Propositional Dynamic Logics for Communicating Concurrent Programs with CCS's Parallel Operator*

Mario R. F. Benevides† L. Menasché Schechter‡

December 18, 2013

Abstract

This work presents three increasingly expressive Dynamic Logics in which the programs are described in a language based on CCS. Our goal is to build dynamic logics that are suitable for the description and verification of properties of communicating concurrent systems, in a similar way as PDL is used for the sequential case. In order to accomplish that, CCS's operators and constructions are added to a basic modal logic. Doing this, the semantics of CCS's parallel operator allows us to build dynamic logics that support communicating and concurrent programs. We build a simple Kripke semantics for these logics, provide complete axiomatizations for them and show that they have the finite model property. This contrasts with other dynamic logics with parallel operators presented in the literature, such as Peleg's Concurrent PDL with Channels, where either the parallel programs cannot communicate, or at least one of the properties mentioned above (simple Kripke semantics, complete axiomatization and finite model property) is missing.

Keywords: Dynamic Logic, Concurrency, Kripke Semantics, Axiomatization, Completeness

1 Introduction

Propositional Dynamic Logic (PDL) [7] plays an important role in formal specification and reasoning about sequential programs and systems. PDL is a multi-modal logic with one modality $\langle\pi\rangle$ for each program π. The logic has a set of basic programs and a set of operators (sequential composition, iteration and nondeterministic choice) that are used to inductively build the set of non-basic programs. PDL has been used to describe and verify properties and behavior of sequential programs and systems. A Kripke semantics can be provided, with a frame $\mathcal{F} = (W, \{R_\pi\}_{\pi\in\Pi})$, where W is a non-empty set of possible program states and, for each program $\pi \in \Pi$, R_π is a binary relation on W such that $(s, t) \in R_\pi$ if and only if there is a computation of π starting in s and terminating in t.

*This work was supported by the Brazilian research agencies CNPq, CAPES and FAPERJ.

†Department of Computer Science and Systems and Computer Engineering Program, Federal University of Rio de Janeiro, Brazil. E-mail: mario@cos.ufrj.br.

‡Department of Computer Science, Federal University of Rio de Janeiro, Brazil. E-mail: luisms@dcc.ufrj.br. *Corresponding author.*

The Calculus for Communicating Systems (CCS) is a well known process algebra, proposed by Robin Milner [14], for the specification of communicating concurrent systems. It models the concurrency and interaction between processes through individual acts of communication. A pair of processes can communicate through a common channel and each act of communication consists simply of a signal being sent at one end of the channel and immediately being received at the other. A CCS specification is a description (in the form of algebraic equations) of the behavior expected from a system, based on the communication events that may occur. As in PDL, CCS has a set of operators (action prefix, parallel composition, nondeterministic choice and restriction on acts of communication) that are used to inductively build process specifications from a set of basic actions. Iteration can also be described through the use of recursive equations.

This work presents three increasingly expressive Dynamic Logics in which the programs are described in a language based on CCS. There are, in the literature, some logics that make use of CCS or other process algebras. However, they use these process algebras as a language for the description of frames and models, while using standard modal logics for the description of properties (see, for example, [14] and [16]). The logics that we develop in the present work use CCS in a distinct way. Its operators and constructions are *added* to a basic modal logic in order to create dynamic logics that are suitable for the description and verification of properties of communicating, concurrent and non-deterministic programs and systems, in a similar way as PDL is used for the sequential case.

Thus, it should be emphasized that the contribution of this work is on the field of dynamic logics and not on the field of process algebras. From process algebras, we just borrow a set of operators that are suitable for the description of communication and concurrency (in particular, CCS's parallel operator is well-suited for this). We use these operators because they have a well-established theory behind them and we can use many of its concepts and results to help us build our logics (many concepts from the theory of CCS will be useful to us).

Our paper falls in the broad category of works that attempt to generalize PDL and build dynamic logics that deal with classes of non-regular programs. As examples of other works in this area, we can mention [11], [10] and [12], that develop decidable dynamic logics for fragments of the class of context-free programs and [13], [9], [18], [17] and [6], that develop dynamic logics for classes of programs with some sort of concurrency.

Our logics are related to Concurrent PDL (CPDL) [18], Channel-CPDL [17] and the logic developed in [6], but have advantages over them. The first can only describe properties of concurrent systems with no communication between the components. While the second is able to describe interesting properties of communicating concurrent systems, it does not have a simple Kripke semantics (in fact, "a formal definition of the semantics of channel-CPDL is rather complicated" [17]). Besides that, its satisfiability problem is undecidable (Π^1_1-hard), which also implies that it does not have a complete axiomatization. The third logic mentioned above makes a semantic distinction between *final* and *non-final* states, which makes its semantics and its axiomatization rather complex. On the other hand, due to the use of the CCS mechanisms of communication and concurrency, our logics have simple Kripke semantics, the finite model property and straightforward axiomatizations. Besides that, our logics can also be seen as extensions of PDL with Interleaving (iPDL)

[13]. In iPDL, the parallel operator that is present in the logic is similar to the parallel operator of CCS, but it only allows interleaving of the actions in parallel programs, while the parallel operator of CCS (in conjunction with the restriction operator) also allows communication and synchronization.

We choose to base our logics in the mechanisms of communication and concurrency of CCS, instead of some other process algebra, for two reasons. First, CCS is built with the philosophy that only those operators that are essential to the description of the basic behaviors of communication and concurrency should be included as primitives in the language, while the operators and behaviors of greater complexity should be derived from the basic ones. Using a small language like CCS, where only the more basic constructions are present, we can study in details what are the problems that may arise when we try to use its operators to build a dynamic logic and what operators and constructions we need to add or remove to correct these problems. Second, the development of dynamic logics using CCS operators can be used as a natural stepping stone to the development of dynamic logics that use the π-Calculus [15] operators and messaging mechanisms. The π-Calculus is a very powerful process algebra that is able to describe not only non-determinism and concurrency, but also *mobility* of programs. The π-Calculus can also be used to encode some powerful programming paradigms, as object-oriented programming and functional programming (λ-Calculus) [15].

The rest of this paper is organized as follows. In section 2, we introduce the necessary background concepts: Propositional Dynamic Logic and the Calculus for Communicating Systems (CCS). Our first logic, together with a complete axiomatic system, is presented in section 3. In this logic, we do not use CCS constants or the CCS restriction operator in the language. In section 4, we present our second logic, in which we allow the presence of CCS constants in the language. We also give an axiomatization for this second logic and prove its completeness using a Fischer-Ladner construction.

The third logic, together with a complete axiomatization for it, is presented in section 5. In this logic, we develop a deeper mix of the traditional PDL operators (like the Kleene star) with the CCS operators, which allows us to solve some issues that appear in the previous logics. This logic is called PPDL*. One of the main results that we show, through the use of reduction axioms, is that every PPDL* formula is equivalent to a PDL formula. However, our result also shows that, while the concurrent operator may be removed from the formulas, in practice it can be very hard to describe a complex concurrent behavior without this operator from the start. Besides that, even though both formulas, with and without the concurrency operator, may be equivalent, the one with it will be much more succinct.

In the last part of section 5 (subsections 5.4 and 5.5), we present two examples of applications of our logics. The first one is an example of a specification of a simple protocol inspired in [19, 24] and illustrates the use of our logics to express some desirable properties of this specification. The second one is based on van Benthem's paradigm of games-as-processes [20] and illustrates a possible use of our logics to the description of simultaneous games, following the ideas presented in [21]. Finally, in section 6, we state our final remarks.

2 Background

This section presents a brief overview of two topics on which the later development is based. First, we make a brief review of the syntax and semantics of PDL [7]. Second, we present the process algebra CCS together with some useful concepts, properties and results from its theory. We do not assume a familiarity with CCS, since process algebras are by no means an universally studied topic among (modal) logicians. We introduce here all that is necessary for our presentation in the next sections, trying to make this work as self-contained as possible.

2.1 Propositional Dynamic Logic

In this section, we present the syntax and semantics of PDL.

Definition 2.1. *The PDL language consists of a set Φ of countably many proposition symbols, a set Π of countably many basic programs, the boolean connectives $\neg$ and $\wedge$, the program constructors* ;, $\cup$ *and* * *and a modality $\langle\pi\rangle$ for every program π. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\pi\rangle\varphi, \text{ with } \pi ::= a \mid \pi_1;\pi_2 \mid \pi_1 \cup \pi_2 \mid \pi^*,$$

where $p \in \Phi$ and $a \in \Pi$.

In all the logics that appear in this paper, we use the standard abbreviations $\bot \equiv \neg\top$, $\varphi \vee \phi \equiv \neg(\neg\varphi \wedge \neg\phi)$, $\varphi \to \phi \equiv \neg(\varphi \wedge \neg\phi)$ and $[\pi]\varphi \equiv \neg\langle\pi\rangle\neg\varphi$.

Definition 2.2. *A* frame *for PDL is a tuple $\mathcal{F} = (W, \{R_a\}_{a\in\Pi})$ where*

- *W is a non-empty set of states;*
- *R_a is a binary relation over W, for each basic program $a \in \Pi$;*
- *Besides the above relations, we also inductively build binary relations R_π, for each non-basic program π, using the following rules:*
  - $R_{\pi_1;\pi_2} = R_{\pi_1} \circ R_{\pi_2}$;
  - $R_{\pi_1\cup\pi_2} = R_{\pi_1} \cup R_{\pi_2}$;
  - $R_{\pi^*} = R_\pi^*$, *where R_π^* denotes the reflexive transitive closure of R_π.*

Definition 2.3. *A* model *for PDL is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is a PDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V} : \Phi \mapsto 2^W$.*

The semantical notion of satisfaction for PDL is defined as follows:

Definition 2.4. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of* satisfaction *of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, can be inductively defined as follows:*

- $\mathcal{M}, w \Vdash p$ *iff* $w \in \mathbf{V}(p)$;
- $\mathcal{M}, w \Vdash \top$ *always;*
- $\mathcal{M}, w \Vdash \neg\varphi$ *iff* $\mathcal{M}, w \nVdash \varphi$;

- $\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ *iff* $\mathcal{M}, w \Vdash \varphi_1$ *and* $\mathcal{M}, w \Vdash \varphi_2$*;*
- $\mathcal{M}, w \Vdash \langle \pi \rangle \varphi$ *iff there is* $w' \in W$ *such that* $w R_\pi w'$ *and* $\mathcal{M}, w' \Vdash \varphi$*.*

It should be noticed that, opposed to the standard presentation of PDL, we do not use the test operator (?) in the language. As CCS does not have a test operator, we choose to leave it out of our presentation, so we could simplify it a little. We feel that the addition of the test operator is an interesting topic of study, but leave it to future work.

2.2 Calculus for Communicating Systems

The Calculus for Communicating Systems (CCS) is a well known process algebra, proposed by Robin Milner [14], for the specification of communicating concurrent systems. It models the concurrency and interaction between processes through individual acts of communication. A CCS specification is a description (in the form of algebraic equations) of the behavior expected from a system, based on the communication events that may occur. For a broad introduction to CCS, [14] can be consulted.

In CCS, a pair of processes can communicate through a common channel and each act of communication consists simply of a signal being sent at one end of the channel and immediately being received at the other.

Let $\mathcal{N} = \{a, b, c, \dots\}$ be a countable set of names. Each channel in a CCS specification is labeled by a name. The labels of the channels are also used to describe the communication actions (sending and receiving signals) performed by the processes. Let $a \in \mathcal{N}$. An action of the form a is called an *input action* and denotes that the process receives a signal through the channel labeled by a. An action of the form $\overline{a}$ is called an *output action* and denotes that the process sends a signal through the channel labeled by a. Besides these communication actions, CCS has only one other action: the silent action, denoted by τ, used to represent any internal action performed by any of the processes that does not involve an act of communication (e.g.: a memory update).

There are two possible semantics for the τ action in CCS: it can be regarded as being observable, in the same way as the communication actions, or it can be regarded as being invisible. We adopt the first one, since it is more generic: in our logical formalism, we are able to represent the second semantics as a particular case of the first.

Definition 2.5. *In our presentation of CCS, process specifications can be built using the following operations:*

$$P ::= \alpha \mid \alpha.P \mid \alpha.A \mid P_1 + P_2 \mid P_1|P_2 \mid P \backslash L,$$

with

$$\alpha ::= a \mid \overline{a} \mid \tau,$$

where $a \in \mathcal{N}$*,* $L \subseteq \mathcal{N}$ *and every constant* A *(taken from a countable set) has a unique* defining equation $A \stackrel{def}{=} P_A$*, where* P_A *is a process specification.*

Remark 2.6. *In this work, every time that a process is linked to a constant* A *through a defining equation, it will be denoted by* P_A*.*

Table 1: Transition Relations of CCS

$\alpha \xrightarrow{\alpha} \surd$	$\alpha.P \xrightarrow{\alpha} P$	$\frac{A \overset{def}{=} P_A}{\alpha.A \xrightarrow{\alpha} P_A}$	$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	$\frac{Q \xrightarrow{\beta} Q'}{P+Q \xrightarrow{\beta} Q'}$
$\frac{P \xrightarrow{\alpha} P'}{P\|Q \xrightarrow{\alpha} P'\|Q}$	$\frac{Q \xrightarrow{\beta} Q'}{P\|Q \xrightarrow{\beta} P\|Q'}$	$\frac{P \xrightarrow{\lambda} P', Q \xrightarrow{\overline{\lambda}} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}$	$\frac{P \xrightarrow{\alpha} P', \alpha \notin L \cup \overline{L}}{P \backslash L \xrightarrow{\alpha} P' \backslash L}$	

The *prefix* operator (.) denotes that the process will first perform the action α and then behave as P or A. The *summation* (or *nondeterministic choice*) operator (+) denotes that the process will make a nondeterministic choice to behave as either P_1 or P_2. The *parallel composition* operator (|) denotes that the processes P_1 and P_2 may proceed independently or may communicate through a common channel. Finally, the *restriction* operator (\) denotes that the channels in L are only accessible inside P. Iteration in CCS is modeled through recursive defining equations, i.e., equations $A \overset{def}{=} P_A$ where A occurs in P_A.

Originally, CCS also defines a null process, denoted by $\mathbf{0}$. It represents the process that is unable to perform any actions. However, because of its somewhat loose definition, which fails to differentiate between a deadlock and a successful termination (unlike other process algebras, as ACP [8] for instance, in which the deadlocked process and the terminated process are different), its use would bring a serious inconvenience to the semantics of our first two logics: the semantics would not be fully compositional. This is shown in details in the next section. Because of that, we drop this null process until our third logic, when we extend CCS with new operators and partially redefine its semantics, obtaining a null process with a much better algebraic behavior. To completely drop the null process, we must also drop the restriction operator, as it may be used to define such a process (e.g. $a \backslash \{a\}$). Hence, the restriction operator will also only be present in our third logic.

We write $P \xrightarrow{\alpha} P'$ to express that the process P can perform the action α and after that behave as P'. We write $P \xrightarrow{\alpha} \surd$ to express that the process P successfully finishes after performing the action α (a notation borrowed from ACP).

A process finishes when there is no possible action left for it to perform. For example, $\beta \xrightarrow{\beta} \surd$. When a process finishes inside a parallel composition, we write P instead of $P|\surd$. We also write $\surd$ instead of $\surd \backslash L$ and $\surd|\surd$. We define the set $\overline{L}$ as $\overline{L} = \{\overline{a} : a \in L\}$. In table 1, we present the semantics for the operators based on this notation. In this table, P, Q and P_A are process specifications, while P' and Q' are process specifications or $\surd$.

In order to motivate the use of CCS, we present a simple example of the use of the language below. Here, we are still using CCS outside of the logical formalisms that are presented in the next sections.

Example 2.7 ([14, 19])**.** *Consider a vending machine where one can put coins of one or two euro and buy a little or a big chocolate bar. After inserting the coins, one must press the little button for a little chocolate or the big button for a big chocolate. The machine is also programmed to shutdown on its own following some internal protocol (represented by a τ action). A CCS term describing the behavior of this machine is the following:*

$$V = 1e.little.\overline{collect}.A + 1e.1e.big.\overline{collect}.A + 2e.big.\overline{collect}.A$$

$$A \stackrel{def}{=} 1e.little.\overline{collect}.A + 1e.1e.big.\overline{collect}.A + 2e.big.\overline{collect}.A + \tau$$

Let us now suppose that Chuck wants to use this vending machine. We could describe Chuck as

$$C = \overline{1e}.\overline{little}.collect + \overline{1e}.\overline{1e}.\overline{big}.collect + \overline{2e}.\overline{big}.collect.$$

Notice that Chuck does not have an iterative behavior. Once he collects the chocolate, he is done. Now, if we want to model the process of Chuck buying a chocolate from the vending machine, we could write $(V|C)\backslash L$*, where* $L = \{1e, 2e, little, big, collect\}$*.*

Definition 2.8. *Let* $\mathcal{P}$ *be the set of all possible process specifications. A set* $Z \subseteq \mathcal{P} \times \mathcal{P}$ *is a* strong bisimulation *if* $(P, Q) \in Z$ *implies the following:*

- *If* $P \xrightarrow{\alpha} P'$ *and* $P' \in \mathcal{P}$*, then there is* $Q' \in \mathcal{P}$ *such that* $Q \xrightarrow{\alpha} Q'$ *and* $(P', Q') \in Z$*;*
- *If* $Q \xrightarrow{\alpha} Q'$ *and* $Q' \in \mathcal{P}$*, then there is* $P' \in \mathcal{P}$ *such that* $P \xrightarrow{\alpha} P'$ *and* $(P', Q') \in Z$*;*
- $P \xrightarrow{\alpha} \surd$ *if and only if* $Q \xrightarrow{\alpha} \surd$*.*

The concept of bisimulation is a key notion in any process algebra. It is an equivalence relation between processes. The intuition is that two bisimilar processes cannot be distinguished by an external observer.

Definition 2.9. *Two process specifications* P *and* Q *are* strongly bisimilar *(or simply* bisimilar*), denoted by* $P \sim Q$*, if there is a strong bisimulation* Z *such that* $(P, Q) \in Z$*.*

Now, we introduce the Expansion Law, which is very important in the definition of the semantics of our logics in the next sections and in their axiomatizations. We present a particular case of the Expansion Law, which is suited to our needs. The most general case of the Expansion Law is presented in [14].

Definition 2.10. *We say that a process is* unrestricted *if it has no occurrences of the* $\backslash$ *operator.*

Theorem 2.11 (Expansion Law (EL))**.** *Let* $P = P_1 \mid P_2$*, where* P *is unrestricted. Then*

$$P \sim \sum_{P_1 \xrightarrow{\alpha} P_1'} \alpha.(P_1' \mid P_2) + \sum_{P_2 \xrightarrow{\beta} P_2'} \beta.(P_1 \mid P_2') + \sum_{R \in A_\tau} \tau.R,$$

where $A_\tau = \{(P_1' \mid P_2') : P_1 \xrightarrow{a} P_1' \text{ and } P_2 \xrightarrow{\overline{a}} P_2', \text{ for some } a \in \mathcal{N}\} \cup \{(P_1' \mid P_2') : P_1 \xrightarrow{\overline{a}} P_1' \text{ and } P_2 \xrightarrow{a} P_2', \text{ for some } a \in \mathcal{N}\}$*. We denote the right side of this bisimilarity by* $Exp(P)$*.*

It should again be emphasized that the Expansion Law presented above is not in its most general form (which can be seen in [14]). The condition that process P is unrestricted allows for a simpler algebraic formula, presented above, and is sufficient for our needs in the axiomatizations of our logics.

The Expansion Law is a very useful property of CCS processes. Its intuition is that processes can be rewritten as a summation of all their possible actions. Suppose we have two processes $A \stackrel{def}{=} a.A'$ and $B \stackrel{def}{=} \overline{a}.B'$. Then, using the Expansion Law, we have that

$$(A \mid B) \sim a.(A' \mid B) + \overline{a}.(A \mid B') + \tau.(A' \mid B').$$

2.3 Action Sequences and Possible Runs

In this section, we introduce the key concept of *finite possible runs* of a process. This concept plays a central role in the semantics of our logics.

Semantics defined in terms of possible runs of programs are sometimes called *trace semantics*. One interesting discussion on trace semantics for *sequential* programs appears on [23]. It defines and compares various semantics for concrete sequential processes and provides algebraic axiomatizations and semantical modal characterizations (no modal axiomatization) for them.

Definition 2.12. *We use the notation $\overrightarrow{\alpha}$ to denote a potentially infinite sequence of actions $\alpha_1.\alpha_2.\cdots.\alpha_n(.\cdots)$ (the empty sequence is denoted by $\overrightarrow{\varepsilon}$). The empty sequence follows the rule $\overrightarrow{\alpha}.\overrightarrow{\varepsilon} = \overrightarrow{\varepsilon}.\overrightarrow{\alpha} = \overrightarrow{\alpha}$, for all $\overrightarrow{\alpha}$. We denote the i-th term of the sequence $\overrightarrow{\alpha}$ by $(\overrightarrow{\alpha})_i$.*

Definition 2.13. *We say that a finite sequence of actions $\overrightarrow{\beta}$ is a prefix of $\overrightarrow{\alpha}$ if there is a non-empty sequence $\overrightarrow{\lambda}$ such that $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$. If $\overrightarrow{\beta}$ is a prefix of $\overrightarrow{\alpha}$, we write $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$.*

Definition 2.14. *We write $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} P'$ to express that the process P may perform the sequence of actions $\overrightarrow{\alpha}$ and after that behave as P'. We write $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$ to express that the process P may successfully finish after performing the sequence of actions $\overrightarrow{\alpha}$ (this, in particular, implies that $\overrightarrow{\alpha}$ is finite).*

Definition 2.15. *We define the set of finite possible runs of a process P, denoted by $\overrightarrow{\mathcal{R}_f}(P)$, as $\overrightarrow{\mathcal{R}_f}(P) = \{\overrightarrow{\alpha} : P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd\}$.*

We want to define semantics for our logics that only take into account the *finite* possible runs of the processes, i.e., situations in which the processes successfully finish. Thus, we present some useful results about finite possible runs.

Definition 2.16. *Let R and S be sets of finite sequences of actions. We can define the following operations on these sets:*

1. $R \circ S = \{\overrightarrow{\alpha}.\overrightarrow{\beta} : \overrightarrow{\alpha} \in R \text{ and } \overrightarrow{\beta} \in S\}$;
2. $R \cup S = \{\overrightarrow{\alpha} : \overrightarrow{\alpha} \in R \text{ or } \overrightarrow{\alpha} \in S\}$;
3. $R^0 = \{\overrightarrow{\varepsilon}\}$, $R^n = R \circ R^{n-1} (n \geq 1)$;
4. $R^* = \bigcup_{n \in \mathbb{N}} R^n$.

Lemma 2.17. *If $P \sim Q$, then $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$ if and only if $Q \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$.*

Proof. We prove this by induction on the length n of $\overrightarrow{\alpha}$. If $n = 0$, then $\overrightarrow{\alpha} = \overrightarrow{\varepsilon}$ and neither P nor Q may successfully finish without executing any action. If $n = 1$, then $\overrightarrow{\alpha} = \alpha$, for some action α. Then, $P \overset{\overrightarrow{\alpha}}{\Rightarrow} \surd \Leftrightarrow P \overset{\alpha}{\rightarrow} \surd$. By the hypothesis that $P \sim Q$, $P \overset{\alpha}{\rightarrow} \surd \Leftrightarrow Q \overset{\alpha}{\rightarrow} \surd$. Finally, $Q \overset{\alpha}{\rightarrow} \surd \Leftrightarrow Q \overset{\overrightarrow{\alpha}}{\Rightarrow} \surd$.

Suppose that the theorem is true for all $n < k$. Let $\overrightarrow{\alpha}$ be a sequence of length k. Let α be the first action of the sequence and let $\overrightarrow{\beta}$ be a sequence of length $k-1$ such that $\overrightarrow{\alpha} = \alpha.\overrightarrow{\beta}$. Then, $P \overset{\overrightarrow{\alpha}}{\Rightarrow} \surd$ if and only if there is a process P' such that $P \overset{\alpha}{\rightarrow} P'$ and $P' \overset{\overrightarrow{\beta}}{\Rightarrow} \surd$. But if $P \overset{\alpha}{\rightarrow} P'$ and $P \sim Q$, then there is a process Q' such that $Q \overset{\alpha}{\rightarrow} Q'$ and $P' \sim Q'$. Now, $\overrightarrow{\beta}$ is a sequence of length shorter than k, so by the induction hypothesis, as $P' \sim Q'$ and $P' \overset{\overrightarrow{\beta}}{\Rightarrow} \surd$, then $Q' \overset{\overrightarrow{\beta}}{\Rightarrow} \surd$. This means that $Q \overset{\overrightarrow{\alpha}}{\Rightarrow} \surd$, proving the theorem. □

Theorem 2.18. *If $P \sim Q$, then $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$.*

Proof. Suppose that $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$. Then, $P \overset{\overrightarrow{\alpha}}{\Rightarrow} \surd$. As $P \sim Q$, this implies, by lemma 2.17, that $Q \overset{\overrightarrow{\alpha}}{\Rightarrow} \surd$, which means that $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(Q)$. Thus, $\overrightarrow{\mathcal{R}_f}(P) \subseteq \overrightarrow{\mathcal{R}_f}(Q)$. The proof that $\overrightarrow{\mathcal{R}_f}(Q) \subseteq \overrightarrow{\mathcal{R}_f}(P)$ is entirely analogous. □

3 sPPDL

This section presents our first dynamic logic for communicating concurrent programs. In this logic, we do not use CCS constants or the CCS restriction operator in the language. We call this logic Small Parallel PDL or sPPDL. Our goal here is to introduce a simple logic and discuss some of the issues concerning the axioms and the relational interpretation of the formulas.

3.1 Language and Semantics

In this section, we present the syntax and semantics of sPPDL.

Definition 3.1. *The sPPDL language consists of a set Φ of countably many proposition symbols, a set $\mathcal{N}$ of countably many names, the silent action τ, the boolean connectives $\neg$ and $\wedge$, the CCS operators ., + and | and a modality $\langle P \rangle$ for every process P. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle P \rangle \varphi, \text{ with } P ::= \alpha \mid \alpha.P \mid P_1 + P_2 \mid P_1 | P_2,$$

where $p \in \Phi$ and $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$.

Definition 3.2. *A* frame *for sPPDL is a tuple $\mathcal{F} = (W, \{R_\alpha\}_{\alpha \in \mathcal{L}})$ where*

- *W is a non-empty set of states;*
- *R_α is a binary relation over W for each basic action $\alpha \in \mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$.*

Definition 3.3. *A* model *for sPPDL is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is an sPPDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V} : \Phi \mapsto 2^W$.*

We now define the semantical notion of satisfaction for sPPDL as follows:

Definition 3.4. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of* satisfaction *of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, can be inductively defined as follows:*

- $\mathcal{M}, w \Vdash p$ *iff* $w \in \mathbf{V}(p)$;
- $\mathcal{M}, w \Vdash \top$ *always;*
- $\mathcal{M}, w \Vdash \neg\varphi$ *iff* $\mathcal{M}, w \nVdash \varphi$;
- $\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ *iff* $\mathcal{M}, w \Vdash \varphi_1$ *and* $\mathcal{M}, w \Vdash \varphi_2$;
- $\mathcal{M}, w \Vdash \langle P \rangle \varphi$ *iff there is a finite path* $(v_0, v_1, \ldots, v_n)$, $n \geq 1$, *such that* $v_0 = w$, $\mathcal{M}, v_n \Vdash \varphi$ *and there is* $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ *of length* n *such that* $(v_{i-1}, v_i) \in R_{(\overrightarrow{\alpha})_i}$, *for* $1 \leq i \leq n$. *We say that such* $\overrightarrow{\alpha}$ matches *the path* $(v_0, \ldots, v_n)$.

As we use the duals of the operators $\langle P \rangle$ in our axiomatization, we also present the satisfaction condition for formulas of the form $[P]\varphi$.

- $\mathcal{M}, w \Vdash [P]\varphi$ iff for all finite paths $(v_0, v_1, \ldots, v_n)$, $n \geq 1$ such that $v_0 = w$ and there is $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ of length n that matches this path, $\mathcal{M}, v_n \Vdash \varphi$.

Intuitively, $\langle P \rangle \varphi$ is true at w, if there exists a finite run of P starting at w and finishing at v_n and at v_n φ is true. Analogously, $[P]\varphi$ is true at w, if all finite runs of P beginning at w finish at some state v_n such that φ is true at v_n.

If $\mathcal{M}, w \Vdash \varphi$ for every state w, we say that φ is *globally satisfied* in the model $\mathcal{M}$, notation $\mathcal{M} \Vdash \varphi$. If φ is globally satisfied in all models $\mathcal{M}$ of a frame $\mathcal{F}$, we say that φ is *valid* in $\mathcal{F}$, notation $\mathcal{F} \Vdash \varphi$. Finally, if φ is valid in all frames, we say that φ is valid, notation $\Vdash \varphi$. Two formulas φ and ψ are *semantically equivalent* if $\Vdash \varphi \leftrightarrow \psi$.

As mentioned in the previous section, there are two possible semantics for the τ action in CCS: it can be regarded as being observable or as being invisible. In our logics, we adopt the first one, since we are able to represent the second semantics as a particular case of the first. In fact, to do that, the only thing that is necessary is to force, in the frames under consideration, R_τ to be the relation $R_\tau = \{(w, w) : w \in W\}$.

Theorem 3.5. $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$ *if and only if* $\Vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$, *for any proposition* p.

Proof. ($\Rightarrow$) Suppose that $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$, but $\nVdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$. Then, we may assume, without loss of generality, that there is a model $\mathcal{M}$ and a state v_0 in this model such that $\mathcal{M}, v_0 \Vdash \langle P \rangle p$ (*), but $\mathcal{M}, v_0 \nVdash \langle Q \rangle p$ (**). By definition 3.4, (*) implies that there is a path $(v_0, v_1, \ldots, v_n)$, $n \geq 1$, in $\mathcal{M}$ such that $\mathcal{M}, v_n \Vdash p$ (***) and there is $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ that matches this path. But as $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$, then $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(Q)$. This and (***) imply, by definition 3.4, that $\mathcal{M}, v_0 \Vdash \langle Q \rangle p$, contradicting (**).

($\Leftarrow$) Suppose that $\Vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$ (*), but $\overrightarrow{\mathcal{R}_f}(P) \neq \overrightarrow{\mathcal{R}_f}(Q)$. Then, we may assume, without loss of generality, that there is $\overrightarrow{\alpha}$ such that $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$, but

$\overrightarrow{\alpha} \notin \overrightarrow{\mathcal{R}_f}(Q)$. Let us build a frame $\mathcal{F}$ that consists solely of a path $(v_0, \ldots, v_n)$, $n \geq 1$, such that $R_\alpha = \{(v_{i-1}, v_i) : 1 \leq i \leq n$ and α is the i-th term of $\overrightarrow{\alpha}\}$. Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, such that $v_n \in \mathbf{V}(p)$ and $v_i \notin \mathbf{V}(p)$, $1 \leq i < n$. Then, we have a path $(v_0, \ldots, v_n)$ such that $\mathcal{M}, v_n \Vdash p$ and $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ matches this path. By definition 3.4, $\mathcal{M}, v_0 \Vdash \langle P \rangle p$. However, $\overrightarrow{\alpha} \notin \overrightarrow{\mathcal{R}_f}(Q)$, so $(v_0, \ldots, v_n)$ is not matched by any sequence in $\overrightarrow{\mathcal{R}_f}(Q)$. Besides that, there is no other path $(v_0, \ldots, v_m)$, $m \geq 1$, in $\mathcal{M}$ such that $\mathcal{M}, v_m \Vdash p$. Thus, by definition 3.4, $\mathcal{M}, v_0 \nVdash \langle Q \rangle p$, which contradicts (*). ☐

Corollary 3.6. *If $P \sim Q$, then $\Vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$, for any proposition p.*

Proof. It follows directly from theorems 2.18 and 3.5. ☐

We present some equalities between sets of finite possible runs that are useful to the soundness proof of our axiomatization and to show why the null process **0** is problematic.

Theorem 3.7. *The following set equalities are true:*

1. $\overrightarrow{\mathcal{R}_f}(\alpha) = \{\alpha\}$;
2. $\overrightarrow{\mathcal{R}_f}(\alpha.P) = \overrightarrow{\mathcal{R}_f}(\alpha) \circ \overrightarrow{\mathcal{R}_f}(P)$;
3. $\overrightarrow{\mathcal{R}_f}(P_1 + P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \cup \overrightarrow{\mathcal{R}_f}(P_2)$.

Proof. The proof is straightforward from table 1. ☐

Theorem 3.8. *The following formulas are valid:*

1. $\langle \alpha.P \rangle p \leftrightarrow \langle \alpha \rangle \langle P \rangle p$
2. $\langle P_1 + P_2 \rangle p \leftrightarrow \langle P_1 \rangle p \vee \langle P_2 \rangle p$

Proof. We only provide the proof for the first formula. The proof for the second formula follows by an analogous line of reasoning, using the third equality in theorem 3.7 instead of the second one.

($\Rightarrow$) Suppose that, for some model $\mathcal{M}$ and some state w in this model, $\mathcal{M}, w \Vdash \langle \alpha.P \rangle p$. Then, by definition 3.4, there is a finite path $(v_0, v_1, \ldots, v_n)$, $n \geq 1$, such that $v_0 = w$, $\mathcal{M}, v_n \Vdash p$ and a sequence $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(\alpha.P)$ that matches this path. Now, by the first and second equalities in theorem 3.7, there is a sequence $\overrightarrow{\beta} \in \overrightarrow{\mathcal{R}_f}(P)$ such that $\overrightarrow{\alpha} = \alpha.\overrightarrow{\beta}$. $\overrightarrow{\beta}$ matches the path $(v_1, \ldots, v_n)$, which implies that $\mathcal{M}, v_1 \Vdash \langle P \rangle p$. Besides that, α matches the path (v_0, v_1), which implies that $\mathcal{M}, w \Vdash \langle \alpha \rangle \langle P \rangle p$. Thus, $\langle \alpha.P \rangle p \rightarrow \langle \alpha \rangle \langle P \rangle p$ is valid.

($\Leftarrow$) This proof is entirely analogous to the previous one, using the second equality in theorem 3.7 in the reverse direction. ☐

Now it is possible to see why, as stated in the previous section, the use of the null process **0** in our logics would be inconvenient. The problems that would appear come from the fact that, as described in [14], in a specification of the form $\alpha.\mathbf{0}$, $\mathbf{0}$ is denoting a process that has successfully terminated, while in a specification of the form $P + \mathbf{0}$, $\mathbf{0}$ is denoting a deadlocked process. This double role cannot be kept

in our logics without sacrificing a very desirable property in a dynamic logic: the compositional semantics, illustrated in theorem 3.8.

The compositional semantics is a direct consequence of the set equalities in theorem 3.7. But when we try to keep them in the presence of **0**, some problems arise. $\overrightarrow{\mathcal{R}_f}(\alpha.\mathbf{0}) = \{\alpha\}$, since **0** denotes successful termination in this case (if **0** denoted a deadlock, then $\overrightarrow{\mathcal{R}_f}(\alpha.\mathbf{0})$ would be $\emptyset$), and $\overrightarrow{\mathcal{R}_f}(P+\mathbf{0}) = \overrightarrow{\mathcal{R}_f}(P)$, since **0** denotes a deadlock in this case (if **0** denoted successful termination, then $\overrightarrow{\mathcal{R}_f}(P+\mathbf{0})$ would be $\overrightarrow{\mathcal{R}_f}(P) \cup \{\overrightarrow{\varepsilon}\}$). To keep the second equality in theorem 3.7, we must have $\{\alpha\} = \overrightarrow{\mathcal{R}_f}(\alpha.\mathbf{0}) = \overrightarrow{\mathcal{R}_f}(\alpha) \circ \overrightarrow{\mathcal{R}_f}(\mathbf{0})$, which implies that $\overrightarrow{\mathcal{R}_f}(\mathbf{0}) = \{\overrightarrow{\varepsilon}\}$ (*). On the other hand, to keep the third equality, we must have $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(P+\mathbf{0}) = \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(\mathbf{0})$, which implies that $\overrightarrow{\mathcal{R}_f}(\mathbf{0}) = \emptyset$ (**).

In the logical formalism, by theorem 3.8, (*) would imply that $\langle\mathbf{0}\rangle\phi$ is semantically equivalent to ϕ, while (**) would imply that $\langle\mathbf{0}\rangle\phi$ is semantically equivalent to $\bot$. The crucial point in this situation is that we would have to either abandon at least one of the equalities in theorem 3.7, substituting it by a pair of equations, one for the case where $P \neq \mathbf{0}$ and the other for the case where $P = \mathbf{0}$, or to somehow change the semantics so that the meaning of a subformula of the form $\langle\mathbf{0}\rangle\phi$ will depend on the context in which it is inserted, being sometimes equivalent to ϕ and sometimes to $\bot$. Both "solutions" would seriously compromise the compositionality of the semantics.

We address this issue of the null process in our third logic, without introducing any of the above problems. There, we redefine the process **0** so that it denotes only a deadlocked process, while defining a new way to denote termination.

3.2 Axiomatic System

We consider the following set of axioms and rules, where p and q are proposition symbols and φ and ψ are formulas.

(PL) Enough propositional logic tautologies

(K) $\vdash [P](p \to q) \to ([P]p \to [P]q)$

(Du) $\vdash [P]p \leftrightarrow \neg\langle P\rangle\neg p$

(Pr) $\vdash \langle\alpha.P\rangle p \leftrightarrow \langle\alpha\rangle\langle P\rangle p$

(NC) $\vdash \langle P_1 + P_2\rangle p \leftrightarrow \langle P_1\rangle p \vee \langle P_2\rangle p$

(PC) If $Exp(P)$ is the result of the application of the Expansion Law to process P, then $\vdash \langle P\rangle p \leftrightarrow \langle Exp(P)\rangle p$

(Sub) If $\vdash \varphi$, then $\vdash \varphi^\sigma$, where σ uniformly substitutes proposition symbols by arbitrary formulas.

(MP) If $\vdash \varphi$ and $\vdash \varphi \to \psi$, then $\vdash \psi$.

(Gen) If $\vdash \varphi$, then $\vdash [P]\varphi$.

Every formula ϕ derivable from an axiomatic system is called a *theorem* (denoted by $\vdash \phi$). A formula ϕ is *consistent* iff $\neg\phi$ is not a theorem, i.e., iff $\nvdash \neg\phi$, and *inconsistent* otherwise. The axiomatic system is said to be *sound* if every satisfiable formula is consistent. The standard way to verify the soundness of an axiomatic system is to check whether all axioms are valid formulas and whether the application of the rules to valid formulas always produces valid formulas. The axiomatic system is said to be *complete* if every consistent formula is satisfiable.

Two formulas ϕ and ψ are equi-consistent if $\vdash \phi \leftrightarrow \psi$. When the axiomatic system is sound, if ϕ and ψ are equi-consistent, then they are also semantically equivalent.

It is important to notice that the theorems $\vdash \langle P_1 + P_2\rangle p \leftrightarrow \langle P_2 + P_1\rangle p$ and $\vdash \langle P_1|P_2\rangle p \leftrightarrow \langle P_2|P_1\rangle p$, which state the commutativity of the $+$ and $|$ operators, are derivable from the axiomatic system above.

The axioms **(PL)**, **(K)** and **(Du)** and the rules **(Sub)**, **(MP)** and **(Gen)** are standard in the modal logic literature. The soundness of **(Pr)** and **(NC)** follows directly from the set equalities in theorem 3.7 and from definition 3.4, as shown in theorem 3.8. Finally, the soundness of **(PC)** follows from theorem 2.11 and corollary 3.6.

The above axiomatic system is also complete with respect to the class of sPPDL frames and the logic has the finite model property. We omit the proofs here, because they are analogous to the proofs presented in section 4, where constants are added to the language.

4 PPDL-c

The logic presented in this section uses the same CCS operators as in the previous section plus CCS constants. This is the Parallel PDL logic with constants, or PPDL-c. Our goal in this section is to build an axiomatic system for PPDL-c and prove its completeness.

4.1 Language and Semantics

In this section, we present the syntax and semantics of PPDL-c.

Definition 4.1. *The PPDL-c language consists of a set Φ of countably many proposition symbols, a set $\mathcal{N}$ of countably many names, the silent action τ, the boolean connectives $\neg$ and $\wedge$, the CCS operators $.$, $+$ and $|$, a set $\mathcal{C}$ of countably many constants, such that each element of $\mathcal{C}$ has its unique corresponding defining equation, and a modality $\langle P\rangle$ for every process P. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle P\rangle\varphi, \text{ with } P ::= \alpha \mid \alpha.P \mid \alpha.A \mid P_1 + P_2 \mid P_1|P_2,$$

where $p \in \Phi$, $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ and $A \in \mathcal{C}$.

The presence of constants in the language allows us to write iterating specifications, as $P = \alpha.A$, with $A \stackrel{def}{=} \alpha.A + \tau$. However, constants can do other things besides describing iterative behaviors. With constants, we are able to write self-replicating specifications, as $P = ((\tau.A) + \tau)|Q$, with $A \stackrel{def}{=} ((\tau.A) + \tau)|Q$. After

the execution of n τ-actions, P is capable of behaving as n Q-processes in parallel, for any $n \in \mathbb{N}$.

The example above is a simple one. It is easy to see that it is possible to write processes with very complex behaviors if we start nesting self-replicating processes.

In order to keep the logic simple, that is, keep the finite model property and a simple and complete axiomatization without having to make changes to our proposed simple Kripke semantics, we restrict the use of constants in PPDL-c in order to prevent self-replicating processes (in [5], Dam enforces a similar syntactic restriction, also to prevent unbounded process growth). The issue of whether it is possible to keep these desirable properties of the logic in the presence of replication remains an open problem and we defer it to a future work, as explained in section 6.

Definition 4.2. *Let P be a process and $\{A_1, \ldots, A_n\}$ be the constants that occur in P. We define $Cons(P)$ as the smallest set of constants such that $Cons(P) \supseteq \{A_1, \ldots, A_n\}$ and, for every constant $A_i \in Cons(P)$, if A_k occurs in P_{A_i}[1], then $A_k \in Cons(P)$.*

Restriction 4.3. *We make the following restrictions to processes in PPDL-c:*

1. *$Cons(P)$ must be a finite set for every process P;*
2. *We only allow defining equations that fit into one of the following models:*
   - $A \stackrel{def}{=} P_A$, *where $A \notin Cons(P_A)$, called* non-recursive equations*;*
   - $A \stackrel{def}{=} \overrightarrow{\alpha}_1.A + \ldots + \overrightarrow{\alpha}_n.A + T_A$, *where $A \notin Cons(T_A)$ and the sequences $\overrightarrow{\alpha}_i$ are nonempty, for all $1 \leq i \leq n$, called* recursive equations.

Intuitively, the second point of the restriction above guarantees that constants are only going to be used to describe iterative behavior and not replication of processes.

The set equalities from theorem 3.7 remains valid, along with the equality

$$\overrightarrow{\mathcal{R}_f}(\alpha.A) = \overrightarrow{\mathcal{R}_f}(\alpha) \circ \overrightarrow{\mathcal{R}_f}(P_A), \tag{1}$$

which also follows from table 1.

However, due to the possibility of iterative behaviors, some set equalities may present themselves as recursive equations, i.e., with a set $\overrightarrow{\mathcal{R}_f}(P)$ appearing on both sides of the equation. In these cases, it is possible to obtain an equivalent non-recursive equality. First, a recursive equation can be rewritten, using the set equalities in theorem 3.7 and equation (1), in the form $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(P') \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(Q)$ (a form that will be useful in lemma 4.9 and theorem 4.10), where $\overrightarrow{\mathcal{R}_f}(Q)$ is not built from $\overrightarrow{\mathcal{R}_f}(P)$ using any of the operations of definition 2.16. Now, as all sequences in $\overrightarrow{\mathcal{R}_f}(P)$, $\overrightarrow{\mathcal{R}_f}(P')$ and $\overrightarrow{\mathcal{R}_f}(Q)$ are finite and $\overrightarrow{\varepsilon} \notin \overrightarrow{\mathcal{R}_f}(P')$, we may use a result known as Arden's Rule, that states that if X, A and B are sets of finite strings and the empty string is not in A, then the equation $X = A \circ X \cup B$ has as its unique solution $X = A^* \circ B$ [1]. Thus, $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(P')^* \circ \overrightarrow{\mathcal{R}_f}(Q)$.

Below, we present the definition of knot processes. These are processes that can iterate themselves or some subprocess of them.

[1] For the definition of P_{A_i}, see remark 2.6.

Definition 4.4. *We say that a process P is a* knot process *if $P = P_A$ for some constant A with a recursive defining equation or if $P = P_1 \mid P_2$ where P_1 or P_2 is a knot process. Otherwise, we say that P is a* non-knot process.

Intuitively, a knot process is a process P that has the possibility of iterating at least one sequence of actions $\overrightarrow{\alpha}$.

Definition 4.5. *We call a non-empty sequence of actions $\overrightarrow{\alpha}$ a* loop *of a knot process P if $P \overset{\overrightarrow{\alpha}}{\Rightarrow} P$. We say that $\overrightarrow{\alpha}$ is a* proper loop *if $\overrightarrow{\alpha}$ is a loop and there is no $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$, with $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, such that $\overrightarrow{\beta}$ and $\overrightarrow{\lambda}$ are loops of P. The set of loops of P is denoted by $Lo(P)$ and the set of proper loops of P is denoted by $PLo(P)$.*

Theorem 4.6. *$\overrightarrow{\alpha} \in Lo(P)$ if and only if $\overrightarrow{\alpha} = \overrightarrow{\alpha_1}.\cdots.\overrightarrow{\alpha_n}$, $n \geq 1$, where $\overrightarrow{\alpha_i} \in PLo(P)$, for all $i \in \{1, \ldots, n\}$.*

Proof. The proof is straightforward from definition 4.5. □

Definition 4.7. *We call a non-empty sequence of actions $\overrightarrow{\alpha}$ a* breaker *of a knot process P if $\overrightarrow{\alpha}$ is a prefix of some finite possible run of P and there is no $\overrightarrow{\beta}$ such that $\overrightarrow{\alpha} \subset \overrightarrow{\beta}$ and $\overrightarrow{\beta}$ is a loop. We say that $\overrightarrow{\alpha}$ is a* proper breaker *if $\overrightarrow{\alpha}$ is a breaker and there is no $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$, with $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, such that $\overrightarrow{\beta}$ is a loop and $\overrightarrow{\lambda}$ is a breaker. Finally, we say that $\overrightarrow{\alpha}$ is a* minimal proper breaker *if $\overrightarrow{\alpha}$ is a proper breaker and there is no $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$ such that $\overrightarrow{\beta}$ is a proper breaker. The set of breakers of P is denoted by $Br(P)$, the set of proper breakers of P is denoted by $PBr(P)$ and the set of minimal proper breakers of P is denoted by $MPBr(P)$.*

Theorem 4.8. *$\overrightarrow{\alpha} \in PBr(P)$ if and only if $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, where $\overrightarrow{\beta} \in MPBr(P)$.*

Proof. The proof is straightforward from definition 4.7. □

Intuitively, a sequence of actions $\overrightarrow{\alpha}$ is a loop of a knot process P if $\overrightarrow{\alpha}$ leads P back to itself and it is a breaker if it leads P to a point P' where it can no longer return to P.

Using the concepts of loops and breakers, we can split a knot process P into two parts: the *looping part*, denoted by L_P, and the *tail part*, denoted by T_P.

$$L_P = \sum\{\overrightarrow{\alpha} : \overrightarrow{\alpha} \in PLo(P)\}.$$

and

$$T_P = \sum\{\overrightarrow{\alpha}.P' : \overrightarrow{\alpha} \in MPBr(P) \text{ and } P \overset{\overrightarrow{\alpha}}{\Rightarrow} P'\}$$

Lemma 4.9. *If P is a knot process, then $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(L_P) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(T_P)$.*

Proof. If $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(L_P) \circ \overrightarrow{\mathcal{R}_f}(P)$, then $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, where $\overrightarrow{\beta} \in \overrightarrow{\mathcal{R}_f}(L_P)$ and $\overrightarrow{\lambda} \in \overrightarrow{\mathcal{R}_f}(P)$. Then, $P \overset{\overrightarrow{\beta}}{\Rightarrow} P$ and $P \overset{\overrightarrow{\lambda}}{\Rightarrow} \surd$, which implies that $P \overset{\overrightarrow{\alpha}}{\Rightarrow} \surd$. Thus, $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$. If $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(T_P)$, then $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, where $P \overset{\overrightarrow{\beta}}{\Rightarrow} P'$ and $P' \overset{\overrightarrow{\lambda}}{\Rightarrow} \surd$, which implies that $P \overset{\overrightarrow{\alpha}}{\Rightarrow} \surd$. Thus, $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$. This proves that $\overrightarrow{\mathcal{R}_f}(L_P) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(T_P) \subseteq \overrightarrow{\mathcal{R}_f}(P)$.

If $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$, then we have two cases:

1. There is $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$, with $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, such that $\overrightarrow{\beta}$ is a loop. Then, by theorem 4.6, $\overrightarrow{\beta} = \overrightarrow{\beta_1}.\overrightarrow{\beta_2}$, where $\overrightarrow{\beta_1} \in PLo(P)$. This means that $\overrightarrow{\beta_1} \in \overrightarrow{\mathcal{R}_f}(L_P)$. If we make $\overrightarrow{\gamma} = \overrightarrow{\beta_2}.\overrightarrow{\lambda}$, then $\overrightarrow{\alpha} = \overrightarrow{\beta_1}.\overrightarrow{\gamma}$ and $P \stackrel{\overrightarrow{\gamma}}{\Rightarrow} \surd$. Thus, $\overrightarrow{\gamma} \in \overrightarrow{\mathcal{R}_f}(P)$ and $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(L_P) \circ \overrightarrow{\mathcal{R}_f}(P)$.

2. There is no $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$, with $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, such that $\overrightarrow{\beta}$ is a loop. This implies that, for all $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$, $\overrightarrow{\beta} \in PBr(P)$. Then, by theorem 4.8, $\overrightarrow{\beta} = \overrightarrow{\beta_1}.\overrightarrow{\beta_2}$, where $\overrightarrow{\beta_1} \in MPBr(P)$. If we make $\overrightarrow{\gamma} = \overrightarrow{\beta_2}.\overrightarrow{\lambda}$, then $\overrightarrow{\alpha} = \overrightarrow{\beta_1}.\overrightarrow{\gamma}$. This means that, if $P \stackrel{\overrightarrow{\beta_1}}{\Rightarrow} P'$, then $P' \stackrel{\overrightarrow{\gamma}}{\Rightarrow} \surd$. Thus, $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(T_P)$.

This proves that $\overrightarrow{\mathcal{R}_f}(P) \subseteq \overrightarrow{\mathcal{R}_f}(L_P) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(T_P)$. □

Theorem 4.10. *If P is a knot process, then $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(L_P)^* \circ \overrightarrow{\mathcal{R}_f}(T_P)$.*

Proof. By lemma 4.9, $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(L_P) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(T_P)$. The result follows from Arden's Rule [1], since $\overrightarrow{\varepsilon} \notin \overrightarrow{\mathcal{R}_f}(L_P)$.

□

We also define the process L'_P, that is capable of iterating L_P.

$$L'_P = \sum\{\overrightarrow{\alpha}.Z_P : \overrightarrow{\alpha} \in PLo(P)\} + L_P,$$

where Z_P is a new constant with defining equation $Z_P \stackrel{def}{=} L'_P$.

The notions of frame, model and satisfaction are defined analogously to definitions 3.2, 3.3 and 3.4. It is straightforward to check that theorem 3.5 and corollary 3.6 remain valid in PPDL-c.

4.2 Axiomatic System

The axiomatic system is similar to the one presented in section 3.2. We consider the following set of axioms and rules, where p, q and r are proposition symbols and φ and ψ are formulas.

- The axioms **(PL)**, **(K)** and **(Du)** and the rules **(Sub)**, **(MP)** and **(Gen)**.
- Axioms for knot processes:

  (Rec) $\vdash \langle P\rangle p \leftrightarrow \langle T_P\rangle p \vee \langle L_P\rangle\langle P\rangle p$

  (FP) $\vdash (r \rightarrow ([T_P]p \wedge [L_P]r)) \wedge [L'_P](r \rightarrow ([T_P]p \wedge [L_P]r)) \rightarrow (r \rightarrow [P]p)$

- Axioms for non-knot processes:

  (sPPDL) The axioms **(Pr)**, **(NC)** and the rule **(PC)**.

  (Cons) $\vdash \langle \alpha.A\rangle p \leftrightarrow \langle\alpha\rangle\langle P_A\rangle p$

The proof of soundness is analogous to the proof of soundness for sPPDL. The soundness of **(Cons)** follows from equation (1) and theorem 3.5. The soundness of **(Rec)** and **(FP)** follows from theorems 4.10 and 3.5. The axiom **(FP)** may seem strange at first, but it is just an adaptation of the so-called induction axiom to our particular situation. Below, we go into a little more detail for the proofs of soundness of the axioms of knot processes.

Theorem 4.11. *The axiom* **(Rec)** *is sound.*

Proof. Suppose that P is a knot process. By lemma 4.9, $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(L_P) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(T_P)$. Now, theorem 3.5 allows us to go from an equality between sets of finite possible runs to semantical equivalence of modal formulas. In fact, from the equality $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(L_P) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(T_P)$, theorem 3.5 allows us to conclude that $\Vdash \langle P\rangle p \leftrightarrow \langle T_P\rangle p \vee \langle L_P\rangle\langle P\rangle p$, for any proposition p, which implies that the axiom **(Rec)** is sound. □

Theorem 4.12. *The axiom* **(FP)** *is sound.*

Proof. Suppose that P is a knot process. Suppose, by contradiction, that there is a model $\mathcal{M}$ and a world w in this model such that $\mathcal{M}, w \nVdash (r \rightarrow ([T_P]p \wedge [L_P]r)) \wedge [L'_P](r \rightarrow ([T_P]p \wedge [L_P]r)) \rightarrow (r \rightarrow [P]p)$. So,

$$\mathcal{M}, w \Vdash (r \rightarrow ([T_P]p \wedge [L_P]r)) \tag{2}$$

$$\mathcal{M}, w \Vdash [L'_P](r \rightarrow ([T_P]p \wedge [L_P]r)) \tag{3}$$

and

$$\mathcal{M}, w \nVdash (r \rightarrow [P]p).$$

Then,

$$\mathcal{M}, w \Vdash r \tag{4}$$

and

$$\mathcal{M}, w \Vdash \langle P\rangle \neg p \tag{5}$$

from (2) and (4) we have that

$$\mathcal{M}, w \Vdash [T_P]p \tag{6}$$

and

$$\mathcal{M}, w \Vdash [L_P]r. \tag{7}$$

From (5), there is a finite path $(v_0, v_1, \ldots, v_n)$, $n \geq 1$, such that $v_0 = w$,

$$\mathcal{M}, v_n \Vdash \neg p \tag{8}$$

and there is $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ of length n that matches this path. By theorem 4.10, $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(L_P)^* \circ \overrightarrow{\mathcal{R}_f}(T_P)$. As $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ then $\overrightarrow{\alpha} = \overrightarrow{\alpha}^1 \cdots \overrightarrow{\alpha}^k \overrightarrow{\alpha}^{k+1}$, for some $k \geq 0$, where $\overrightarrow{\alpha}^i \in \overrightarrow{\mathcal{R}_f}(L_P)$, for all $0 \leq i \leq k$ and $\overrightarrow{\alpha}^{k+1} \in \overrightarrow{\mathcal{R}_f}(T_P)$. We then define $\overrightarrow{\alpha}^j_L = \overrightarrow{\alpha}^1 \cdots \overrightarrow{\alpha}^j$, for every $1 \leq j \leq k$, and $\overrightarrow{\alpha}_T = \overrightarrow{\alpha}^{k+1}$. By the definition of L'_P, we have that $\overrightarrow{\alpha}^j_L \in \overrightarrow{\mathcal{R}_f}(L'_P)$, for every $1 \leq j \leq k$.

If $k = 0$, then by the definition of satisfaction, $\mathcal{M}, w \Vdash \langle T_P \rangle \neg p$, which contradicts (6).

If $k \geq 1$, as $\overrightarrow{\alpha}$ matches the path $(v_0, v_1, \ldots, v_n)$, $n \geq 1$, and $\overrightarrow{\alpha} = \overrightarrow{\alpha}_L^k \overrightarrow{\alpha}_T$, then there are states $s_j \in \{v_1, \ldots, v_{n-1}\}$, for each $1 \leq j \leq k$, such that $\overrightarrow{\alpha}_L^j \in \overrightarrow{\mathcal{R}_f}(L'_P)$ matches the path $(v_0, \ldots, s_j)$ and $\overrightarrow{\alpha}_T \in \overrightarrow{\mathcal{R}_f}(T_P)$ matches the path $(s_k, \ldots, v_n)$. Applying the definition of satisfaction to (3) and the paths $(v_0, \ldots, s_j)$ we get that

$$\mathcal{M}, s_j \Vdash r \rightarrow ([T_P]p \wedge [L_P]r), \text{ for all } 1 \leq j \leq k. \tag{9}$$

Now, from (2) and (4), we have that $\mathcal{M}, w \Vdash [L_P]r$. Now, as $\overrightarrow{\alpha}^1 \in \overrightarrow{\mathcal{R}_f}(L_P)$ matches the path (v_0, s_1), this implies that $\mathcal{M}, s_1 \Vdash r$. From this, together with (9), we get that $\mathcal{M}, s_1 \Vdash [L_P]r$. Now, repeating the above reasoning with $\overrightarrow{\alpha}^2$, $\overrightarrow{\alpha}^3$ and so on, we get that $\mathcal{M}, s_j \Vdash r$, for all $1 \leq j \leq k$. Now, from $\mathcal{M}, s_k \Vdash r$ together with (9), we get $\mathcal{M}, s_k \Vdash [T_P]p$. As $\overrightarrow{\alpha}_T \in \overrightarrow{\mathcal{R}_f}(T_P)$ matches the path $(s_k, \ldots, v_n)$, the definition of satisfaction allows us to conclude that $\mathcal{M}, v_n \Vdash p$, which is a contradiction with (8).

□

Theorem 4.13 (Completeness)**.** *Every consistent formula is satisfiable in a finite PPDL-c model.*

Proof. The proof is presented in the appendix A. □

It is important to highlight an unusual property of the above axiomatization. It has two separate sets of axioms and rules, one for modalities with processes that match the definition of knot processes and another for modalities with processes that do not match this definition. This is another motivation for the presentation of our next logic in the following section. The axiomatization for that logic does not present this problem.

5 PPDL*

As it was shown in section 3, the use of the null process **0** of CCS in our first two logics would bring a serious inconvenience to their semantics: their compositionality would be compromised. This problem also affects our ability to include the restriction operator in these logics. Besides that, in PPDL-c, we have to define two distinct sets of axioms, depending on whether the process under consideration is a knot process or not.

In this section, our goal is to solve these two problems that occur in the previous logics. In order to accomplish this, first we extend the language of CCS with new operators (traditional operators from PDL, as the Kleene star) and a new type of action and slightly redefine its semantics. We call this new process algebra *extended CCS* or XCCS. Then, we define a dynamic logic in which the programs are described in the language of XCCS (PPDL*). Because of the refined definition of the null process **0** in XCCS, we can include it in this logic, as well as the restriction operator. Besides that, one of the new operators of XCCS, the iteration operator, allows us to deal with all sorts of processes with just one set of axioms and to also drop the constants and all its elaborated theory from the language.

Table 2: Transition Relations of XCCS

$\alpha.P \xrightarrow{\alpha} P$	$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	$\frac{Q \xrightarrow{\beta} Q'}{P+Q \xrightarrow{\beta} Q'}$	
$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$	$\frac{Q \xrightarrow{\beta} Q'}{P \mid Q \xrightarrow{\beta} P \mid Q'}$	$\frac{P \xrightarrow{\lambda} P', Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$	$\frac{P \xrightarrow{\alpha} P', \alpha \notin L \cup \overline{L}}{P \backslash L \xrightarrow{\alpha} P' \backslash L}$
$\frac{P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\alpha} P';Q}$	$\frac{P \xrightarrow{\alpha} P'}{P^* \xrightarrow{\alpha} P';P^*}$		
$\text{END} \xrightarrow{\text{END}} \surd$	$P^* \xrightarrow{\text{END}} \surd$	$\frac{P \xrightarrow{\text{END}} \surd, Q \xrightarrow{\text{END}} \surd}{P;Q \xrightarrow{\text{END}} \surd}$	$\frac{P \xrightarrow{\text{END}} \surd, Q \xrightarrow{\alpha} Q'}{P;Q \xrightarrow{\alpha} Q'}$
$\frac{P \xrightarrow{\text{END}} \surd}{P+Q \xrightarrow{\text{END}} \surd}$	$\frac{Q \xrightarrow{\text{END}} \surd}{P+Q \xrightarrow{\text{END}} \surd}$	$\frac{P \xrightarrow{\text{END}} \surd, Q \xrightarrow{\text{END}} \surd}{P \mid Q \xrightarrow{\text{END}} \surd}$	$\frac{P \xrightarrow{\text{END}} \surd}{P \backslash L \xrightarrow{\text{END}} \surd}$

5.1 XCCS

In CCS, we have the set of actions $\mathcal{A} = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$. In XCCS, we denote this set of actions as $\mathcal{A}_R$, the set of *running actions*. In XCCS, we have an extra action, besides the ones in $\mathcal{A}_R$, called the *ending action* and denoted by END. A process in XCCS can only successfully finish after performing the action END and it always successfully finishes after performing such action. If a process cannot perform any running action and cannot successfully finish, it is called a *deadlocked* process.

Definition 5.1. *In XCCS, process specifications can be built using the following operations:*

$$P ::= \mathbf{0} \mid END \mid \alpha.P \mid P_1; P_2 \mid P_1 + P_2 \mid P_1|P_2 \mid P^* \mid P\backslash L,$$

with

$$\alpha ::= a \mid \overline{a} \mid \tau,$$

where $a \in \mathcal{N}$ and $L \subseteq \mathcal{N}$.

$\mathbf{0}$ is the *null process*. It is a deadlocked process, since it is incapable of performing any running action and of successfully finishing. END is a process that is incapable of performing any running action, but it is capable of successfully finishing.

It is important to notice that α alone is not a program in XCCS. It can now be written as α.END. Also, the constants are not necessary any longer, because now we have the iteration operator (*).

Table 2 presents the semantics of the XCCS operators. The first two rows contain the standard CCS operators introduced in section 2.2. The third row presents the transition rules for the sequential composition operator (;) and the iteration operator (*). Finally, the last two rows are concerned with the semantics of the END program. They are all quite intuitive as, for instance, the one at the last row and third column asserts that a parallel composition can only successfully terminate if both sides terminate.

From table 2, one can see that now the null process $\mathbf{0}$ denotes only a deadlocked process, i.e, a process that can not perform any action and can not successfully terminate. As explained in section 3, the situation in standard CCS is different, since there, in a specification of the form $\alpha.\mathbf{0}$, $\mathbf{0}$ is denoting a process that has successfully terminated. This is no longer the case. In XCCS, a specification of the

form $\alpha.\mathbf{0}$ denotes that a process performs the action α and then deadlocks, while a specification of the form $\alpha.\text{END}$ denotes that a process performs the action α and then successfully terminates. This slight extension of the language allows for the null process and for the restriction operator to be used in our third logic without compromising the compositionality of its semantics.

In [14] and [15], Milner uses a clever syntactic construction to define a form of sequential composition. It is slightly different to the form presented in table 2 and it is not a primitive operator. He also uses the notation ; for it, but we denote his construction with a : instead, so we can easily differentiate between his and our constructions. Milner's construction depends on a number of things. First, we must consider a new name $z \notin \mathcal{N}$. Second, every process must perform the action $\overline{z}$ as its last action before termination and may not perform z or $\overline{z}$ at any other point of execution. Third, we must perform syntactic substitutions of names in processes, where $P[b/a]$ denotes the substitution of every occurrence of a ($\overline{a}$) in P by b ($\overline{b}$). Then, sequential composition is defined in the following way:

$$P : Q = (P[a/z] \mid a.Q)\backslash\{a\},$$

where a must be a name that does not occur in neither P nor Q.

The main difference between the two forms of sequential composition is that, as tables 1 and 2 easily show, $\overrightarrow{\mathcal{R}_f}(P;Q) = \overrightarrow{\mathcal{R}_f}(P) \circ \overrightarrow{\mathcal{R}_f}(Q)$, while $\overrightarrow{\mathcal{R}_f}(P : Q) = \overrightarrow{\mathcal{R}_f}(P) \circ \{\tau\} \circ \overrightarrow{\mathcal{R}_f}(Q)$. The extra τ would also be present in the finite runs of a process P^*, as we use sequential composition to define the semantics of the iteration operator (table 2). These extra τ's appearing between the finite runs of the subprocesses would be a complication to the semantics of our logic, as some intuitive validities, such as $\langle A\rangle\langle B\rangle\varphi \rightarrow \langle A; B\rangle\varphi$, would be false. Since we are already introducing the END process to solve the previous problems with the null process, there is no reason why we should not also use it to build a simpler and more convenient form of sequential composition and a simple form of iteration, as it is done in table 2.

Now, we need to make slight adjustments to the notion of strong bisimulation and to the Expansion Law.

Definition 5.2. *Let $\mathcal{P}$ be the set of all possible process specifications. A set $Z \subseteq \mathcal{P} \times \mathcal{P}$ is a* strong bisimulation *if $(P, Q) \in Z$ implies, for all $\alpha \in \mathcal{A}_R$,*

- *If $P \xrightarrow{\alpha} P'$ and $P' \in \mathcal{P}$, then there is $Q' \in \mathcal{P}$ such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in Z$;*
- *If $Q \xrightarrow{\alpha} Q'$ and $Q' \in \mathcal{P}$, then there is $P' \in \mathcal{P}$ such that $P \xrightarrow{\alpha} P'$ and $(P', Q') \in Z$;*

and

- *$P \xrightarrow{END} \surd$ if and only if $Q \xrightarrow{END} \surd$.*

The definition of strong bisimilarity is analogous to definition 2.9, using the new notion of strong bisimulation stated above.

In the presence of the iteration operator, a weaker version of the Expansion Law is now sufficient for our needs.

Theorem 5.3 (Expansion Law (EL)). *Let $P = P_1 \mid P_2$, where P is unrestricted* and $\mid$ does not occur in P_1 and P_2. *Then*

$$P \sim \sum_{P_1 \xrightarrow{\alpha} P_1'} \alpha.(P_1' \mid P_2) + \sum_{P_2 \xrightarrow{\beta} P_2'} \beta.(P_1 \mid P_2') + \sum_{R \in A_\tau} \tau.R + E_P,$$

where $A_\tau = \{(P_1' \mid P_2') : P_1 \xrightarrow{a} P_1'$ *and* $P_2 \xrightarrow{\overline{a}} P_2'$, *for some* $a \in \mathcal{N}\} \cup \{(P_1' \mid P_2') : P_1 \xrightarrow{\overline{a}} P_1'$ *and* $P_2 \xrightarrow{a} P_2'$, *for some* $a \in \mathcal{N}\}$ *and* $E_P = END$, *if* $P_1 \xrightarrow{END} \surd$ *and* $P_2 \xrightarrow{END} \surd$ *or* $E_P = \mathbf{0}$, *otherwise. Again, we denote the right side of this bisimilarity by* $Exp(P)$.

Again, it is important to emphasize that the Expansion Law presented above is not in its most general form. The conditions that process P is unrestricted and that the $\mid$ operator does not occur in P_1 and P_2 allow for a simpler algebraic formula, presented above, and is sufficient for our needs in the axiomatization of our logic.

Finally, because of the presence of the action END, we need to slightly adjust the definition of the composition $R \circ S$ of two sets R and S of finite sequences of actions.

Definition 5.4. *Let* $\natural(\overrightarrow{\alpha}) = \overrightarrow{\lambda}$, *if* $\overrightarrow{\alpha} = \overrightarrow{\lambda}.END$ *and* $\natural(\overrightarrow{\alpha}) = \overrightarrow{\alpha}$, *otherwise. Then,*

$$R \circ S = \{\natural(\overrightarrow{\alpha}).\overrightarrow{\beta} : \overrightarrow{\alpha} \in R \text{ and } \overrightarrow{\beta} \in S\}$$

Now, we define some concepts that are useful to the axiomatization of our third logic.

Definition 5.5. *We say that a relation $\cong$ between processes is a* congruence *if it is an equivalence relation and it is preserved by all of XCCS operators, that is, if $P \cong Q$, then $\alpha.P \cong \alpha.Q$, $P + R \cong Q + R$ and so on.*

Definition 5.6. *A syntactic substitution of a* restricted name *by a* fresh name *(a name that does not occur in the process specification) in a restriction set L and in every occurrence of the name in the scope of the correspondent restriction $\backslash L$ is called an* alpha conversion.

Definition 5.7. Restriction congruence, *or* r-congruence, *denoted by $\equiv_r$, is a relation between processes defined by the following set of axioms and rules, where $n(P)$ denotes the set of names that occur in P as part of both input and output actions.*

1. *It is a congruence;*
2. *It is closed under alpha conversion;*
3. *If $\alpha \notin L \cup \overline{L}$, $(\alpha.P)\backslash L \equiv_r \alpha.(P\backslash L)$;*
4. *If $\alpha \in L \cup \overline{L}$, $(\alpha.P)\backslash L \equiv_r \mathbf{0}$;*
5. *$(P; Q)\backslash L \equiv_r (P\backslash L); (Q\backslash L)$;*
6. *$(P + Q)\backslash L \equiv_r (P\backslash L) + (Q\backslash L)$;*
7. *If $n(P) \cap (L \cup \overline{L}) = \emptyset$, $P|(Q\backslash L) \equiv_r (P|Q)\backslash L$;*
8. *$(P^*)\backslash L \equiv_r (P\backslash L)^*$;*
9. *$P\backslash L\backslash M \equiv_r P\backslash(L \cup M)$;*
10. *If $n(P) \cap (L \cup \overline{L}) = \emptyset$, $P\backslash L \equiv_r P$.*

We should also explicitly state two important particular cases of rule 10: $\mathbf{0}\backslash L \equiv_r \mathbf{0}$ and END$\backslash L \equiv_r$ END.

Definition 5.8. *We say that a process is in* r-external form *if it has the form $P\backslash L$, where P is unrestricted.*

Theorem 5.9. *Every process is r-congruent to a process in r-external form and every process with no occurrences of the $|$ operator is r-congruent to an unrestricted process.*

Proof. The proof follows from definition 5.7. □

Theorem 5.10. *If $P \equiv_r Q$, then $P \sim Q$.*

Proof. The proof follows from table 2 and definition 5.2. We have to prove that all axioms and inference rules in definition 5.7 hold when we replace $\sim$ for $\equiv_r$. For instance, in the case of axiom 8, we have to prove that $(P+Q)\backslash L \sim (P\backslash L)+(Q\backslash L)$, which follows directly from table 2 using definition 5.2. □

5.2 Language and Semantics

In this section, we present the syntax and semantics of PPDL*.

Definition 5.11. *The PPDL* language consists of a set Φ of countably many proposition symbols, a set $\mathcal{N}$ of countably many names, the silent action τ, the ending action END, the boolean connectives $\neg$ and $\wedge$, the XCCS operators ., ;, +, $|$, * and $\backslash$, a modality $\langle\alpha\rangle$ for every $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ and a modality $\langle P\rangle$ for every process P, including the atomic processes $\mathbf{0}$ and END. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\alpha\rangle\varphi \mid \langle P\rangle\varphi,$$

with

$$P ::= \mathbf{0} \mid END \mid \alpha.P \mid P_1; P_2 \mid P_1 + P_2 \mid P_1|P_2 \mid P^* \mid P\backslash L,$$

where $p \in \Phi$, $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ and $L \subseteq \mathcal{N}$.

Although α alone is not a XCCS program, $\langle\alpha\rangle\varphi$ is a formula of PPDL*. This is necessary to keep the axioms (**Pr**) and (**SC**) (presented below) simple and compositional.

Definition 5.12. *A* frame *for PPDL* is a tuple $\mathcal{F} = (W, \{R_\alpha\}_{\alpha\in\mathcal{L}}, R_{END}, R_{\mathbf{0}})$ where*

- *W is a non-empty set of states;*
- *R_α, for each $\alpha \in \mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$, R_{END} and $R_{\mathbf{0}}$ are the basic binary relations, where $R_{END} = \{(w, w) : w \in W\}$ and $R_{\mathbf{0}} = \emptyset$.*

The notion of model is defined analogously to definition 3.3. We define the semantical notion of satisfaction for PPDL* as follows:

Definition 5.13. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of* satisfaction *of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, can be inductively defined as follows:*

- $\mathcal{M}, w \Vdash p$ *iff* $w \in \mathbf{V}(p)$*;*
- $\mathcal{M}, w \Vdash \top$ *always;*
- $\mathcal{M}, w \Vdash \neg\varphi$ *iff* $\mathcal{M}, w \nVdash \varphi$*;*
- $\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ *iff* $\mathcal{M}, w \Vdash \varphi_1$ *and* $\mathcal{M}, w \Vdash \varphi_2$*;*
- $\mathcal{M}, w \Vdash \langle\alpha\rangle\varphi$ *iff there is* $w' \in W$ *such that* $wR_\alpha w'$ *and* $\mathcal{M}, w' \Vdash \varphi$*, where* $\alpha \in \mathcal{N} \cup \overline{N} \cup \{\tau\}$*;*
- $\mathcal{M}, w \Vdash \langle P\rangle\varphi$ *iff there is a finite path* $(v_0, v_1, \ldots, v_n)$*,* $n \geq 1$*, such that* $v_0 = w$*,* $\mathcal{M}, v_n \Vdash \varphi$ *and there is* $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ *of length* n *such that* $(v_{i-1}, v_i) \in R_{(\overrightarrow{\alpha})_i}$*, for* $1 \leq i \leq n$*. We say that such* $\overrightarrow{\alpha}$ matches *the path* $(v_0, \ldots, v_n)$*.*

It is important to notice that theorem 3.5 and corollary 3.6 remain valid in PPDL*.

Theorem 5.14. *The following set equalities are true:*

1. $\overrightarrow{\mathcal{R}_f}(\mathbf{0}) = \emptyset$*;*
2. $\overrightarrow{\mathcal{R}_f}(END) = \{END\}$*;*
3. $\overrightarrow{\mathcal{R}_f}(\alpha.P) = \overrightarrow{\mathcal{R}_f}(\alpha) \circ \overrightarrow{\mathcal{R}_f}(P)$*;*
4. $\overrightarrow{\mathcal{R}_f}(P_1; P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \circ \overrightarrow{\mathcal{R}_f}(P_2)$*;*
5. $\overrightarrow{\mathcal{R}_f}(P_1 + P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \cup \overrightarrow{\mathcal{R}_f}(P_2)$*;*
6. $\overrightarrow{\mathcal{R}_f}(P^*) = \overrightarrow{\mathcal{R}_f}(P)^*$*;*
7. $\overrightarrow{\mathcal{R}_f}(P_1|P_2) = \bigcup\{\overrightarrow{\mathcal{R}_f}(\overrightarrow{\alpha}|\overrightarrow{\beta}) : \overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P_1)$ *and* $\overrightarrow{\beta} \in \overrightarrow{\mathcal{R}_f}(P_2)\}$*;*
8. *If* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$*, then* $\overrightarrow{\mathcal{R}_f}(P\backslash L) = \overrightarrow{\mathcal{R}_f}(Q\backslash L)$*;*
9. *If* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(A) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(B)$ *and* $END \notin \overrightarrow{\mathcal{R}_f}(A)$*, then* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(A)^* \circ \overrightarrow{\mathcal{R}_f}(B)$*.*

Proof. The proof of the first eight items is straightforward from table 2 and theorem 2.18. The ninth item is Arden's Rule [1] applied in our context. □

5.3 Axiomatic System

We consider the following set of axioms and rules, where p and q are proposition symbols and φ and ψ are formulas.

(sPPDL) The axioms **(PL)**, **(K)**, **(Du)**, **(Pr)** and **(NC)** and the rules **(PC)**, **(Sub)**, **(MP)** and **(Gen)**

(0) $\vdash \neg\langle\mathbf{0}\rangle p$

(END) $\vdash \langle \text{END}\rangle p \leftrightarrow p$

(SC) $\vdash \langle P_1; P_2\rangle p \leftrightarrow \langle P_1\rangle\langle P_2\rangle p$

(Rec) $\vdash \langle P^*\rangle p \leftrightarrow p \vee \langle P\rangle\langle P^*\rangle p$

(FP) $\vdash p \wedge [P^*](p \to [P]p) \to [P^*]p$

(PCSub) If $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$, then $\vdash \langle P|R\rangle p \leftrightarrow \langle Q|R\rangle p$ and $\vdash \langle R|P\rangle p \leftrightarrow \langle R|Q\rangle p$

(RSub) If $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$, then $\vdash \langle P\backslash L\rangle p \leftrightarrow \langle Q\backslash L\rangle p$

(Ard) If $\vdash \langle P\rangle p \leftrightarrow \langle A; P+B\rangle p$ and $A \overset{\text{END}}{\not\to} \surd$, then $\vdash \langle P\rangle p \leftrightarrow \langle A^*; B\rangle p$

(Con) If $P \equiv_r Q$, then $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$

The proof of soundness is analogous to the proof of soundness for sPPDL and PPDL-c. The axioms **(PL)**, **(K)** and **(Du)** and the rules **(Sub)**, **(MP)** and **(Gen)** are standard in the modal logic literature. The soundness of **(Pr)**, **(NC)**, **(0)**, **(SC)**, **(Rec)**, **(FP)**, **(PCSub)**, **(RSub)** and **(Ard)** follows from the set equalities in theorem 5.14 and theorem 3.5. The soundness of **(END)** also follows from the two previous results with the help of definition 5.12. The soundness of **(PC)** and **(Con)** follows from theorems 5.3 and 5.10 with the help of corollary 3.6. The only rule that may require special attention is **(PCSub)**.

Theorem 5.15. *The rule* **(PCSub)** *is sound.*

Proof. By theorem 5.14, $\overrightarrow{\mathcal{R}_f}(P_1|P_2) = \bigcup\{\overrightarrow{\mathcal{R}_f}(\overrightarrow{\alpha}|\overrightarrow{\beta}) : \overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P_1) \text{ and } \overrightarrow{\beta} \in \overrightarrow{\mathcal{R}_f}(P_2)\}$. Now, suppose that $\Vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$, but $\nVdash \langle P|R\rangle p \leftrightarrow \langle Q|R\rangle p$. Then, by theorem 3.5, $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$, but $\overrightarrow{\mathcal{R}_f}(P|R) \neq \overrightarrow{\mathcal{R}_f}(Q|R)$. We may assume, without loss of generality, that there is $\overrightarrow{\lambda}$ such that $\overrightarrow{\lambda} \in \overrightarrow{\mathcal{R}_f}(P|R)$ (*), but $\overrightarrow{\lambda} \notin \overrightarrow{\mathcal{R}_f}(Q|R)$ (**). (*) implies that there is $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ and $\overrightarrow{\beta} \in \overrightarrow{\mathcal{R}_f}(R)$ such that $\overrightarrow{\lambda} \in \overrightarrow{\mathcal{R}_f}(\overrightarrow{\alpha}|\overrightarrow{\beta})$. But then $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(Q)$, which implies that $\overrightarrow{\lambda} \in \overrightarrow{\mathcal{R}_f}(Q|R)$, contradicting (**). □

Definition 5.16. *We define the following relation between processes: $P \leftrightarrow Q$ iff $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$, for any proposition p.*

Theorem 5.17. $\leftrightarrow$ *is a* congruence*.*

Proof. This relation is clearly an equivalence relation and the axioms **(Pr)**, **(SC)**, **(NC)**, **(Rec)** and **(FP)** and the rules **(PCSub)** and **(RSub)** enforce the preservation results needed to satisfy definition 5.5. □

Definition 5.18. *Let $\Omega_k = \{P_1, \ldots, P_k\}$ be a set of processes such that $P_i \not\equiv_r P_j$, if $i \neq j$. Let $E(\Omega_k) = \{E_1, \ldots, E_k\}$ such that $E_i = (P_i, T_i)$, $P_i \leftrightarrow T_i$, $T_i = \sum_j A^i_j; Q^i_j$ and, for all (i, j), A^i_j has no occurrence of $|$. We say that $E(\Omega_k)$ is* closed *if, for all (i, j), $Q^i_j \in \Omega_k$.*

Intuitively, we can think of $E(\Omega_k)$ as a system of equations, where each P_i, $1 \leq i \leq k$, is a variable representing a process and each E_i, $1 \leq i \leq k$, is an equation $P_i = T_i$, where T_i gives a description of the process P_i in terms of all of the variables $\{P_1, \ldots, P_k\}$. In this analogy, the processes A^i_j would be the coefficients of the equations. Below, in theorem 5.19, we show a way to "solve" a particular system of k equations and k variables built like this, such that we find a description T'_1 for P_1 without the occurrence of any of the variables $\{P_1, \ldots, P_k\}$ and also without the occurrence of the $|$ operator.

If we start from a single process $P = P_1|P_2$, where P is unrestricted, we can build a finite set Ω_k such that $P \in \Omega_k$ and $E(\Omega_k)$ is closed using the rule **(PC)**. The Expansion Law applied to P will give us a summation in the form of a process T_i as

described in the definition above, such that the processes A^i_j will be atomic actions and the processes Q^i_j will be added to Ω_k (if one such process already belongs to Ω_k, it is not added again). We then apply **(PC)** to the new processes added to Ω_k and proceed like this until no new process is added to Ω_k. This procedure always stops eventually because the initial process P cannot generate an infinite set of *distinct* processes during its execution.

Theorem 5.19. *Let $P = P_1 \mid P_2$, where P is unrestricted. Then, there is a $\overline{P}$ such that $P \leftrightarrow \overline{P}$ and $\overline{P}$ has no occurrences of the $\mid$ operator.*

Proof. The proof is by induction on the number n of occurrences of the $\mid$ operator in P. If $n = 0$, then $\overline{P} = P$ and there is nothing to be done.

If $n = 1$, then EL can be applied to P. Then, we can use **(PC)** to build pairs (P_i, T_i) that satisfy definition 5.18. Let $P_1 = P$ and Ω_k be the smallest set such that $P_1 \in \Omega_k$ and $E(\Omega_k)$ is closed. Take the pair E_k. If there is no $Q^k_j = P_k$ (*), then we can substitute in the processes T_i, $1 \leq i < k$, all the occurrences of P_k by T_k. Otherwise, we can use **(Ard)** to substitute the pair (P_k, T_k) by a pair (P_k, T'_k) where (*) holds and then proceed as in the previous case. We then continue this process with the pair E_{k-1} and so on, until we finally get a pair (P_1, T'_1) such that no process in Ω_k occurs in T'_1. By the use of **(PC)** to build the initial pairs and the fact that neither **(Ard)** nor the substitution process introduce new $\mid$ operators, we have $\overline{P} = T'_1$. This method, based on the solution of a "system of equations", was inspired by Brzozowski's algebraic method to obtain the regular expression that describes the language accepted by a finite automaton [4].

Suppose that the theorem is true for all $n < k$. Let P have k occurrences of $\mid$. As $P = P_1 \mid P_2$, we can obtain $\overline{P}$ as $\overline{\overline{P_1} \mid \overline{P_2}}$. □

Theorem 5.20 (Completeness)**.** *Every consistent formula is satisfiable in a finite PPDL* model.*

Proof. Let φ be a consistent formula and let $\mathbf{P}(\varphi)$ be the set of processes that appear in φ. For all $P \in \mathbf{P}(\varphi)$, we can use **(Con)**, **(RSub)** and theorems 5.9, 5.17 and 5.19 to get a sequence $P \leftrightarrow P' \leftrightarrow P'' \leftrightarrow P'''$, where P' is r-external form, P'' is also without any occurrence of the $\mid$ operator and P''' is like P'' but unrestricted. We can then obtain an equi-consistent formula $\varphi' = \varphi[P'''/P, P \in \mathbf{P}(\varphi)]$ in which the only XCCS operators that appear are ., ;, + and *. The axioms that deal with all of these operators are analogous to the axioms that deal with the operators in standard PDL. **(Pr)** and **(SC)** are analogous to the axiom of the PDL ; operator, **(NC)** is analogous to the axiom of the PDL $\cup$ operator and **(Rec)** and **(FP)** are analogous to the axioms of the PDL * operator. Thus, we can follow the completeness proof of standard PDL (the PDL axioms and its completeness proof are presented in details in [3]), treating the actions as basic PDL programs, to show that φ' is satisfiable in a finite model. As φ and φ' are equi-consistent and the axiomatic system is sound, they are also semantically equivalent, which means that φ is also satisfied in that same finite model. □

5.4 First Example

Verification of communication protocols is an important subject. In this section, we describe a simple protocol presented in [19, 24]. The protocol has a sender, a

receiver and a channel. The sender wants to send a message to the receiver through the channel. The channel is not reliable and can lose messages.

The sender starts waiting for a message from the application (port *in*), then sends it to the channel (port *smsg*) and waits for an acknowledgment (port *sack*). The channel may lose the message and ask the sender to retransmit it (port *sendagain*). Whenever the receiver receives a message it outputs it and acknowledges it to the channel. In the first version of our protocol we model the loss of a message by the channel as a silent action τ. After performing the τ action the channel sends a *timeout* signal to the sender and the message is retransmitted. We assume that acknowledgment messages cannot be lost. The components of the protocol are illustrated in figure 1 and their CCS specifications (using constants) are given in figure 2. Rewriting the specifications replacing the CCS constants with the iteration operator (we use the operator $^+$ to denote one or more iterations, instead of the operator *, which denotes zero or more iterations), we obtain the specifications in figure 3.

In figures 2, 3, 5 and 6, some terms in summations appear with a parameter x, like the only term in $Sender$. Supposing that the messages received in channel in are from a finite domain X, we can think of in as an abbreviation for $|X|$ different input channels in_i, $i \in X$. So, whenever a term in a summation contains the parameter x, this term is also an abbreviation for the summation of terms where, in each term, the x is replaced by one of the values in X. So, for instance, the term $in(x).\overline{smsg}(x).S_1(x)$ in $Sender$ is an abbreviation to $\sum_{i \in X} in_i.\overline{smsg_i}.S_{1_i}$.

Our $Protocol$ is the parallel composition of the processes $Sender$, $Channel$ and $Receiver$ restricting the ports $smsg, rmsg, sack, rack$ and $sendagain$ to internal communications: $Protocol \equiv (Sender|Channel|Receiver)\backslash L$, where $L = \{smsg, rmsg, sack, rack, sendagain\}$.

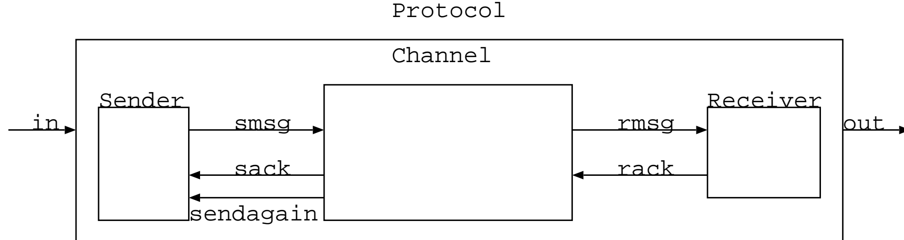

Figure 1: Simple Protocol

Suppose we want to refine the specification of the channel. Instead of using the τ action to model the loss of a message, we use a $Timer$ that sends a *timeout* message after three units of time. $Timer$ uses an internal $Clock$ to count the units of time. The specifications of the $Sender$ and $Receiver$ remains the same. The new channel model is illustrated in figure 4 and their CCS specifications (using constants) are given in figure 5. Rewriting the specification replacing the CCS constants with the iteration operator, we obtain the specifications in figure 6.

The $Protocol'$ is the parallel composition of the processes $Sender$, $Channel'$ and $Receiver$ restricting the ports $\{smsg, rmsg, sack, rack, sendagain\}$ to internal communications: $Protocol' \equiv (Sender|Channel'|Receiver)\backslash L$, where $L =$

Sender

$$Sender \overset{def}{=} in(x).\overline{smsg}(x).S_1(x)$$
$$S_1(x) \overset{def}{=} sack.Sender \ + \ sendagain.\overline{smsg}(x).S_1(x)$$

Channel

$$Channel \overset{def}{=} smsg(x).Ch_1(x)$$
$$Ch_1(x) \overset{def}{=} \overline{rmsg}(x).rack.\overline{sack}.Channel \ + \ \tau.\overline{sendagain}.Channel$$

Receiver

$$Receiver \overset{def}{=} rmsg(x).\overline{out}(x).\overline{rack}.Receiver$$

Figure 2: Specifications for the Simple Protocol (with constants)

Sender

$$Sender \equiv (in(x).\overline{smsg}(x).(sack + (sendagain.\overline{smsg}(x))^+))^+$$

Channel

$$Channel \equiv (smsg(x).(\overline{rmsg}(x).rack.\overline{sack} \ + \ \tau.\overline{sendagain}))^+$$

Receiver

$$Receiver \equiv (rmsg(x).\overline{out}(x).\overline{rack})^+$$

Figure 3: Specifications for the Simple Protocol (with the iteration operator)

$\{smsg, rmsg, \ sack, rack, sendagain\}$.

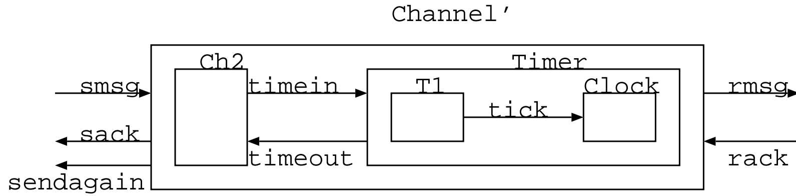

Figure 4: Refined Channel Model

Important parts of the work with processes is to prove the equivalence between two process specifications and to verify whether a concrete system (in our modeling, a Kripke frame) is respecting the desired specification. Given a specification of a process and a refinement of it (a more detailed specification) and given a Kripke frame $\mathcal{F}$ that represents the concrete implementation of the system, it would be interesting to prove that, in $\mathcal{F}$, the two specifications result in equivalent behavior. In our example, we would like to prove the following property:

$$\mathcal{F} \Vdash \langle Protocol \rangle \varphi \leftrightarrow \langle Protocol' \rangle \varphi,$$

for any formula φ.

It can also be interesting to prove that these two protocols produce in $\mathcal{F}$ a behavior which is equivalent to a more abstract specification, as in the formulas below:

$$\mathcal{F} \Vdash \langle Protocol \rangle \varphi \leftrightarrow \langle (in(x).\tau^*.out(x))^+ \rangle \varphi$$
$$\mathcal{F} \Vdash \langle Protocol' \rangle \varphi \leftrightarrow \langle (in(x).\tau^*.out(x))^+ \rangle \varphi$$

Sender

$$
\begin{array}{lll}
Sender & \stackrel{def}{=} & in(x).\overline{smsg}(x).S_1(x) \\
S_1(x) & \stackrel{def}{=} & sack.Sender \ + \ sendagain.\overline{smsg}(x).S_1(x)
\end{array}
$$

Channel

$$
\begin{array}{lll}
Ch_2 & \stackrel{def}{=} & smsg(x).Ch_1(x) \\
Ch_1(x) & \stackrel{def}{=} & \overline{timein}.\overline{rmsg}(x).rack.timeout.\overline{sack}.Ch_2 \ + \\
 & & \overline{timein}.timeout.\overline{sendagain}.Ch_2 \\
Channel' & \stackrel{def}{=} & (Ch_2|Timer)\backslash\{timein, timeout\}
\end{array}
$$

Timer

$$
\begin{array}{lll}
Timer & \stackrel{def}{=} & (T_1|Clock)\backslash\{tick\} \\
T_1 & \stackrel{def}{=} & timein.\overline{tick}.\overline{tick}.\overline{tick}.\overline{timeout}.T_1 \\
Clock & \stackrel{def}{=} & tick.Clock
\end{array}
$$

Receiver

$$
\begin{array}{lll}
Receiver & \stackrel{def}{=} & rmsg(x).\overline{out}(x).\overline{rack}.Receiver
\end{array}
$$

Figure 5: Specifications for the Refined Protocol (with constants)

5.5 Second Example

The parallel composition operator (|) has a dual role in CCS. It represents both interleaving and synchronization of processes. In van Benthem's paradigm of *games-as-processes* [20], this could be used to represent simultaneous games, where each player chooses his actions unaware of what are the actions taken by the other player. We consider a concrete example. Let $\mathcal{M}$ be a game board and the proposition symbols w_i, $i = 1, 2$, denote that player i wins if the game reaches a state where w_i is satisfied. Let $\{a, b, c\}$ be the possible actions for player 1 and $\{d, e, f\}$ the possible actions for player 2. Each player has to perform a sequence of three actions, completely unaware of which actions the other player performed or even how many of the three actions the other player performed so far. This means that the two sequences are interleaved in an arbitrary order. Then, $\mathcal{M}, w \Vdash \langle a.b.b.\text{END} + a.b.c.\text{END} \mid d.d.d.\text{END}\rangle w_1$ means that if the game starts in w, there is an interleaved sequence of a, b, b and d, d, d or a, b, c and d, d, d that leads to a victory of player 1. On the other hand, $\mathcal{M}, w \Vdash [a.b.b.\text{END} \mid d.d.d.\text{END}]w_1$ means that, if the game starts in w, no matter in what order the six actions take place, if player 1 plays a, b, b and player 2 plays d, d, d, player 1 is guaranteed to win. This can also be generalized to games with more than two players.

The | operator can also represent synchronization of processes. This allows us to model richer games, where we can get "rounds" of blind, simultaneous games explicitly divided by some synchronization between the players. For instance, in the game described above, suppose now that the two players may select their actions from the same set $\{a, b\}$. To differentiate between the sequences of actions of each player, each sequence starts with p_i, $i = 1, 2$. Besides that, each player will now perform the three actions in the following way: each player performs two actions freely, with no regard to how many actions the other player has performed so far, then each one only performs its third action after they have both performed the

Sender

$$
\begin{array}{lll}
Sender & \equiv & (in(x).\overline{smsg}(x).(sack + (sendagain.\overline{smsg}(x))^{+}))^{+}
\end{array}
$$

Channel

$$
\begin{array}{lll}
Ch_2 & \equiv & (smsg(x).(\overline{timein}.\overline{rmsg}(x).rack.timeout.sack\ + \\
 & & \overline{timein}.timeout.\overline{sendagain}))^{+} \\
Channel' & \equiv & (Ch_2|Timer)\backslash\{timein, timeout\}
\end{array}
$$

Timer

$$
\begin{array}{lll}
Timer & \equiv & (T_1|Clock)\backslash\{tick\} \\
T_1 & \equiv & (timein.\overline{tick}.\overline{tick}.\overline{tick}.\overline{timeout})^{+} \\
Clock & \equiv & (tick)^{+}
\end{array}
$$

Receiver

$$
\begin{array}{lll}
Receiver & \equiv & (rmsg(x).\overline{out}(x).\overline{rack})^{+}
\end{array}
$$

Figure 6: Specifications for the Refined Protocol (with the iteration operator)

first two (synchronization). Then we can express properties of this game using formulas of our logic, in an analogous way to the paragraph above. For instance, $\mathcal{M}, w \Vdash [(p_1.b.b.\overline{s}.a.\text{END} \mid p_2.a.b.s.a.\text{END} + p_2.a.b.s.b.\text{END})\backslash\{s\}]w_1$ means that if the game starts in w, no matter in what order the four initial and the two final actions take place, if player 2 starts with a, b, then player 1 can always win by playing b, b and then finishing with a.

This interplay between interleaving and synchronization can then be used to describe a fairly large group of games. A recent paper [21] also works with this idea that concurrency operators can be used to model simultaneous games. The authors use CPDL [18] as a stepping stone to build a concurrent dynamic game logic. However, since CPDL does not admit communication, their logic also has that limitation. As the generalization from CPDL to channel-CPDL has as drawbacks the loss of decidability and the loss of a complete axiomatization, our logic may be better suited for the generalization of the logic presented in [21] to deal with simultaneous games with communication, as we briefly illustrated.

6 Final Remarks and Future Work

In this work, we present three increasingly expressive Dynamic Logics in which the programs are described in a language based on CCS. From the point of view of dynamic logics, these logics represent an improvement on the current scenario, as previous dynamic logics could not effectively deal with both concurrency and communication. CPDL [18] dealt with concurrency, but there was no possibility of communication between the components of a concurrent system. Channel-CPDL [17] models concurrency and communication but it has a "rather complicated" [17] semantics, is undecidable and lacks a complete axiomatization. On the other hand, we are able to provide a simple Kripke semantics for our logics, based on the idea of finite possible runs of processes, build complete axiomatizations for them and show that they have the finite model property.

We also provide a method, in a language with a iteration (*) and sequential composition (;) operators, to rewrite any process specification to a form without the parallel composition operator (|) while preserving the set of finite possible runs

of the process. This method is based on Brzozowski's algorithm to find the regular expression that corresponds to a finite automaton [4]. We feel that this is an interesting and original application of Brzozowski's idea and that it provides an elegant proof to a key result to the completeness of our last axiomatization.

It should also be noticed that, while the $|$ operator can be eliminated from the specifications, in practice it can be very hard to describe a complex concurrent behavior without it from the start. Besides that, even though both specifications, with and without $|$, may be equivalent, the one with $|$ will be much more succinct. We leave it to future work the interesting problem of determining an asymptotic bound on the relative succinctness of PPDL* as compared do PDL.

As a continuation of the present work, it would be interesting to study the complexity of the satisfiability and model-checking problems for these logics, possibly relating them to the satisfiability and model-checking problems for standard PDL. Another line of work would be to try to develop an automatic theorem prover for these logics. This would involve, among other things, an efficient algorithmic method to deal with the expansion of parallel processes and, in the particular case of PPDL-c, an efficient algorithmic method to determine the processes L_P and T_P related to a knot process P. It would also be interesting to study the effects of the addition of the PDL test operator (?) to this family of logics.

A natural extension of CCS is the π-Calculus [15], in which the acts of communication are more complex than in CCS. With this extended expressive power, the π-Calculus is a very powerful process algebra that is able to describe not only nondeterminism and concurrency, but also *mobility* of processes and that can also be used to encode some powerful programming paradigms, as object-oriented programming and functional programming (λ-Calculus) [15]. Besides that, the π-Calculus has a specific operator to denote that a process has the ability to self-replicate. So, even though we have already developed a simple extension of our third logic to deal with a subset of π-Calculus processes in [2], we feel that the study of Dynamic Logics based on the π-Calculus can and should be further developed. For instance, it would be interesting to develop a logic that could deal explicitly with the mobility of processes or programs. The π-Calculus could also provide an interesting context to analyze in more depth the issue of self-replicating processes, which was left out of the present work, since its replication operator provides a simpler way of describing self-replicating behavior than the constants of CCS.

As another interesting future project, the possibility of modeling communication, concurrency and synchronization offered by both CCS and the π-Calculus could be used to adapt the logics presented in this present work to build a Dynamic Epistemic Logic (DEL) [22] with a concurrency operator.

Finally, we would also like to study in more detail the possible connections between our logic and the ideas presented in [21] and analyze how to use our logic as a tool for the description of simultaneous games with communication.

References

[1] D. N. Arden. Delayed logic and finite state machines. In *Theory of Computing Machine Design*, pages 1–35. University of Michigan Press, 1960.

[2] M. R. F. Benevides and L. M. Schechter. A propositional dynamic logic for concurrent programs based on the pi-calculus. In *Proceedings of the VI Workshop on Methods for Modalities*, volume 262 of *Electronic Notes in Theoretical Computer Science*, pages 49–64. Elsevier, 2010.

[3] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Theoretical Tracts in Computer Science. Cambridge University Press, 2001.

[4] J. A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[5] M. Dam. On the decidability of process equivalences for the pi-calculus. *Theoretical Computer Science*, 183(2):215–228, 1997.

[6] V. L. P. dos Santos. *Concorrência e Sincronização para Lógica Dinâmica de Processos*. PhD thesis, Federal University of Rio de Janeiro, 2005.

[7] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[8] W. J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science. Springer, 2000.

[9] R. Goldblatt. Parallel action: Concurrent dynamic logic with independent modalities. *Studia Ligica*, 51(3–4):551–578, 1992.

[10] D. Harel and M. Kaminsky. Strengthened results on nonregular PDL. Technical Report MCS99-13, Faculty of Mathematics and Computer Science, Weizmann Institute of Science, 1999.

[11] D. Harel and D. Raz. Deciding properties of nonregular programs. *SIAM Journal on Computing*, 22(4):857–874, 1993.

[12] C. Löding, C. Lutz, and O. Serre. Propositional dynamic logic with recursive programs. *Journal of Logic and Algebraic Programming*, 73(1–2):51–69, 2007.

[13] A. J. Mayer and L. J. Stockmeyer. The complexity of PDL with interleaving. *Theoretical Computer Science*, 161(1–2):109–122, 1996.

[14] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[15] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[16] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171, 1993.

[17] D. Peleg. Communication in concurrent dynamic logic. *Journal of Computer and System Sciences*, 35(1):23–58, 1987.

[18] D. Peleg. Concurrent dynamic logic. *Journal of the Association for Computing Machinery*, 34(2):450–479, 1987.

[19] C. Stirling. *Modal and Temporal Properties of Processes.* Texts in Computer Science. Springer, 2001.

[20] J. van Benthem. Extensive games as process models. *Journal of Logic, Language and Information*, 11(3):289–313, 2002.

[21] J. van Benthem, S. Ghosh, and F. Liu. Modelling simultaneous games in dynamic logic. *Synthese*, 165(2):247–268, 2008.

[22] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic.* Springer, 2007.

[23] R. J. van Glabbeek. The linear time - branching time spectrum I: The semantics of concrete, sequential processes. In *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.

[24] D. Walker. An introduction to a calculus of communicating systems. Technical Report ECS-LFCS-87-22, Dept. of Computer Science - University of Edinburgh, 1987.

A Completeness Proof for PPDL-c

Definition A.1. *Let ϕ be a formula. We define the formula $\overline{\phi}$ as $\overline{\phi} = \psi$, if $\phi = \neg\psi$, or $\overline{\phi} = \neg\phi$, otherwise.*

Definition A.2 (Fischer-Ladner Closure)**.** *Let Γ be a set of formulas. The* Fischer-Ladner Closure *of Γ, notation $C(\Gamma)$, is the smallest set of formulas that contains Γ and satisfies the following conditions:*

- *$C(\Gamma)$ is closed under sub-formulas;*
- *if $\phi \in C(\Gamma)$, then $\overline{\phi} \in C(\Gamma)$;*
- *For knot processes:*
  - *If $\langle P\rangle\varphi \in C(\Gamma)$, then $\langle T_P\rangle\varphi \vee \langle L_P\rangle\langle P\rangle\varphi \in C(\Gamma)$.*
- *For non-knot processes:*
  - *If $\langle \alpha.P\rangle\varphi \in C(\Gamma)$, then $\langle\alpha\rangle\langle P\rangle\varphi \in C(\Gamma)$;*
  - *If $\langle \alpha.A\rangle\varphi \in C(\Gamma)$, then $\langle\alpha\rangle\langle P_A\rangle\varphi \in C(\Gamma)$;*
  - *If $\langle P_1 + P_2\rangle\varphi \in C(\Gamma)$, then $\langle P_1\rangle\varphi \vee \langle P_2\rangle\varphi \in C(\Gamma)$;*
  - *If $\langle P_1 \mid P_2\rangle\varphi \in C(\Gamma)$, then $\bigvee_{P_1 \xrightarrow{\alpha} P_1'}\langle\alpha\rangle\langle P_1' \mid P_2\rangle\varphi \vee \bigvee_{P_2 \xrightarrow{\alpha} P_2'}\langle\alpha\rangle\langle P_1 \mid P_2'\rangle\varphi \vee \bigvee_{\substack{P_1 \xrightarrow{\lambda} P_1' \\ P_2 \xrightarrow{\overline{\lambda}} P_2'}}\langle\tau\rangle\langle P_1' \mid P_2'\rangle\varphi \in C(\Gamma)$.*

The proof that if Γ is finite, then the closure $C(\Gamma)$ is also finite is standard. We assume Γ to be finite from now on.

Definition A.3. *A set of formulas $\mathcal{A}$ is said to be an* atom *over Γ if it is a maximal consistent subset of $C(\Gamma)$. The set of all atoms over Γ is denoted by $At(\Gamma)$. We denote the conjunction of all the formulas in an atom $\mathcal{A}$ as $\bigwedge\mathcal{A}$.*

Lemma A.4. *Every atom $\mathcal{A} \in At(\Gamma)$ has the following properties:*

1. *For every $\phi \in C(\Gamma)$, exactly one of ϕ and $\neg\phi$ belongs to $\mathcal{A}$.*
2. *For every $\phi \wedge \psi \in C(\Gamma)$, $\phi \wedge \psi \in \mathcal{A}$ iff $\phi \in \mathcal{A}$ and $\psi \in \mathcal{A}$.*

Proof. This follows immediately from the definition of atoms as maximal consistent subsets of $C(\Gamma)$. □

Lemma A.5. *If $\Delta \subseteq C(\Gamma)$ and Δ is consistent then there exists an atom $\mathcal{A} \in At(\Gamma)$ such that $\Delta \subseteq \mathcal{A}$.*

Proof. We can construct the atom $\mathcal{A}$ as follows. First, we enumerate the elements of $C(\Gamma)$ as $\phi_1, \ldots, \phi_n$. We start the construction making $\mathcal{A}_0 = \Delta$. Then, for $0 \leq i < n$, we know that $\bigwedge \mathcal{A}_i \leftrightarrow (\bigwedge \mathcal{A}_i \wedge \phi_{i+1}) \vee (\bigwedge \mathcal{A}_i \wedge \overline{\phi_{i+1}})$ is a tautology and therefore either $\mathcal{A}_i \cup \{\phi_{i+1}\}$ or $\mathcal{A}_i \cup \{\overline{\phi_{i+1}}\}$ is consistent. We take $\mathcal{A}_{i+1}$ as the consistent extension. At the end, we make $\mathcal{A} = \mathcal{A}_n$. □

Corollary A.6. *If $\varphi \in C(\Gamma)$ is a consistent formula, then there is an atom $\mathcal{A} \in At(\Gamma)$ such that $\varphi \in \mathcal{A}$.*

Definition A.7 (Canonical model over Γ)**.** *Let Γ be a finite set of formulas. The* canonical model *over Γ is the tuple $\mathcal{M}^\Gamma = (At(\Gamma), \{S_\alpha\}, \mathbf{V})$ where, for all elements $p \in \Phi$, we have $\mathbf{V}(p) = \{\mathcal{A} \in At(\Gamma) \mid p \in \mathcal{A}\}$ and for all atoms $\mathcal{A}, \mathcal{B} \in At(\Gamma)$,*

$$\mathcal{A} S_\alpha \mathcal{B} \text{ iff } \bigwedge \mathcal{A} \wedge \langle \alpha \rangle \bigwedge \mathcal{B} \text{ is consistent.}$$

$\mathbf{V}$ *is called the* canonical valuation *and S_α the* canonical relations*, where α is a CCS action.*

Definition A.8. *We write $\mathcal{A} \overset{P}{\twoheadrightarrow} \mathcal{B}$ if and only if $\bigwedge \mathcal{A} \wedge \langle P \rangle \bigwedge \mathcal{B}$ is consistent. We also write $S_P = \{(\mathcal{A}, \mathcal{B}) : \mathcal{A} \overset{P}{\twoheadrightarrow} \mathcal{B}\}$.*

Lemma A.9 (Existence Lemma for Basic Processes)**.** *Let $\mathcal{A}$ be an atom and let α be an action. Then, for all formulas $\langle \alpha \rangle \phi \in C(\Gamma)$, $\langle \alpha \rangle \phi \in \mathcal{A}$ iff there is a $\mathcal{B} \in At(\Gamma)$ such that $\mathcal{A} S_\alpha \mathcal{B}$ and $\phi \in \mathcal{B}$.*

Proof. ($\Rightarrow$) Suppose $\langle \alpha \rangle \phi \in \mathcal{A}$. We can build an appropriate atom $\mathcal{B}$ by forcing choices. Enumerate the formulas in $C(\Gamma)$ as $\phi_1, \ldots, \phi_n$. Define $\mathcal{B}_0 = \{\phi\}$. Suppose, as an inductive hypothesis that $\mathcal{B}_m$ is defined such that $\bigwedge \mathcal{A} \wedge \langle \alpha \rangle \bigwedge \mathcal{B}_m$ is consistent, for $0 \leq m < n$. We have that

$$\vdash \langle \alpha \rangle \bigwedge \mathcal{B}_m \leftrightarrow \langle \alpha \rangle ((\bigwedge \mathcal{B}_m \wedge \phi_{m+1}) \vee (\bigwedge \mathcal{B}_m \wedge \overline{\phi_{m+1}})),$$

thus

$$\vdash \langle \alpha \rangle \bigwedge \mathcal{B}_m \leftrightarrow (\langle \alpha \rangle (\bigwedge \mathcal{B}_m \wedge \phi_{m+1}) \vee \langle \alpha \rangle (\bigwedge \mathcal{B}_m \wedge \overline{\phi_{m+1}})).$$

Therefore, either for $\mathcal{B}' = \mathcal{B}_m \cup \{\phi_{m+1}\}$ or for $\mathcal{B}' = \mathcal{B}_m \cup \{\overline{\phi_{m+1}}\}$, we have that $\bigwedge \mathcal{A} \wedge \langle \alpha \rangle \bigwedge \mathcal{B}'$ is consistent. We take $\mathcal{B}_{m+1}$ as the consistent extension. At the end, we make $\mathcal{B} = \mathcal{B}_n$. We have that $\phi \in \mathcal{B}$ and, as $\bigwedge \mathcal{A} \wedge \langle \alpha \rangle \bigwedge \mathcal{B}$ is consistent, $\mathcal{A} S_\alpha \mathcal{B}$, by definition A.7.

($\Leftarrow$): Suppose that there is an atom $\mathcal{B}$ such that $\phi \in \mathcal{B}$ and $\mathcal{A} S_\alpha \mathcal{B}$. Then $\bigwedge \mathcal{A} \wedge \langle \alpha \rangle \bigwedge \mathcal{B}$ is consistent by definition A.7. As ϕ is one of the conjuncts of $\bigwedge \mathcal{B}$, $\bigwedge \mathcal{A} \wedge \langle \alpha \rangle \phi$ is also consistent. As $\langle \alpha \rangle \phi$ is in $C(\Gamma)$, it must also be in $\mathcal{A}$, since $\mathcal{A}$ is a maximal consistent subset of $C(\Gamma)$. □

Lemma A.10. *For all knot processes P, $S_P \subseteq S'_P$, where $S'_P = S^*_{L_P} \circ S_{T_P}$.*

Proof. For an atom $\mathcal{B} \in At(\Gamma)$ and a relation S, we denote the set of atoms $\{\mathcal{A} \mid \mathcal{A}S\mathcal{B}\}$ as $\langle S\rangle\mathcal{B}$. Suppose there are two atoms $\mathcal{A}, \mathcal{B} \in At(\Gamma)$ such that $\mathcal{A} \in \langle S_P\rangle\mathcal{B}$, but $\mathcal{A} \notin \langle S'_P\rangle\mathcal{B}$. Let $V = \{\mathcal{C} \in At(\Gamma) \mid \mathcal{C} \in \langle S_P\rangle\mathcal{B}$ but $\mathcal{C} \notin \langle S'_P\rangle\mathcal{B}\} \cup \{\mathcal{C} \in At(\Gamma) \mid \mathcal{C} \notin \langle S_P\rangle\mathcal{B}\}$ and $\overline{V} = At(\Gamma) \setminus V = \{\mathcal{C} \in At(\Gamma) \mid \mathcal{C} \in \langle S_P\rangle\mathcal{B}$ and $\mathcal{C} \in \langle S'_P\rangle\mathcal{B}\}$. Thus, $\mathcal{A} \in V$.

Let $r = \bigvee\{\bigwedge\mathcal{C} \mid \mathcal{C} \in V\}$. The way we define the formula r, it is satisfied in a state $\mathcal{C}$ of the model if and only if $\mathcal{C} \in V$. So, $\neg r$ is going to be satisfied in a state $\mathcal{C}$ if and only if $\mathcal{C} \notin V$, or, equivalently, if and only if $\mathcal{C} \in \overline{V}$. Hence, $\neg r$ is semantically equivalent to the formula $\bigvee\{\bigwedge\mathcal{C} \mid \mathcal{C} \in \overline{V}\}$.

First, we have that $\vdash r \to [T_P]\neg\bigwedge\mathcal{B}$. Otherwise, $\neg(r \to [T_P]\neg\bigwedge\mathcal{B}) \equiv r \wedge \langle T_P\rangle\bigwedge\mathcal{B}$ is consistent. This means that there is $\mathcal{A}' \in V$ such that $\bigwedge\mathcal{A}' \wedge \langle T_P\rangle\bigwedge\mathcal{B}$ is consistent. On one hand, this implies, by **(Rec)**, that $\bigwedge\mathcal{A}' \wedge \langle P\rangle\bigwedge\mathcal{B}$ is consistent, which means that $\mathcal{A}' \in \langle S_P\rangle\mathcal{B}$. On the other hand, it implies that $\mathcal{A}'S_{T_P}\mathcal{B}$, which means that $\mathcal{A}' \in \langle S'_P\rangle\mathcal{B}$. These two conclusions contradict the fact that $\mathcal{A}' \in V$.

Second, we also have that $\vdash r \to [L_P]r$. Otherwise, $\neg(r \to [L_P]r) \equiv r \wedge \langle L_P\rangle\neg r$ is consistent. This means that there are $\mathcal{A}' \in V$ and $\mathcal{B}' \in \overline{V}$ such that $\bigwedge\mathcal{A}' \wedge \langle L_P\rangle\bigwedge\mathcal{B}'$ is consistent, which implies that $\mathcal{A}'S_{L_P}\mathcal{B}'$. Since $\mathcal{B}' \in \overline{V}$, $\mathcal{B}'S_P\mathcal{B}$ and $\mathcal{B}'S'_P\mathcal{B}$. On one hand, $\mathcal{A}'S_{L_P}\mathcal{B}'$ and $\mathcal{B}'S'_P\mathcal{B}$ imply that $\mathcal{A}'S'_P\mathcal{B}$ (*). On the other hand, $\mathcal{A}'S_{L_P}\mathcal{B}'$ and $\mathcal{B}'S_P\mathcal{B}$ imply that $\bigwedge\mathcal{A}' \wedge \langle L_P\rangle\langle P\rangle\bigwedge\mathcal{B}$ is consistent, which, by **(Rec)**, implies that $\bigwedge\mathcal{A}' \wedge \langle P\rangle\bigwedge\mathcal{B}$ is consistent, which means that $\mathcal{A}'S_P\mathcal{B}$ (**). The conclusions in (*) and (**) contradict the fact that $\mathcal{A}' \in V$.

Taking these two results together, we conclude that $\vdash r \to ([T_P]\neg\bigwedge\mathcal{B} \wedge [L_P]r)$. By **(Gen)**, **(PL)**, **(FP)** and **(MP)**, $\vdash r \to [P]\neg\bigwedge\mathcal{B}$. But, as $\mathcal{A} \in V$, $\vdash \bigwedge\mathcal{A} \to r$, which means that $\vdash \bigwedge\mathcal{A} \to [P]\neg\bigwedge\mathcal{B}$. This implies that $\bigwedge\mathcal{A} \wedge \langle P\rangle\bigwedge\mathcal{B}$ is inconsistent, contradicting the fact that $\mathcal{A}S_P\mathcal{B}$. Thus, there cannot be a pair of atoms $\mathcal{A}, \mathcal{B} \in At(\Gamma)$ such that $\mathcal{A} \in \langle S_P\rangle\mathcal{B}$, but $\mathcal{A} \notin \langle S'_P\rangle\mathcal{B}$. □

Definition A.11. *We write $\mathcal{A} \overset{P}{\rightsquigarrow} \mathcal{B}$ if and only if there is a path in the canonical model starting in $\mathcal{A}$ and ending in $\mathcal{B}$ such that there is $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ that matches it. We also write $R_P = \{(\mathcal{A}, \mathcal{B}) : \mathcal{A} \overset{P}{\rightsquigarrow} \mathcal{B}\}$. Finally, it also follows from this definition that $\mathcal{M}^\Gamma, \mathcal{A} \Vdash \langle P\rangle\varphi$ if and only if there is $\mathcal{B}$ such that $(\mathcal{A}, \mathcal{B}) \in R_P$ and $\mathcal{M}^\Gamma, \mathcal{B} \Vdash \varphi$.*

The proofs of the following two lemmas proceed by induction. In order to define a suitable induction order for these proofs, we present the notion of *magnitude* of a process P, denoted by $\mathbb{M}(P)$. This is necessary because, in these lemmas, the induction is not done, as in most cases, directly in the structure of the logical formulas or of the processes. It is done as a traditional induction over the natural numbers. We associate to each process a positive natural number, that we call the *magnitude* of the process, and we define this magnitude of the processes in a way that, in the induction proof of the lemmas, the proof of the lemma for a given formula depends only of the proof of the lemma for formulas with processes that have magnitude strictly smaller than the magnitude of the process in the original formula. The magnitude is a function from the set of processes to the set of positive natural numbers. The function has several sub-cases, depending on the format of the process, but it is well-defined in the sense that there is always exactly one of its sub-cases that can be applied to a certain process (the function is total and is not ambiguous).

Definition A.12. *The notion of* magnitude *of a process P, denoted by $\mathbb{M}(P)$, is recursively defined as follows:*

- *If P is an atomic action, then $\mathbb{M}(P) = 1$;*
- *If P is a non-knot process:*
  - *If $P = \alpha.P'$, then $\mathbb{M}(P) = \mathbb{M}(P') + 1$;*
  - *If $P = \alpha.A$, then $\mathbb{M}(P) = \mathbb{M}(P_A) + 1$;*
  - *If $P = P_1 + P_2$, then $\mathbb{M}(P) = max(\mathbb{M}(P_1), \mathbb{M}(P_2)) + 1$;*
  - *If $P = P_1|P_2$, then $\mathbb{M}(P) = \mathbb{M}(Exp(P))$;*
- *If P is a knot process, then $\mathbb{M}(P) = max(\mathbb{M}(L_P), \mathbb{M}(T_P)) + 1$.*

Lemma A.13. *For all processes P, $S_P \subseteq R_P$.*

Proof. The proof is by induction on the magnitude of the process P. In order to prove the lemma for a process P, we use, as the induction hypothesis, that the lemma is true for all processes that have smaller magnitude than P.

- If P is an action α, then the proof is straightforward. First, $\overrightarrow{\mathcal{R}_f}(P) = \{\alpha\}$. Now, if $\mathcal{A}S_\alpha\mathcal{B}$, then there is a path in the canonical model starting in $\mathcal{A}$ and ending in $\mathcal{B}$ such that there is $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ that matches it. Hence, $\mathcal{A}R_\alpha\mathcal{B}$ is true as well.
- P is a non-knot process:
  - Suppose $\mathcal{A}S_{\alpha.P}\mathcal{B}$, that is, $\bigwedge\mathcal{A}\wedge\langle\alpha.P\rangle\bigwedge\mathcal{B}$ is consistent. By **(Pr)**, $\bigwedge\mathcal{A}\wedge\langle\alpha\rangle\langle P\rangle\bigwedge\mathcal{B}$ is consistent as well. Using a "forcing choices" argument (as exemplified in lemma A.9), we can construct an atom $\mathcal{C}$ such that $\bigwedge\mathcal{A}\wedge\langle\alpha\rangle\bigwedge\mathcal{C}$ and $\bigwedge\mathcal{C}\wedge\langle P\rangle\bigwedge\mathcal{B}$ are both consistent. But then, by the inductive hypothesis, $\mathcal{A}R_\alpha\mathcal{C}$ and $\mathcal{C}R_P\mathcal{B}$. It follows that $\mathcal{A}R_{\alpha.P}\mathcal{B}$ as required.
  - Suppose $\mathcal{A}S_{\alpha.A}\mathcal{B}$, that is, $\bigwedge\mathcal{A}\wedge\langle\alpha.A\rangle\bigwedge\mathcal{B}$ is consistent. By **(Cons)**, $\bigwedge\mathcal{A}\wedge\langle\alpha\rangle\langle P_A\rangle\bigwedge\mathcal{B}$ is consistent as well. Using a "forcing choices" argument, we can construct an atom $\mathcal{C}$ such that $\bigwedge\mathcal{A}\wedge\langle\alpha\rangle\bigwedge\mathcal{C}$ and $\bigwedge\mathcal{C}\wedge\langle P_A\rangle\bigwedge\mathcal{B}$ are both consistent. But then, by the inductive hypothesis, $\mathcal{A}R_\alpha\mathcal{C}$ and $\mathcal{C}R_{P_A}\mathcal{B}$. It follows that $\mathcal{A}R_{\alpha.A}\mathcal{B}$ as required.
  - Suppose $\mathcal{A}S_{P_1+P_2}\mathcal{B}$, that is, $\bigwedge\mathcal{A}\wedge\langle P_1+P_2\rangle\bigwedge\mathcal{B}$ is consistent. By **(NC)**, $\bigwedge\mathcal{A}\wedge\langle P_1\rangle\bigwedge\mathcal{B}$ is consistent or $\bigwedge\mathcal{A}\wedge\langle P_2\rangle\bigwedge\mathcal{B}$ is consistent. But then, by the inductive hypothesis, $\mathcal{A}R_{P_1}\mathcal{B}$ or $\mathcal{A}R_{P_2}\mathcal{B}$. It follows that $\mathcal{A}R_{P_1+P_2}\mathcal{B}$ as required.
  - Suppose $\mathcal{A}S_{P_1|P_2}\mathcal{B}$, that is, $\bigwedge\mathcal{A}\wedge\langle P_1 \mid P_2\rangle\bigwedge\mathcal{B}$ is consistent. By **(PC)**, $\bigwedge\mathcal{A}\wedge\langle\alpha\rangle\langle P'\rangle\bigwedge\mathcal{B}$ is consistent for some basic process α and some process P'. Using a "forcing choices" argument, we can construct an atom $\mathcal{C}$ such that $\bigwedge\mathcal{A}\wedge\langle\alpha\rangle\bigwedge\mathcal{C}$ and $\bigwedge\mathcal{C}\wedge\langle P'\rangle\bigwedge\mathcal{B}$ are both consistent. But then, by the inductive hypothesis, $\mathcal{A}R_\alpha\mathcal{C}$ and $\mathcal{C}R_{P'}\mathcal{B}$. It follows that $\mathcal{A}R_{\alpha.P'}\mathcal{B}$, which means that $\mathcal{A}R_{P_1|P_2}\mathcal{B}$ as required.

- Suppose $\mathcal{A}S_P\mathcal{B}$, where P is a knot process. By lemma A.10, $S_P \subseteq S'_P$, where $S'_P = S^*_{L_P} \circ S_{T_P}$. By the induction hypothesis, $S_{L_P} \subseteq R_{L_P}$ and $S_{T_P} \subseteq R_{T_P}$. This implies that $S'_P \subseteq R_P$, which proves the result.

□

Lemma A.14 (Existence Lemma)**.** *For all atoms $\mathcal{A} \in At(\Gamma)$ and all formulas $\langle P \rangle \phi \in C(\Gamma)$, $\langle P \rangle \phi \in \mathcal{A}$ iff there is $\mathcal{B} \in At(\Gamma)$ such that $\mathcal{A}R_P\mathcal{B}$ and $\phi \in \mathcal{B}$.*

Proof. ($\Rightarrow$) Suppose $\langle P \rangle \phi \in \mathcal{A}$. We can build an atom $\mathcal{B}$ such that $\phi \in \mathcal{B}$ and $\mathcal{A}S_P\mathcal{B}$ by "forcing choices". But, by lemma A.13, $S_P \subseteq R_P$, thus $\mathcal{A}R_P\mathcal{B}$ as well.

($\Leftarrow$) We proceed by induction on the magnitude of the process P. In order to prove the lemma for a process P, we use, as the induction hypothesis, that the lemma is true for all processes that have smaller magnitude than P.

- The base case is just the Existence Lemma for basic processes.
- P is a non-knot process:
  - Suppose P has the form $\alpha.P'$, $\mathcal{A}R_{\alpha.P'}\mathcal{B}$ and $\phi \in \mathcal{B}$. Thus, there is an atom $\mathcal{C}$ such that $\mathcal{A}R_\alpha\mathcal{C}$ and $\mathcal{C}R_{P'}\mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle P' \rangle \phi \in C(\Gamma)$, hence by the induction hypothesis, $\langle P' \rangle \phi \in C$. Similarly, as $\langle \alpha \rangle \langle P' \rangle \phi \in C(\Gamma)$, $\langle \alpha \rangle \langle P' \rangle \phi \in \mathcal{A}$. Hence, by **(Pr)**, $\langle \alpha.P' \rangle \phi \in \mathcal{A}$.
  - Suppose P has the form $\alpha.A$, $\mathcal{A}R_{\alpha.A}\mathcal{B}$ and $\phi \in \mathcal{B}$. Thus, there is an atom $\mathcal{C}$ such that $\mathcal{A}R_\alpha\mathcal{C}$, $\mathcal{C}R_{P_A}\mathcal{B}$ and $\phi \in \mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle P_A \rangle \phi \in C(\Gamma)$, hence by the induction hypothesis, $\langle P_A \rangle \phi \in C$. Similarly, as $\langle \alpha \rangle \langle P_A \rangle \phi \in C(\Gamma)$, $\langle \alpha \rangle \langle P_A \rangle \phi \in \mathcal{A}$. Hence, by **(Cons)**, $\langle \alpha.A \rangle \phi \in \mathcal{A}$.
  - Suppose P has the form $P_1 + P_2$, $\mathcal{A}R_{P_1+P_2}\mathcal{B}$ and $\phi \in \mathcal{B}$. Thus, $\mathcal{A}R_{P_1}\mathcal{B}$ or $\mathcal{A}R_{P_2}\mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle P_1 \rangle \phi, \langle P_2 \rangle \phi \in C(\Gamma)$, hence by the inductive hypothesis, $\langle P_1 \rangle \phi \in \mathcal{A}$ or $\langle P_2 \rangle \phi \in \mathcal{A}$. Hence, by **(NC)**, $\langle P_1 + P_2 \rangle \phi \in \mathcal{A}$.
  - Suppose P has the form $P_1 \mid P_2$, $\mathcal{A}R_{P_1 \mid P_2}\mathcal{B}$ and $\phi \in \mathcal{B}$. Thus, $\mathcal{A}R_{\alpha.P'}\mathcal{B}$ for some process α and some process P'. Then, there is an atom $\mathcal{C}$ such that $\mathcal{A}R_\alpha\mathcal{C}$ and $\mathcal{C}R_{P'}\mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle \alpha.P' \rangle \phi, \langle \alpha \rangle \langle P' \rangle \phi, \langle P' \rangle \phi \in C(\Gamma)$, hence by the inductive hypothesis, $\langle P' \rangle \phi \in \mathcal{C}$ and $\langle \alpha \rangle \langle P' \rangle \phi \in \mathcal{A}$. Hence, by **(Pr)**, $\langle \alpha.P' \rangle \phi \in \mathcal{A}$ and, by **(PC)**, $\langle P_1 \mid P_2 \rangle \phi \in \mathcal{A}$.
- Suppose P is a knot process, $\mathcal{A}R_P\mathcal{B}$ and $\phi \in \mathcal{B}$. Then, there is a finite sequence of atoms $\mathcal{C}_0 \dots \mathcal{C}_n$ such that $\mathcal{A} = \mathcal{C}_0 R_{L_P} \mathcal{C}_1 \dots \mathcal{C}_{n-1} R_{L_P} \mathcal{C}_n R_{T_P} \mathcal{B}$. We prove by a sub-induction on n that $\langle P \rangle \phi \in \mathcal{C}_i$, for all i. The desired result for $\mathcal{A} = \mathcal{C}_0$ follows immediately.
  - Base case: $n = 0$. This means $\mathcal{A}R_{T_P}\mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle T_P \rangle \phi \in C(\Gamma)$, hence by the inductive hypothesis, $\langle T_P \rangle \phi \in \mathcal{A}$. Hence, by **(Rec)**, $\langle P \rangle \phi \in \mathcal{A}$.

– Inductive step: Suppose the result holds for $k < n$, and that $\mathcal{A} = \mathcal{C}_0 R_{L_P} \mathcal{C}_1 \ldots R_{L_P} \mathcal{C}_n R_{T_P} \mathcal{B}$. By the inductive hypothesis, $\langle P \rangle \phi \in \mathcal{C}_1$. Hence $\langle L_P \rangle \langle P \rangle \phi \in \mathcal{A}$, as $\langle L_P \rangle \langle P \rangle \phi \in C(\Gamma)$. By **(Rec)**, we have that$\langle P \rangle \phi \in \mathcal{A}$.

□

Lemma A.15 (Truth Lemma)**.** *Let* $\mathcal{M}^\Gamma = (At(\Gamma), \{S_\alpha\}, \mathbf{V})$ *be the canonical model over* Γ*. For all atoms* $\mathcal{A} \in At(\Gamma)$ *and all formulas* $\varphi \in C(\Gamma)$*,* $\mathcal{M}^\Gamma, \mathcal{A} \Vdash \varphi$ *iff* $\varphi \in \mathcal{A}$*.*

Proof. The proof is by induction on the structure of the formula φ.

- ϕ is a proposition symbol: The proof follows directly from the definition of $\mathbf{V}$.
- $\phi = \neg\psi$ or $\phi = \psi_1 \wedge \psi_2$: The proof follows directly from lemma A.4.
- $\phi = \langle P \rangle \psi$:

  ($\Rightarrow$) Suppose that $\mathcal{M}^\Gamma, \mathcal{A} \Vdash \langle P \rangle \psi$. Then, there exists $\mathcal{A}' \in \mathcal{M}^\Gamma$ such that $\mathcal{A} R_P \mathcal{A}'$ and $\mathcal{M}^\Gamma, \mathcal{A}' \Vdash \psi$. By the induction hypothesis, we know that $\psi \in \mathcal{A}'$ and, by the Existence Lemma, we have that $\langle P \rangle \psi \in \mathcal{A}$.

  ($\Leftarrow$) Suppose that $\langle P \rangle \psi \in \mathcal{A}$. Then, by the Existence Lemma, there is $\mathcal{A}' \in \mathcal{M}^\Gamma$ such that $\mathcal{A} R_P \mathcal{A}'$ and $\psi \in \mathcal{A}'$. By the induction hypothesis, $\mathcal{M}^\Gamma, \mathcal{A}' \Vdash \psi$, which implies $\mathcal{M}^\Gamma, \mathcal{A} \Vdash \langle P \rangle \psi$.

□

Theorem A.16 (Completeness)**.** *Every consistent formula is satisfiable in a finite PPDL-c model.*

Proof. Let φ be a consistent formula. Let $C(\varphi)$ be its closure under the conditions of definition A.2. As φ is consistent, by corollary A.6, there is an atom $\mathcal{A} \in At(\varphi)$ such that $\varphi \in \mathcal{A}$. Let $\mathcal{M}^\varphi$ be the canonical model over φ. Then, by the Truth Lemma (lemma A.15), as $\varphi \in \mathcal{A}$, we conclude that $\mathcal{M}^\varphi, \mathcal{A} \Vdash \varphi$, which proves the theorem. □