A Propositional Dynamic Logic for CCS Programs

Mario R. F. Benevides and L. Menasché Schechter

{mario,luis}@cos.ufrj.br

Abstract

This work presents a Propositional Dynamic Logic in which the programs are CCS terms (CCS-PDL). Its goal is to reason about properties of concurrent systems specified in CCS. CCS is a process algebra that models the concurrency and interaction between processes through individual acts of communication. In a first step, we consider only CCS processes without constants and give a complete axiomatization for this logic, which is very similar to $*$-free PDL. Then, we proceed to include CCS processes with constants. In this case, we impose some restrictions on the form of the recursive equations that can be built with those constants. We also give an axiomatization for this second logic and prove its completeness using a Fischer-Ladner construction. Unlike Concurrent PDL (with channels) [4, 5], our logic has a simple Kripke semantics, a complete axiomatization and the finite model property.

1 Introduction

Propositional Dynamic Logic (PDL) plays an important role in formal specification and reasoning about programs and actions. PDL is a multi-modal logic with one modality $\langle\pi\rangle$ for each program π. The logic has a finite 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. PDL has been used in formal specification to reason about properties of programs and their behaviour. Correctness, termination, fairness, liveness and equivalence of programs are among the properties that one usually wants to verify. A Kripke semantics can be provided, with a frame $\mathcal{F} = \langle W, R_\pi \rangle$, where W is a set of possible program states and, for each program π, 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.

The Calculus for Communicating Systems (CCS) is a well known process algebra, proposed by Robin Milner [2], for the specification of 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 behaviour 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) to build more complex specifications from simpler ones. Iteration can also be described through the use of recursive equations.

This work presents a Propositional Dynamic Logic in which the programs are CCS terms (CCS-PDL). Its goal is to reason about properties of concurrent systems specified in CCS. The idea is to bring together the logical and the algebraic formalisms. Quoting from Milner [2]:

> "The calculus of this book is indeed largely algebraic, but I make no claim that everything can be done by algebra. (...) It is perhaps equally true that not everything can be done by logic; thus one of the outstanding challenges in concurrency is to find the right marriage between logical and behavioural approaches."

CCS-PDL is related to Concurrent PDL (with channels) [4, 5] and the logic developed in [1]. Both of these logics are expressive enough to represent interesting properties of concurrent systems. However, neither of them has a simple Kripke semantics. The first has a semantics based on *super-states and super-processes* and its satisfiability problem can be proved undecidable (in fact, it is Π^1_1-hard) [4]. Also, it does not have a complete axiomatization [4]. The second 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 full use of CCS mechanism of communication, CCS-PDL has a simple Kripke semantics and the finite model property.

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. A first version of our logic, together with an axiomatic system, is presented in section 3. In this preliminary version, we do not use constants in the CCS processes. In section 4, we present the full logic, in which we allow the presence of constants in the CCS processes. We also give an axiomatization for this second logic and prove its completeness using a Fischer-Ladner construction. Finally, in section 5, we state our final remarks.

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 process algebra CCS and the Expansion Law, which is closely related to the way we deal with the parallel composition operator.

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* $\Phi = \{p, q, \ldots\}$ *of countably many propositional symbols, a finite set of basic programs* $\Pi = \{a, b, \ldots\}$*, 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:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\pi\rangle\varphi$$

$$\pi ::= a \mid \pi_1;\pi_2 \mid \pi_1 \cup \pi_2 \mid \pi^*$$

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 $[\pi]\varphi \equiv \neg\langle\pi\rangle\neg\varphi$.

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

- *W is a non-empty set of states;*
- *R_a is a binary relation for each basic program a;*
- *R_π is a binary relation for each non-basic program π, inductively built as:*
  - *$R_{\pi_1;\pi_2} = R_{\pi_1} \circ R_{\pi_2}$;*
  - *$R_{\pi_1 \cup \pi_2} = R_{\pi_1} \cup R_{\pi_2}$;*
  - *$R_{\pi^*} = R^*_\pi$, 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 $wR_\pi w'$ and $\mathcal{M}, w' \Vdash \varphi$.*

2.2 Calculus for Communicating Systems

The Calculus for Communicating Systems (CCS) is a well known process algebra, proposed by Robin Milner [2], for the specification of concurrent systems. It models the concurrency and interaction between processes through individual actions of communication. A CCS specification is a description (in the form of algebraic equations) of the behaviour expected from a system, based on the communication events that may occur.

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. Each process connects itself to a channel through a port.

Let $\mathcal{N} = \{a, b, c, \ldots\}$ be a set of names and $\overline{\mathcal{N}} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$ be the correspondent set of co-names. Each port in a CCS specification is labelled by either a name or a co-name. Using the convention that $\overline{\overline{a}} = a$, if a port connected to an end of a channel is labelled with $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}}$, then a port connected to the other end of this same channel must be labelled with $\overline{\alpha}$. Moreover, two ports of a process cannot have the same label. Channels in CCS can only transmit signals

in one direction. By convention, signals are sent through the ports labelled by co-names and received through ports labelled by names.

The labels of the ports are also used to describe the communication actions performed by the processes. For example, if a process is connected to a channel through port $\overline{\alpha}$, the action of sending a signal into this channel is also denoted by $\overline{\alpha}$. So, in terms of actions, α means that the process receives a signal through the port labelled with α and $\overline{\alpha}$ means that the process sends a signal through the port labelled with $\overline{\alpha}$. A process can never perform a communication action α without its complementary action $\overline{\alpha}$ also being performed at the same time by some other process.

Besides the actions denoted by the elements of $\mathcal{N} \cup \overline{\mathcal{N}}$, CCS admits only one other action: the silent action, denoted by τ. The silent action is used to represent any internal action performed by any of the processes that does not involve any act of communication (e.g.: a memory update). Thus, we have that the set of CCS actions is $\mathcal{A} = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$.

There are two different ways in CCS to consider the τ action: it can be regarded as being observable, in the same way as the communication actions or it can be regarded as being invisible. We will adopt the first point of view, since it simplifies the semantics considerably.

In CCS, process specifications can be built using the following operations ($\alpha \in \mathcal{A}$ is an action, P, P_1 and P_2 are process specifications, A is a constant and $L \subseteq \mathcal{N}$):

$$P ::= \alpha \mid \alpha.P \mid \alpha.A \mid P_1 + P_2 \mid P_1|P_2 \mid P\backslash L^1,$$

where every constant A has a (unique) *defining equation* $A \stackrel{def}{=} P_A$, where P_A is a process specification. 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.

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 pair of complementary ports (one performing an action α and the other $\overline{\alpha}$). Finally, the *restriction* operator ($\backslash$) denotes that for all ports α such that $\alpha \in L$ or $\overline{\alpha} \in L$, α is unreachable outside P. Iteration in CCS is modeled through recursive defining equations, i.e., equations $A \stackrel{def}{=} P_A$ where A occurs in P_A.

We write $P \stackrel{\alpha}{\rightarrow} P'$ to express that the process P can perform the action α and after that behave as P'. We write $P \stackrel{\alpha}{\rightarrow} \mathbf{0}$ to express that the process P finishes after performing the action α. A process only finishes when there is not any possible action left for it to perform. For example, $\beta \stackrel{\beta}{\rightarrow} \mathbf{0}$. When a process finishes inside a parallel composition, we write P instead of $P|\mathbf{0}$. In table 1, we present the semantics for the operators based on this notation.

The notion of equality between process specifications is defined through the concept of *strong bisimulation*.

Definition 2.5. *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 $\alpha \in \mathcal{A}$,*

[1] Originally, CCS has also a relabelling operator.

Action	$\frac{}{\alpha \xrightarrow{\alpha} \mathbf{0}}$	Prefix	$\frac{}{\alpha.P \xrightarrow{\alpha} P}$
Constant	$\frac{}{\alpha.A \xrightarrow{\alpha} P_A}(A \stackrel{def}{=} P_A)$	Summation (1)	$\frac{E \xrightarrow{\alpha} E'}{E+F \xrightarrow{\alpha} E'}$
Summation (2)	$\frac{F \xrightarrow{\beta} F'}{E+F \xrightarrow{\beta} F'}$	Par. Comp. (1)	$\frac{E \xrightarrow{\alpha} E'}{E\|F \xrightarrow{\alpha} E'\|F}$
Par. Comp. (2)	$\frac{F \xrightarrow{\beta} F'}{E\|F \xrightarrow{\beta} E\|F'}$	Par. Comp. (3)	$\frac{E \xrightarrow{\lambda} E', F \xrightarrow{\overline{\lambda}} F'}{E\|F \xrightarrow{\tau} E'\|F'}$
Restriction	$\frac{E \xrightarrow{\alpha} E'}{E\backslash L \xrightarrow{\alpha} E'\backslash L}(\alpha, \overline{\alpha} \notin L)$		

Table 1: Transition Relations of CCS

- *Whenever* $P \xrightarrow{\alpha} P'$ *then, for some* Q', $Q \xrightarrow{\alpha} Q'$ *and* $(P', Q') \in Z$
- *Whenever* $Q \xrightarrow{\alpha} Q'$ *then, for some* P', $P \xrightarrow{\alpha} P'$ *and* $(P', Q') \in Z$

Definition 2.6. *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* $(P, Q) \in Z$*. Two process specifications are considered "equal" if they are bisimilar.*

Bisimilarity is preserved by all of CCS operators:

Theorem 2.7 ([2]). *Let* $P_1 \sim P_2$*. Then*

1. $\alpha.P_1 \sim \alpha.P_2$;
2. $P_1 + Q \sim P_2 + Q$;
3. $P_1|Q \sim P_2|Q$;
4. $P_1\backslash L \sim P_2\backslash L$.

A very useful property of CCS processes is that they can be rewritten as a summation of all their possible actions. This is what states the Expansion Law below. This theorem will be very important in the definition of the semantics of our logic in the next section.

Theorem 2.8 (Expansion Law [2]). *Let* $P = (P_1 \mid P_2)$*. Then*

$$P \sim \sum\{\alpha.(P_1' \mid P_2) : P_1 \xrightarrow{\alpha} P_1'\} + \sum\{\alpha.(P_1 \mid P_2') : P_2 \xrightarrow{\alpha} P_2'\} +$$

$$+ \sum\{\tau.(P_1' \mid P_2') : P_1 \xrightarrow{\lambda} P_1', P_2 \xrightarrow{\overline{\lambda}} P_2'\}$$

In the rest of this paper, we consider that the restriction operator does not occur in the processes. In order to motivate the use of CCS, we present a simple example below:

Example 2.9 (Vending Machine [2, 6]). *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 logic (represented by a* τ *action). A CCS term describing the behaviour 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 behaviour. 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 $(V|C)$.

3 PDL for CCS Programs without Constants

This section presents a suitable fragment of CCS-PDL. In this fragment, all CCS processes do not use constants. We call this fragment Small CCS-PDL or SCCS-PDL. Our goal is to introduce a smaller version of our logic and discuss some of the issues concerning the axioms and the relational interpretation of formulas.

3.1 Language and Semantics

Definition 3.1. *The SCCS-PDL language consists of a set* $\Phi = \{p, q, \ldots\}$ *of countably many propositional symbols, a finite set of actions* $\mathcal{A} = \{\alpha, \beta, \ldots\}$ *that includes 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:*

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

$$P ::= \alpha \mid \alpha.P \mid P_1 + P_2 \mid P_1|P_2$$

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 $[P]\varphi \equiv \neg\langle P\rangle\neg\varphi$.

Going back to the example of the vending machine, we could use this language to express that after the insertion of two one euro coins, the machine does not accept coins anymore. This can be expressed with the formula

$$[1e.1e][1e + 2e]\bot.$$

Definition 3.2. *We define the length of a process* P*, denoted by* $||P||$*, as the number of symbols in* P*. We also define the length of a finished process as* 0*.*

Theorem 3.3. *If* P *is a process without any constants and* $P \stackrel{\alpha}{\rightarrow} P'$*, then* $||P'|| < ||P||$*.*

Definition 3.4. *A* frame *for SCCS-PDL is a tuple* $\mathcal{F} = (W, R_\alpha, R_P)$ *where*

- W *is a non-empty set of states;*
- R_α *is a binary relation for each basic action* α*, including* τ*;*
- R_P *is a binary relation for each non-basic process* P*, inductively built as:*
  - $R_{\alpha.P} = R_\alpha \circ R_P$*;*

– $R_{(P_1+P_2)} = R_{P_1} \cup R_{P_2}$;
– $R_{(P_1|P_2)} = \bigcup_{P_1 \xrightarrow{\alpha} P_1'} R_\alpha \circ R_{(P_1'|P_2)} \cup \bigcup_{P_2 \xrightarrow{\alpha} P_2'} R_\alpha \circ R_{(P_1|P_2')} \cup \bigcup_{\substack{P_1 \xrightarrow{\lambda} P_1' \\ P_2 \xrightarrow{\overline{\lambda}} P_2'}} R_\tau \circ R_{(P_1'|P_2')}$

It should be noticed that the Expansion Law stated in theorem 2.8 is what allows us to define a simple relational semantic to the parallel composition operator. Besides that, theorem 3.3 guarantees that we can fully define the relation R_P in terms of the relations R_α, for any process P, applying the above rules a finite number of times.

Definition 3.5. *A* model *for SCCS-PDL is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is a SCCS-PDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V} : \Phi \mapsto 2^W$.*

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

Definition 3.6. *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* $w' \in W$ *such that* wR_Pw' *and* $\mathcal{M}, w' \Vdash \varphi$.

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 $\Vdash \varphi$.

3.2 Proof Theory

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

Axioms

(PL) Enough propositional logic tautologies

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

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

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

(PC) $\langle P_1 \mid P_2\rangle p \leftrightarrow \bigvee_{P_1 \xrightarrow{\alpha} P_1'} \langle\alpha\rangle\langle P_1' \mid P_2\rangle p \vee \bigvee_{P_2 \xrightarrow{\alpha} P_2'} \langle\alpha\rangle\langle P_1 \mid P_2'\rangle p \vee \bigvee_{\substack{P_1 \xrightarrow{\lambda} P_1' \\ P_2 \xrightarrow{\overline{\lambda}} P_2'}} \langle\tau\rangle\langle P_1' \mid P_2'\rangle p$

Inference Rules:

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

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

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

These axioms are closely related to the conditions imposed in Definition 3.4. The second axiom is the K axiom for modal logics. The third and fourth axioms are well-known from PDL literature and define sequential composition and choice operators respectively. The fifth axiom corresponds to the Expansion Law for the parallel composition operator.

The logic presented in this section is sound and complete w.r.t. the class of frames described in Definition 3.4. It also 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 PDL for CCS Programs

The logic presented in this section uses the same CCS operators as in the previous section plus constants. This is the full CCS-PDL logic. Our goal in this section is to build an axiomatic system to CCS-PDL and prove its completeness.

4.1 Language and Semantics

Definition 4.1. *The CCS-PDL language consists of a set $\Phi = \{p, q, ...\}$ of countably many propositional symbols, a finite set of actions $\mathcal{A} = \{a, b, \ldots\}$ that includes the silent action τ, the boolean connectives $\neg$ and $\wedge$, the CCS operators $.$, $+$ and $|$, constants, with its correspondent defining equations, and a modality $\langle P\rangle$ for every process P. The formulas are defined as follows:*

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

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

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 $[P]\varphi \equiv \neg\langle P\rangle\neg\varphi$.

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}, then $A_k \in Cons(P)$. We make the restriction that $Cons(P)$ must be a finite set for every process P.

Another restriction concerns the construction of defining equations. 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*, or $A \stackrel{def}{=} \vec{\alpha}_1.A + \ldots + \vec{\alpha}_n.A + T_A$, called *recursive equations*, where $\vec{\alpha}_i$ denotes a sequence of actions $\alpha^i_1.\alpha^i_2 \ldots \alpha^i_{m_i}$ and $A \notin Cons(T_A)$.

Definition 4.2. *We say that a process P is a* looping 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 looping process. Otherwise, we say that P is a* non-looping process.

Definition 4.3. *We write $P \stackrel{\vec{\alpha}}{\Rightarrow} P'$ to express that the process P can perform the sequence of actions $\vec{\alpha}$ and after that behave as P'. We write $P \stackrel{\vec{\alpha}}{\Rightarrow} \mathbf{0}$ to express that the process P finishes after performing the sequence of actions $\vec{\alpha}$.*

Definition 4.4. *We call a sequence $\vec{\alpha}$ a* breaker *for a looping process P if $P = P_A$, $P \stackrel{\vec{\alpha}}{\Rightarrow} P'$ and $A \notin Cons(P')$ (or $P \stackrel{\vec{\alpha}}{\Rightarrow} \mathbf{0}$) or if $P = P_1 \mid P_2$, P_i ($i \in \{1, 2\}$) is a looping process and $\vec{\alpha}$ is a breaker for P_i. These sequences are called breakers because after performing $\vec{\alpha}$, there is no sequence of actions $\vec{\alpha}'$ such that $P' \stackrel{\vec{\alpha}'}{\Rightarrow} P$, i. e., the looping in P is broken by $\vec{\alpha}$.*

Using the concept of a breaker, we can split a looping process P into two parts: the *looping part*, denoted by L_P, and the *tail part*, denoted by T_P. Informally, L_P describes one loop of P and T_P describes the behaviour of P when it stops looping. Let $Br(P)$ be the set of all breakers for P. Then,

$$T_P = \sum\{\vec{\alpha}.P' : \vec{\alpha} \in Br(P), P \stackrel{\vec{\alpha}}{\Rightarrow} P' \text{ and } \forall \vec{\alpha}' \subsetneq \vec{\alpha}, P \stackrel{\vec{\alpha}'}{\not\Rightarrow} P\} \text{ and}$$

$$L_P = \sum\{\vec{\alpha} : P \stackrel{\vec{\alpha}}{\Rightarrow} P \text{ and } \forall \vec{\alpha}' \subsetneq \vec{\alpha}, P \stackrel{\vec{\alpha}'}{\not\Rightarrow} P\}.$$

If $P = P_A = \vec{\alpha}_1.A + \ldots + \vec{\alpha}_n.A + T_A$, it is not difficult to see that $L_P = \vec{\alpha}_1 + \ldots + \vec{\alpha}_n$ and $T_P = T_A$. We also define the process L'_P, that describes one or more loops of P, as

$$L'_P = \sum\{\vec{\alpha}.Z_P : P \stackrel{\vec{\alpha}}{\Rightarrow} P \text{ and } \forall \vec{\alpha}' \subsetneq \vec{\alpha}, P \stackrel{\vec{\alpha}'}{\not\Rightarrow} P\} + L_P,$$

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

Definition 4.5. *We say that a process* can unfold a constant *if it has the form $\alpha.A$ or if it has the form $P_1 \mid P_2$ where P_1 or P_2 can unfold a constant.*

Theorem 4.6. *If P is a process that cannot unfold a constant and $P \stackrel{\alpha}{\rightarrow} P'$, then $||P'|| < ||P||$.*

Definition 4.7. *A* **frame** *for CCS-PDL is a tuple $\mathcal{F} = (W, Z, R_\alpha, R_P)$ where:*

- *W is a non-empty set of states;*
- *R_α is a binary relation for each basic action α, including τ;*
- *R_P is a binary relation for each non-basic process P, inductively built as:*
  - *For looping processes:*
    - *$R_P = R^*_{L_P} \circ R_{T_P}$*
  - *For non-looping processes:*
    - *$R_{\alpha.P}$, $R_{P_1+P_2}$ and $R_{P_1|P_2}$ as in definition 3.4;*
    - *$R_{\alpha.A} = R_\alpha \circ R_{P_A}$*

It should be noticed that the restriction on the set $Cons(P)$ and the restriction on the formation of the defining equations, both presented in the beginning of the section, together with the definition of R_P for looping processes P based on R_{L_P} and R_{T_P}, which are both non-looping processes, and theorem 4.6 guarantee that we can fully define the relation R_P in terms of the relations R_α, for any process P, applying the above rules a finite number of times and that we can build well-founded proofs by induction on the structure of a process P.

The notions of models and satisfaction are defined analogously to Definitions 3.5 and 3.6.

4.2 Proof Theory

The proof theory is similar to the 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.

Axioms

- Axioms for looping processes:

  (Rec) $\langle P\rangle p \leftrightarrow \langle T_P\rangle p \vee \langle L_P\rangle\langle P\rangle p$
  (FP) $(r \rightarrow ([T_P]\neg p \wedge [L_P]r)) \wedge [L'_P](r \rightarrow ([T_P]\neg p \wedge [L_P]r)) \rightarrow (r \rightarrow [P]\neg p)$

- Axioms for non-looping processes:

  (SCCS) The axioms **(PL)**, **(K)**, **(Pr)**, **(NC)** and **(PC)** from section 3.2
  (Cons) $\langle \alpha.A\rangle p \leftrightarrow \langle \alpha\rangle\langle P_A\rangle p$

The inference rules remain the same as in section 3.2: **(Sub)**, **(MP)** and **(Gen)**.

The proof of soundness is analogous to the proof of soundness for PDL and CTL, as all of our axioms are very closely related to axioms for these logics.

Theorem 4.8 (Completeness). *Every consistent formula is satisfiable in a finite model that respects definition 4.7.*

Proof. See the appendix. □

5 Final Remarks and Future Work

In this work, we present a PDL-like logic in which the programs are CCS terms (CCS-PDL). We provide a simple Kripke semantics for it and also give an axiomatization for this logic. We prove the completeness of the axiomatic system and the finite model property for the logic using a Fischer-Ladner construction.

As a continuation of this work, it would be interesting to study the complexity of the satisfiability problem for this logic, possibly relating it to the satisfiability problem for standard PDL.

We would also like to investigate some extension of CCS-PDL to deal with the restriction operator and a PDL for π-Calculus programs [3]. It would be interesting to develop an automatic theorem prover for CCS-PDL. This would involve, among other things, efficient algorithmic methods to determine the processes L_P and T_P related to a looping process P and to deal with the expansion of parallel processes.

References

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

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

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

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

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

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

A Completeness Proof

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 looping 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-looping 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 $\bigvee_{P_1 \xrightarrow{\alpha} P_1'}\langle\alpha\rangle\langle P_1' \mid P_2\rangle\varphi \vee \bigvee_{P_2 \xrightarrow{\alpha} P_2'}\langle\alpha\rangle\langle P_1 \mid P_2'\rangle\varphi \vee \bigvee_{\substack{P_1 \xrightarrow{\lambda} P_1' \\ P_2 \xrightarrow{\overline{\lambda}} P_2'}}\langle\tau\rangle\langle P_1' \mid P_2'\rangle\varphi$*

It is not difficult to prove that if Γ is finite, then the closure $C(\Gamma)$ is also finite. We assume Γ to be finite from now on.

Definition A.3. *Every formula ϕ that is derivable from the set of axioms and rules in section 4.2 is called a* theorem *and denoted as $\vdash \phi$. We say that a formula ϕ is* consistent *iff $\neg\phi$ is not a theorem, i.e., iff $\nvdash \neg\phi$ and* inconsistent *otherwise. A set of formulas $\Delta = \{\phi_1, \ldots, \phi_n\}$ is said to be consistent iff $\psi = \phi_1 \wedge \ldots \wedge \phi_n$ is consistent.*

Definition A.4. *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.5. *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.6. *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.7. *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.8 (Canonical model over Γ). *Let Γ be a finite set of formulas. The* canonical model *over Γ is the tuple $\mathcal{M}^\Gamma = (At(\Gamma), \{S_P\}, \mathbf{V})$ 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 $\mathcal{A}, \mathcal{B} \in At(\Gamma)$,*

$$\mathcal{A} S_P \mathcal{B} \quad \textit{iff} \quad \bigwedge \mathcal{A} \wedge \langle P \rangle \bigwedge \mathcal{B} \textit{ is consistent.}$$

$\mathbf{V}$ is called the canonical valuation *and S_P the* canonical relations*, where P is a CCS process.*

Definition A.9 (CCS-PDL model over Γ). *The* CCS-PDL model *over Γ is the tuple $\mathcal{N}^\Gamma = (At(\Gamma), \{R_P\}, \mathbf{V})$, where R_P is defined as $R_\alpha = S_\alpha$ for all basic processes α and according to definition 4.7 for all complex processes. $\mathbf{V}$ is the canonical valuation.*

If $\Gamma = \{\varphi\}$, we write $C(\varphi)$, $At(\varphi)$, $\mathcal{M}^\varphi$ and $\mathcal{N}^\varphi$ instead of $C(\{\varphi\})$, $At(\{\varphi\})$, $\mathcal{M}^{\{\varphi\}}$ and $\mathcal{N}^{\{\varphi\}}$.

Lemma A.10 (Existence Lemma for Basic Processes). *Let $\mathcal{A}$ be an atom, and let α be a basic process. 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 $\mathcal{A} R_\alpha \mathcal{B}$ 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.8, which implies that $\mathcal{A} R_\alpha \mathcal{B}$.

($\Leftarrow$): Suppose that there is an atom $\mathcal{B}$ such that $\phi \in \mathcal{B}$ and $\mathcal{A} R_\alpha \mathcal{B}$. Then $\mathcal{A} S_\alpha \mathcal{B}$ and $\bigwedge \mathcal{A} \wedge \langle\alpha\rangle \bigwedge \mathcal{B}$ is consistent by definition A.8. 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.11. *For all looping 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 $r = \bigvee\{\bigwedge \mathcal{C} \mid \mathcal{C} \in V\}$. It is not difficult to see that $\neg r = \bigvee\{\bigwedge \mathcal{C} \mid \mathcal{C} \in \overline{V}\}$.

First, we have that $\vdash r \rightarrow [T_P]\neg\bigwedge \mathcal{B}$. Otherwise, $\neg(r \rightarrow [T_P]\neg\bigwedge \mathcal{B}) \equiv r \wedge \langle T_P\rangle \bigwedge \mathcal{B}$ 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 $\vdash r \rightarrow [L_P]r$. Otherwise, $\neg(r \rightarrow [L_P]r) \equiv r \wedge \langle L_P\rangle\neg r$ 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 $\vdash r \rightarrow ([T_P]\neg\bigwedge \mathcal{B} \wedge [L_P]r)$. By **(Gen)**, **(PL)**, **(FP)** and **(MP)**, $\vdash r \rightarrow [P]\neg\bigwedge \mathcal{B}$. But, as $\mathcal{A} \in V$, $\vdash \bigwedge \mathcal{A} \rightarrow r$, which means that $\vdash \bigwedge \mathcal{A} \rightarrow [P]\neg\bigwedge \mathcal{B}$. 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 $\mathcal{A} \notin \langle S'_P\rangle\mathcal{B}$. □

Lemma A.12. *For all processes P, $S_P \subseteq R_P$.*

Proof. The proof is by induction on the structure of the process P.

- The base case is immediate, for we defined $R_\alpha = S_\alpha$ for all basic processes α.
- P is a non-looping 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.10), 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 looping process. By lemma A.11, $S_P \subseteq S'_P$, where $S'_P = S^*_{L_P} \circ S_{T_P}$, where L_P and T_P are non-looping processes. 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.13 (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.12, $S_P \subseteq R_P$, thus $\mathcal{A} R_P \mathcal{B}$ as well.

($\Leftarrow$) We proceed by induction on the structure of P.

- The base case is just the Existence Lemma for basic processes.

- P is a non-looping 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 $P_1 \mid P_2$, $\mathcal{A}R_{P_1|P_2}\mathcal{B}$ 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, $\langle P\rangle\phi \in C$ 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 looping process, $\mathcal{A}R_P\mathcal{B}$ and $\phi \in \mathcal{B}$. Then, there is a finite sequence of atoms $\mathcal{C}_0 \ldots \mathcal{C}_n$ such that $\mathcal{A} = \mathcal{C}_0 R_{L_P} \mathcal{C}_1 \ldots \mathcal{C}_{n-1} R_{L_P} \mathcal{C}_n R_{T_P} \mathcal{B}$. 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)**, $\langle P\rangle\phi \in \mathcal{A}$. □

Lemma A.14 (Truth Lemma). *Let $\mathcal{N}^\Gamma = (At(\Gamma), \{R_P\}, \mathbf{V})$ be the CCS-PDL model over Γ. For all atoms $\mathcal{A} \in At(\Gamma)$ and all formulas $\varphi \in C(\Gamma)$, $\mathcal{N}^\Gamma, \mathcal{A} \Vdash \varphi$ 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.5.

- $\phi = \langle P\rangle\psi$:

  ($\Rightarrow$) Suppose that $\mathcal{N}^\Gamma, \mathcal{A} \Vdash \langle P\rangle\psi$. Then, there exists $\mathcal{A}' \in \mathcal{N}^\Gamma$ such that $\mathcal{A}R_P\mathcal{A}'$ and $\mathcal{N}^\Gamma, \mathcal{A}' \Vdash \psi$. 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 $\mathcal{A}' \in \mathcal{N}^\Gamma$ such that $\mathcal{A}R_P\mathcal{A}'$ and $\psi \in \mathcal{A}'$. By the induction hypothesis, $\mathcal{N}^\Gamma, \mathcal{A}' \Vdash \psi$, which implies $\mathcal{N}^\Gamma, \mathcal{A} \Vdash \langle P\rangle\psi$. □

Theorem A.15 (Completeness). *Every consistent formula is satisfiable in a finite model that respects definition 4.7.*

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.7, there is an atom $\mathcal{A} \in At(\varphi)$ such that $\varphi \in \mathcal{A}$. Let $\mathcal{N}^\varphi$ be the CCS-PDL model over φ. Then, by the Truth Lemma (lemma A.14), as $\varphi \in \mathcal{A}$, we conclude that $\mathcal{N}^\varphi, \mathcal{A} \Vdash \varphi$, which proves the theorem. □