iii) $p = recX.q(X)$; *only if*:
$recX.q(X) \overset{a}{\not\longrightarrow} \forall a$ implies $q(recX.q(X)) \overset{a}{\not\longrightarrow} \forall a$. For the induction hypothesis we have that $\mathcal{M}(q(recX.q(X))) \equiv AX\, false$ and so $\mathcal{M}(recX.q(X)) = \mathcal{M}(q(recX.q(X))) \equiv AX\, false$.
If:
$\mathcal{M}(recX.q(X)) \equiv AX\, false$
since $\mathcal{M}(recX.q(X)) = \mathcal{M}(q(recX.q(X)))$ we have that $\mathcal{M}(q(recX.q(X))) \equiv AX\, false$ and for the induction hypothesis $q(recX.q(X)) \overset{a}{\not\longrightarrow} \forall a$. This implies $recX.q(X) \overset{a}{\not\longrightarrow} \forall a$. ∎

Lemma A.10 $\mathcal{M}(p) \not\equiv AX\, false$ *implies* $\exists\, p_1,\ldots,p_m, a_1,\ldots,a_m$:

$$\mathcal{M}(p) \equiv \wedge_{i=1,\ldots,m}\, EX!\,(a_i \wedge \mathcal{M}(p_i)) \wedge AX\,(\vee_{i=1,\ldots,m}\,(a_i \wedge \mathcal{M}(p_i))).$$

Proof.

By structural induction:

(i) $p \neq recX.q(X),\ p \neq p_1 \cdot p_2$
$\mathcal{M}(p) = \mathcal{S}(p) \wedge AX\, \mathcal{T}(p)$ and the result follows by the Lemmas A.5, A.6 and A.9;

(ii) $p = p_1 \cdot p_2$
if $\mathcal{M}(p_1 \cdot p_2) \not\equiv AX\,false$ then from the Lemma A.9 we have that $\exists\, p_1, \ldots, p_m$, $a_1, \ldots, a_m$ such that the transitions of $p_1 \cdot p_2$ are $p_1 \cdot p_2 \xrightarrow{a_i} p_i$ $(i = 1, \ldots, m)$; this implies that $\exists\, p'_1, \ldots, p'_m, a_1, \ldots, a_m$ such that $p_1 \xrightarrow{a_i} p'_i$ where $p_i = p'_i \cdot p_2$ $(i = 1, \ldots, m)$. Since from the Lemma A.9 we have $\mathcal{M}(p_1) \not\equiv AX\,false$, for the induction hypothesis we have that $\mathcal{M}(p_1) \equiv \wedge_{i=1,\ldots,m}\, EX!\,(a_i \wedge \mathcal{M}(p'_i)) \wedge AX\,(\vee_{i=1,\ldots,m}\,(a_i \wedge \mathcal{M}(p'_i)))$; therefore $\mathcal{M}(p_1 \cdot p_2) = \mathcal{M}(p_1)\, C_{\surd}\, \mathcal{M}(p_2) \equiv (\wedge_{i=1,\ldots,m}\, EX!\,(a_i \wedge \mathcal{M}(p'_i)) \wedge AX\,(\vee_{i=1,\ldots,m}\,(a_i \wedge \mathcal{M}(p'_i))))\, C_{\surd}\, \mathcal{M}(p_2) \equiv \wedge_{i=1,\ldots,m}\, EX!\,(a_i \wedge \mathcal{M}(p'_i \cdot p_2)) \wedge AX\,(\vee_{i=1,\ldots,m}\,(a_i \wedge \mathcal{M}(p'_i \cdot p_2))) = \wedge_{i=1,\ldots,m}\, EX!\,(a_i \wedge \mathcal{M}(p_i)) \wedge AX\,(\vee_{i=1,\ldots,m}\,(a_i \wedge \mathcal{M}(p_i)))$;

(iii) $p = recX.\,q(X)$
$\mathcal{M}(recX.\,q(X)) = \mathcal{M}(q(recX.\,q(X))) \not\equiv AX\,false$; for the induction hypothesis we have that $\exists\, p_1, \ldots, p_m, a_1, \ldots, a_m$ such that $\mathcal{M}(q(recX.\,q(X))) \equiv \wedge_{i=1,\ldots,m}\, EX!\,(a_i \wedge \mathcal{M}(p_i)) \wedge AX\,(\vee_{i=1,\ldots,m}\,(a_i \wedge \mathcal{M}(p_i)))$. This implies that $\mathcal{M}(recX.\,q(X)) = \mathcal{M}(q(recX.\,q(X))) \equiv \wedge_{i=1,\ldots,m}\, EX!\,(a_i \wedge \mathcal{M}(p_i)) \wedge AX\,(\vee_{i=1,\ldots,m}\,(a_i \wedge \mathcal{M}(p_i)))$. ∎

Lemma A.11 *$\mathcal{M}(p) \equiv \mathcal{M}(q)$ implies p and q are bisimilar ($p \leftrightarroweq q$ in the following).*
Proof.

For the Lemma A.10 we have only two cases:

1. $\mathcal{M}(p) \equiv \mathcal{M}(q) \equiv AX\,false$
   for the Lemma A.9 we have $p \overset{a}{\not\longrightarrow} \forall a$ and $q \overset{a}{\not\longrightarrow} \forall a$ and so $p \leftrightarroweq q$;
2. $\mathcal{M}(p) \equiv \mathcal{M}(q) \equiv \wedge_{i=1,\ldots,m}\, EX!\,(a_i \wedge \mathcal{M}(p_i)) \wedge AX\,(\vee_{i=1,\ldots,m}\,(a_i \wedge \mathcal{M}(p_i)))$
   let $p \xrightarrow{b} p'$; for the Lemma A.5 and A.6 we have that $\exists\, i \in \{1, \ldots, m\}$ such that $a_i = b$ and $\mathcal{M}(p_i) \equiv \mathcal{M}(p')$ and, for the Lemma A.7 and A.8, we have that $\exists\, q'$ such that $q \xrightarrow{a_i} q'$ with $\mathcal{M}(q') \equiv \mathcal{M}(p_i)$. Thus $\mathcal{M}(p') \equiv \mathcal{M}(q')$ and for the induction hypothesis $p' \leftrightarroweq q'$; wecan repeat this reasoning for any transition of p and q and so we have $p \leftrightarroweq q$. ∎

Lemma A.12 *$p \leftrightarroweq q$ implies $\mathcal{M}(p) \equiv \mathcal{M}(q)$.*
Proof.

We will apply the semantic functions $\mathcal{M}$, $\mathcal{S}$ and $\mathcal{T}$ to the left and the right sides of bisimulation's axioms of $\mathrm{BPA}_{\delta,rec}$ and we will show that these functions preserves the equalities:

(i) $x + y = y + x$
$\mathcal{M}(x + y) = \mathcal{S}(x + y) \wedge AX\,\mathcal{T}(x + y) = \mathcal{S}(x) \wedge \mathcal{S}(y) \wedge AX\,(\mathcal{T}(x) \vee \mathcal{T}(y)) \equiv \mathcal{S}(y) \wedge \mathcal{S}(x) \wedge AX\,(\mathcal{T}(y) \vee \mathcal{T}(x)) = \mathcal{S}(y + x) \wedge AX\,\mathcal{T}(y + x) = \mathcal{M}(y + x)$;

(ii) $(x + y) + z = x + (y + z)$
$\mathcal{M}((x + y) + z) = \mathcal{S}((x + y) + z) \wedge AX\,\mathcal{T}((x + y) + z) = \mathcal{S}(x + y) \wedge \mathcal{S}(z) \wedge AX\,(\mathcal{T}(x + y) \vee \mathcal{T}(z)) = \mathcal{S}(x) \wedge \mathcal{S}(y) \wedge \mathcal{S}(z) \wedge AX\,(\mathcal{T}(x) \vee \mathcal{T}(y) \vee \mathcal{T}(z)) \equiv \mathcal{S}(x) \wedge \mathcal{S}(y + z) \wedge AX\,(\mathcal{T}(x) \vee \mathcal{T}(y + z)) \equiv \mathcal{S}(x + (y + z)) \wedge AX\,\mathcal{T}(x + (y + z)) = \mathcal{M}(x + (y + z))$;

(iii) $x + x = x$
$\mathcal{M}(x+x) = \mathcal{S}(x+x) \wedge AX\,\mathcal{T}(x+x) = \mathcal{S}(x) \wedge \mathcal{S}(x) \wedge AX\,(\mathcal{T}(x) \vee \mathcal{T}(x)) \equiv \mathcal{S}(x) \wedge AX\,\mathcal{T}(x) = \mathcal{M}(x)$;

(iv) $(x + y) \cdot z = x \cdot z + y \cdot z$
$\mathcal{M}((x+y)\cdot z) = \mathcal{M}(x+y)\,C_{\surd}\,\mathcal{M}(z) = (\mathcal{S}(x+y) \wedge AX\,\mathcal{T}(x+y))\,C_{\surd}\,\mathcal{M}(z) = (\mathcal{S}(x) \wedge \mathcal{S}(y) \wedge AX\,(\mathcal{T}(x) \vee \mathcal{T}(y)))\,C_{\surd}\,\mathcal{M}(z) \equiv (\mathcal{S}(x)\,C_{\surd}\,\mathcal{M}(z)) \wedge (\mathcal{S}(y)\,C_{\surd}\,\mathcal{M}(z)) \wedge AX\,((\mathcal{T}(x)\,C_{\surd}\,\mathcal{M}(z)) \vee (\mathcal{T}(y)\,C_{\surd}\,\mathcal{M}(z))) = \mathcal{S}(x\cdot z) \wedge \mathcal{S}(y\cdot z) \wedge AX\,(\mathcal{T}(x\cdot z) \vee \mathcal{T}(y\cdot z)) = \mathcal{M}(x\cdot z + y\cdot z)$;

(v) $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
$\mathcal{M}((x\cdot y)\cdot z) = \mathcal{M}(x\cdot y)\,C_{\surd}\,\mathcal{M}(z) = (\mathcal{M}(x)\,C_{\surd}\,\mathcal{M}(y))\,C_{\surd}\,\mathcal{M}(z) \equiv \mathcal{M}(x)\,C_{\surd}\,(\mathcal{M}(y)\,C_{\surd}\,\mathcal{M}(z)) = \mathcal{M}(x)\,C_{\surd}\,\mathcal{M}(y\cdot z) = \mathcal{M}(x\cdot(y\cdot z))$;

(vi) $x + \delta = x$
$\mathcal{M}(x+\delta) = \mathcal{S}(x+\delta) \wedge AX\,\mathcal{T}(x+\delta) = \mathcal{S}(x) \wedge \mathcal{S}(\delta) \wedge AX\,(\mathcal{T}(x) \vee \mathcal{T}(\delta)) = \mathcal{S}(x) \wedge \mathit{true} \wedge AX\,(\mathcal{T}(x) \vee \mathit{false}) \equiv \mathcal{S}(x) \wedge AX\,\mathcal{T}(x) = \mathcal{M}(x)$;

(vii) $\delta \cdot x = \delta$
$\mathcal{M}(\delta\cdot x) = \mathcal{M}(\delta)\,C_{\surd}\,\mathcal{M}(x) = (AX\,\mathit{false})\,C_{\surd}\,\mathcal{M}(x) \equiv AX\,\mathit{false} = \mathcal{M}(\delta)$;

(viii) $recX.p(X) = p(recX.p(X))$
$\mathcal{M}(recX.p(X)) = \nu x.\,\mathcal{S}(p(X)) \wedge AX\,\mathcal{T}(p(X)) \equiv \mathcal{S}(p(recX.p(X))) \wedge AX\,\mathcal{T}(p(recX.p(X))) = \mathcal{M}(p(recX.p(X)))$. ∎

Theorem A.13 *The temporal semantics $\mathcal{M}$ is fully abstract with respect to bisimulation semantics of* $\mathrm{BPA}_{\delta,\mathrm{rec}}$, *i.e.* $\mathcal{M}(p) \equiv \mathcal{M}(q)$ *if and only if* $p \leftrightarrows q$.
Proof.

Follow by Lemmas A.11 and A.12.