On Propositional Dynamic Logic and Concurrency

MATTEO ACCLAVIO*, University of Sussex, United Kingdom
FABRIZIO MONTESI, University of Southern Denmark, Denmark
MARCO PERESSOTTI, University of Southern Denmark, Denmark

Dynamic logic is a powerful approach to reasoning about programs and their executions, obtained by extending classical logic with modalities that can express program executions as formulas. However, the use of dynamic logic in the setting of concurrency has proved problematic because of the challenge of capturing interleaving. This challenge stems from the fact that, traditionally, programs are represented by their sets of traces. These sets are then expressed as elements of a Kleene algebra, for which it is not possible to decide equality in the presence of the commutations required to model interleaving.

In this work, we generalise propositional dynamic logic (PDL) to a logic framework we call operational propositional dynamic logic (OPDL), which departs from tradition by distinguishing programs from their traces. Traces are generated by an arbitrary operational semantics that we take as a parameter, making our approach applicable to different program syntaxes and semantics. To develop our framework, we provide the first proof of cut-elimination for a finitely-branching non-wellfounded sequent calculus for PDL. Thanks to this result we can effortlessly prove adequacy for PDL, and extend these results to OPDL. We conclude by discussing OPDL for two representative cases of concurrency: the Calculus of Communicating Systems (CCS), where interleaving is obtained by parallel composition, and Choreographic Programming, where interleaving is obtained by out-of-order execution.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: Propositional Dynamic Logic, Concurrency, Cut-elimination, Process Calculi, Choreographies

ACM Reference Format:
Matteo Acclavio, Fabrizio Montesi, and Marco Peressotti. 2024. On Propositional Dynamic Logic and Concurrency. 1, 1 (November 2024), 26 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Logic, and in particular proof theory, offers several approaches to reason about different computational properties of programs. In the Curry-Howard correspondence, programs are represented by proofs, thus providing a strong foundation for the development of type systems [12, 27, 75]. In logic programming, a program is an inference system, which allows for using proof search as the means of execution [50, 55]. In *dynamic logic* (DL), programs are part of the language of formulas itself, which enables the direct use of the logic to reason about the semantics of programs [29]. Under the latter view, the purpose of programs is to change the truth value of a formula. At the syntactic level, each program α defines the modalities $[\alpha]$ and $\langle\alpha\rangle$ and a formula $[\alpha]\,\phi$ is interpreted as 'every state reached after executing

*Supported by Villum Fonden, grant no. 50079

Authors' addresses: Matteo Acclavio, University of Sussex, Brighton, United Kingdom; Fabrizio Montesi, University of Southern Denmark, Odense, Denmark; Marco Peressotti, University of Southern Denmark, Odense, Denmark.

Manuscript submitted to ACM

α satisfies the formula ϕ' while a formula $\langle\alpha\rangle\,\phi$ is interpreted as 'there is a state reached after executing α satisfying the formula F'. This idea has been of profound inspiration in the field of formal verification [13, 70]. In this work, we are interested in the propositional fragment of dynamic logic (Propositional Dynamic Logic, or PDL) [45].

PDL *and the concurrency problem.* While PDL has been successfully applied to the study of sequential programs, extending this approach to concurrent programs remains challenging. In standard PDL, a program is represented by a regular expression that describes its set of possible traces. In other words, programs are elements of a free Kleene algebra. This works well for sequential programs, because one obtains that the theory of equational reasoning for Kleene algebras is a complete system for reasoning about *trace equivalence* [38, 41, 44, 68]. Trace equivalence is therefore captured by logical equivalence in PDL:

$$\alpha \text{ and } \beta \text{ have the same traces} \quad \text{iff} \quad \vdash_{\mathsf{PDL}} [\alpha]\,\phi \Leftrightarrow [\beta]\,\phi \qquad \text{for all } \phi\,. \tag{1}$$

However, the case of concurrent programs with an interleaving semantics is more problematic. In the presence of interleaving, one expects traces differing by interleaving to be equivalent modulo equations of the form $\alpha;\beta = \beta;\alpha$ (called *commutations*). Unfortunately, the word problem in a Kleene algebra enriched with an equational theory containing such commutations is known to be undecidable[1], which makes undecidable checking whether two modalities in PDL are the same. For the same reason, a general treatment of concurrency is still elusive also for simpler approaches than PDL, like equational reasoning on programs represented as terms in Kleene algebras [10, 36, 37, 42].[2]

As a consequence of this problem, applications of PDL to concurrency fall short of the expected level of expressivity from established theories, like CCS [56] and the π-calculus [57]. For example, previous works lack nested parallel composition, synchronisation, or recursion [8, 9, 51, 63–65]. In general, adding any new concurrency feature (e.g., a construct in the language of programs or a law defining its semantics) requires great care and effort in establishing the meta-theoretical properties of the logic. The result: a literature of various propositional dynamic logics, all independently useful, but with different limitations and dedicated technical developments.

Main contributions and structure of the paper. In this work, we significantly advance the line of work on PDL by developing *operational propositional dynamic logic* (OPDL). The key innovation of OPDL is to distinguish and separate reasoning on programs from reasoning on their traces. Thanks to this distinction, we circumvent previous limitations and finally obtain a PDL that can be applied to established concurrency models, such as CCS [56] and choreographic programming [58]. Crucially, OPDL is a general framework: it is parameterised on the operational semantics used to generate traces from programs, yielding a simple yet reusable approach to characterise trace reasoning.

We proceed as described next.

After recalling the axiomatization and semantics of PDL in Section 2, in Section 3 we provide a proof of its soundness and completeness with respect to the sequent calculus introduced [19]. For this purpose, we provide the first cut-elimination result for this non-wellfounded calculus, by adapting the technique developed in [2].[3] This allows us to prove our results by reasoning on the axiomatisation and the sequent system, without directly relying on semantic arguments.

[1] This is proven in [43] by reducing the Post correspondence problem to the word problem by combining sequential composition, iteration, and commutations.

[2] While the pure algebraic setting based on Kleene algebras is strictly less expressive than PDL, the missing expressivity (propositional reasoning on states reached after performing actions in a given trace) can be recovered by considering the (much) more complex structure of Kleene modules [22].

[3] A cut-elimination result for another sequent calculus for PDL is provided in [34], but that calculus is fundamentally different: it employs nested sequents and contains rules with an infinite number of premises.

$\phi, \psi \coloneqq$	$\top$		true
$\mid$	$\bot$		false
$\mid$	$p \in \mathcal{A}$	(with $p \in \mathcal{A}$)	atom (literal)
$\mid$	$\overline{p}$	(with $p \in \mathcal{A}$)	negated atom (literal)
$\mid$	$\phi \vee \psi$		disjunction
$\mid$	$\phi \wedge \psi$		conjunction
$\mid$	$[\alpha]\,\phi$		box
$\mid$	$\langle\alpha\rangle\,\phi$		diamond

$\alpha, \beta \coloneqq$	ϵ	terminated program
$\mid$	$\varnothing$	stacked program
$\mid$	$a \in \mathbb{I}$	instruction
$\mid$	$t \in \mathbb{T}$	test
$\mid$	$\alpha;\beta$	sequential composition
$\mid$	α^*	iteration
$\mid$	$\alpha \oplus \beta$	(non-deterministic) choice

Fig. 1. Grammar generating formulas

Then, in Section 4, we extend PDL with an additional axiom allowing us to encapsulate an operational semantics for a set of programs into the trace reasoning. We call the resulting logic *operational propositional dynamic logic* (or OPDL), providing a general framework encompassing various previous works [9, 31, 51].

We show that the expressive power of our framework goes beyond the state of the art in Section 5, by instantiating it for two use cases of archetypes of concurrent programming languages: the Calculus of Communicating Systems (CCS) [56], representative of the process algebra approach, and the textbook presentation of choreographic programming [59], representative of languages inspired by the *Alice-and-Bob* notation that originates from security protocols. These two cases are interesting because they model concurrency in completely different ways: in CCS, concurrency is obtained through an explicit parallel operator equipped with an interleaving semantics, while in choreographic programming concurrency is obtained implicitly by executing instructions out of order whenever they involve different processes. Thus, OPDL advances the study of PDL with leaps in both expressivity and versatility.

We conclude and discuss future work in Section 6.

Related work is discussed, where relevant, as part of our development (in addition to the works mentioned in this introduction).

2 PRELIMINARY NOTIONS ON PROPOSITIONAL DYNAMIC LOGIC

In this section we recall standard definitions and results for PDL as presented in [29].

We consider the set $\mathcal{F}_{\mathsf{PDL}}$ of **formulas** generated from a countable set $\mathcal{A}$ of **propositional atoms**, a set of **atomic programs** $\mathbb{I}$ and a set of **tests** $\mathbb{T} = \{\phi? \mid \phi \in \mathcal{F}_{\mathsf{PDL}}\}$ by the grammars in Figure 1. The **(logical) implication** $A \Rightarrow B \coloneqq (\overline{A}) \vee B$ is defined by extending the negation from atoms to formulas via the **De Morgan laws**:

$$\overline{\top} = \bot \qquad \overline{(\overline{\phi})} = \phi \qquad \overline{\phi \wedge \psi} = (\overline{\phi}) \vee (\overline{\psi}) \qquad \overline{[\alpha]\,\phi} = \langle\alpha\rangle\,\overline{\phi}\,. \tag{2}$$

We write $\vdash_{\mathsf{PDL}} \phi$ if the formula ϕ is derivable from the axioms in Figure 2 using the rules **modus ponens** (MP), **necessitation** (NEC), **loop invariance** (LI), from the same figure. The **propositional dynamic logic** (or PDL) is defined as the logic of formulas satisfying $\vdash_{\mathsf{PDL}} \phi$.

Remark 1. The axiomatization of PDL is often presented by replacing the loop invariance rule with an additional axiom (scheme) $\mathbf{A_{Ind}}$: $(\phi \wedge [\alpha^*]\,(\phi \Rightarrow [\alpha]\,\phi)) \Rightarrow [\alpha^*]\,\phi$ reminding the induction axiom (scheme) in Peano arithmetic. Prove that the two formulations are equivalent is an exercise which can be found in [29].

Semantically speaking, while a *model* of propositional classical logic is simply an evaluation function assigning a truth value to each formula, models for PDL are given by *Kripke frames*. A Kripke frame for classical modal logic is

$$
\begin{array}{ll}
\textbf{PL} & : \text{Axiomatization of propositional classical logic} \\
\textbf{Neg} & : [\alpha]\,\phi \Leftrightarrow \overline{\left(\langle\alpha\rangle\,\overline{\phi}\right)} \\
\mathbf{K} & : ([\alpha]\,(\phi \Rightarrow \psi)) \Rightarrow ([\alpha]\,\phi \Rightarrow [\alpha]\,\psi) \\
\mathbf{A}_{\varnothing} & : [\varnothing]\,\phi \\
\mathbf{A}_{\epsilon} & : [\epsilon]\,\phi \Leftrightarrow \phi \\
\mathbf{A}_{?} & : [\psi?]\,\phi \Leftrightarrow (\overline{\psi} \vee \phi) \\
\mathbf{A}_{\oplus} & : [\alpha \oplus \beta]\,\phi \Leftrightarrow ([\alpha]\,\phi \wedge [\beta]\,\phi) \\
\mathbf{A}_{;} & : [\alpha;\beta]\,\phi \Leftrightarrow [\alpha]\,[\beta]\,\phi \\
\mathbf{A}_{*} & : [\alpha^*]\,\phi \Leftrightarrow (\phi \wedge [\alpha]\,[\alpha^*]\phi)
\end{array}
$$

$$
\text{MP}\ \frac{\vdash \phi \quad \vdash \phi \Rightarrow \psi}{\vdash \psi}
\qquad
\text{NEC}\ \frac{\vdash \phi}{\vdash [\alpha]\,\phi}
\qquad
\text{LI}\ \frac{\vdash \phi \Rightarrow [\alpha]\,\phi}{\vdash \phi \Rightarrow [\alpha^*]\,\phi}
$$

Fig. 2. Axioms and rules for Propositional Dynamic Logic.

$$
\begin{array}{ll}
\mathfrak{m}(\top) = W & \mathfrak{m}(\epsilon) = \{(v,v) \mid v \in W\} \\
\mathfrak{m}(\bot) = \varnothing & \mathfrak{m}(\varnothing) = \varnothing \\
\mathfrak{m}\left(\overline{\phi}\right) = W \setminus \mathfrak{m}(\phi) & \mathfrak{m}(\phi?) = \{(v,v) \mid v \in \mathfrak{m}(\phi)\} \\
\mathfrak{m}(\phi \vee \psi) = \mathfrak{m}(\phi) \cup \mathfrak{m}(\psi) & \mathfrak{m}(\alpha;\beta) = \{(u,w) \mid \text{exists } v \text{ s.t. } (u,v) \in \mathfrak{m}(\alpha) \text{ and } (v,w) \in \mathfrak{m}(\beta)\} \\
\mathfrak{m}(\phi \wedge \psi) = \mathfrak{m}(\phi) \cap \mathfrak{m}(\psi) & \mathfrak{m}(\alpha \oplus \beta) = \mathfrak{m}(\alpha) \cup \mathfrak{m}(\beta) \\
\mathfrak{m}([\alpha]\,\phi) = \{v \mid w \in \mathfrak{m}(\phi) \text{ for all } w \text{ s.t. } (v,w) \in \mathfrak{m}(\alpha)\} & \mathfrak{m}(\alpha^*) = \bigcup_{n \geq 0} \mathfrak{m}(\alpha^n) \quad (\text{where } \alpha^0 = \epsilon) \\
\mathfrak{m}(\langle\alpha\rangle\,\phi) = \{v \mid w \in \mathfrak{m}(\phi) \text{ for a } w \text{ s.t. } (v,w) \in \mathfrak{m}(\alpha)\} &
\end{array}
$$

Fig. 3. Inductive definition of the meaning of compound formulas and programs in the Kripke frames.

given by a set of *worlds*, an *accessibility relation* between worlds, and an *evaluation function* assigning to each formula the set of worlds in which it is true. Intuitively, a classical model can be seen as a single-world Kripke frame in which the evaluation function assigns to each formula a set containing the unique world of the frame only if it the formula is evaluated as true. We recall here the formal definition of model for PDL.

Definition 2. A **Kripke frame** (or **model**) $\mathfrak{M} = \langle W, \mathfrak{m}\rangle$ is given by a set of **worlds** W, a **meaning function** associating to each atom $p \in \mathcal{A}$ a set of worlds $\mathfrak{m}(p) \subseteq W$ (in which p holds), and to each instruction $a \in \mathbb{I}$ an accessibility relation $\mathfrak{m}(a) \subseteq W \times W$. The meaning of compound formulas and programs is defined as shown in the left of Figure 3. We write $\mathfrak{M}, w \vDash \phi$ if $w \in \mathfrak{m}(\phi)$ (or simply $w \vDash \phi$ if $\mathfrak{M}$ is clear from the context), and $\mathfrak{M} \vDash \phi$ if $\mathfrak{M}, w \vDash \phi$ holds for any world w of $\mathfrak{M}$. Finally we write $\vDash \phi$ if $\mathfrak{M} \vDash \phi$ holds for any possible model $\mathfrak{M}$.

The proof of soundness and completeness result of the axioms of PDL with respect to the semantics can be found in [29]. In particular, completeness is shown by constructing a model $\mathfrak{M}_\phi$ for each consistent formula ϕ such that the formula ϕ holds in at least a world (i.e. $\mathfrak{m}(\phi) \neq \varnothing$ in $\mathfrak{M}_\phi$).

Theorem 3. *Let ϕ be a* PDL*-formula. Then $\vdash_{\mathsf{PDL}} \phi$ iff $\vDash_{\mathsf{PDL}} \phi$.*

We conclude by showing the following result, which allows us to interpreted the modality $[\alpha^*]$ as a fixpoint for the modality $[\alpha]$.

Lemma 4. *If $\nvdash_{\mathsf{PDL}} \phi \Rightarrow [\alpha^*]\,\psi$, then there is $n \in \mathbb{N}$ such that $\nvdash_{\mathsf{PDL}} \phi \Rightarrow [\alpha^n]\,\psi$.*

Proof. By Theorem 3, if $\nvdash_{\mathsf{PDL}} \phi \Rightarrow [\alpha^*]\,\psi$, then $\nvDash \phi \Rightarrow [\alpha^*]\,\psi$. By definition, this means that there is a model $\mathfrak{M} = \langle W, \mathfrak{m}\rangle$ such that $\mathfrak{m}(\phi \Rightarrow [\alpha^*]\,\psi) \neq W$. Now assume that $\vdash_{\mathsf{PDL}} \phi \Rightarrow \left[\alpha^k\right]\psi$ for any $k \in \mathbb{N}$. By Theorem 3 this implies $\vDash \phi \Rightarrow \left[\alpha^k\right]\psi$ for any $k \in \mathbb{N}$; therefore that in any model $\mathfrak{M} = \langle W, \mathfrak{m}\rangle$ we have $W = \mathfrak{m}\left(\phi \Rightarrow \left[\alpha^k\right]\psi\right)$ for all

$$\top \frac{}{\vdash \top} \qquad \mathrm{AX} \frac{}{\vdash \phi, \overline{\phi}} \qquad \mathrm{W} \frac{\vdash \Gamma}{\vdash \Gamma, \phi} \qquad \vee \frac{\vdash \Gamma, \phi, \psi}{\vdash \Gamma, \phi \vee \psi} \qquad \wedge \frac{\vdash \Gamma, \phi \quad \vdash \Gamma, \psi}{\vdash \Gamma, \phi \wedge \psi}$$

$$\mathrm{K}_\alpha \frac{\vdash \Gamma, \phi}{\vdash \langle \alpha \rangle \Gamma, [\alpha] \phi} \ \alpha \notin \{\epsilon, \varnothing\} \qquad \mid \qquad \mathrm{cut} \frac{\vdash \Gamma, \phi \quad \vdash \Gamma, \overline{\phi}}{\vdash \Gamma}$$

$$[\epsilon] \frac{\vdash \Gamma, \phi}{\vdash \Gamma, [\epsilon] \phi} \qquad [\varnothing] \frac{}{\vdash [\varnothing] \phi} \qquad [?] \frac{\vdash \Gamma, \overline{\phi} \vee \psi}{\vdash \Gamma, [\phi?] \psi}$$

$$[\oplus] \frac{\vdash \Gamma, [\alpha] \phi \quad \vdash \Gamma, [\beta] \phi}{\vdash \Gamma, [\alpha \oplus \beta] \phi} \qquad [;] \frac{\vdash \Gamma, [\alpha] [\beta] \phi}{\vdash \Gamma, [\alpha; \beta] \phi} \qquad [*] \frac{\vdash \Gamma, \phi \quad \vdash \Gamma, [\alpha; \alpha^*] \phi}{\vdash \Gamma, [\alpha^*] \phi}$$

$$\langle \epsilon \rangle \frac{\vdash \Gamma, \phi}{\vdash \Gamma, \langle \epsilon \rangle \phi} \qquad \langle \varnothing \rangle \frac{\vdash \Gamma, \psi}{\vdash \Gamma, \psi, \langle \varnothing \rangle \phi} \qquad \langle ? \rangle \frac{\vdash \Gamma, \phi \wedge \psi}{\vdash \Gamma, \langle \phi? \rangle \psi}$$

$$\langle \oplus \rangle \frac{\vdash \Gamma, \langle \alpha \rangle \phi, \langle \beta \rangle \phi}{\vdash \Gamma, \langle \alpha \oplus \beta \rangle \phi} \qquad \langle ; \rangle \frac{\vdash \Gamma, \langle \alpha \rangle \langle \beta \rangle \phi}{\vdash \Gamma, \langle \alpha; \beta \rangle \phi} \qquad \langle * \rangle \frac{\vdash \Gamma, \phi, \langle \alpha; \alpha^* \rangle \phi}{\vdash \Gamma, \langle \alpha^* \rangle \phi}$$

Fig. 4. Sequent calculus rules of the sequent system $\mathsf{LPD}^{\mathsf{cut}} = \mathsf{LPD} \cup \{\mathsf{cut}\}$.

$k \in \mathbb{N}$. This would be absurd since

$$W = \bigcap_{k \geq 0} W = \bigcap_{k \geq 0} \left(\mathfrak{m} \left(\phi \Rightarrow \left[\alpha^k \right] \psi \right) \right) = \bigcap_{k \geq 0} \left(\mathfrak{m} \left(\overline{\phi} \right) \cup \mathfrak{m} \left(\left[\alpha^k \right] \psi \right) \right) =$$

$$= \mathfrak{m} \left(\overline{\phi} \right) \cup \left(\bigcap_{k \geq 0} \mathfrak{m} \left(\left[\alpha^k \right] \psi \right) \right) = \mathfrak{m} \left(\overline{\phi} \right) \cup \mathfrak{m} \left(\left[\alpha^* \right] \psi \right) = \mathfrak{m} \left(\phi \Rightarrow \left[\alpha^* \right] \psi \right) \neq W \qquad \square$$

3 SEQUENT CALCULUS FOR PDL

In this section we recall the definition for (possibly infinite) derivations in a sequent system. We then consider the sequent system LPD given by the rules in Figure 4 introduced in [18–20] (for the fragment of PDL without the programs ϵ and $\varnothing$), as an adaptation of the sequent calculus for the modal μ-calculus given in [71].

To prove soundness and completeness of the sequent system LPD with respect to the axiomatization of PDL, we rely on the subformula property following from the admissibility of the cut rule. To prove admissibility of cut, we provide the first cut-elimination result for LPD by adapting the technique developed in [2].[4] More precisely, these adequacy results are proven by translating the winning conditions of the *provability games* for PDL and for modal μ-calculus (respectively defined in [48] and [60]) into correctness criteria for non-wellfounded derivations[5].

3.1 Definitions and Notations for Derivations

We assume the reader to be familiar with the terminology of sequent calculus (see, e.g., [72]) and non-wellfounded sequent calculi (see, e.g., [2, 7]). We recall here the formalism we adopt in this paper.

[4]Note that the system LPD from [19] (as well as the system studied in [71], and the labeled cyclic proof system from [21]) does not contain the rule cut, and its soundness and completeness is not proven with respect to axiomatization but with respect to the semantics.
[5]More precisely, as explicitly shown in [1] in the case of intuitionistic logic, translations of winning games correspond to derivations in *focused* sequent systems, that is, a systems in which the order of rules in proof search is subject to specific restriction.

A **sequent** is a set of formulas. A **sequent system** is given by a set of **rules** of the form $\mathsf{r}\,\dfrac{}{\vdash \Gamma}$ or $\mathsf{r}\,\dfrac{\vdash \Gamma_1}{\vdash \Gamma}$ or $\mathsf{r}\,\dfrac{\vdash \Gamma_1 \quad \vdash \Gamma_2}{\vdash \Gamma}$, where the sequents Γ_1 and Γ_2 are called **premises** and the sequent Γ is called **conclusion** of the rule r. A formula is **active** (resp. **principal**) for a rule if it occurs in a premise but not in the conclusion (resp. it occurs in the conclusion but in none of its premises). A rule r is **admissible** in S if its conclusion is derivable in S whenever its premises are.

Definition 5. A **tree** $\mathcal{T}$ is a prefix-closed set of words over the alphabet $\{1, \dots, n\}$ such that if $vk \in \mathcal{T}$, then $vm \in \mathcal{T}$ for all $m < k$. The elements of $\mathcal{T}$ are called **nodes**, the empty word $\epsilon \in \mathcal{T}$ is called **root**. A node $v \in \mathcal{T}$ is **below** $w \in \mathcal{T}$ if $w = vv'$ with $v' \neq \epsilon$. The **height** of a node is the number of nodes below it. A **child** of $v \in \mathcal{T}$ is a node of the form $vk \in \mathcal{T}$ with $k \in \{1, \dots, n\}$. A **branch** is a prefix-closed totally ordered (w.r.t to the prefix order) set of nodes.

A **derivation** (resp. **open derivation**) is a labeling $\mathcal{D}$ of a tree $\mathcal{T}_\mathcal{D}$ with nodes labeled by sequents in such a way for each node v (resp. for each non-leaf node v) the sequent $\mathcal{D}(v)$ with is the conclusion sequent of a rule with premises the sequents $\mathcal{D}(v_1), \dots, \mathcal{D}(v_n)$ where $v_1, \dots, v_n$ are the children of v. The sequent $\mathcal{D}(\epsilon)$ is called the **conclusion** of $\mathcal{D}$ and a leaf v such that $\mathcal{D}(v)$ is not the conclusion of a rule is called an **open premise**. We identify (an occurence of) a **rule** r in $\mathcal{D}$ with the nodes corresponding to its conclusion and premises. A node is **below** a rule if it is its conclusion or any node below, and we may refer to a node of $\mathcal{D}$ as a node in the underlying tree $\mathcal{T}_\mathcal{D}$.

A **sub-derivation** of $\mathcal{D}$ is a derivation $\mathcal{D}'$ such that $\mathcal{D}'(w) = \mathcal{D}(vw)$ for a $v \in \mathcal{T}_\mathcal{D}$. A derivation is **regular** if it has finitely many distinct sub-derivations. An open derivation $\mathcal{D}'$ is an **approximation**[6] of $\mathcal{D}$ (denoted $\mathcal{D}' \preceq \mathcal{D}$) if $\mathcal{T}_{\mathcal{D}'} \subseteq \mathcal{T}_\mathcal{D}$ and $\mathcal{D}'(v) = \mathcal{D}(v)$ for any $v \in \mathcal{T}_{\mathcal{D}'}$.

If X is a set of derivations, then we say that Γ is **provable** in X (denoted $\vdash_\mathsf{X} \Gamma$) if there is a derivation of Γ in X. For this purpose, we may identify a sequent system S with the set of derivations over S.

Notation 6. We may denote a derivation with conclusion Γ (resp. an open derivation with open premise Δ and conclusion Γ) by $\begin{matrix}\mathcal{D}\,\|\\ \vdash \Gamma\end{matrix}\left(\text{resp. } \begin{matrix}\vdash \Delta\\ \mathcal{D}\,\|\\ \vdash \Gamma\end{matrix}\right)$.

A regular derivation can be represented as a finite (directed) graph of sequents, by identifying nodes of its tree which are conclusions of two identical sub-derivations. In this case we label the bottom-most rules of identical derivations by the same symbol (see the derivation on the left of Equation (3) for an example).

3.2 A Sequent System for PDL

The sequent systems $\mathsf{LPD}^{\mathsf{cut}}$ is defined by the set of rules in Figure 4, while LPD is the sub-system without the rule cut.

As standard as soon as we allow us to consider infinite derivations, we could be able to construct unsound derivations in a sequent system. By means of example, consider the following derivations allowing us to derive in LK any formula ϕ using ax and cut, or the (admissible) contraction rule c

$$\dfrac{\dfrac{}{\vdash \phi, \bar{\phi}}\,\mathsf{ax} \qquad \dfrac{}{\vdash \phi}\,\mathsf{cut}\,\star}{\vdash \phi}\,\mathsf{cut}\,\star \qquad\qquad \dfrac{\dfrac{\vdots}{\vdash \phi, \phi}\,\mathsf{c}}{\vdash \phi}\,\mathsf{c} \tag{3}$$

In order to recover correctness, we introduce the following *progressiveness* criterion.

[6]The name is meant to suggest that infinite derivations can be seen as the limit of their approximations. See Definition 12.

Definition 7. Let ϕ be a formula occurring in a sequent of a $\mathcal{D} \in \mathsf{LPD}$ conclusion of a rule r. We say that a formula ψ occurring in a premise of r is an **immediate ancestor** of ϕ whenever one of the following holds:

- ψ is an active formula of $\mathsf{r} \neq \mathrm{K}_\alpha$ with principal formula ϕ;
- ψ is an active formula of K_α-rule and $\phi \in \{[\alpha]\,\psi, \langle\alpha\rangle\,\psi\}$;
- ψ is the unique occurrence of ϕ in the sequent.

A **thread** in a derivation $\mathcal{D}$ is a maximal sequence of formulas occurring in sequents of $\mathcal{D}$ totally ordered with respect to the immediate ancestor relation. Its first element is called **starting point**. A thread is **progressing** if its starting point is a formula $\phi = [\alpha^*]\,\psi$ (also called the **principal formula** of the thread) which occurs as active formula of K_α-rules[7] infinitely often. A derivation is **progressing** if each infinite branch contains a progressing thread. We denote by pLPD the set of progressing derivations.

Lemma 8. *Each $\mathcal{D} \in \mathsf{pLPD}$ is of the following shape*

$$\dfrac{\begin{array}{ccc}\mathcal{D}_1\,\|\!\| & & \mathcal{D}_n\,\|\!\| \\ \vdash \Gamma_1, [\alpha_1^*]\,\phi_1 & \cdots & \vdash \Gamma_n, [\alpha_n^*]\,\phi_n\end{array}}{\begin{array}{c}\mathcal{D}_0 \\ \vdash \Gamma\end{array}}$$

where $\mathcal{D}_0$ is a finite open derivation with n open premises of the form $\Gamma_i, [\alpha_i^*]\,\phi_i$ and such that $[\alpha_i^*]\,\phi_i$ is the starting point of a progressing thread in $\mathcal{D}$ for all $i \in \{1, \ldots, n\}$.

Proof. By definition, each infinite branch of $\mathcal{D}$ contains a progressing thread, which must have a starting point. We conclude by letting $\mathcal{D}_0$ be the approximation of $\mathcal{D}$ with open premises all such nodes. □

We can easily prove that if a formula is valid in PDL, then it is derivable in $\mathsf{pLPD}^{\mathsf{cut}}$.

Lemma 9. *The set of derivations* $\mathsf{pLPD} \cup \mathsf{cut}$ *is complete for* PDL.

Proof. Each axiom in Figure 2 is derivable in pLPD, that is, there is a derivation with conclusion the axiom formula defined as follows (see also Figure 5):

- **PL**: the sub-system $\{\top, \textsc{ax}, \textsc{w}, \wedge, \vee\}$ is a well-known sound and complete sequent system for classical logic (see [72], where the system is called the system G_3p);
- **Neg** is immediate by definition of the negation;
- $\mathbf{A}_\varnothing$ is proven using a single instance of $[\varnothing]$;
- $\mathbf{A}_\epsilon$, $\mathbf{A}_;$ and $\mathbf{K}$ are straightforward using rule $\textsc{ax}$, $\wedge$, $\vee$ and K_α;
- $\mathbf{A}_?$ is also straightforward using rule $\textsc{ax}$, $\wedge$, $\vee$ and $[?]$ and $\langle ? \rangle$;
- $\mathbf{A}_\oplus$ (resp. $\mathbf{A}_\star$) require the use of rules rule $\textsc{ax}$, $\wedge$, $\vee$, and both $[\oplus]$ and $\langle\oplus\rangle$ (resp. $[*]$ and $\langle * \rangle$) plus the rule $\textsc{w}$.

Moreover, each rule in Figure 2 is derivable in pLPD, that is, there is an open derivation in pLPD with the same conclusion of the rule and with a single open premise which is the same of the premise of the rule. Rules $\textsc{mp}$ and $\textsc{nec}$ are derivable as shown in Equation (4) below, while the loop-invariance rule ($\textsc{li}$) can be simulated by the progressive infinite derivation shown in Figure 6. Note that right premise of the cut-rule at the bottom of the derivation in Figure 6 is the axiom $\mathbf{A}_{\mathbf{Ind}}$ mentioned in Remark 1.

[7] It is easy to show that the principal formula of a progressing thread which is active for K_α-rules infinitely often, is also principal for $[*]$-rules infinitely often. More precisely, occurrence of these rules are interleaved and we can easily show that our progress condition is equivalent to the one in [19] formulated by means of $[*]$-rules.

$$\mathbf{K}: \dfrac{\dfrac{\dfrac{}{\vdash \phi, \overline{\phi}}\,\text{AX} \quad \dfrac{}{\vdash \psi, \overline{\psi}}\,\text{AX}}{\vdash \phi \wedge \overline{\psi}, \overline{\phi}, \psi}\,\wedge}{\vdash \langle\alpha\rangle\,(\phi \wedge \overline{\psi}), \langle\alpha\rangle\,\overline{\phi}, [\alpha]\,\psi}\,\kappa_\alpha}{\vdash \langle\alpha\rangle\,(\phi \wedge \overline{\psi}) \vee \left(\langle\alpha\rangle\,\overline{\phi} \vee [\alpha]\,\psi\right)}\,2\times\vee$$

$$\mathbf{A}_\epsilon: \dfrac{\dfrac{\dfrac{}{\vdash \overline{\phi}, \phi}\,\text{AX}}{\vdash \langle\epsilon\rangle\,\overline{\phi}, \phi}\,\langle\epsilon\rangle \quad \dfrac{\dfrac{}{\vdash \overline{\phi}, \phi}\,\text{AX}}{\vdash [\epsilon]\,\phi, \overline{\phi}}\,[\epsilon]}{\vdash \langle\epsilon\rangle\,\overline{\phi} \vee \phi}\,\vee \quad \dfrac{}{\vdash [\epsilon]\,\phi \vee \overline{\phi}}\,\vee}{\vdash (\langle\epsilon\rangle\,\overline{\phi} \vee \phi) \wedge ([\epsilon]\,\phi \vee \overline{\phi})}\,\wedge$$

$$\mathbf{A}_?: \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \psi \wedge \overline{\phi}, \overline{\psi} \vee \phi}\,\text{AX}}{\vdash \langle\psi?\rangle\,\overline{\phi}, (\overline{\psi} \vee \phi)}\,\langle?\rangle}{\vdash \langle\psi?\rangle\,\overline{\phi} \vee (\overline{\psi} \vee \phi)}\,\vee \quad \dfrac{\dfrac{}{\vdash \overline{\psi} \vee \phi, (\psi \wedge \overline{\phi})}\,\text{AX}}{\vdash [\psi?]\,\phi, (\psi \wedge \overline{\phi})}\,[?]}{\vdash [\psi?]\,\phi \vee (\psi \wedge \overline{\phi})}\,\vee}{\vdash (\langle\psi?\rangle\,\overline{\phi} \vee (\overline{\psi} \vee \phi)) \wedge ([\psi?]\,\phi \vee (\psi \wedge \overline{\phi}))}\,\wedge$$

$$\mathbf{A}_;: \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \langle\alpha\rangle\,\langle\beta\rangle\,\overline{\phi}, [\alpha]\,[\beta]\,\phi}\,\text{AX}}{\vdash \langle\alpha;\beta\rangle\,\overline{\phi}, [\alpha]\,[\beta]\,\phi}\,\langle?\rangle}{\vdash \langle\alpha;\beta\rangle\,\overline{\phi} \vee [\alpha]\,[\beta]\,\phi}\,\vee \quad \dfrac{\dfrac{}{\vdash [\alpha]\,[\beta]\,\phi, \langle\alpha\rangle\,\langle\beta\rangle\,\overline{\phi}}\,\text{AX}}{\vdash [\alpha;\beta]\,\phi, \langle\alpha\rangle\,\langle\beta\rangle\,\overline{\phi}}\,[?]}{\vdash [\alpha;\beta]\,\phi \vee \langle\alpha\rangle\,\langle\beta\rangle\,\overline{\phi}}\,\vee}{\vdash (\langle\alpha;\beta\rangle\,\overline{\phi} \vee [\alpha]\,[\beta]\,\phi) \wedge ([\alpha;\beta]\,\phi \vee \langle\alpha\rangle\,\langle\beta\rangle\,\overline{\phi})}\,\wedge$$

$$\mathbf{A}_\oplus: \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \langle\alpha\rangle\,\overline{\phi}\,[\alpha]\,\phi}\,\text{AX}}{\vdash \langle\alpha\rangle\,\overline{\phi}, \langle\beta\rangle\,\overline{\phi}, [\alpha]\,\phi}\,\text{w} \quad \dfrac{\dfrac{}{\vdash \langle\beta\rangle\,\overline{\phi}, [\alpha]\,\phi}\,\text{AX}}{\vdash \langle\alpha\rangle\,\overline{\phi}, \langle\beta\rangle\,\overline{\phi}, [\alpha]\,\phi}\,\text{w}}{\vdash \langle\alpha\rangle\,\overline{\phi}, \langle\beta\rangle\,\overline{\phi}, [\alpha]\,\phi \wedge [\beta]\,\phi}\,\wedge}{\vdash \langle\alpha \oplus \beta\rangle\,\overline{\phi}, [\alpha]\,\phi \wedge [\beta]\,\phi}\,\langle\oplus\rangle}{\vdash \langle\alpha \oplus \beta\rangle\,\overline{\phi} \vee ([\alpha]\,\phi \wedge [\beta]\,\phi)}\,\vee \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash [\alpha]\,\phi, \langle\alpha\rangle\,\overline{\phi}}\,\text{AX}}{\vdash [\alpha]\,\phi, \langle\alpha\rangle\,\overline{\phi}, \langle\beta\rangle\,\overline{\phi}}\,\text{w} \quad \dfrac{\dfrac{}{\vdash [\beta]\,\phi, \langle\beta\rangle\,\overline{\phi}}\,\text{AX}}{\vdash [\beta]\,\phi, \langle\alpha\rangle\,\overline{\phi}, \langle\beta\rangle\,\overline{\phi}}\,\text{w}}{\vdash [\alpha]\,\phi, \langle\alpha\rangle\,\overline{\phi}, \langle\beta\rangle\,\overline{\phi}}\,[\oplus]}{\vdash [\alpha \oplus \beta]\,\phi, \langle\alpha\rangle\,\overline{\phi}, \langle\beta\rangle\,\overline{\phi}}\,2\times\vee}{\vdash [\alpha \oplus \beta]\,\phi \vee (\langle\alpha\rangle\,\overline{\phi} \vee \langle\beta\rangle\,\overline{\phi})}\,\vee}{\left(\langle\alpha \oplus \beta\rangle\,\overline{\phi} \vee ([\alpha]\,\phi \wedge [\beta]\,\phi)\right) \wedge \left([\alpha \oplus \beta]\,\phi \vee (\langle\alpha\rangle\,\overline{\phi} \vee \langle\beta\rangle\,\overline{\phi})\right)}\,\wedge$$

$$\mathbf{A}_*: \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \overline{\phi}, \phi}\,\text{AX}}{\vdash \overline{\phi}, \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi}, \phi}\,\text{w} \quad \dfrac{\dfrac{}{\vdash \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi}, [\alpha]\,[\alpha^*]\phi}\,\text{AX}}{\vdash \overline{\phi}, \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi}, [\alpha]\,[\alpha^*]\phi}\,\text{w}}{\vdash \overline{\phi}, \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi}, (\phi \wedge [\alpha]\,[\alpha^*]\phi)}\,\wedge}{\vdash \langle\alpha^*\rangle\,\overline{\phi}, (\phi \wedge [\alpha]\,[\alpha^*]\phi)}\,\langle*\rangle}{\vdash \langle\alpha^*\rangle\,\overline{\phi} \vee (\phi \wedge [\alpha]\,[\alpha^*]\phi)}\,\vee \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \phi, \overline{\phi}}\,\text{AX}}{\vdash \phi, \overline{\phi}, \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi}}\,\text{w} \quad \dfrac{\dfrac{}{\vdash [\alpha]\,[\alpha^*]\,\phi, \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi}}\,\text{AX}}{\vdash [\alpha]\,[\alpha^*]\,\phi, \overline{\phi}, \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi}}\,\text{w}}{\vdash [\alpha^*]\,\phi, \overline{\phi}, \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi}}\,[*]}{\vdash [\alpha^*]\,\phi \vee (\overline{\phi} \vee \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi})}\,2\times\vee}{\vdash \langle\alpha^*\rangle\,\overline{\phi} \vee (\phi \wedge [\alpha]\,[\alpha^*]\phi)) \wedge \left([\alpha^*]\,\phi \vee (\overline{\phi} \vee \langle\alpha\rangle\,\langle\alpha^*\rangle\,\overline{\phi})\right)}\,\wedge$$

Fig. 5. Non-trivial derivations of the axioms of PDL in LPD.

MP	NEC
$\dfrac{\vdash \phi \qquad \dfrac{\vdash \overline{\phi} \vee \psi \qquad \dfrac{\dfrac{}{\vdash \phi, \overline{\phi}}\,\text{AX} \quad \dfrac{}{\vdash \overline{\psi}, \psi}\,\text{AX}}{\vdash \phi \wedge \overline{\psi}, \overline{\phi}, \psi}\,\wedge}{\vdash \overline{\phi}, \psi}\,\text{cut}}{\vdash \psi}\,\text{cut}$	$\dfrac{\vdash \phi}{\vdash [\alpha]\,\phi}\,\kappa_\alpha$

(4)

This allows us to conclude because $\vdash_{\mathsf{PDL}} \phi$ iff there is a derivation in the Hilbert system made of the axioms and rules in Figure 2, and each of such a derivation can be translated into a derivation in $\mathsf{pLPD}^{\mathsf{cut}}$ by replacing each axiom and rule with the ones provided. □

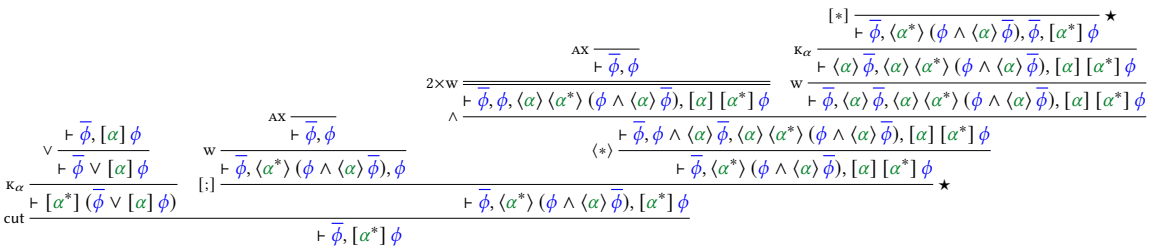


Fig. 6. Derivability of the loop invariance rule in $\mathsf{LPD}^{\mathsf{cut}}$.

3.3 Cut-Elimination in $\mathsf{LPD}^{\mathsf{cut}}$

In order to prove cut-elimination in LPD, we adapt the proof in [2, 3] to define an infinitary rewriting defined from the cut-elimination steps in Figure 7 able to remove all cut-rules from progressing derivations in $\mathsf{LPD}^{\mathsf{cut}}$.

Remark 10. To reduce the cases taken into account in Figure 7, we restrain the rule κ_α to atomic programs $\alpha \in \mathbb{I}$. The general instance of this rule is derivable reasoning by induction on the structure of $\alpha \in \mathbb{P}$ using this atomic version of the rule κ_α and rules $[\oplus]\,, \langle\oplus\rangle\,, [;]\,, \langle;\rangle\,, [*]\,, \langle*\rangle$.

The proof of cut-elimination can be summarized as follows:

- we prove that the set of approximations of derivations of a same sequent Γ is a Scott domain;
- we then define *maximal (non-deterministic) cut-elimination strategies* as specific sets of *views*, that is, maximal sequences of open derivations, obtained by applying cut-elimination steps to open derivations over $\mathsf{LPD}^{\mathsf{cut}}$. In these strategies we require a *coherence* condition ensuring that each view in the strategy starting from an open derivation $\mathcal{D}$ can be 'projected' (resp. 'lifted') to a view in the strategy over an open derivation $\mathcal{D}'$ such that $\mathcal{D}$ is an approximation of $\mathcal{D}'$ (resp. $\mathcal{D}'$ is an approximation of $\mathcal{D}$).
- we prove that each view σ in a cut-elimination strategy $\mathfrak{h}$, where cut-elimination steps are applied *bottom-up*, defines a Scott-continuous function f_σ, which associates to each derivation $\mathcal{D}$ the derivation which is the limit of succession of the greatest cut-free approximations the derivations in view σ;
- we conclude by showing that each $f_\sigma(\mathcal{D})$ is a well-defined and progressing derivations.

Remark 11. Note that in [2] the authors rely on the confluence of cut-elimination over finite approximations. This property is due to the fact that the system is inspired by the parsimonious linear logic [53, 54], a variant of linear logic [25] following the tradition of light and soft linear logic [26, 47, 52].

However, such a desirable feature is not possible in LPD since this system is an extension of a sequent calculus for classical logic (see Remark 16). There we have to define *non-deterministic* cut-elimination strategies. Note that the lack of confluence does not jeopardize our results because we are interested in proving cut-elimination, not in studying a Curry-Howard correspondence for PDL – which would require a different approach because the denotational semantics of LPD should extend the one of LK (see, e.g., [62]).

We first recall standard definitions on Scott domains and Scott-continuous functions.

Definition 12. Let S be a set, S' be a subset of S, and let $<$ be a partial order over S. We say that S is a **direct set** if for all $x, y \in S$ there is $z \in S$ such that $x \leq z$ and $y \leq x$. An **upper bound** of S' is an element $x \in S$ such that $y \leq x$ for all

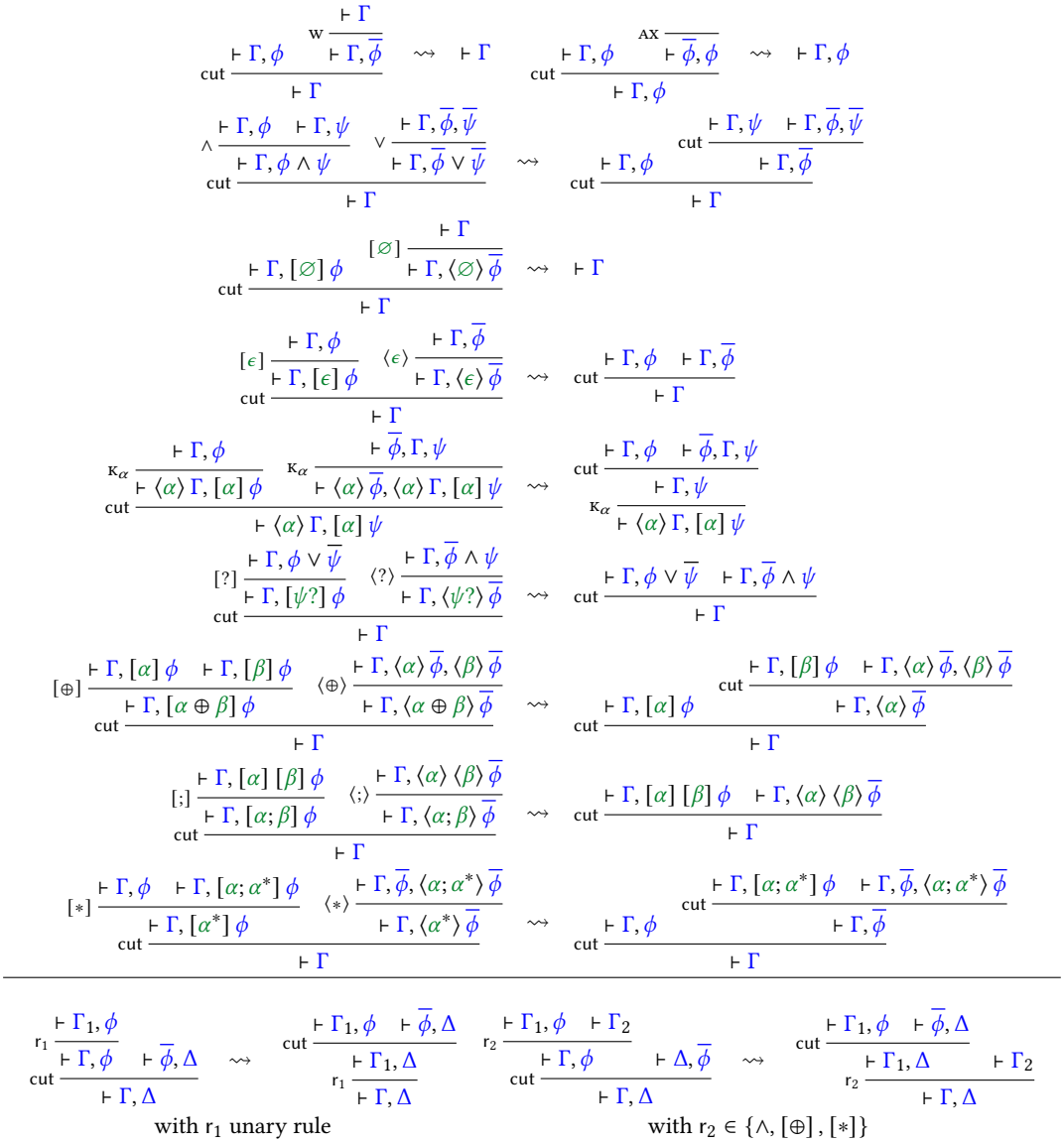


Fig. 7. Cut-elimination steps in LPD (with κ_α restricted on $\alpha \in \mathbb{I}$). The steps in the bottom-most row are called *commutative*.

$y \in S$; a **supremum** of S' (also denoted $\bigsqcup S'$) is an upper bound x such that $x \leq y$ for any y upper bound of S'. A $c \in S$ is **compact** if for all direct subset $S' \subseteq S$ such that if $\bigsqcup S'$ is defined and $c \leq \bigsqcup S'$, then $c \leq x$ for a $x \in S$.

A **Scott domain** is a pair $\mathfrak{D} = \langle S, < \rangle$ such that:

- $\mathfrak{D}$ is **directed complete**: every directed subset of S has a supremum;
- $\mathfrak{D}$ is **bounded complete**: every subset which has an upper bound has a supremum;
- $\mathfrak{D}$ is **algebraic**: every element in $\mathfrak{D}$ can be seen as the supremum of a direct set of compact elements of $\mathcal{D}$.

A function f over a Scott domain $\mathfrak{D}$ is **Scott-continuous** if it preserves suprema, that is, if $f(\bigsqcup S') = \bigsqcup(f(S'))$.

Notation 13. We denote by oLPD the set of open derivations over $\mathsf{LPD}^{\mathsf{cut}}$ and by $\mathcal{K}_{\mathcal{D}}$ the set of approximations of a $\mathcal{D} \in$ oLPD. If $\mathcal{D} \in$ oLPD, we denote by $\mathrm{cf}(\mathcal{D})$ the greatest (w.r.t. $\preceq$) cut-free approximation of $\mathcal{D}$.

Proposition 14. *The set of open derivations in* oLPD *with conclusion* Γ *is a Scott domain (w.r.t.* $\prec$*) with compact elements the open derivations with conclusion* Γ.

Proof. Directed and bounded completeness follows by definition of $\preceq$. Algebricity follows by the remark that each open derivation $\mathcal{D}$ can be seen as the supremum of the set $\mathcal{K}_{\mathcal{D}}$. □

We now can define maximal cut-elimination strategies as sequences of open derivations obtained by applying the cut-elimination steps to open derivations.

Definition 15. A **cut-elimination view** for $\mathcal{D} \in$ oLPD is a countable sequence $\sigma = \sigma_0 \cdot \sigma_1 \cdots \sigma_n \cdots$ (with length $\ell(\sigma) \in \mathbb{N} \cup \{\infty\}$) of open derivations in oLPD with $\sigma_0 = \mathcal{D}$ and such that σ_{i+1} is obtained by applying a cut-elimination step to σ_i.

Let $\mathfrak{s}$ be a family of views. We denote by $\mathfrak{s}_{\mathcal{D}}$ the set of views starting with $\mathcal{D}$, that is, $\mathfrak{s}_{\mathcal{D}} \coloneqq \{\sigma \in \mathfrak{s} \mid \sigma_0 = \mathcal{D}\}$. We say that $\mathfrak{s}$ is a **maximal cut-elimination strategy** (or mces) if:

- $\mathfrak{s}$ is **total**: $\mathfrak{s}_{\mathcal{D}} \neq \varnothing$ for each $\mathcal{D} \in$ oLPD;
- $\mathfrak{s}$ contains **maximal views** only: no cut-elimination step can be applied to $\sigma_{\ell(\sigma)}$ for any $\sigma \in \mathfrak{s}$;
- $\mathfrak{s}$ is **memory-less**: if $\sigma \in \mathfrak{s}$ and $\sigma_{i+1} = \mathcal{D}$, then $\sigma_0 \cdots \sigma_i \cdot \sigma' \in \mathfrak{s}$ for each $\sigma' \in \mathfrak{s}_{\mathcal{D}}$.
- $\mathfrak{s}$ is **coherent** over approximations: if $\mathcal{D}' \prec \mathcal{D}$ and r is a cut-rule occurring in both $\mathcal{D}$ and $\mathcal{D}'$, then there is a $\sigma \in \mathfrak{s}_{\mathcal{D}}$ such that σ_1 is obtained by applying a cut-elimination step to r iff there is a $\sigma' \in \mathfrak{s}_{\mathcal{D}'}$ such that σ'_1 is obtained by applying the same cut-elimination step to r.

A mces $\mathfrak{s}$ is **bottom-up** if in each $\sigma \in \mathfrak{s}$, each derivation σ_{i+1} is obtained applying a cut-elimination step to a bottom-most reducible cut-rule in σ_i.

The non-deterministic function over open derivations $f_{\mathfrak{s}}$ (with $\mathfrak{s}$ a bottom-up mces) is defined by letting

$$f_\sigma(\mathcal{D}) = \bigsqcup_{i=0}^{\ell(\sigma)} \mathrm{cf}(\sigma_i) \quad \text{and} \quad f_{\mathfrak{s}}(\mathcal{D}) = \bigcup_{\sigma \in \mathfrak{s}_{\mathcal{D}}} f_\sigma(\mathcal{D}) \quad .$$

Remark 16. The requirement for strategies of being memory-less (together with maximal and total) ensures that the suffix of any cut-elimination view in $\mathfrak{s}$ is also a cut-elimination view in $\mathfrak{s}$, that is, for all $\sigma \in \mathfrak{s}$ and $i \in \mathbb{N}$ there is a view $\sigma^{i+} \in \mathfrak{s}$ such that $\sigma_{i+k} = \sigma^{i+}_k$ for all $k \in \mathbb{N}$.

The coherence condition over approximations guarantees that from every view in a strategy starting from $\mathcal{D}$ we can extract views starting from any approximation of $\mathcal{D}$, and conversely that each view starting from $\mathcal{D}$ can be seen as an upper bound of all views starting from a proper approximation of $\mathcal{D}$. This condition is not present in [2], where the cut-elimination is confluent, but it is required here to guarantee that limits can be defined. In fact, none of the other conditions on strategies ensures any sort of *completeness* – intended as the property that a strategy takes into account all sequences of derivation obtained by applying all possible cut-elimination steps – nor a weaker of this property demanding that if $\mathcal{D}$ reduces to $\mathcal{D}'$ and $\mathcal{D}'$ via two cut-elimination steps applied to a same cut-rule of $\mathcal{D}$, then we must have (at least) a σ' and a σ' in $\mathfrak{s}_{\mathcal{D}}$ such that $\sigma'_1 = \mathcal{D}'$ and $\sigma'_1 = \mathcal{D}'$. For example, we could have a derivation $\mathcal{D}$ as

shown below where $\mathcal{D}$ could, a priori, reduce either to $\mathcal{D}'$ or to $\mathcal{D}'$ by applying a same cut-elimination step to the same cut-rule (i.e., $\mathcal{D}'$ and $\mathcal{D}'$ are a *Lafont pair* [27]), and a strategy $\mathfrak{s}$ containing $\mathcal{D} \cdot \mathcal{D}'$ but not $\mathcal{D} \cdot \mathcal{D}'$.

$$\dfrac{\dfrac{\mathcal{D}' \left[\!\left[\vdash \Gamma \right]\!\right]}{\mathrm{w}\dfrac{}{\vdash \Gamma, \phi}} \quad \dfrac{\mathcal{D}' \left[\!\left[\vdash \Gamma \right]\!\right]}{\mathrm{w}\dfrac{}{\vdash \Gamma, \overline{\phi}}}}{\mathrm{cut}\dfrac{}{\vdash \Gamma}} \qquad \text{with } \mathcal{D}' \text{ and } \mathcal{D}' \text{ cut-free}$$

Proposition 17. *If $\mathfrak{h}$ is a bottom-up mces, then $f_{\mathfrak{h}}$ is Scott-continuous.*

Proof. For each $i \in \mathbb{N}$ and $\sigma^{\mathcal{D}} \in \mathfrak{h}$ with $\mathcal{D} \in$ oLPD, the derivation σ_i is obtained by applying a finite number of cut-elimination steps to the bottom-most cut-rules in $\sigma_0 = \mathcal{D}$. We let $\mathsf{k}_i(\mathcal{D})$ be defined as the greatest approximation of $\mathcal{D}$ containing all nodes of $\mathcal{D}$ which are not above cut-rules (of $\mathcal{D}$) which involved in a cut-elimination step applied to reach σ_i. Then we have $\mathrm{cf}(\sigma_i) = \mathrm{cf}(\sigma_i^{\mathsf{k}_i(\mathcal{D})}) \preceq f_{\sigma^{\mathsf{k}_i(\mathcal{D})}}(\mathsf{k}_i(\mathcal{D}))$ for a suitable $\sigma^{\mathsf{k}_i(\mathcal{D})} \in \mathfrak{h}$ which exists because of the coherence of $\mathfrak{h}$. Thus $f_\sigma(\mathcal{D}) \preceq \bigsqcup_{i\geq 0} f_{\sigma^i}(\mathsf{k}_i(\mathcal{D}))$, henceforth $f_\sigma(\mathcal{D}) = \bigsqcup_{i\geq 0} f_{\sigma^{\mathsf{k}_i(\mathcal{D})}}(\mathsf{k}_i(\mathcal{D}))$. Thanks to the coherence condition, this allows us to conclude because $f_\sigma(\mathcal{D}) \succeq \bigsqcup_{\mathcal{D}' \in \mathcal{K}_{\mathcal{D}}} f_{\sigma'}(\mathcal{D}')$, by definition (assuming $\sigma' \in \mathfrak{h}_{\mathcal{D}'}$), and $f_{\mathfrak{h}}(\mathcal{D}) = \bigsqcup_{\mathcal{D}' \in \mathcal{K}_{\mathcal{D}}} f(\mathcal{D}')$. □

Theorem 18. *The rule* cut *is admissible in* pLPD.

Proof. Since we can always define a bottom-up mces, to conclude it suffices to prove that if $\mathcal{D} \in$ pLPD, then $f_\sigma(\mathcal{D})$ is a well-defined progressing derivation for any $\sigma \in \mathfrak{h}_{\mathcal{D}}$. For this purpose, we prove that each branch $\mathcal{B}$ in $f_\sigma(\mathcal{D})$ does not end with an open premise and, if infinite, it contains a progressing thread.

If there is a $k \in \mathbb{N}$ such that $\mathcal{B}$ occurs in σ_k (therefore in all σ_i with $i \geq k$), then, either $\mathcal{B}$ is finite, ending with a ax-rule (since $\mathcal{D}$ has no open branches), or infinite. Moreover, $\mathcal{B}$ contains a progressing thread since progressing threads are preserved by (finitely many) cut-elimination steps.

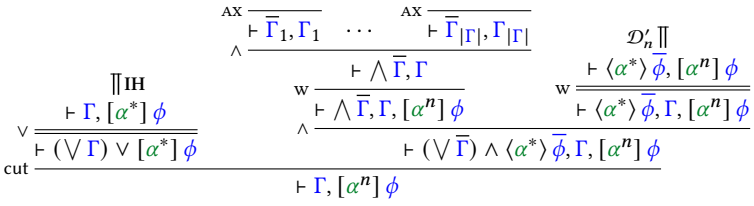
Otherwise, we define the *open branch* $\mathcal{B}_i$ as the set of nodes in σ_i containing the nodes in $\mathcal{B}$ (seen as set of nodes) strictly below any cut-rule in σ_i. Note that each $\mathcal{B}_i$ can be seen as a finite subset of nodes in $\mathcal{B}$, and that the sequence $(\mathcal{B}_i)_{i\geq 0}$ is well-ordered with supremum $\mathcal{B}$ by definition. The existence of a progressing thread ρ in the infinite (therefore not ending with an open premise) branch $\mathcal{B}$ is proven by remarking that each cut-elimination steps either do not interact with κ_α-rules, or it is a cut-elimination step of the form κ_α-vs-κ_α. In the latter case, the sequence of nodes in $\mathcal{B}_{i+1}$ strictly longer than $\mathcal{B}_i$ and it contains an additional 'progressing point' of the progressive thread of $\mathcal{B}$ with respect to $\mathcal{B}_i$, that is, the number of κ_α-rules in $\mathcal{B}_{i+1}$ with principal formula the one of the progressive thread of $\mathcal{B}$ is one more than the one in $\mathcal{B}_i$. This ensures progressiveness of $\mathcal{B}$, which is the supremum of $(\mathcal{B}_i)_{i\geq 0}$.

Details can easily obtained by adapting the technique developed in [3], where the modality ! (resp. ?) can be considered as a box (resp. a diamond), and the rule c!p can plays the same role of κ_α (and $[*]$) to define the progressing condition. □

3.4 Soundness and Completeness of pLPD

We conclude by proving soundness and completeness of pLPD with respect to PDL relying on the cut-elimination result. The omitted details of the proofs provided in **??**.

Lemma 19. *If $\vdash_{\mathsf{pLPD}} \Gamma, [\alpha^*]\,\phi$, then $\vdash_{\mathsf{pLPD}} \Gamma, [\alpha^n]\,\phi$ for any $n \in \mathbb{N}$.*



where

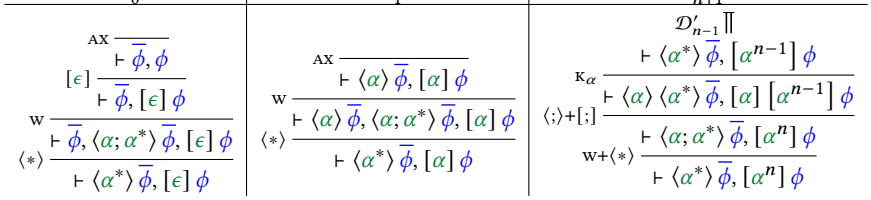


Fig. 8. Derivations proving that if $\vdash_{\mathsf{pLPD}} \Gamma, [\alpha^*]\,\phi$, then $\vdash_{\mathsf{pLPD}} \Gamma, [\alpha^n]\,\phi$ for any $n \in \mathbb{N}$.

Proof. It follows from Theorem 18 since we have a derivation $\mathcal{D}_n$ defined as in Figure 8. □

To prove soundness and completeness of pLPD with respect to pLPD, we use the notion of **Fischer-Ladner closure** of a formula ϕ. This is defined as the smallest set of formulas $\mathsf{FL}(\phi)$ containing ϕ and such that the conditions in Equation (5) hold.

$$
\begin{array}{ll}
\mathsf{FL}(p) = p & \mathsf{FL}(\overline{p}) = \overline{p} \\
\mathsf{FL}(\phi \wedge \psi) \supset (\mathsf{FL}(\phi) \cup \mathsf{FL}(\psi)) & \mathsf{FL}(\phi \vee \psi) \supset (\mathsf{FL}(\phi) \cup \mathsf{FL}(\psi)) \\
\mathsf{FL}([\alpha]\,\phi) \supset \mathsf{FL}(\phi) & \mathsf{FL}(\langle\alpha\rangle\,\phi) \supset \mathsf{FL}(\phi) \\
\mathsf{FL}([\phi?]\,\psi) \supset (\mathsf{FL}(\phi) \cup \mathsf{FL}(\psi)) & \mathsf{FL}(\langle\phi?\rangle\,\psi) \supset (\mathsf{FL}(\phi) \cup \mathsf{FL}(\psi)) \\
\mathsf{FL}([\alpha \oplus \beta]\,\phi) \supset (\mathsf{FL}([\alpha]\,\phi) \cup \mathsf{FL}([\beta]\,\phi)) & \mathsf{FL}(\langle\alpha \oplus \beta\rangle\,\phi) \supset (\mathsf{FL}(\langle\alpha\rangle\,\phi) \cup \mathsf{FL}(\langle\beta\rangle\,\phi)) \\
\mathsf{FL}([\alpha;\beta]\,\phi) \supset (\mathsf{FL}([\alpha]\,[\beta]\,\phi)) & \mathsf{FL}(\langle\alpha;\beta\rangle\,\phi) \supset (\mathsf{FL}(\langle\alpha\rangle\,\langle\beta\rangle\,\phi)) \\
\mathsf{FL}([\alpha^*]\,\phi) \supset (\mathsf{FL}([\alpha]\,[\alpha^*]\,\phi)) & \mathsf{FL}(\langle\alpha^*\rangle\,\phi) \supset (\mathsf{FL}(\langle\alpha\rangle\,\langle\alpha^*\rangle\,\phi))
\end{array}
\tag{5}
$$

If Γ is a sequent, then $\mathsf{FL}(\Gamma) \coloneqq \bigcup_{\phi\in\Gamma} \mathsf{FL}(\phi)$.

Remark 20 (Fisher-Ladner Analyticity)**.** By rules inspection, each sequent occurring in a derivation $\mathcal{D} \in \mathsf{LPD}$ with conclusion Γ is a subset of $\mathsf{FL}(\Gamma) = \bigcup_{\phi\in\Gamma} \mathsf{FL}(\phi)$. More precisely, if Δ is a premise of a rule with conclusion Γ, then $\mathsf{FL}(\Delta) \subseteq \mathsf{FL}(\Gamma)$.

Theorem 21. *Let Γ be a sequent. Then $\vdash_{\mathsf{pLPD}} \Gamma$ iff $\vdash_{\mathsf{PDL}} \Gamma$.*

Proof. Completeness of pLPD with respect to PDL is a consequence of Lemma 9 and cut-admissibility (Theorem 18).

To prove that pLPD is sound for PDL, we first observe that each rule in pLPD is locally sound, that is, if each premise of a rule is valid in PDL, then its conclusion is. As a consequence, if a sequent Γ is a conclusion of a derivation $\mathcal{D}$ in pLPD is not valid in PDL, we deduce that $\mathcal{D}$ must be infinite. Then, by Lemma 8, $\mathcal{D}$ can be written as a finite open derivation with open premises of the form $\Gamma, [\alpha^*]\,\phi$ which are derivable in pLPD. We deduce that if the conclusion of $\mathcal{D}$ is not valid in PDL, then there must exist a sequent of the form $\Gamma, [\alpha^*]\,\phi$ which is derivable in pLPD (via an infinite

derivation) but whose conclusion is not valid in PDL. Therefore to prove soundness it suffices to prove the the statement for infinite derivations in pLPD with conclusion a sequent of the form $\Gamma, [\alpha^*]\,\phi$.

Let $\Gamma, [\alpha^*]\,\phi$ and such that $\vdash_{\mathsf{pLPD}} \Gamma, [\alpha^*]\,\phi$ but $\nvdash_{\mathsf{PDL}} \Gamma, [\alpha^*]\,\phi$. We can assume $\Gamma, [\alpha^*]\,\phi$ to be minimal with respect to the well-founded partial order over sequents defined by the inclusion of the Fisher-Lander closures of the sequents (see Remark 20). By Lemma 4 there is a $n \in \mathbb{N}$ minimal such that $\nvdash_{\mathsf{PDL}} \Gamma, [\alpha^n]\,\phi$, while by Lemma 19 we have that if $\vdash_{\mathsf{pLPD}} \Gamma, [\alpha^*]\,\phi$; then we must have that $\vdash_{\mathsf{pLPD}} \Gamma, [\alpha^*]\,\phi$ but $\nvdash_{\mathsf{PDL}} \Gamma, [\alpha^*]\,\phi$. This would only be possible if $\Gamma, [\alpha^*]\,\phi$ is not minimal since all rules in pLPD are analytic as intended in Remark 20. Absurd. □

4 EMBEDDING OPERATIONAL SEMANTICS IN PROPOSITIONAL DYNAMIC LOGIC

We consider a new set of formulas defined as PDL-formulas where the programs in $\mathbb{P}$ are provided with an *operational semantics*.

Definition 22. Let $\mathbb{P}$ be a set of programs possibly containing a set of **tests** $\mathbb{T}$. An **operational semantics** for a set of programs $\mathbb{P}$ with labels in $\mathbb{L}$[8] is a labeled binary relation between programs $\mathcal{O} \subset \mathbb{P} \times \mathbb{L} \times \mathbb{P}$ whose elements are called (labeled) transitions and may be written as $\alpha \xrightarrow{\beta} \gamma$ instead of (α, β, γ). We assume $\mathbb{P}$ contains a distinguished program ϵ (called **terminated program**) such that $(\epsilon, \beta, \gamma) \notin \mathcal{O}$ for any $\beta \in \mathbb{L}$ and $\gamma \in \mathbb{P}$. An operational semantics is **finitely branching** if the set of $\{(\beta, \gamma) \mid \alpha \xrightarrow{\beta} \gamma\}$ is finite for all $\alpha \in \mathbb{P}$ [4, Def. 2.2].

A **trace** is a sequential composition of labels[9]. A trace α is **valid** for a program β if there is a trace α' such that $\alpha = a; \alpha'$ which is valid for a β' such that $\beta \xrightarrow{a} \beta'$. We denote by $\mathsf{Tr}(\alpha)$ the set of traces valid for α. Two programs are **trace equivalent** (denoted $\alpha \sim_{\mathsf{Tr}} \beta$) if $\mathsf{Tr}(\alpha) = \mathsf{Tr}(\beta)$.

Definition 23. The **(operational) Fisher-Ladner closure** of a formula ϕ is defined as the smallest set of formulas closed with respect to conditions given for the Fisher-Ladner closure in Equation (5) plus the following:

$$\mathsf{FL}([\alpha]\,\phi) \supseteq (\mathsf{FL}([\beta]\,[\gamma]\,\phi)) \quad \text{for all } \beta \text{ such that } \alpha \xrightarrow{\beta} \gamma$$
$$\mathsf{FL}(\langle\alpha\rangle\,\phi) \supseteq (\mathsf{FL}(\langle\beta\rangle\,\langle\gamma\rangle\,\phi)) \quad \text{for all } \beta \text{ such that } \alpha \xrightarrow{\beta} \gamma$$

Definition 24 (Dynamic Operational Logic)**.** Let $\mathcal{O}$ be an finitely branching operational semantics for a set of programs $\mathbb{P}$. The set $\mathcal{F}_{\mathbb{P}}$ of $\mathbb{P}$**-formulas** (or simply **formulas** when clear) is defined by the same grammar in Figure 1 by letting $\mathbb{I} = \mathbb{L} \cup (\mathbb{P} \setminus \mathbb{T})$ and by assuming that the set of propositional atoms $\mathcal{A}$ is such that $\mathbb{T} \subseteq \{\phi? \mid \phi \in \mathcal{F}_{\mathbb{P}}\}$.[10]

We write $\vdash_{\mathsf{OPDL}} \phi$ if ϕ is derivable using rules and axioms of PDL (Figure 2) plus the following axiom

$$\mathbf{A}_{\mathcal{O}} \;:\; [\alpha]\,\phi \Leftrightarrow \left(\bigwedge_{\alpha \xrightarrow{\beta} \gamma} [\beta]\,[\gamma]\,\phi \right) \tag{6}$$

The **operational propositional dynamic logic** of $\mathcal{O}$ (denoted $\mathcal{O}$PDL, or simply OPDL if $\mathcal{O}$ is clear) is the set of formulas such that $\vdash_{\mathsf{OPDL}} \phi$.

Remark 25. In this paper we consider finitely branching operational semantics only. At the syntactical level, this guarantees that the axiom $\mathbf{A}_{\mathcal{O}}$ is a finite formula, at the semantical level, that the set of reachable states from a state is

[8] Unless specified otherwise, we can assume $\mathbb{L} \subseteq \mathbb{P}$.
[9] We may use the color green for traces whenever we want to distinguish them from general programs (in red).
[10] The condition on the set of propositional atoms ensures us that any evaluation of a conditional or a guard required in the operational semantics can be evaluated in the logic itself without the need of an external language. See Section 5.2.

finite. Note that this condition does not guarantee the so-called *small world property* for models of OPDL, nor that the Fisher-Ladner closure of a sequent is finite.

Example 26. The standard PDL can be recovered as the OPDL where the set of programs $\mathbb{P}$ is the set of regular programs generated from a set on instructions $\mathbb{I}$ and a set of tests $\mathbb{T}$ (i.e. a Kleene algebra with tests) provided with the following operational semantics $\mathcal{K}$:

$$\begin{array}{lll} a;\beta \xrightarrow{a} \beta & \alpha \oplus \beta \xrightarrow{\epsilon} \alpha & \alpha \oplus \beta \xrightarrow{\epsilon} \beta \\ \phi?;\beta \xrightarrow{\phi?} \beta & \alpha^* \xrightarrow{\epsilon} \epsilon & \alpha^* \xrightarrow{\epsilon} \alpha;\alpha^* \end{array} \tag{7}$$

Note that if we identify the sequential composition, choice and iteration in $\mathbb{P}$ with the analogous operations we use to generate regular languages, then each instance of the axiom $\mathbf{A}_{\mathcal{O}}$ in $\mathcal{K}$PDL is derivable using the axioms $\mathbf{A}_{;}$, $\mathbf{A}_{*}$ and $\mathbf{A}_{\oplus}$.

Example 27. In [8, 51] the authors study different versions of PDL for fragments of CCS* [11], the restriction of Milner's CCS [56] that replaces recursion with iteration à la Kleene star. These versions can be recovered in OPDL by instantiating it with the operational semantics considered in those papers. In particular, in [51] the parallel constructor (see Figure 12) is restricted to be at the root of the syntactic tree of the terms representing processes, while in [8] the parallel is entirely removed.

Example 28. The logic πDL in [9] is the OPDL of the unconventional version of the π-calculus where the *replication* constructor (usually denoted by !) is replaced by the *iteration* (denoted by $*$). That is, the version of the π-calculus they consider stays at the standard π-calculus as CCS* stays at the standard CCS.

Example 29. The logic APDL in [31] (based on an idea proposed in [66]) reminds a test-free fragment of the OPDL where a program α^i_j is a path over a finite state automaton from a state i to a state j. However, the induction axiom in APDL, which can be reformulated in the following way

$$\mathbf{A_5}: \quad \left(\bigwedge_{\alpha^i_j \xrightarrow{\alpha^i_k} \alpha^k_l \text{ transition}} [\alpha^i_k] \left(\phi_k \Rightarrow [\alpha^k_l] \phi_l \right) \right) \Rightarrow \left(\phi_i \Rightarrow [\alpha^i_j] \phi_j \right)$$

would be derivable in a OPDL with programs defined as paths over a finite state automaton only if $\phi_i = \phi_j = \phi_k = \phi_l$. Note that $\mathbf{A_5}$ require that α^k_l is a transition in the automaton but there is no such condition on α^i_k (nor that α^i_k is an elementary path). This may be problematic in an automaton containing loops.

4.1 Soundness and Completeness of OPDL

We conclude this section by proving soundness and completeness of the axiomaxization of OPDL with respect to its semantics. We then consider define a sequent system extending LPD and we prove its soundness and completeness with respect to OPDL by lifting the method used in the previous section.

Definition 30. A **model (for** OPDL**)** is a Kripke frame defined similarly to Definition 2 with the following differences:

- the meaning of each label $\mathbb{L}$ is defined by an accessibility relation $\mathfrak{m}(a) \subseteq W \times W$;
- the meaning of a (non-atomic) program $\alpha \in \mathbb{P}$ is defined as

$$\mathfrak{m}(\alpha) = \bigcup_{\alpha \xrightarrow{\beta} \gamma} \mathfrak{m}(\beta;\gamma) \ . \tag{8}$$

The satisfability relation $\models$ is defined analogously to Definition 2 by considering OPDL models.

In order to prove soundness and completeness, we recall that a formula ϕ is said **refutable** if $\vdash \overline{\phi}$, and **consistent** if not refutable, that is, $\nvdash \overline{\phi}$. Similarly, a (possibly infinite) set of formulas A is *consistent* if there is no refutable finite subset $\{\phi_1, \dots, \phi_n\}$ of A, that is, if $\nvdash \overline{\phi_1 \wedge \dots \wedge \phi_n}$.

Theorem 31. *Let ϕ be a formula. Then $\vdash_{\mathsf{OPDL}} \phi$ iff $\models_{\mathsf{OPDL}} \phi$.*

Proof. Soundness of $\vdash_{\mathsf{OPDL}}$ with respect to $\models_{\mathsf{OPDL}}$ follows by definition of models. To prove the completeness, we adapt the proof of completeness for PDL in [45]. In particular, proving completeness is equivalent to prove that if ϕ is consistent, then $\not\models \overline{\phi}$. By definition $\not\models \overline{\phi}$ holds iff there is a model $\mathfrak{M}$ and a world w of $\mathfrak{M}$ such that $\mathfrak{M}, w \not\models \overline{\phi}$, therefore $\mathfrak{M}, w \models \phi$. For each formula ϕ construct such a model $\mathfrak{M}_\phi$ as follows:

- the model $\mathfrak{M}_\phi$ has a world w_A for each maximal consistent sets of formulas A such that, for each $\psi \in \mathsf{FL}(\phi)$, either $\psi \in A$ or $\overline{\psi} \in A$;
- for each $a \in \mathbb{L}$ we have $(w_A, w_B) \in \mathfrak{m}(a)$ iff $A \cup \langle a \rangle B$ is a consistent set of formulas.

In $\mathfrak{M}_\phi$, the following properties hold:

- if $A \cup \langle \alpha \rangle B$ is consistent, then $(w_A, w_B) \in \mathfrak{m}(\alpha)$;
- if $\langle \alpha \rangle \psi \in \mathsf{FL}(\phi)$ and $w_A \in \mathfrak{m}(\langle \alpha \rangle \psi)$, then there is a w_B such that $(w_A, w_B) \in \mathfrak{m}(\alpha)$;
- for any $\psi \in \mathsf{FL}(\phi)$, we have that $w_A \in \mathfrak{m}(\psi)$ iff $\psi \in A$;

We conclude that if ϕ is consistent, then $\mathfrak{m}(\phi) \neq \varnothing$; therefore there is w_A such that $\mathfrak{M}_\phi, w_A \models \phi$. □

Definition 32. We define the sequent system $\mathsf{LOPD} = \mathsf{LPD} \cup \{[O], \langle O \rangle\}$ defined by the rules in Figure 4 plus the two following rules capturing the axiom $\mathbf{A}_O$,

$$[O]\ \frac{\vdash \Gamma, [\beta_1]\,[\gamma_1]\,\phi \quad \cdots \quad \vdash \Gamma, [\beta_n]\,[\gamma_n]\,\phi}{\vdash \Gamma, [\alpha]\,\phi}\,\dagger \qquad \langle O \rangle\ \frac{\vdash \Gamma, \langle \beta_1 \rangle \langle \gamma_1 \rangle \phi, \dots, \langle \beta_n \rangle \langle \gamma_n \rangle \phi}{\vdash \Gamma, \langle \alpha \rangle \phi}\,\dagger \tag{9}$$

where the side condition $\dagger$ requires that the set $\{(\beta_i, \gamma_i) \mid i \in \{1, \dots, n\}\} = \{(\beta, \gamma) \mid \alpha \twoheadrightarrow_{\beta} \gamma\}$ is finite.

The definition of progressive derivation in LOPD is obtained by replacing any occurrence of LPD in Definition 7 with LOPD. We denote by pLOPD the set of progressing derivations in LOPD.

Remark 33. The restriction on the operational semantics discussed in Remark 25 also guarantees that the sequent rules in Equation (9) capturing the axiom $\mathbf{A}_O$ have a finite number of premises (in the case of the rule $[O]$) and finite-sequent premise (in the case of the rule $\langle O \rangle$).

Theorem 34. *The rule* cut *is admissible in* LOPD.

Proof. The proof of cut-elimination for progressing derivations in $\mathsf{LOPD} \cup \{\mathsf{cut}\}$ is similar to the one provided Section 3.3 and it only requires to consider the additional cut-elimination step in Figure 11, which do not affect any of the reasoning on threads which are crucial to guaranteeing the preservation of progressing condition in Section 3.3. □

Theorem 35. *Let Γ be a sequent. Then $\vdash_{\mathsf{pLOPD}} \Gamma$ iff $\vdash_{\mathsf{OPDL}} \Gamma$.*

Proof. We conclude by Theorem 34 since the axiom $\mathbf{O}_{\mathsf{Atom}}$ is derivable in LOPD (see Figure 9). Note that contrary to what happens in PDL, the Fisher-Ladner closure of a formula in OPDL may be infinite. However, the partial order over sequents in Remark 20 used in the proof of Theorem 21 is still well-founded. □

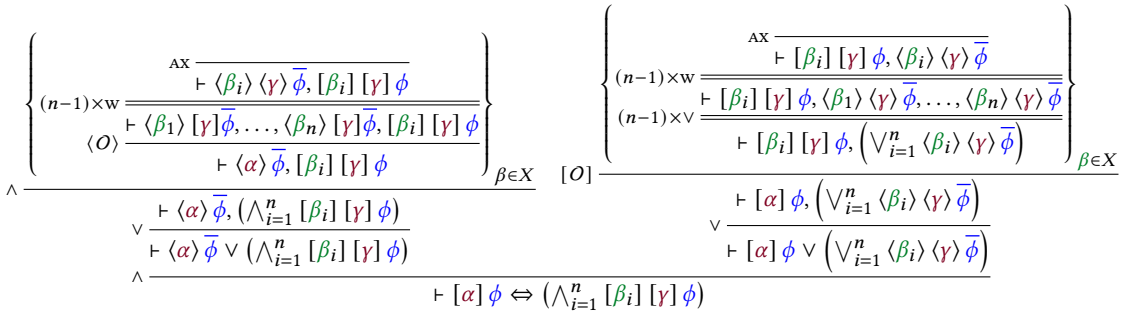


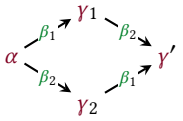
Fig. 9. Derivation in pLOPD of the axiom $\mathbf{O}_{\text{Atom}}$, where we are assuming $\{\beta_1, \ldots, \beta_n\} = \{\beta \in \mathbb{L} \mid \alpha \xrightarrow{\beta} \gamma\}$.

Theorem 36. *Let $\alpha, \beta \in \mathbb{P}$. Then $\vdash_{\text{OPDL}} [\alpha]\,\phi \Leftrightarrow [\beta]\,\phi$ iff $\alpha \sim_{\text{Tr}} \beta$.*

Proof. By definition of $\sim_{\text{Tr}}$ we have that $\alpha \xrightarrow{\alpha_1} \alpha'$ iff $\beta \xrightarrow{\alpha_1} \beta'$. We conclude by induction on size length of the (finite) prefixes of traces in $\text{Tr}(\alpha) = \text{Tr}(\beta)$. □

Remark 37. As written, the rule $[O]$ introduces (bottom-up) a branching during proof search which corresponds to the branching in the label transition system of the program execution. However, it would be desirable to refine such a rule in order to distinguish the branching due to interleaving concurrency from the branching due to internal choices of the system. More precisely, using the terminology from [6, 33, 49], interleaving concurrency is a form of 'don't care' non-determinism, depending on inessential choices introduced by the syntax because of its limitations in handling concurrency, while internal choices cause a 'don't know' non-determinism, requiring us to take into account all possible evolution of the system in order to overcome this lack of knowledge about the next state of a computation. In proof theory, the 'don't care' non-determinism is considered inessential in defining a notion of equivalence for proofs, and it is usually captured by simple independent rule permutations (see Figure 10) while the 'don't know' non-determinism is the responsible of having different proofs.

For this purpose, it would suffice to define a notion of *concurrency* between two elements $(\alpha, \beta_1, \gamma_1)$ and $(\alpha, \beta_2, \gamma_2)$ in O by requiring the existence of a program γ' such that



with $(\gamma_1, \beta_2, \gamma'), (\gamma_2, \beta_1, \gamma') \in O$

and restrict the side condition † of the rules $[O]$ and $\langle O \rangle$ to sets of pairs such that $(\alpha, \beta_1, \gamma_1)$ and $(\alpha, \beta_2, \gamma_2)$ are concurrent for each for each (β_1, γ_1) and (β_2, γ_2) in †. The adequacy result for the calculus with such a restricted rule is proven by showing, modulo rule permutations, that the general and restricted version of the rules are inter-definable.

5 CONCURRENCY THEORY MEETS PDL

In this section we provide two case studies of languages for concurrent systems: Milner's *Calculus of Communicating Systems* (CCS) [56], and a theory of *Choreographic Programming* [59]. The first provides an archetypal case of concurrency

$$\mathsf{r}_2\dfrac{\mathsf{r}_1\dfrac{\vdash \Gamma, \Delta', \Sigma'}{\vdash \Gamma, \Delta', \Sigma}}{\vdash \Gamma, \Delta, \Sigma} \equiv \mathsf{r}_1\dfrac{\mathsf{r}_2\dfrac{\vdash \Gamma, \Delta', \Sigma'}{\vdash \Gamma, \Delta, \Sigma'}}{\vdash \Gamma, \Delta, \Sigma} \qquad \mathsf{r}_2\dfrac{\mathsf{r}_1\dfrac{\Gamma', \phi}{\Gamma, \phi} \quad \Delta, \psi}{\Gamma, \Delta, \Sigma} \equiv \mathsf{r}_1\dfrac{\mathsf{r}_2\dfrac{\Gamma', \phi \quad \Delta, \psi}{\Gamma', \Delta, \Sigma}}{\Gamma, \Delta, \Sigma} \qquad \mathsf{r}_2\dfrac{\mathsf{r}_1\dfrac{\Gamma_1, \phi \quad \Gamma_2, \psi}{\Gamma_1, \Gamma_2, \Delta} \quad \Gamma_3, \chi}{\Gamma_1, \Gamma_2, \Gamma_3, \Delta, \Sigma} \equiv \mathsf{r}_1\dfrac{\Gamma_1, \phi \quad \mathsf{r}_2\dfrac{\Gamma_2, \psi \quad \Gamma_3, \chi}{\Gamma_2, \Gamma_3, \Sigma}}{\Gamma_1, \Gamma_2, \Gamma_3, \Delta, \Sigma}$$

Fig. 10. Independent rule permutations.

$$\mathsf{cut}\dfrac{[O]\dfrac{\vdash \Gamma, [\beta_1]\,[\gamma_1]\,\overline{\phi} \quad \cdots \quad \vdash \Gamma, [\beta_n]\,[\gamma_n]\,\overline{\phi}}{\vdash \Gamma, [\alpha]\,\overline{\phi}} \quad \langle O\rangle\dfrac{\vdash \Gamma, \langle\beta_1\rangle\langle\gamma_1\rangle\,\phi, \ldots, \langle\beta_n\rangle\langle\gamma_n\rangle\,\phi}{\vdash \Gamma, \langle\alpha\rangle\,\phi}}{\vdash \Gamma}$$

$$\wr$$

$$\mathsf{cut}\dfrac{\vdash \Gamma, [\beta_1]\,[\gamma_1]\,\overline{\phi} \qquad \mathsf{cut}\dfrac{\vdash \Gamma, \langle\beta_2\rangle\langle\gamma_2\rangle\,\phi \qquad \mathsf{cut}\dfrac{\vdots \qquad \mathsf{cut}\dfrac{\vdash \Gamma, [\beta_n]\,[\gamma_n]\,\overline{\phi} \quad \vdash \Gamma, \langle\beta_1\rangle\langle\gamma_1\rangle\,\phi, \ldots, \langle\beta_n\rangle\langle\gamma_n\rangle\,\phi}{\vdots}}{\vdash \Gamma, \langle\beta_1\rangle\langle\gamma_1\rangle\,\phi}}{\vdash \Gamma}$$

Fig. 11. Additional cut-elimination step in LOPD.

Processes

$P, Q \coloneqq$	$\mathbf{0}$	terminated process
$\mid$	$\lambda.P$	action prefix
$\mid$	$P \mid Q$	parallel composition
$\mid$	$P + Q$	choice
$\mid$	$P \backslash a$	action restriction
$\mid$	X	process name

Labels

$\lambda \coloneqq$	a	actions ($a \in \mathit{Act}$)
$\mid$	$\overline{a}$	co-actions ($a \in \mathit{Act}$)
$\mid$	τ	silent

Reduction rules

PRE	$\lambda.P$	$\xrightarrow{\lambda}$	P	
PAR$_1$	$P \mid Q$	$\xrightarrow{\lambda}$	$P' \mid Q$	if $P \xrightarrow{\lambda} P'$
PAR$_2$	$P \mid Q$	$\xrightarrow{\lambda}$	$P \mid Q'$	if $Q \xrightarrow{\lambda} Q'$
COM	$P \mid Q$	$\xrightarrow{\tau}$	$P' \mid Q'$	if $P \xrightarrow{a} P'$ and $Q \xrightarrow{\overline{a}} Q'$
SUM$_1$	$P + Q$	$\xrightarrow{\lambda}$	P'	if $P \xrightarrow{\lambda} P'$
SUM$_2$	$P + Q$	$\xrightarrow{\lambda}$	Q'	if $Q \xrightarrow{\lambda} Q'$
RES	$P\backslash a$	$\xrightarrow{\lambda}$	$P'\backslash a$	if $P \xrightarrow{\lambda} P'$ and $\lambda \notin \{a, \overline{a}\}$
REC	X	$\xrightarrow{\lambda}$	P'	if $X \overset{\text{def}}{=} P$ and $P \xrightarrow{\lambda} P'$

Fig. 12. Syntax and operational semantics of CCS.

via parallel composition of processes and the second an illustrative example of concurrency via out-of-order execution of non-interfering actions.

5.1 Concurrency via parallel composition

CCS is a process calculus where processes interact via synchronisations where two parties perform complementary actions (often thought of as sending and receiving). Concurrency is achieved via explicit parallel composition of processes equipped with interleaving semantics.

Processes in CCS (with recursion) are described by the terms generated by the grammar in Figure 12. The definition is parametrised in a countable set Act of symbols denoting the synchronisation **actions** that processes can perform. The set is equipped with an involution $(\bar{\cdot})$ mapping each action its complementary action, or **co-action** for short. The definition is also parametrised in a set of **process definitions** (objects of the form $X \overset{\text{def}}{=} P$) which are used to express infinite behaviours via recursion. The semantics of processes is given as the labelled transition system (or LTS) with processes as states and as transition relation the smallest relation closed under the derivation rules reported in Figure 12. Both syntax and semantics are standard and we briefly discuss them below.

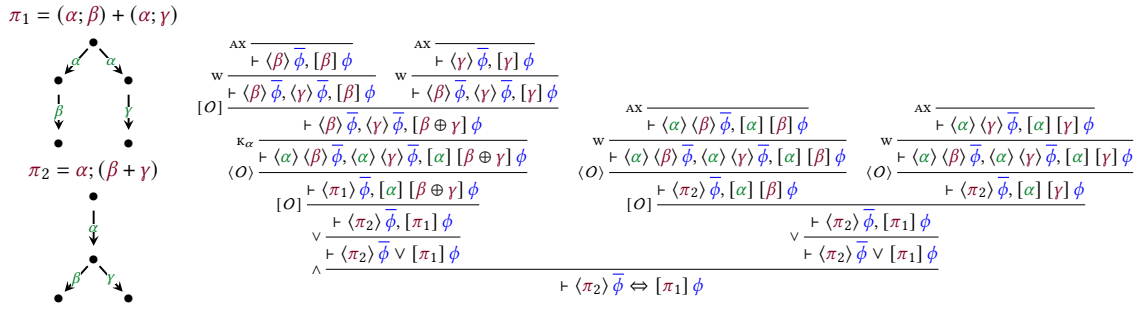


Fig. 13. The derivation proving $\pi_1 \sim_{\mathsf{Tr}} \pi_2$.

The term $\mathbf{0}$ denotes the **terminated process** and has no transitions. A term $\lambda.P$ denotes a process ready to perform the action λ before continuing as P as specified by rule PRE. A term $P \mid Q$ denotes the **parallel** composition of processes P and Q which are executed by interleaving (rules PAR_1 and PAR_2) or synchronising their actions (rule COM). Rule PAR_1 allows $P \mid Q$ to perform a transition where P performs an action (evolving into P') independently from Q and symmetrically for rule PAR_2. Rule COM describes transitions where P and Q synchronise by performing matching actions. To model that synchronisations are binary, transitions derived with this rule are given the label τ which is separate from Act and is traditionally used in process algebras to denote steps that do not interact with the context of a process (hence named **silent** or **internal**). A term $P + Q$ denotes a **choice** between actions performed by P and Q where performing an action from one process disregards the other as specified by rules SUM_1 and SUM_2. A term $P\backslash a$ denotes a process where synchronisations using the action a are **restricted** to its subterm P as prescribed by rule RES which requires λ to be neither a nor $\overline{a}$. A term X denotes the process P associated to the **process name** X by the process definition $X \stackrel{\text{def}}{=} P$ and has the same semantics as P (rule REC). To ensure that the resulting LTS is finitely branching, we assume, as common practice (see, e.g., [4, 28]), that process definitions are *guarded* meaning that every process name occurring in the body of a process definition occurs under an action prefix.

We denote by O_{CCS} the operational semantics over the set of CCS processes (the set of tests is empty) with labels defined as in Figure 12 (by letting $\tau = \epsilon$) defined as in Figure 12. Then, we obtain as an instance of Theorem 36, that logical equivalence in O_{CCS}PDL captures trace equivalence in CCS.

Corollary 38. *Let P and Q be process. Then,*

$$P \sim_{\mathsf{Tr}} Q \text{ iff } \vdash_{O_{\text{CCS}}\text{PDL}} [P]\phi \Leftrightarrow [Q]\phi \text{ for any formula } \phi\,.$$

Example 39. The two processes shown on the left of Figure 13 are a textbook example of the different discriminating power of bisimilarity and trace equivalence: only the first can separate them. The derivation on the right of the figure proves, by Corollary 38, that π_1 and π_2 are indeed trace equivalent.

Although we considered a version of CCS where infinite behaviours are achieved via recursion, instantiating our results to replication ($\text{CCS}^!$) and iteration (CCS^*) is straightforward. In particular, the latter corresponds to the settings considered in [8, 51], as discussed in Example 27, which sits at the bottom of the expressiveness hierarchy formed by these three approaches [11]. Corollary 38 subsumes results from [8, 51] stating that structural congruence ($\equiv$) and strong bisimilarity ($\sim$) are sound w.r.t. logical equivalence. Moreover, our treatment of CCS is standard: parallel composition is

choreographies

$C \coloneqq \mathbf{0}$	inactive process	$\mathsf{pn}(C) = \varnothing$
$\mid I; C$	sequential composition	$\mathsf{pn}(C) = \mathsf{pn}(I) \cup \mathsf{pn}(C)$
$\mid \text{if } \mathsf{p}.b \text{ then } C_1 \text{ else } C_2$	conditional	$\mathsf{pn}(C) = \{\mathsf{p}\} \cup \mathsf{pn}(C_1) \cup \mathsf{pn}(C_2)$
$\mid X$	call	$\mathsf{pn}(C)$ where $X \stackrel{\text{def}}{=} C$

instructions

$I \coloneqq \mathsf{p}.x \coloneqq e$	local assignment	$\mathsf{pn}(I) = \{\mathsf{p}\}$
$\mid \mathsf{p}.e \to \mathsf{q}.x$	communication	$\mathsf{pn}(I) = \{\mathsf{p}, \mathsf{q}\}$
$\mid \mathsf{p} \to \mathsf{q}[\iota]$	selection	$\mathsf{pn}(I) = \{\mathsf{p}, \mathsf{q}\}$
$\mid \mathsf{p} : X$	(call continuation, runtime)	$\mathsf{pn}(I) = \{\mathsf{p}\}$
$\mid \mathsf{p}.b?$	test ($\mathbb{T}$)	$\mathsf{pn}(I) = \{\mathsf{p}\}$
$\mid \mathsf{p}.\overline{b}?$	(negative) test	$\mathsf{pn}(I) = \{\mathsf{p}\}$

Fig. 14. Syntax of choreographies.

$O[C]$	ATOMIC	I	$\xrightarrow{I}$	ϵ	
	COND-THEN	$\text{if } \mathsf{p}.b \text{ then } C_1 \text{ else } C_2$	$\xrightarrow{\mathsf{p}.b?}$	C_1	
	COND-ELSE	$\text{if } \mathsf{p}.b \text{ then } C_1 \text{ else } C_2$	$\xrightarrow{\mathsf{p}.\overline{b}?}$	C_2	
	CALL	X	$\xrightarrow{\mathsf{q}: X}$	$\mathsf{p}_1 : X; \ldots; \mathsf{p}_n : X; C$	if $X \stackrel{\text{def}}{=} C$ and $\mathsf{pn}(X) = \{\mathsf{q}, \mathsf{p}_1, \ldots, \mathsf{p}_n\}$
	I	$I; C$	$\xrightarrow{\mu}$	C	if $I \xrightarrow{\mu} \epsilon$
	DELAY-I	$I; C$	$\xrightarrow{\mu}$	$I; C'$	if $C \xrightarrow{\mu} C'$ and $\mathsf{pn}(I) \cap \mathsf{pn}(\mu) = \varnothing$
	DELAY-COND	$\text{if } \mathsf{p}.b \text{ then } C_1 \text{ else } C_2; C$	$\xrightarrow{\mu}$	$\text{if } \mathsf{p}.b \text{ then } C'_1 \text{ else } C'_2; C$	if $C_i \xrightarrow{\mu} C'_i$ and $\mathsf{p} \notin \mathsf{pn}(\mu)$
$O[\Sigma]$	Σ-ASG	Σ	$\xrightarrow{\mu}$	$\Sigma[\mathsf{p}.x \mapsto v]$	if $\Sigma(\mathsf{p}) \vdash e \downarrow v$ and $\mu = \mathsf{p}.x \coloneqq e$
	Σ-COM	Σ	$\xrightarrow{\mu}$	$\Sigma[\mathsf{q}.x \mapsto v]$	if $\Sigma(\mathsf{p}) \vdash e \downarrow v$ and $\mu = \mathsf{p}.e \to \mathsf{q}.x$
	Σ-POSTEST	Σ	$\xrightarrow{\mathsf{p}.b?}$	Σ	if $\Sigma(\mathsf{p}) \vdash e \downarrow v$ and $v = \text{true}$
	Σ-NEGTEST	Σ	$\xrightarrow{\mathsf{p}.\overline{b}?}$	Σ	if $\Sigma(\mathsf{p}) \vdash e \downarrow v$ and $v \neq \text{true}$
	Σ-SELCALL	Σ	$\xrightarrow{\mu}$	Σ	if $\mu = \mathsf{p} \to \mathsf{q}[\iota]$ or $\mu = \mathsf{q} : X$
$O[\langle C, \Sigma \rangle]$	SC	$\langle C, \Sigma \rangle$	$\xrightarrow{\mu}$	$\langle C', \Sigma' \rangle$	if $C \xrightarrow{\mu} C'$ and $\Sigma \xrightarrow{\mu} \Sigma'$

Fig. 15. Operational semantics of choreographies $O[C]$, for memory storage $O[\Sigma]$, and for stateful choreographies $O[\langle C, \Sigma \rangle]$.

a primitive of the calculus whereas in [8] it is encoded using choices between sequential programs, an approach that is limited to CCS* and results in exponentially larger formulas.

5.2 Concurrency via out-of-order execution

Choreographies, in general, are coordination plans that define the expected collective behaviour of concurrent and distributed systems [59, 61, 74]. In the programming paradigm of Choreographic Programming, choreographies are programs that describe the interaction and local computation of processes participating in the system and that can be compiled to executable implementations for each participant (a procedure called endpoint projection) [59]. The standard way of supporting concurrency in choreographic programming is to execute independent instructions out of order w.r.t. their syntactic position in the program. This is an example of a technique found in many programming languages, compilers, and CPUs, to parallelise the execution of code written as sequential.

We consider a powerful theory of choreographic programming from [59] that includes out-of-order execution, recursion, and stateful local computations. We adopt a presentation of the semantics of choreographic programs that defines separately the dynamics of programs and of memory storage, as done for example in [23]; this separation allows for simpler rules, and for uniform reasoning on both models that abstract over memory or that track precisely its evolution. Except for this presentational difference, our definitions are essentially as for the tail-recursive language given in [59].

Choreographic programs (or just choreographies for short) are described by the terms in Figure 14. Their semantics is given as the LTS induced by the derivation rules in Figure 15. Both definitions are parametrised in a shared language for expressions that are evaluated by processes locally (i.e., without accessing the state of other processes) and which are used to model local computation. We write $\Sigma(\mathsf{p}) \vdash e \downarrow v$ to denote that the expression e evaluates to the value v

given the assignment $\Sigma(\mathsf{p})$ for the variables local to process p. Both definitions are also parametrised in a shared set of **choreography definitions** (objects of the form $X \stackrel{\text{def}}{=} C$ where $\mathsf{pn}(C)$ is not empty) which are used to express infinite behaviours via recursion.

Instructions are performed atomically and describe interactions among processes and, when included in the model, with the memory Σ (via rule SC that interfaces choreographies and memory). An instruction $\mathsf{p} \to \mathsf{q}[\mathrm{L}]$ describes the communication of a constant value L used to communicate a local **selection** from process p to process q (without requiring any interaction with the memory, cf. rule Σ-SELCALL). An instruction $\mathsf{p}.e \to \mathsf{q}.x$ describes the **communication** of a value computed by p evaluating the expression e to q which stores it into its local variable x (cf., rule Σ-COM). An instruction $\mathsf{p}.x \coloneqq e$ represents the **local assignment** at p (cf., rule Σ-ASG). An instruction $\mathsf{p}.b?$ denotes a **test** where p evaluates the condition b proceeding only if successful. Likewise, $\overline{\mathsf{p}.b?}$ represents a negative test. Test instructions are not part of the language [59]; we decided to include them to illustrate the use of tests in OPDL.

The term $\mathbf{0}$ denotes the **terminated choreography** and has no transitions. A term $I;C$ denotes the **sequential composition** of the instruction I (discussed below) and choreography C. The resulting choreography can execute I before continuing as C via rule ATOMIC (similarly action prefixes and rule PRE in CCS) or delay I by executing a transition of C that does not involve any of the processes occurring in I via rule DELAY-I. (Labels are instructions and thus carry all the information required to determine, using the function pn, which processes are involved in a transition.) This relaxed semantics for sequential composition is an instance of the out-of-order execution of instructions and introduces concurrency in the model by allowing the programs to interleave the execution of instructions at distinct processes (while instructions within the same process remain sequential). A term $\mathsf{if}\ \mathsf{p}.b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$ denotes a **conditional** where either C_1 or C_2 is chosen depending on whether the test b performed by process p is successful (rule COND-THEN) or not (rule COND-ELSE). A term X denotes a recursive **call** to the choreography definition $X \stackrel{\text{def}}{=} C$. Its semantics is rather more involved than recursive process calls in CCS because recursive choreography calls involve multiple processes that can join the call concurrently without coordination (this is to capture the decentralised nature of the underlying process model). The standard device used to achieve this behaviour is the (**runtime**) instruction $\mathsf{p} : X$, a syntactic gadget introduced by the first unfolding of a call and used to track processes have yet to join (and prevent erroneous applications of DELAY-I). The resulting semantics is finitely branching, we assume each choreography definition involves finitely many processes (i.e., $\mathsf{pn}(X)$ is finite for any $X \stackrel{\text{def}}{=} C$).

The operational semantics $\mathcal{O}[C]$ for this theory of choreographic programming (abstracting from memory configurations[11]) has choreographies as programs, the instructions of the form $\mathsf{p}.b?$ or $\overline{\mathsf{p}.b?}$ as tests, and the set of instructions. By Theorem 36, logical equivalence in $\mathcal{O}[\mathsf{C}]\mathsf{PDL}$ captures trace equivalence for choreographies.

Corollary 40. $\vdash_{\mathcal{O}[\mathsf{C}]\mathsf{PDL}} [C_1]\,\phi \Leftrightarrow [C_2]\,\phi$ *iff* $C_1 \sim_{\mathsf{Tr}} C_2$.

Likewise, we instantiate OPDL to the theory of choreographic programming with memory updates simply following the steps above while pairing choreographies and memory configurations.

Example 41. Consider the choreographies $(\mathsf{p}.e \to \mathsf{q}.x; \mathsf{r}.e' \to \mathsf{s}.y)$ and $(\mathsf{r}.e' \to \mathsf{s}.y; \mathsf{p}.e \to \mathsf{q}.x)$. The communications are the same save for their syntactic position and, since they involve distinct processes, out-of-order execution (rule DELAY-I) ensures that these can fire concurrently. Indeed, these two choreographies are trace equivalent as shown,

[11] Works on choreographic programming with memory updates usually consider the semantics of choreographies equipped with memory configurations (as in $\mathcal{O}[\langle C, \Sigma\rangle]$ Figure 15). However, the separation adopted in this presentation does not limit the precision of results expected from a theory of choreographies e.g., the correctness of EndPoint Projection: one only needs to ensure that labels used to interface programs and memory are used coherently by the target language. In other words, a presentation like ours treats memory configuration as part of the context of the computation whether programs are expressed as choreographies or their projection.

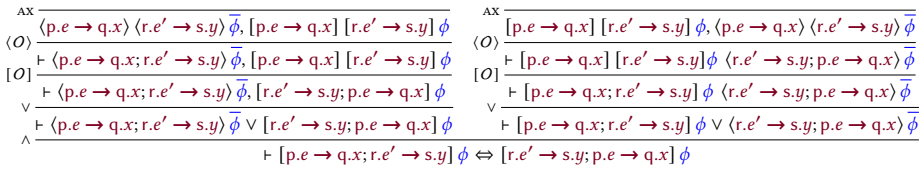


Fig. 16. The derivation proving $(\mathrm{p}.e \rightarrow \mathrm{q}.x; \mathrm{r}.e' \rightarrow \mathrm{s}.y) \sim_{\mathsf{Tr}} (\mathrm{r}.e' \rightarrow \mathrm{s}.y; \mathrm{p}.e \rightarrow \mathrm{q}.x)$ (because of the out-of-order execution).

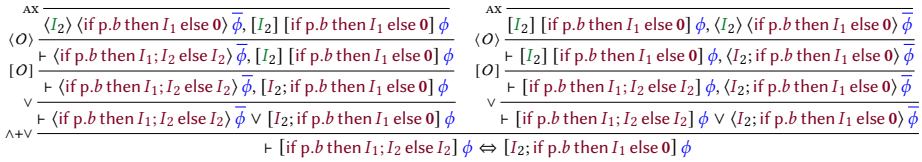


Fig. 17. The derivation proving $(\text{if } \mathrm{p}.b \text{ then } (I_1; I_2) \text{ else } I_2) \sim_{\mathsf{Tr}} (I_2; \text{if } \mathrm{p}.b \text{ then } I_1 \text{ else } \mathbf{0})$ whenever $\mathsf{pn}(I_2) \cap (\{\mathrm{p}\} \cup \mathsf{pn}(I_2)) = \varnothing$.

invoking Corollary 40, by the derivation reported in Figure 16. A similar case that illustrates concurrent conditionals and instructions is shown in Figure 17.

6 CONCLUSION

We have extended PDL by decoupling reasoning on programs from reasoning on traces, bridged by a new axiom that integrates the two aspects. This decoupling allowed us to create an axiom scheme parameterised on the operational semantics of the programs under consideration. The result, OPDL, subsumes a number of previous extensions of PDL by seeing them as particular instantiations of this schema. Furthermore, OPDL can be instantiated for programming languages out of reach of previous approaches, because of problematic standard features such as recursion, interleaving, or out-of-order execution. Thus, we are hopeful that OPDL can be a useful tool for the future study of dynamic logic and formal methods. We mention next a few interesting perspectives.

OPDL, like standard PDL, captures trace equivalence. Trace equivalence can be used to capture finer equivalences by decorating traces with information about choices [40, 73], which for example was used in the context of PDL and a simpler iterative process calculus (CCS*) in [8]. We plan to investigate this in the more general setting of OPDL.

Having captured CCS, a natural next step would be investigating how to capture even richer process calculi. The prime example would be the π-calculus [67], which allows for dynamically creating and transmitting actions. Work on PDL for the π-calculus covers iteration [9], but neither of the standard constructs for infinite behaviours, i.e., recursion and replication. While OPDL can be directly instantiated with the standard π-calculus (retracing the steps for CCS), the resulting notion of equivalence merits attention: the π-calculus has a richer behavioural theory than CCS, which for example introduces the problem of equating traces up to action equivalence.

Likewise, there are numerous choreographic programming languages that would be interesting to study in OPDL, because they pose additional challenges on top of out-of-order execution. Examples include dynamic process spawning [16], parametric recursive procedures [16, 59], and higher-order composition [14, 24, 35, 69]. As we mentioned, a key aspect of choreographic programming is endpoint projection: a mechanical mapping of choreographies into distributed implementations, usually given in terms of a process calculus. Proving that endpoint projection is correct (an operational correspondence result) requires tedious work [17]: OPDL could provide a unifying framework for these

proofs, obtained by instantiating it with the union of the choreographic and target process languages. Adopting this approach might make proofs more robust and reusable.

OPDL inherits the feature from PDL that Hoare clauses $\{\phi\}\alpha\{\psi\}$ can be encoded as $\phi \Rightarrow [\alpha]\,\psi$. All rules in Hoare logic are then derivable. Thus, for example, our instantiation of OPDL with choreographic programming yields a direct generalisation of the previous development of a Hoare logic for choreographies [15], providing a basis for its extension to more sophisticated languages.

Another line of future work is the study of the decision problem in (instantiations of) OPDL. In PDL the so-called *small world model* is constructed using (the finiteness of) the Fisher-Ladner closure of a formula and provides a naive deterministic decidability procedure for the satisfability problem. In OPDL the Fisher-Ladner closure is not guaranteed to be finite, an aspect that depends on the operational semantics under consideration. In general, as shown in [32], any non-regular program add expressiveness power to PDL, and the decision problem for a PDL in which programs may have non-regular set of traces is known to be already Π^1_1-complete [30]. The validity problem for context-free PDL is undecidable because so is the equivalence problem for general context-free languages [39, 46]. This is not surprising, since logical equivalence in PDL captures trace equivalence. In concurrency theory, there is an extensive literature on the relation between the design of concurrent languages and decidability of different program equivalences [5, 11]. The methods studied therein might be useful for exploring decision problems in OPDL, for example by establishing properties on specific operational semantics and how they are defined (rule formats, etc.).

Finally, it would be interesting to model a similar separation between trace reasoning and the operational semantics of programs in algebraic approaches for proving program equivalence. For this we foresee the possibility of defining structures in which an operational semantics is 'nested' inside a Kleene algebra. Intuitively, such structures should be defined as Kleene algebras freely generated by a set of programs $\mathbb{P}$ and a set of atomic actions $\mathbb{I}$ provided with a relation $O \subseteq \mathbb{P} \times \mathbb{I} \times \mathbb{P}$ (in general, a coalgebra $O \subseteq \mathbb{P} \to \mathcal{B}(\mathbb{I} \times \mathbb{P})$ representing the operational semantics of the set of programs.

REFERENCES

[1] Matteo Acclavio and Davide Catta. 2023. Lorenzen-Style Strategies as Proof-Search Strategies. In *Multi-Agent Systems*, Vadim Malvone and Aniello Murano (Eds.). Springer Nature Switzerland, Cham, 150–166.

[2] Matteo Acclavio, Gianluca Curzi, and Giulio Guerrieri. 2023. Infinitary cut-elimination via finite approximations. *CoRR* abs/2308.07789 (2023). https://doi.org/10.48550/ARXIV.2308.07789 arXiv:2308.07789

[3] Matteo Acclavio, Gianluca Curzi, and Giulio Guerrieri. 2024. Infinitary cut-elimination via finite approximations (extended version). arXiv:2308.07789 [cs.LO] https://arxiv.org/abs/2308.07789

[4] Luca Aceto, Wan J. Fokkink, and Chris Verhoef. 2001. Structural Operational Semantics. In *Handbook of Process Algebra*, Jan A. Bergstra, Alban Ponse, and Scott A. Smolka (Eds.). North-Holland / Elsevier, 197–292. https://doi.org/10.1016/B978-044482830-9/50021-7

[5] Luca Aceto, Anna Ingolfsdottir, and Jirí Srba. 2011. *The algorithmics of bisimilarity.* Cambridge University Press, 100–172.

[6] Jean-Marc Andreoli. 1992. Logic programming with focusing proofs in linear logic. *Journal of logic and computation* 2, 3 (1992), 297–347.

[7] David Baelde, Amina Doumane, and Alexis Saurin. 2016. Infinitary Proof Theory: the Multiplicative Additive Case. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France (LIPIcs, Vol. 62)*, Jean-Marc Talbot and Laurent Regnier (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 42:1–42:17. https://doi.org/10.4230/LIPIcs.CSL.2016.42

[8] Mario Benevides. 2017. Bisimilar and logically equivalent programs in PDL with parallel operator. *Theoretical Computer Science* 685 (2017), 23–45. https://doi.org/10.1016/j.tcs.2017.02.037 Logical and Semantic Frameworks with Applications.

[9] Mario R.F. Benevides and L. Menasché Schechter. 2010. A Propositional Dynamic Logic for Concurrent Programs Based on the π-Calculus. *Electronic Notes in Theoretical Computer Science* 262 (2010), 49–64. https://doi.org/10.1016/j.entcs.2010.04.005 Proceedings of the 6th Workshop on Methods for Modalities (M4M-6 2009).

[10] Paul Brunet, Damien Pous, and Georg Struth. 2017. On Decidability of Concurrent Kleene Algebra. In *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany (LIPIcs, Vol. 85)*, Roland Meyer and Uwe Nestmann (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:15. https://doi.org/10.4230/LIPICS.CONCUR.2017.28

[11] Nadia Busi, Maurizio Gabbrielli, and Gianluigi Zavattaro. 2004. Comparing Recursion, Replication, and Iteration in Process Calculi. In *Automata, Languages and Programming*, Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg,

307–319.

[12] Luís Caires and Frank Pfenning. 2010. Session Types as Intuitionistic Linear Propositions. In *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6269)*, Paul Gastin and François Laroussinie (Eds.). Springer, 222–236. https://doi.org/10.1007/978-3-642-15375-4_16

[13] Sjoerd Cranen, Jan Friso Groote, Jeroen J. A. Keiren, Frank P. M. Stappers, Erik P. de Vink, Wieger Wesselink, and Tim A. C. Willemse. 2013. An Overview of the mCRL2 Toolset and Its Recent Advances. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7795)*, Nir Piterman and Scott A. Smolka (Eds.). Springer, 199–213. https://doi.org/10.1007/978-3-642-36742-7_15

[14] Luís Cruz-Filipe, Eva Graversen, Lovro Lugovic, Fabrizio Montesi, and Marco Peressotti. 2023. Modular Compilation for Higher-Order Functional Choreographies. In *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States (LIPIcs, Vol. 263)*, Karim Ali and Guido Salvaneschi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 7:1–7:37. https://doi.org/10.4230/LIPICS.ECOOP.2023.7

[15] Luís Cruz-Filipe, Eva Graversen, Fabrizio Montesi, and Marco Peressotti. 2023. Reasoning About Choreographic Programs. In *Coordination Models and Languages (Lecture Notes in Computer Science, Vol. 13908)*, Sung-Shik Jongmans and Antónia Lopes (Eds.). Springer, 144–162. https://doi.org/10.1007/978-3-031-35361-1_8

[16] Luís Cruz-Filipe and Fabrizio Montesi. 2017. Procedural Choreographic Programming. In *Formal Techniques for Distributed Objects, Components, and Systems - 37th IFIP WG 6.1 International Conference, FORTE 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10321)*, Ahmed Bouajjani and Alexandra Silva (Eds.). Springer, 92–107. https://doi.org/10.1007/978-3-319-60225-7_7

[17] Luís Cruz-Filipe, Fabrizio Montesi, and Marco Peressotti. 2023. A Formal Theory of Choreographic Programming. *Journal of Automated Reasoning* 67, 21 (2023), 1–34. https://doi.org/10.1007/s10817-023-09665-3

[18] Anupam Das and Marianna Girlando. 2022. Cyclic Proofs, Hypersequents, and Transitive Closure Logic. arXiv:2205.08616 [cs.LO]

[19] Anupam Das and Marianna Girlando. 2022. Cyclic Proofs, Hypersequents, and Transitive Closure Logic. In *Automated Reasoning*, Jasmin Blanchette, Laura Kovács, and Dirk Pattinson (Eds.). Springer International Publishing, Cham, 509–528.

[20] Anupam Das and Marianna Girlando. 2023. Cyclic Hypersequent System for Transitive Closure Logic. *Journal of Automated Reasoning* 67, 3 (2023), 27. https://doi.org/10.1007/s10817-023-09675-1

[21] Simon Docherty and Reuben N. S. Rowe. 2019. A Non-wellfounded, Labelled Proof System for Propositional Dynamic Logic. In *Automated Reasoning with Analytic Tableaux and Related Methods*, Serenella Cerrito and Andrei Popescu (Eds.). Springer International Publishing, Cham, 335–352.

[22] Thorsten Ehm, Bernhard Möller, and Georg Struth. 2004. Kleene Modules. In *Relational and Kleene-Algebraic Methods in Computer Science*, Rudolf Berghammer, Bernhard Möller, and Georg Struth (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 112–123.

[23] Saverio Giallorenzo, Fabrizio Montesi, and Maurizio Gabbrielli. 2018. Applied Choreographies. In *Formal Techniques for Distributed Objects, Components, and Systems - 38th IFIP WG 6.1 International Conference, FORTE 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10854)*, Christel Baier and Luís Caires (Eds.). Springer, 21–40. https://doi.org/10.1007/978-3-319-92612-4_2

[24] Saverio Giallorenzo, Fabrizio Montesi, and Marco Peressotti. 2024. Choral: Object-oriented Choreographic Programming. *ACM Trans. Program. Lang. Syst.* 46, 1, Article 1 (Jan. 2024), 59 pages. https://doi.org/10.1145/3632398

[25] Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science* 50, 1 (1987), 1–101. https://doi.org/10.1016/0304-3975(87)90045-4

[26] Jean-Yves Girard. 1998. Light Linear Logic. *Information and Computation* 143, 2 (1998), 175–204. https://doi.org/10.1006/inco.1998.2700

[27] Jean-Yves Girard, Paul Taylor, and Yves Lafont. 1989. *Proofs and types*. Vol. 7. Cambridge university press Cambridge.

[28] Jan Friso Groote and Frits W. Vaandrager. 1992. Structured Operational Semantics and Bisimulation as a Congruence. *Inf. Comput.* 100, 2 (1992), 202–260. https://doi.org/10.1016/0890-5401(92)90013-6

[29] David Harel, Dexter Kozen, and Jerzy Tiuryn. 2002. *Dynamic Logic*. Springer Netherlands, Dordrecht, 99–217. https://doi.org/10.1007/978-94-017-0456-4_2

[30] David Harel, Amir Pnueli, and Jonathan Stavi. 1983. Propositional dynamic logic of nonregular programs. *J. Comput. System Sci.* 26, 2 (1983), 222–243.

[31] D. Harel and R. Sherman. 1985. Propositional dynamic logic of flowcharts. *Information and Control* 64, 1 (1985), 119–135. https://doi.org/10.1016/S0019-9958(85)80047-4 International Conference on Foundations of Computation Theory.

[32] David Harel and Eli Singerman. 1996. More on nonregular PDL: Finite models and Fibonacci-like programs. *information and computation* 128, 2 (1996), 109–118.

[33] David Hemer, Robert Colvin, Ian Hayes, and Paul Strooper. 2002. Don't care non-determinism in logic program refinement. *Electronic Notes in Theoretical Computer Science* 61 (2002), 101–121.

[34] Brian Hill and Francesca Poggiolesi. 2010. A Contraction-free and Cut-free Sequent Calculus for Propositional Dynamic Logic. *Studia Logica* 94, 1 (2010), 47–72. https://doi.org/10.1007/s11225-010-9224-z

[35] Andrew K. Hirsch and Deepak Garg. 2022. Pirouette: higher-order typed functional choreographies. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–27. https://doi.org/10.1145/3498684

[36] Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. 2011. Concurrent Kleene Algebra and its Foundations. *J. Log. Algebraic Methods Program.* 80, 6 (2011), 266–296. https://doi.org/10.1016/J.JLAP.2011.04.005

[37] Tony Hoare, Stephan van Staden, Bernhard Möller, Georg Struth, and Huibiao Zhu. 2016. Developments in concurrent Kleene algebra. *J. Log. Algebraic Methods Program.* 85, 4 (2016), 617–636. https://doi.org/10.1016/J.JLAMP.2015.09.012

[38] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. 2001. Introduction to automata theory, languages, and computation. *Acm Sigact News* 32, 1 (2001), 60–65.

[39] John E Hoperoft and Jeffrey D Ullman. 1979. Introduction to automata theory, languages, and computation. *Addison-Welsey, NY* (1979).

[40] Xiao Jun Chen and Rocco De Nicola. 2001. Algebraic characterizations of trace and decorated trace equivalences over tree-like structures. *Theoretical Computer Science* 254, 1 (2001), 337–361. https://doi.org/10.1016/S0304-3975(99)00300-X

[41] Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. 2019. Kleene Algebra with Observations. In *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands (LIPIcs, Vol. 140)*, Wan J. Fokkink and Rob van Glabbeek (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 41:1–41:16. https://doi.org/10.4230/LIPICS.CONCUR.2019.41

[42] Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. 2020. Concurrent Kleene Algebra with Observations: From Hypotheses to Completeness. In *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12077)*, Jean Goubault-Larrecq and Barbara König (Eds.). Springer, 381–400. https://doi.org/10.1007/978-3-030-45231-5_20

[43] Dexter Kozen. 1996. Kleene algebra with tests and commutativity conditions. In *Tools and Algorithms for the Construction and Analysis of Systems*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 14–33.

[44] Dexter Kozen. 1997. Kleene Algebra with Tests. *ACM Trans. Program. Lang. Syst.* 19, 3 (1997), 427–443. https://doi.org/10.1145/256167.256195

[45] Dexter Kozen and Rohit Parikh. 1981. An elementary proof of the completeness of PDL. *Theoretical Computer Science* 14, 1 (1981), 113–118. https://doi.org/10.1016/0304-3975(81)90019-0

[46] Dexter C Kozen. 2007. *Automata and computability.* Springer Science & Business Media.

[47] Yves Lafont. 2004. Soft linear logic and polynomial time. *Theoretical computer science* 318, 1-2 (2004), 163–180.

[48] Martin Lange. 2003. Games for modal and temporal logics. (2003).

[49] Chuck Liang and Dale Miller. 2021. Focusing Gentzen's LK proof system. (Nov. 2021). https://hal.science/hal-03457379 working paper or preprint.

[50] John W Lloyd. 2012. *Foundations of logic programming.* Springer Science & Business Media.

[51] Alain J. Mayer and Larry J. Stockmeyer. 1996. The complexity of PDL with interleaving. *Theoretical Computer Science* 161, 1 (1996), 109–122. https://doi.org/10.1016/0304-3975(95)00095-X

[52] Damiano Mazza. 2006. Linear logic and polynomial time. *Mathematical Structures in Computer Science* 16, 6 (2006), 947–988. https://doi.org/10.1017/S0960129506005688

[53] Damiano Mazza. 2015. Simple Parsimonious Types and Logarithmic Space. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015 (LIPIcs, Vol. 41)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 24–40. https://doi.org/10.4230/LIPIcs.CSL.2015.24

[54] Damiano Mazza and Kazushige Terui. 2015. Parsimonious Types and Non-uniform Computation. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9135)*, Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann (Eds.). Springer, 350–361. https://doi.org/10.1007/978-3-662-47666-6_28

[55] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. 1991. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic* 51, 1 (1991), 125–157. https://doi.org/10.1016/0168-0072(91)90068-W

[56] Robin Milner. 1980. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science, Vol. 92. Springer. https://doi.org/10.1007/3-540-10235-3

[57] Robin Milner, Joachim Parrow, and David Walker. 1992. A calculus of mobile processes, I. *Information and Computation* 100, 1 (1992), 1–40. https://doi.org/10.1016/0890-5401(92)90008-4

[58] Fabrizio Montesi. 2013. *Choreographic Programming.* Ph.D. Thesis. IT University of Copenhagen. https://www.fabriziomontesi.com/files/choreographic-programming.pdf.

[59] Fabrizio Montesi. 2023. *Introduction to Choreographies.* Cambridge University Press. https://doi.org/10.1017/9781108981491

[60] Damian Niwiński and Igor Walukiewicz. 1996. Games for the μ-calculus. *Theoretical Computer Science* 163, 1 (1996), 99–116. https://doi.org/10.1016/0304-3975(95)00136-0

[61] Object Management Group. 2011. Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0/.

[62] Michel Parigot. 1992. $\lambda\mu$-Calculus: An algorithmic interpretation of classical natural deduction. In *Logic Programming and Automated Reasoning*, Andrei Voronkov (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 190–201.

[63] David Peleg. 1987. Communication in concurrent dynamic logic. *J. Comput. System Sci.* 35, 1 (1987), 23–58. https://doi.org/10.1016/0022-0000(87)90035-3

[64] David Peleg. 1987. Concurrent dynamic logic. *J. ACM* 34, 2 (apr 1987), 450–479. https://doi.org/10.1145/23005.23008

[65] David Peleg. 1987. Concurrent program schemes and their logics. *Theoretical Computer Science* 55, 1 (1987), 1–45. https://doi.org/10.1016/0304-3975(87)90088-0

[66] V. R. Pratt. 1982. Using graphs to understand PDL. In *Logics of Programs*, Dexter Kozen (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 387–396.

[67] Davide Sangiorgi and David Walker. 2001. *The Pi-Calculus - a theory of mobile processes.* Cambridge University Press.

[68] Todd Schmid, Tobias Kappé, and Alexandra Silva. 2023. A Complete Inference System for Skip-free Guarded Kleene Algebra with Tests. In *Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 13990)*, Thomas Wies (Ed.). Springer, 309–336. https://doi.org/10.1007/978-3-031-30044-8_12

[69] Gan Shen, Shun Kashiwa, and Lindsey Kuper. 2023. HasChor: Functional Choreographic Programming for All (Functional Pearl). *Proc. ACM Program. Lang.* 7, ICFP (2023), 541–565. https://doi.org/10.1145/3607849

[70] Colin Stirling and David Walker. 1991. Local model checking in the modal mu-calculus. *Theoretical Computer Science* 89, 1 (1991), 161–177.

[71] Thomas Studer. 2008. On the Proof Theory of the Modal mu-Calculus. *Studia Logica: An International Journal for Symbolic Logic* 89, 3 (2008), 343–363. http://www.jstor.org/stable/40268983

[72] A. S. Troelstra and H. Schwichtenberg. 2000. *Basic Proof Theory* (2 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9781139168717

[73] R. J. van Glabbeek. 1990. The linear time - branching time spectrum. In *CONCUR '90 Theories of Concurrency: Unification and Extension*, J. C. M. Baeten and J. W. Klop (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 278–297.

[74] W3C. 2004. WS Choreography Description Language. http://www.w3.org/TR/ws-cdl-10/.

[75] Philip Wadler. 2015. Propositions as types. *Commun. ACM* 58, 12 (2015), 75–84. https://doi.org/10.1145/2699407