Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 262 (2010) 49–64
www.elsevier.com/locate/entcs

A Propositional Dynamic Logic for Concurrent Programs Based on the π-Calculus

Mario R. F. Benevides[1,2]

Computer Science Department and Systems and Computer Engineering Program, Federal University of Rio de Janeiro, Brazil

L. Menasché Schechter[1,3]

Systems and Computer Engineering Program, Federal University of Rio de Janeiro, Brazil

Abstract

This work presents a Propositional Dynamic Logic (πDL) in which the programs are described in a language based on the π-Calculus without replication. Our goal is to build a dynamic logic that is suitable for the description and verification of properties of communicating concurrent systems, in a similar way as PDL is used for the sequential case. We build a simple Kripke semantics for this logic, provide a complete axiomatization for it and show that it has the finite model property.

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 $\langle P \rangle$ for each program P. The logic has a set of basic programs and a set of operators (sequential composition, iteration and nondeterministic choice) that can be used to build more complex programs from simpler ones. A Kripke semantics can be provided, with a frame $\mathcal{F} = (W, R_P)$, where W is a non-empty set of possible program states and, for each program P, R_P is a

[1] The authors were supported by grants from the Brazilian research agency CNPq.
[2] Email: mario@cos.ufrj.br
[3] Email: luis@cos.ufrj.br

doi:10.1016/j.entcs.2010.04.005

binary relation on W such that $(s,t) \in R_P$ if and only if there is a computation of P starting in s and terminating in t.

The π-Calculus is a well known process algebra, proposed by Milner, Parrow and Walker [11], for the specification of communicating concurrent systems. It is an extension of Milner's CCS [10] that is able to describe not only non-determinism and concurrency, but also *mobility* of processes. 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 of a message (which, in the π-Calculus, is also a channel name) being sent at one end of the channel and immediately being received at the other. A π-Calculus specification is a description of the behaviour expected from a system, based on the communication events that may occur. As in PDL, the π-Calculus 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.

This work presents a Propositional Dynamic Logic (πDL [4]) in which the programs are described in a language based on the π-Calculus without replication. There are, in the literature, some other logics that make use of CCS or the π-Calculus. 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, [12] and [6]). The logic that we develop in the present work uses the π-Calculus in a distinct way. Its operators and constructions are *added* to a basic modal logic in order to create a dynamic logic where it is simple to describe and verify properties of communicating, concurrent, non-deterministic and mobile programs and systems, in a similar way as PDL is used for the sequential case. As such, this logic is an extension of the logics from our previous works [3] and [2], which develop propositional dynamic logics based on CCS.

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. 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 logic.

Our logic is related to Concurrent PDL (CPDL) [15] and Channel-CPDL [14], but has advantages over both. The former can only describe properties of concurrent systems with no communication between the components and while the latter 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" [14]) and its satisfiability problem can be proved undecidable (Π^1_1-hard), which also implies that it does not have a complete axiomatization. On the other hand, due to the use of the π-Calculus mechanisms of communication and concurrency, our logic has a simple Kripke semantics, the finite

[4] The pun here comes from the fact that the name of the letter π in Greek and the name of the letter P in English are pronounced exactly the same way.

model property, a straightforward axiomatization and can also deal with *mobility*. Our logic can also be seen as an extension of PDL with Interleaving (iPDL) [9]. In iPDL, the parallel operator that is present in the logic is similar to the parallel operator of the π-Calculus, but it only allows interleaving of the actions in parallel programs, while the parallel operator of the π-Calculus (in conjunction with the restriction operator) also allows communication, synchronization and mobility.

The rest of this paper is organized as follows. In Section 2, we introduce the necessary background concepts: Propositional Dynamic Logic and the π-Calculus. Our logic (πDL), together with a couple of simple examples of its application and a complete axiomatic system, is presented in Section 3. Finally, in Section 4, we state our final remarks. We omit some of the proofs in the text, when they follow directly from previously stated results.

2 Background

This section presents two important subjects. First, we make a brief review of the syntax and semantics of PDL. Second, we present the π-Calculus together with some useful concepts, properties and results from its theory. We do not assume a familiarity with the π-Calculus, since process algebras are by no means a 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. For a more detailed account, [4] can be consulted.

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

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle P\rangle\varphi, \text{ with } P ::= a \mid P_1; P_2 \mid P_1 \cup P_2 \mid P^*,$$

where $p \in \Phi$ and $a \in \mathbb{P}$.

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

Definition 2.2 A *frame* for PDL is a tuple $\mathcal{F} = (W, \{R_a\}_{a\in\mathbb{P}})$ where W is a non-empty set of states and R_a is a binary relation for each basic program a. Besides that, we inductively build binary relations R_P, for each non-basic program P, using the rules $R_{P_1;P_2} = R_{P_1} \circ R_{P_2}$, $R_{P_1\cup P_2} = R_{P_1} \cup R_{P_2}$ and $R_{P^*} = R_P^*$, where R_P^* denotes the reflexive transitive closure of R_P.

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 $\mathbf{V} : \Phi \mapsto 2^W$.

Definition 2.4 Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The semantical notion of *satisfaction* of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, is defined in PDL in the standard way for modal logics [4] for the atomic formulas and the boolean operators. The following rule takes care of the modalities: $\mathcal{M}, w \Vdash \langle P \rangle \varphi$ iff there is $w' \in W$ such that $wR_P w'$ and $\mathcal{M}, w' \Vdash \varphi$.

2.2 The π-Calculus

The π-Calculus is a well known process algebra, proposed by Milner, Parrow and Walker [11], for the specification of communicating concurrent systems. It is an extension of Milner's CCS [10] that is able to describe not only non-determinism and concurrency, but also *mobility* of processes. A π-Calculus specification is a description of the behaviour expected from a system, based on the communication events that may occur. For a broad introduction to the π-Calculus, [13] can be consulted.

In the π-Calculus, a pair of processes can communicate through a common channel and each act of communication consists of a message (which, in the π-Calculus, is also a channel name) being sent at one end of the channel and immediately being received at the other.

Let $\mathcal{N} = \{a, b, c, \ldots\}$ be a set of names. Each channel in a π-Calculus specification is labelled by a name. The labels of the channels are also used to describe the communication actions (sending and receiving messages) performed by the processes, as is shown below. Besides these communication actions, the π-Calculus 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).

Definition 2.5 In our presentation of the π-Calculus, process specifications can be built using the following operations:

$$P ::= \mathbf{0} \mid END \mid \alpha.P \mid P_1; P_2 \mid P_1 + P_2 \mid P_1 | P_2 \mid P^* \mid (\nu a)P,$$

with

$$\alpha ::= a(x) \mid \overline{a}\langle x \rangle \mid \overline{a}\langle \nu x \rangle \mid \tau,$$

where $a, x \in \mathcal{N}$.

Usually, the π-Calculus is presented with a replication operator (!), that denotes the ability of a process to generate multiple copies of itself, or with constants, that may be used to describe recursion. In [2], in the context of CCS, we present a dynamic logic that uses processes with constants. However, in order to keep the finite model property and a complete axiomatization, we had to restrict the interaction between constants and the | operator in order to prevent potentially self-replicating processes. Besides that, with constants, the axiomatization and the theory behind its completeness proof became considerably more complex. On the other hand, the issue of whether it is possible to keep the finite model property and

a complete axiomatization in the presence of replication remains open and we defer it to a future work, as explained in Section 4.

Thus, at the present time, we restrict ourselves to the language without constants and the replication operator. However, as is also shown in [2], it is much simpler to deal with iteration (which is a restricted form of recursion) in the logic than with recursion in its more general form and the resulting axiomatization is more elegant. So, in order to express iterative behaviours, we add to our presentation of the π-Calculus the PDL-inspired operators * and ;.

As it is explained in details in [2], the somewhat loose definition of the null process $\mathbf{0}$ in the π-Calculus, 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), can get in the way of a fully compositional semantics for a dynamic logic based on the π-Calculus. To solve this, we introduce an extra action, with a special meaning: the *ending action*, denoted by *END*. All other actions are called *running actions*. A process 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.

$\mathbf{0}$ 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. The *prefix* operator (.) denotes that the process will first perform the running action α and then behave as P. The *sequential composition* operator (;) denotes that the process will first behave as P_1 and if and when P_1 successfully terminates, it will proceed behaving as P_2. The *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. The *iteration* operator (*) denotes that the process P is capable of being iterated zero or more times. Finally, the *restriction* operator (νa) denotes that the channel a is only accessible inside P (the scope of a is P).

The action $a(x)$, called *input action*, denotes that the process receives a name through the channel labelled by a and the name x marks, in P, the places where the received name should be put. The actions $\overline{a}\langle x\rangle$, called *free output action*, and $\overline{a}\langle \nu x\rangle$, called *bound output action*, both denote that the process sends the name x through the channel labelled by a. The difference between the two is that, in $\overline{a}\langle \nu x\rangle$, x is a restricted name, with this action working as an abbreviation for $(\nu x)\overline{a}\langle x\rangle$. Finally, τ denotes the silent action. We define the bound output action as a primitive action because, as is shown below, under certain circumstances, the only form of restriction that is needed is the one provided by bounded outputs.

We say that the actions $a(x)$ and $\overline{a}\langle \nu x\rangle$ and the restriction (νx) *bind* the name x, calling them *binders*. We say that a name is *bound* in P if it occurs inside the scope of an action or a restriction that binds it. Otherwise, we say that a name is *free* in P. We denote by $f(P)$ the set of free names in P and by $b(P)$ the set

of bound names in P. Similarly, we denote by $f(\alpha)$ and $b(\alpha)$ the free and bound names in an action α. In the actions $a(x)$, $\overline{a}\langle x\rangle$ and $\overline{a}\langle\nu x\rangle$, a is called the *subject*, denoted by $s(\alpha)$, where α is the action, and x is called the *object*, denoted by $o(\alpha)$. The τ action has neither subject nor object.

Definition 2.6 We say that a relation $\cong$ between processes is a *congruence* if it is an equivalence relation and it is preserved by all π-Calculus operators, that is, if $P \cong Q$, then $\alpha.P \cong \alpha.Q$, $P + R \cong Q + R$ and so on.

Definition 2.7 A syntactic substitution of a *bound name* by a *fresh name* (a name that does not occur in the process specification) in its binder and in every occurrence of the name in the scope of this binder is called an *alpha-conversion*.

Definition 2.8 *Structural congruence*, denoted by $\equiv$, is a relation between processes defined by the following set of axioms and rules:

(i) It is a congruence;
(ii) It is closed under alpha-conversion;
(iii) Commutativity of $+$ and $|$;
(iv) If $a \neq x$, $(\nu x)\overline{a}\langle x\rangle.P \equiv \overline{a}\langle\nu x\rangle.P$;
(v) $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$;
(vi) If $x \notin f(P)$, $(\nu x)P \equiv P$.

Definition 2.9 We say that a process P is *clean* if no name appears both free and bound in P and no name is bound by more than one binder. We say that a process is *unrestricted* if it has no occurrences of the ν operator. We say that a clean process is in *ν-prefix form* if it has the form $(\nu x_1)\ldots(\nu x_n)P$, $n \geq 0$, where P is unrestricted. Finally, we say that a clean process is in *ν-standard* form if the only occurrences of the ν operator are inside bound output prefixes.

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 $P \xrightarrow{END} \surd$ to express that the process P can perform the action *END* and successfully finish. In Table 1, we present the semantics for the operators of the π-Calculus based on this notation. This semantics is called *late semantics*. For more details on this and other semantics, [13] can be consulted.

$\frac{P' \equiv P, P \xrightarrow{\alpha} Q, Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$	$\alpha.P \xrightarrow{\alpha} P$	$END \xrightarrow{END} \surd$	$P^* \xrightarrow{END} \surd$
$\frac{P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\alpha} P';Q}$	$\frac{P \xrightarrow{END} \surd, Q \xrightarrow{\alpha} Q'}{P;Q \xrightarrow{\alpha} Q'}$	$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	$\frac{P \xrightarrow{\alpha} P', b(\alpha) \cap f(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$
$\frac{P \xrightarrow{a(x)} P', Q \xrightarrow{\overline{a}\langle u\rangle} Q'}{P \mid Q \xrightarrow{\tau} P'\{u/x\} \mid Q'}$	$\frac{P \xrightarrow{a(x)} P', Q \xrightarrow{\overline{a}\langle \nu u\rangle} Q'}{P \mid Q \xrightarrow{\tau} (\nu u)(P'\{u/x\} \mid Q')}$	$\frac{P \xrightarrow{\alpha} P'}{P^* \xrightarrow{\alpha} P';P^*}$	$\frac{P \xrightarrow{\alpha} P', x \notin \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$
$\frac{P \xrightarrow{END} \surd, Q \xrightarrow{END} \surd}{P;Q \xrightarrow{END} \surd}$	$\frac{P \xrightarrow{END} \surd}{P+Q \xrightarrow{END} \surd}$	$\frac{Q \xrightarrow{END} \surd}{P+Q \xrightarrow{END} \surd}$	$\frac{P \xrightarrow{END} \surd, Q \xrightarrow{END} \surd}{P \mid Q \xrightarrow{END} \surd}$
$\frac{P \xrightarrow{END} \surd}{(\nu x)P \xrightarrow{END} \surd}$			

Table 1
Transition Relations of the π-Calculus

From Table 1, we can see that we have a clear distinction between deadlock and termination. 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 $\alpha.END$ denotes that a process performs the action α and then successfully terminates.

Definition 2.10 Let $\mathcal{P}$ be the set of all possible process specifications. A *late bisimulation* is a *symmetric* binary relation $Z \subseteq \mathcal{P} \times \mathcal{P}$ such that

(i) If $(P, Q) \in Z$ and $P \xrightarrow{\alpha} P'$, where $b(\alpha)$ is fresh in P and Q, then
  (a) If $\alpha = a(x)$, then there is $Q' \in \mathcal{P}$ such that $Q \xrightarrow{\alpha} Q'$ and for all $u \in \mathcal{N}$, $(P'\{u/x\}, Q'\{u/x\}) \in Z$;
  (b) If α is not an input action, then there is $Q' \in \mathcal{P}$ such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in Z$;

(ii) If $(P, Q) \in Z$ and $P \xrightarrow{END} \surd$, then $Q \xrightarrow{END} \surd$.

The reason why the only running actions that need to be considered are the ones that satisfy the freshness condition above is explained in details in [13].

Definition 2.11 Two process specifications P and Q are *late bisimilar* (or simply *bisimilar*), denoted by $P \dot{\sim} Q$, if there is a late bisimulation Z such that $(P, Q) \in Z$. In the π-Calculus, bisimilarity is an equivalence relation but is not a congruence.

Theorem 2.12 *If $P \equiv Q$, then $P \dot{\sim} Q$.*

It should be noticed that while $\mathbf{0}$ is the neutral element for the $+$ operator, that is, $P + \mathbf{0} \dot{\sim} P$, END is the neutral element for the $|$ operator, as $P|END \dot{\sim} P$ [5].

We now present a few particular bisimilarities that are going to be useful in the axiomatization of our logic. We start with the Expansion Law.

Theorem 2.13 (Expansion Law [13]) *Let $P = P_1 \mid P_2$, where P is clean and unrestricted and $|$ does not occur in P_1 and P_2. Then*

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

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

Definition 2.14 *ν-bisimilarity*, denoted by $P \overset{\nu}{\sim} Q$, is a relation between processes defined by the following set of axioms and rules, where $x \notin \alpha$ denotes that x is neither the subject nor the object of the action α:

(i) If $P \equiv Q$, then $P \overset{\nu}{\sim} Q$;
(ii) $(\nu x)\mathbf{0} \overset{\nu}{\sim} \mathbf{0}$;
(iii) $(\nu x)END \overset{\nu}{\sim} END$;
(iv) If $x \notin \alpha$, $(\nu x)\alpha.P \overset{\nu}{\sim} \alpha.(\nu x)P$;
(v) If $x = s(\alpha)$, $(\nu x)\alpha.P \overset{\nu}{\sim} \mathbf{0}$;
(vi) $(\nu x)(P; Q) \overset{\nu}{\sim} (\nu x)P; (\nu x)Q$;

[5] Notice that, according to table 1, $P|\mathbf{0}$ can never successfully finish, so $P|\mathbf{0} \not\dot{\sim} P$ in general.

(vii) $(\nu x)(P+Q) \overset{\nu}{\sim} (\nu x)P + (\nu x)Q$; $(P|Q)$;

(viii) If $x \not\in f(P)$, $P|(\nu x)Q \overset{\nu}{\sim} (\nu x)$ (ix) $(\nu x)P^* \overset{\nu}{\sim} ((\nu x)P)^*$.

It follows from Table 1 and Definition 2.11 that items (ii)-(ix) are indeed bisimilarities. Hence, the relation of ν-bisimilarity is a subset of the relation of bisimilarity. ν-bisimilarity is a convenient relation because it has a simple axiomatization and it is sufficient for all our needs in this work.

Theorem 2.15 *Every process is structurally congruent to a clean process. Every clean process is ν-bisimilar to a process in ν-prefix form. Finally, every clean process with no occurrences of the $|$ operator is ν-bisimilar to a process in ν-standard form.*

3 πDL

In this section, we define a Propositional Dynamic Logic (πDL) in which the programs are built in a language based on the π-Calculus without replication (Definition 2.5). First, we introduce the key concept of *finite possible runs* of a process. We then proceed to describe, using this concept, the syntax and semantics of πDL and provide a couple of simple examples of its application. Finally, we present an axiomatization for πDL and prove its soundness and completeness.

3.1 Action Sequences and Possible Runs

Here, we introduce the concept of *finite possible runs* of a process.

Definition 3.1 We use the notation $\overrightarrow{\alpha}$ to denote a potentially infinite sequence of actions $\alpha_1.\alpha_2.\ldots.\alpha_n(.\cdots)$ (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 3.2 For a finite sequence of actions $\overrightarrow{\alpha}$, we write $P \overset{\overrightarrow{\alpha}}{\Rightarrow} P'$ to express that the process P may perform the sequence $\overrightarrow{\alpha}$ and after that behave as P'. Besides that, we write $P \overset{\overrightarrow{\alpha}}{\Rightarrow} \surd$ to express that the process P may successfully finish after performing the sequence $\overrightarrow{\alpha}$.

We can define notions of alpha-conversion of bound names in a sequence of actions and of a *clean* sequence of actions, in analogy with Definitions 2.7 and 2.9. We can also extend our notation and write $b(\overrightarrow{\gamma})$ and $f(\overrightarrow{\gamma})$ for the sets of bound and free names in the sequence $\overrightarrow{\gamma}$. If a sequence of actions $\overrightarrow{\gamma}$ can be alpha-converted to a sequence $\overrightarrow{\sigma}$, we write $\overrightarrow{\gamma} \equiv_\alpha \overrightarrow{\sigma}$. It is not difficult to see that, if $P \overset{\overrightarrow{\gamma}}{\Rightarrow} \surd$, then $P \overset{\overrightarrow{\sigma}}{\Rightarrow} \surd$, where $\overrightarrow{\gamma} \equiv_\alpha \overrightarrow{\sigma}$ and $\overrightarrow{\sigma}$ is clean. Let $S(P)$ be the set of all such $\overrightarrow{\sigma}$. Then, it is also not difficult to see that, if we establish the convention that $\overrightarrow{\sigma} \equiv_\alpha \overrightarrow{\sigma}$, $\equiv_\alpha$ is an equivalence relation for the elements of the set $S(P)$.

Definition 3.3 We define the set of finite possible runs of a process P, denoted by $\overrightarrow{\mathcal{R}_f}(P)$, as the quotient set $\overrightarrow{\mathcal{R}_f}(P) = S(P)/\equiv_\alpha$. If $\overrightarrow{\gamma} \in S(P)$, then $[\overrightarrow{\gamma}] \in \overrightarrow{\mathcal{R}_f}(P)$ denotes its equivalence class.

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

Definition 3.4 Let $\overrightarrow{\alpha}$ and $\overrightarrow{\sigma}$ be two sequences of actions and let P and Q be two process specifications. If $b(\overrightarrow{\alpha})$ is fresh in Q and $b(\overrightarrow{\sigma})$ is fresh in P, we write $(\overrightarrow{\alpha}, \overrightarrow{\sigma}) \bowtie (P, Q)$.

Definition 3.5 Let $T = \overrightarrow{\mathcal{R}_f}(P)$ and $U = \overrightarrow{\mathcal{R}_f}(Q)$ and let $\natural(\overrightarrow{\alpha}) = \overrightarrow{\lambda}$, where $\overrightarrow{\alpha} = \overrightarrow{\lambda}.END$. We can define the following operations on the sets T and U:

- $T \circ U = \{[\natural(\overrightarrow{\alpha}).\overrightarrow{\beta}] : [\overrightarrow{\alpha}] \in T, [\overrightarrow{\beta}] \in U \text{ and } (\overrightarrow{\alpha}, \overrightarrow{\beta}) \bowtie (P, Q)\}$;
- $T \cup U = \{[\overrightarrow{\alpha}] : [\overrightarrow{\alpha}] \in T \text{ or } [\overrightarrow{\alpha}] \in U\}$;
- $R^0 = \{[\overrightarrow{\varepsilon}]\}, R^n = R \circ R^{n-1} (n \geq 1)$ and $R^* = \bigcup_{n \in \mathbb{N}} R^n$.

Lemma 3.6 *If $P \dot{\sim} Q$ and $b(\overrightarrow{\alpha})$ is fresh in P and Q, then $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$ iff $Q \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$.*

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 $\overrightarrow{\alpha} = END$. Then, $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd \Leftrightarrow P \stackrel{END}{\rightarrow} \surd$. By the hypothesis that $P \dot{\sim} Q$, $P \stackrel{END}{\rightarrow} \surd \Leftrightarrow Q \stackrel{END}{\rightarrow} \surd$. Finally, $Q \stackrel{END}{\rightarrow} \surd \Leftrightarrow Q \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$.

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, $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$ if and only if there is a process P' such that $P \stackrel{\alpha}{\rightarrow} P'$ and $P' \stackrel{\overrightarrow{\beta}}{\Rightarrow} \surd$. But if $P \stackrel{\alpha}{\rightarrow} P'$ and $P \dot{\sim} Q$, then there is a process Q' such that $Q \stackrel{\alpha}{\rightarrow} Q'$ and $P' \dot{\sim} Q'$. Now, $\overrightarrow{\beta}$ is a sequence of length shorter than k, so by the induction hypothesis, as $P' \dot{\sim} Q'$ and $P' \stackrel{\overrightarrow{\beta}}{\Rightarrow} \surd$, then $Q' \stackrel{\overrightarrow{\beta}}{\Rightarrow} \surd$. This means that $Q \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$, proving the theorem. □

Theorem 3.7 *If $P \dot{\sim} Q$, then $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$.*

Proof. Suppose that $[\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P)$. Then, there is a clean sequence $\overrightarrow{\sigma}$ such that $[\overrightarrow{\sigma}] = [\overrightarrow{\alpha}]$ (*) and $b(\overrightarrow{\sigma})$ is fresh in P and Q (**). By (*), $P \stackrel{\overrightarrow{\sigma}}{\Rightarrow} \surd$. As $P \dot{\sim} Q$, this, together with (**) and Lemma 3.6, implies that $Q \stackrel{\overrightarrow{\sigma}}{\Rightarrow} \surd$, which means that $[\overrightarrow{\alpha}] = [\overrightarrow{\sigma}] \in \overrightarrow{\mathcal{R}_f}(Q)$. 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. □

We present some equalities between sets of finite possible runs that are useful to the development of our axiomatization.

Theorem 3.8 *The following set equalities are true:*

(i) $\overrightarrow{\mathcal{R}_f}(\mathbf{0}) = \emptyset$;

(ii) $\overrightarrow{\mathcal{R}_f}(END) = \{[END]\}$;

(iii) $\overrightarrow{\mathcal{R}_f}(\alpha.P) = \{[\alpha.END]\} \circ \overrightarrow{\mathcal{R}_f}(P)$;

(iv) $\overrightarrow{\mathcal{R}_f}(P_1; P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \circ \overrightarrow{\mathcal{R}_f}(P_2)$;

(v) $\overrightarrow{\mathcal{R}_f}(P_1 + P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \cup \overrightarrow{\mathcal{R}_f}(P_2)$; (vi) $\overrightarrow{\mathcal{R}_f}(P^*) = (\overrightarrow{\mathcal{R}_f}(P))^*$;

(vii) $\overrightarrow{\mathcal{R}_f}(P_1|P_2) = \bigcup\{\overrightarrow{\mathcal{R}_f}(\overrightarrow{\alpha} \mid \overrightarrow{\beta}) : [\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P_1)$ *and* $[\overrightarrow{\beta}] \in \overrightarrow{\mathcal{R}_f}(P_2)\}$;

(viii) *If* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$, *then* $\overrightarrow{\mathcal{R}_f}((\nu x)P) = \overrightarrow{\mathcal{R}_f}((\nu x)Q)$;

(ix) *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 1 and Theorem 3.7. The ninth item is simply Arden's Rule [1] applied in our context. This application is sound, since the elements of $\overrightarrow{\mathcal{R}_f}(P)$, for any P, are classes of finite strings. □

3.2 Language and Semantics

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

Definition 3.9 The πDL 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 π-Calculus operators ., ;, +, |, * and ν, a pair of modalities $\langle a\rangle$ and $\langle\overline{a}\rangle$, for each $a \in \mathcal{N}$, and a modality $\langle P\rangle$ for every process P, including the atomic processes $\mathbf{0}$ and *END*. The formulas are defined as follows:

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle a\rangle\varphi \mid \langle\overline{a}\rangle\varphi \mid \langle P\rangle\varphi,$$

where $p \in \Phi$ and P is built as in Definition 2.5.

Definition 3.10 A *frame* for πDL is a tuple $\mathcal{F} = (W, \{R_a, R_{\overline{a}}\}_{a\in\mathcal{N}}, R_{END}, R_\tau)$ where

- W is a non-empty set of states;
- R_a, $R_{\overline{a}}$, for each $a \in \mathcal{N}$, R_{END} and R_τ are the basic binary relations, where $R_{END} = \{(w, w) : w \in W\}$.

Definition 3.11 A *model* for πDL is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is a πDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V} : \Phi \mapsto 2^W$.

Definition 3.12 We define the *core* of an action α, denoted by $c(\alpha)$, in the following way: $c(a(x)) = a$, $c(\overline{a}\langle x\rangle) = c(\overline{a}\langle\nu x\rangle) = \overline{a}$ and $c(\alpha) = \alpha$, if $\alpha = \tau$ or $\alpha = END$.

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

Definition 3.13 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 $\mathcal{M}, w \Vdash \varphi_2$;
- $\mathcal{M}, w \Vdash \langle\kappa\rangle\varphi$ iff there is $w' \in W$ such that $wR_\kappa w'$ and $\mathcal{M}, w' \Vdash \varphi$, where $\kappa = a$ or $\kappa = \overline{a}$, for some $a \in \mathcal{N}$, or $\kappa = \tau$;
- $\mathcal{M}, w \Vdash \langle P\rangle\varphi$ 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 $\overrightarrow{\alpha}$ such that $[\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P)$, the length of $\overrightarrow{\alpha}$ is n and $(v_{i-1}, v_i) \in R_\kappa$ if and only if $c((\overrightarrow{\alpha})_i) = \kappa$, for $1 \leq i \leq n$. We say that such $\overrightarrow{\alpha}$ *matches* the path $(v_0, \ldots, v_n)$.

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 $\Vdash \varphi \leftrightarrow \psi$.

Theorem 3.14 *$\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$ if and only if $\Vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$.*

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 3.13, (*) 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 a sequence $\overrightarrow{\alpha}$, such that $[\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P)$, 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 3.13, 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 a clean sequence $\overrightarrow{\alpha}$ 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 $R_a = \{(v_{i-1}, v_i) : 1 \leq i \leq n \text{ and } c((\overrightarrow{\alpha})_i) = a\}$. Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, such that $v_n \in \mathbf{V}(p)$ and $v_i \notin \mathbf{V}(p)$, $0 \leq i < n$. Then, we have a path $(v_0, \ldots, v_n)$ such that $\mathcal{M}, v_n \Vdash p$ and $\overrightarrow{\alpha}$ matches this path. By Definition 3.13, $\mathcal{M}, v_0 \Vdash \langle P\rangle p$. 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 3.13, $\mathcal{M}, v_0 \nVdash \langle Q\rangle p$, which contradicts (*). □

Corollary 3.15 *If $P \dot{\sim} Q$, then $\Vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$.*

3.3 Examples

In this section, we present two simple examples of applications of πDL.

Example 3.16 Let $\mathcal{M}$ be a Kripke model representing the behaviour of a local network in a business office. Suppose that, in this network, there are three computers and two printers, managed by a common server. For a restricted analysis of $\mathcal{M}$ with respect only to the printing protocols, we may assume that all that the employees of the office do at the computers is to print documents.

Let s_1 be the communication channel between the computers and the server and s_2 be the communication channel between the server and the printers. Then, $C_i = (\nu c_i)\overline{s_1}\langle c_i\rangle.c_i(p).\overline{p}\langle d\rangle.END$ describes a program that sends a document that can be retrieved through channel d to be printed, $S = (s_1(n).s_2(p).\overline{n}\langle p\rangle\ .END)^*$ describes a program that controls the server and distributes the printers following requests from the computers and $P_j = (\nu p_j)\overline{s_2}\langle p_j\rangle.p_j(d).\tau.END$ describes a program that controls the printer and prints the document (the act of printing is represented by τ) that can be retrieved through channel d.

We may want to verify if $\mathcal{M}$ allows for any computer to print a sequence of any number of documents at any time, no matter what state $\mathcal{M}$ is presently in. If we assume that each computer cannot simultaneously request the printing of more than one document, this is equivalent to checking whether the formula $\langle C_1^* \mid C_2^* \mid C_3^* \mid S \mid P_1^* \mid P_2^*\rangle\top$ is globally satisfied in $\mathcal{M}$. We may also want to verify that the actions of one computer do not affect the actions of the others. For this, we could check, for instance, if $\langle C_i\rangle\top \leftrightarrow [(C_j \mid C_k)^*]\langle C_i\rangle\top$, $i \neq j \neq k$, is globally satisfied in $\mathcal{M}$.

Example 3.17 The parallel composition operator ($|$) has a dual role in the π-Calculus. It represents both interleaving and synchronization of processes. In van Benthem's paradigm of *games-as-processes* [16], 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, $\mathcal{M}, w \Vdash \langle a.b.b.END + a.b.c.END \mid d.d.d.END\rangle w_1$ 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, $\mathcal{M}, w \Vdash [a.b.b.END \mid d.d.d.END]w_1$ 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 separated by some exchange of information between the players. For instance, in the game described above, suppose now that the two players may select their actions from the same set $\{a, b, c\}$. 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: a player performs two actions, then informs the other player of one of them and performs the final action. Then we can express properties of this game using formulas of our logic, in an analogous way to the paragraph above. For instance, $\mathcal{M}, w \Vdash [(\nu x)(\nu y)(p_1.b.b.\overline{x}\langle b\rangle.y(w).w.END \mid p_2.a.b.x(z).\overline{y}\langle a\rangle.b.END + p_2.a.b.x(z).\overline{y}\langle b\rangle.b.END)]w_1$ 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 and finishes with b, then player 1 can always win by playing b, b and then finishing with the action informed by player 2.

This interplay between interleaving and synchronization can then be used to describe a fairly large group of games. A recent paper [17] also works with this idea that concurrency operators can be used to model simultaneous games. The authors use CPDL [15] 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 [17] to deal with simultaneous games with communication, as we briefly illustrated.

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

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

(END) $\vdash \langle END\rangle p \leftrightarrow p$

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

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

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

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

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

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

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

(νBi) If $P \overset{\nu}{\sim} Q$, then $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$

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

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

(RSub) If $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$, then $\vdash \langle(\nu x)P\rangle p \leftrightarrow \langle(\nu x)Q\rangle p$

(Exp) If the Expansion Law can be applied to P, then $\vdash \langle P\rangle p \leftrightarrow \langle Exp(P)\rangle p$

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

The axioms **(PL)**, **(K)** and **(Du)** and the rules **(MP)**, **(Gen)** and **(Sub)** are standard in the modal logic literature. The soundness of the other axioms and rules follows directly from the set equalities in Theorem 3.8 (with the help of Theorem 3.14), Theorem 2.13, Corollary 3.15 and Definition 3.10.

Definition 3.18 We define the following relation between processes: $P \leftrightarrow Q$ iff $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$.

Theorem 3.19 $\leftrightarrow$ *is a* congruence.

Definition 3.20 Let $\Omega_k = \{P_1, \ldots, P_k\}$ be a set of processes such that $P_i \not\equiv P_j$, if $i \neq j$. Let $E(\Omega_k) = \{E_1, \ldots, E_k\}$ such that $E_i = (P_i, T_i)$, $P_i \leftrightarrow T_i$, $T_i = \sum_j A_j^i; Q_j^i$

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), $Q^i_j \in \Omega_k$.

Theorem 3.21 *Let $P = P_1 \mid P_2$, where P is clean and unrestricted. Then there is $\overline{P}$ such that $P \leftrightarrow \overline{P}$ and $\overline{P}$ has no occurrence 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 the Expansion Law can be applied to P, so we can use **(Exp)** to build pairs (P_i, T_i) that satisfy Definition 3.20. Let $P_1 = P$ and Ω_k be the smallest set such that $P_1 \in \Omega_k$ and $E(\Omega_k)$ is closed. Such a set always exists, as otherwise there would be an infinite set of processes $\{A_i : i \in \mathbb{N}\}$ such that $A_i \not\equiv A_j$, if $i \neq j$, and $P \xrightarrow{\alpha_0} A_0 \xrightarrow{\alpha_1} A_1 \ldots$, which, by a careful inspection of Table 1, cannot happen.

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 **(Exp)** 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 [5].

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

Two formulas ϕ and ψ are equi-consistent if $\vdash \phi \leftrightarrow \psi$. By soundness, if ϕ and ψ are equi-consistent, then they are also semantically equivalent.

Theorem 3.22 (Completeness) *Every consistent formula is satisfiable in a finite πDL 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 **($\nu$Bi)**, **(RSub)**, **(MP)** and Theorems 2.15, 3.19 and 3.21 to get a sequence $P \leftrightarrow P' \leftrightarrow P'' \leftrightarrow P'''$, where P' is clean and in ν-prefix form, P'' is also without any occurrence of the $\mid$ operator and P''' is like P'' but is instead in ν-standard form. We can then obtain an equi-consistent formula $\varphi' = \varphi[P'''/P, P \in \mathbf{P}(\varphi)]$ in which the only π-Calculus operators that appear are ., ;, + and *. All of these operators and its correspondent axioms are analogous to the operators and axioms in standard PDL. Thus, we can follow the completeness proof of standard PDL (the PDL axioms and its completeness proof are presented in details in [4]), treating the actions as basic PDL programs, to show that φ' is satisfiable in a finite model. As φ and φ' are equi-consistent, they are also semantically equivalent, which means that φ is also satisfied in that same finite model. □

4 Final Remarks and Future Work

In this work, we present a Propositional Dynamic Logic for communicating concurrent systems (πDL) in which the programs are described in a language based on the π-Calculus without replication. From the point of view of dynamic logics, this logic represents an improvement on the current scenario, as previous dynamic logics could not effectively deal with both concurrency and communication. CPDL [15] dealt with concurrency, but there was no possibility of communication between the components of a concurrent system. Channel-CPDL [14] models concurrency and communication but it has a "rather complicated" [14] semantics, is undecidable and lacks a complete axiomatization. On the other hand, we are able to provide a simple Kripke semantics for our logic, based on the idea of finite possible runs of processes, build a complete axiomatization for it and show that it has the finite model property.

We also provide a method, in a language with 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 [5]. 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 axiomatization.

It should also be noticed that, while the | operator can be written out of the specifications, in practice it can be very hard to describe a complex concurrent behaviour without it from the start. Besides that, even though both specifications, with and without |, may be equivalent, the one with | will be more succinct.

It would be interesting to study the complexities of the satisfiability and model-checking problems for this logic and the ones in [3] and [2]. It would also be interesting to develop an automatic model-checker for these logics, which would involve an efficient algorithmic method to deal with the expansion of parallel processes. We would also like to analyze the issue of self-replicating processes, which was left out of the present work. We would like to study what would change in the logic with the addition of the π-Calculus replication operator (!).

Finally, we would also like to study in more detail the possible connections between our logic and the ideas presented in [17] and analyze how to use our logic as a tool for the description of simultaneous games with communication.

References

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

[2] Benevides, M. R. F. and L. M. Schechter, *CCS-based dynamic logics for communicating concurrent programs*, URL: http://arxiv.org/abs/0904.0034v1.

[3] Benevides, M. R. F. and L. M. Schechter, *A propositional dynamic logic for CCS programs*, in: *Proceedings of the XV Workshop on Logic, Language, Information and Computation (Wollic 2008)*, LNAI **5110** (2008), pp. 83–97.

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

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

[6] Dam, M., *Model checking mobile processes*, Information and Computation **129** (1996), pp. 35–51.

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

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

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

[10] Milner, R., "Communication and Concurrency," Prentice Hall, 1989.

[11] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes - part I and part II*, Information and Computation **100** (1992), pp. 1–77.

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

[13] Parrow, J., *An introduction to the π-calculus*, in: J. A. Bergstra, A. Ponse and S. A. Smolka, editors, *Handbook of Process Algebra*, Elsevier, 2001 pp. 479–543.

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

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

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

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