Denotational Semantics for Probabilistic and Concurrent Programs

NOAM ZILBERSTEIN*, Cornell University, USA
DANIELE GORLA, "Sapienza" Università di Roma, Italy
ALEXANDRA SILVA, Cornell University, USA

We develop a novel denotational model for *probabilistic and concurrent imperative programs*, a class of programs with standard control flow via conditionals and while-loops, as well as probabilistic actions and parallel composition. Whereas semantics for concurrent or randomized programs in isolation is well studied, understanding their combination remains challenging. The crux of the problem is that interactions between control flow, probabilistic actions, and concurrent execution cannot be captured by straightforward generalizations of prior work on pomsets and convex languages, prominent models for those effects, individually. Our work introduces the first denotational model for probabilistic concurrent programs with unbounded looping, a major step towards verification of these programs. We prove that the model has good domain theoretic properties, important for semantics of unbounded loops, as well as two adequacy theorems, showing that the model subsumes typical powerdomain semantics for concurrency and convex powerdomain semantics for probabilistic nondeterminism. We illustrate the reasoning power of the new model with an analysis of the randomized solution to Dijkstra's Dining Philosophers problem, adapted from Lehmann and Rabin.

CCS Concepts: • **Theory of computation → Parallel computing models**; **Denotational semantics**; **Probabilistic computation**.

1 Introduction

Program semantics is a longstanding research area, underpinning many static analysis and formal verification techniques. From simple imperative languages, to higher-order functional ones, to the more recent probabilistic and quantum paradigms, the development of programming languages is often accompanied by a close study of the mathematical objects that capture the meaning of the language's syntactic constructs. The underlying semantic objects not only influence verification techniques, but also how a language can be extended.

Semantics are typically presented in either an *operational* or *denotational* style. Operational semantics give a *local* account of the program behavior by defining the possible atomic steps in each configuration. By contrast, *denotational* models are more global, giving a full view of all the behaviors of a completed program in one of the aforementioned carefully selected semantic domains. That domain may give insights beyond the final memory configuration, such as whether or not the program is guaranteed to terminate, the probability of reaching each end state, or information about other observable effects. The need for well-defined semantics of unbounded loops also motivated the development of *domain theory*, an area of research providing technical tools to represent infinite computations via fixed points [Scott 1970; Scott and Strachey 1971]. With increasingly complex program constructs, the need for more involved domains grew; two prime examples are probabilistic and concurrent programs.

Probabilistic programs have been widely studied—going back to the seminal works of Kozen [1979] and McIver and Morgan [2005]—and their semantic domains require extra algebraic structure to capture the *distribution* of outcomes generated by operations such as sampling and coin flips. The added convex structure brings additional complexity to questions of language extensions and program analysis, but the last decades have brought success stories on taming the complexity

*Also with Stevens Institute of Technology.

Authors' Contact Information: Noam Zilberstein, nzilbers@stevens.edu, Cornell University, USA; Daniele Gorla, gorla@di.uniroma1.it, "Sapienza" Università di Roma, Italy; Alexandra Silva, alexandra.silva@cornell.edu, Cornell University, USA.

$$
\begin{array}{rll}
C ::= & \textbf{skip} & \textit{(No-op)} \\
\mid & a & \textit{(Atomic Actions)} \\
\mid & C_1 ; C_2 & \textit{(Sequential Composition)} \\
\mid & C_1 \mid C_2 & \textit{(Parallel Composition)} \\
\mid & C_1 \,\&\, C_2 & \textit{(Nondeterministic Choice)} \\
\mid & C_1 \oplus_p C_2 & \textit{(Probabilistic Choice)} \\
\mid & \textbf{if } b \textbf{ then } C_1 \textbf{ else } C_2 & \textit{(Guarded Branching)} \\
\mid & \textbf{while } b \textbf{ do } C & \textit{(Looping)}
\end{array}
$$

Fig. 1. Syntax of a concurrent imperative programming language $C \in \mathit{cmd}$.

through program logics and predicate transformer calculi [Barthe et al. 2018; den Hartog 1999; McIver and Morgan 2005; Morgan et al. 1996a; Zilberstein 2025; Zilberstein et al. 2025b, 2026].

Concurrent programs have prominent applications in distributed systems and have long been a target of program analysis, as concurrency bugs are hard to detect and prevent. Many approaches to concurrency semantics have been pursued, some arising from process algebra research and some from more practical hardware considerations (*e.g.,* memory models). In terms of denotational semantics, partially ordered multisets (*pomsets*) [Brookes 2004; Gischer 1988; Grabowski 1981; Pratt 1976] provide a prominent model, expressing causality between actions and parallel branching as a partial order.

However, the situation is more subtle when programs have not only concurrent behavior, but also control-flow-structures—*i.e.,* if-then-else and while-loops. For example, in recent work on Concurrent Kleene Algebra with Tests (CKAT) [Jipsen and Moshier 2016], the proposed model (series-parallel guarded strings) turned out not to adequately capture the behavior of concurrent programs: using CKAT, one can prove that the equation $b \cdot C \cdot \overline{b} \equiv 0$ holds for any test b and program C. In other words, no program can change the outcome of any test or, equivalently (and undesirably), any test is an invariant of any program! This oversight in the interaction between concurrency and tests highlights the need for semantic models that accurately capture the meaning of programs.

Semantics of concurrent and probabilistic programs, separately, are subtle and complex. Unsurprisingly, their combination introduces additional challenges, exacerbated when programs also have control-flow constructs. In this paper, we develop a denotational model for programs that mix probabilistic and concurrent constructs in the presence of while loops and conditionals. We work with a simple concurrent imperative programming language, whose syntax is shown in Figure 1. In addition to the usual control flow operations, the language also includes: parallel execution $C_1 \mid C_2$; nondeterministic branching $C_1 \,\&\, C_2$, where a scheduler chooses whether to execute C_1 or C_2; and probabilistic branching $C_1 \oplus_p C_2$, where C_1 is executed with probability p and C_2 is executed with probability $1 - p$. The actions $a \in \mathit{act}$ can perform basic operations like variable assignments; atomic operations like compare-and-set; and probabilistic operations such as random sampling.

Studying the semantics of the language in Figure 1 is not only interesting from a theoretical point of view; for decades, randomization has been used to enhance the capabilities of concurrent programs [Morris 1978; Rabin 1980, 1982]. For example, Dijkstra's famous *Dining Philosophers Problem*—a distributed synchronization scheme—has no fully distributed deterministic solution [Dijkstra 1971], but it has a simple randomized one. Despite the simplicity of the randomized algorithm, the correctness argument is quite intricate. As Lehmann and Rabin [1981] remarked:

> *"Proofs of correctness for probabilistic distributed systems are extremely slippery; in fact the proof presented here (hopefully correct) is only the last one in a sequence of incorrect proofs."*

Further, their proof was based on an informal understanding of the algorithm, not on a formal model extracted from the syntax of a program. Later efforts based on Markov chains [Hart et al. 1983] also do not formally link to program syntax. Given that bugs often arise when translating algorithms into code [Chandra et al. 2007; Liu et al. 2007], semantic models for reasoning directly about programs provide a higher level of assurance.

Despite the importance of mixing randomization and concurrency, denotational models for probabilistic concurrent programs with unbounded looping did not previously exist. We present the first such model in this paper, taking a major step in the development of verification techniques to analyze behaviors of many concurrent randomized protocols—including the Dining Philosophers—which avoid deadlock with probability 1 (they admit infinite traces where processes never really progress but whose probability converges to 0). We draw insights from prior work on concurrent and probabilistic semantics, but solve several challenges unique to their combination and the interaction with control flow constructs. In a nutshell, the contributions of this paper are:

- In Section 3, we introduce *pomsets with formulae*, our new semantic structure for capturing the meaning of probabilistic concurrent programs.
- In Section 4, we prove that pomsets with formulae have good domain-theoretic properties, meaning that they are suitable for reasoning about infinite computations such as while loops. One key result is an *extension lemma* for lifting monotone operations on finite pomsets to Scott continuous operations on infinite ones.
- We define guarded choice, and sequential and parallel composition operations on pomsets with formulae (Section 5); those operations are then used to define the denotational semantics for the concurrent imperative language from Figure 1, with uninterpreted actions (Section 6.1).
- We define a linearization operation (via the extension lemma), parametric on the computational domain, to convert pomsets into state transformers (Section 6.2). Whereas pomsets capture the parallelism between actions in the program, linearization collapses that parallelism into all the possible outcomes of interleaving the threads in different ways.
- In Section 7, we instantiate our model in the *convex powerdomain* and prove our first adequacy result (Theorem 7.2): the parallel-free fragment of our linearized model is equivalent to prior probabilistic semantic models [He et al. 1997]. We also provide two examples of how our model can be used to discover the probabilistic outcomes of concurrent programs.
- In Section 8, we instantiate our model in a nondeterministic powerdomain, and give our second adequacy result (Theorem 8.2): the non-probabilistic fragment of our model is equivalent to prior pomset models of concurrency.
- As a final, larger case study, in Section 9 we show how pomsets with formulae allow us to reason about the randomized Dining Philosophers; differently from prior work [Lehmann and Rabin 1981], here the protocol is implemented as an actual program with a formal semantics, rather than informal pseudocode.
- The results from Sections 3 to 6 are mechanized in Lean 4 [Moura and Ullrich 2021]. The Lean mechanization is available as an artifact in the ACM Digital Library, and also in an actively maintained Github Repository [Zilberstein 2026a]. A map from the results of the paper to their respective definitions in Lean is available in the `README.md` file of the artifact. The structure of the Lean development follows the paper and was written by hand. Some of the Lemmas in the artifact were proven with the help of Harmonic's Aristotle tool [Achim

et al. 2025]. A full account of the AI use is documented in the git history (see Zilberstein [2026a]); all commits that used AI have Aristotle listed as a co-author.

We discuss related work and conclude in Sections 10 and 11. All proofs are provided in the appendices. An earlier version of this work was presented at the CONCUR 2025 conference [Zilberstein et al. 2025a]. This version extends the conference paper by giving full definitions throughout the paper, generalizing the approach to support the $C_1 \,\&\, C_2$ and $C_1 \oplus_p C_2$ programming constructs, providing more insights on our model, more examples to help motivate our choices, and has an accompanying Lean implementation.

2 An Overview of Probabilistic Concurrency Semantics

There are several challenges and pitfalls when it comes to interpreting programs that are both probabilistic and concurrent. To illustrate the subtleties, consider the following program:

$$C \triangleq x :\approx \mathtt{flip}\left(\tfrac{1}{2}\right); (y := 0 \mid y := 1); \mathtt{if}\ y\ \mathtt{then\ skip\ else}\ x := x + 2 \tag{1}$$

C sequentially composes three sub-programs: first, the result of a fair coin flip is stored in x; then, two concurrent threads race to determine the value of y; and finally, a conditional based on the value of y may result in incrementing the value of x by 2. One can see that after executing C, the value of x can be any integer between 0 and 3, and that value depends both on random sampling and nondeterminism (from the concurrent scheduling). The parity of x is fixed after the coin flip (returning 0 or 1, each with probability $\frac{1}{2}$), but then the scheduler can influence the final value of x by choosing in which order to run the updates to y. Importantly though, x will be even with probability exactly $\frac{1}{2}$, regardless of the behavior of the scheduler.

Any semantic domain for interpreting program (1) must encompass probability distributions of program states, while at the same time accounting for nondeterminism introduced from parallel branching. One might attempt to give semantics to program (1) using a naive composition of distributions and powersets (as monads), but, as it is well known from prior work, such a combination leads to non-compositional semantics [Parlant 2020; Varacca and Winskel 2006; Zwart 2020; Zwart and Marsden 2019]. While there are several domains for combining probability and nondeterminism, we use the *convex powerdomain*, which is quite well-studied in that it is a monad [Jacobs 2008] for sequential composition, it is a directed complete partial order (DCPO) [Keimel and Plotkin 2017; Tix 1999, 2000; Tix et al. 2009] for finding fixed points of iterated computations, and it has well behaved equational laws [Bonchi et al. 2021; Mislove 2000; Mislove et al. 2004].

2.1 The Convex Powerdomain

We now give a basic account of the convex powerdomain $\mathcal{C}$. At its core, the convex powerdomain consists of nonempty sets of probability distributions. The sets represent nondeterministic possibilities, and the distributions represent the random outcomes that can occur for each nondeterministic choice. Additional properties ensure that $\mathcal{C}$ is a monad (and thus has well-behaved sequential composition) and a DCPO (and can thus represent infinitary computations). For more details on the DCPO structure and monad operations, refer to Section 7 and Zilberstein et al. [2025b, §3].

Given a countable set X, a discrete probability distribution on X is a map $\mu \in \mathcal{D}(X) \triangleq X \to [0,1]$ such that $\sum_{x \in X} \mu(x) = 1$. Convex combinations are defined for distributions $(\mu \oplus_p \nu)(x) = p \cdot \mu(x) + (1-p) \cdot \nu(x)$ and sets of distributions $S \oplus_p T \triangleq \{\mu \oplus_p \nu \mid \mu \in S, \nu \in T\}$. A set $S \subseteq \mathcal{D}(X)$ is *convex* if $S = S \oplus_p S$ for all $p \in [0,1]$, *i.e.,* S contains all convex combinations of its elements. Convexity ensures that sequential computations can be defined compositionally. See Varacca and Winskel [2006, §6.5] for a counterexample showing how sequential composition fails without convexity.

The *convex powerdomain* on X consists of nonempty convex sets of distributions[1]:

$$C(X) \triangleq \{S \subseteq \mathcal{D}(X \cup \{\bot\}) \mid S \text{ is nonempty, convex, } \dots\}$$

Above, $\bot$ is a distinguished element, which represents nontermination. The domain C allows us to give semantics to all atomic actions, by using a function of the form[2] $[\![\cdot]\!]_{act} : act \to \mathcal{S} \to C(\mathcal{S})$, where $\mathcal{S} \triangleq var \to val$ is the set of variable valuations. For example, the semantics of the coin flip action $x :\approx \mathbf{flip}\,(p)$, for any $p \in [0,1]$, is given by a set containing one single distribution, where x is set to 1 with probability p and it is set to 0 with probability $1-p$:

$$[\![x :\approx \mathbf{flip}\,(p)]\!]_{act}\,(s) \triangleq \left\{ \begin{bmatrix} s[x := 1] & \mapsto & p \\ s[x := 0] & \mapsto & 1-p \end{bmatrix} \right\}$$

Since $C(\mathcal{S})$ is comprised of *sets of distributions*, it can also represent nondeterministic choice, which we denote by $\&$. Operationally, $S \,\&\, T$ is not simply a choice between S and T, but rather it is a choice of a *distribution* over S and T, corresponding to a scheduler that can use biased coin flips to make decisions [Varacca 2002]:

$$S \,\&\, T \triangleq \bigcup_{p \in [0,1]} S \oplus_p T$$

2.2 The Need for a New Model: Pomsets with Formulae

Our goal is to develop a denotational model to compositionally reason about concurrency in a manner that is compatible with the convex powerdomain. Pomsets are the predominant denotational technique for concurrent programs [Brookes 2004; Gischer 1988; Grabowski 1981; Pratt 1976], however standard pomsets cannot handle the mixture of control flow and probabilistic computation. The key issue is that existing pomset models do not track control flow, and instead rely on the following equation, stating that a command followed by an if-statement can be decomposed into two traces, with the split lifted to the top:

$$C; \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2 \quad \equiv \quad (C; \mathbf{assume}\ b; C_1)\ \&\ (C; \mathbf{assume}\ \neg b; C_2) \tag{2}$$

However, if C performs a probabilistic action, then Equation (2) amounts to reordering probabilistic and nondeterministic effects without obeying the proper distributive law [Mislove 2000].

More concretely, prior denotational models for concurrency $[\![-]\!]_{\mathsf{PL}} : cmd \to \mathcal{P}(pom)$ (given later in Figure 7) map programs to *pomset languages* (sets of pomsets) where each pomset $\boldsymbol{\alpha} \in [\![C]\!]_{\mathsf{PL}}$ corresponds to a particular resolution of the tests in the program. For program (1), the pomset language approach yields the semantics on the left of Figure 2, where the problematic Equation (2) is implicitly applied. Arrows $a_1 \to a_2$ indicate causality, *i.e.,* a_1 must occur before a_2. When there is no arrow between two actions, they can be interleaved in any order, as is the case with $y := 0$ and $y := 1$ in Figure 2. There is a significant problem in $[\![C]\!]_{\mathsf{PL}}$: as we will see soon, there is no straightforward way to piece together the entire distribution of outcomes from the two distinct structures.

Crucially, our new pomset structure tracks the results of control-flow tests (which can rely on randomization). Tests are tracked symbolically by associating a Boolean formula to each node of the pomset to record the resolution of tests needed to reach that point. We call these new structures *pomsets with formulae*, which capture both parallel composition and guarded branching[3].

[1] The actual definition relies on a few more properties, which we report in Section 7 and which are needed to ensure a DCPO structure.

[2] Here and in the rest of the paper, we usually curry function definitions, e.g. by writing $f : A \to B \to C$ instead of $f : A \times B \to C$; moreover, as usual, the arrow type associates at right, so $A \to B \to C$ is a shorthand for $A \to (B \to C)$.

[3] Formulae contain more information than what conflict relations usually provide, *e.g.,* in event structures: a conflict can only exclude execution of some branches, whereas formulae record the entire outcome of tests that lead to every node.

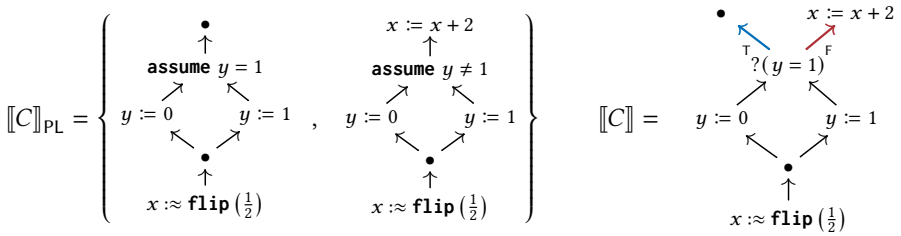


Fig. 2. Pomset language model (left) vs pomset with formulae model (right) for program (1).

As such, one single structure describes all the possible program traces, avoiding the need for a set of structures and simultaneously remaining compositional in the presence of probabilistic actions.

The semantics in our new model then takes the form $[\![-]\!] : \mathit{cmd} \to \mathit{pom}$; the semantics for program (1) is shown on the right of Figure 2. Compared to $[\![C]\!]_{\mathsf{PL}}$, our new model has merged the set of structures into one single structure, where the two opposing tests are replaced by a single node labelled $?(y = 1)$. The outgoing edges are labelled T and F, indicating that they can only be followed if the test passes or fails, respectively.

2.3 Linearization and Infeasibility of Prior Approaches

Now, we may wish to know the possible probabilities for each outcome resulting from running program (1). To that end, in Section 6.2 we develop a *linearization* procedure $\mathcal{L} : \mathit{pom} \to \mathcal{S} \to C(\mathcal{S})$ for interpreting a pomset as a state transformer, mapping a start state to the set of distributions of end states that could be obtained by different interleavings of the concurrent actions. Linearizing $[\![C]\!]$ from Figure 2, we get the following set of distributions:

$$\mathcal{L}([\![C]\!])(s) = \left\{ \begin{bmatrix} s[x := 0, y := 1] & \mapsto & \frac{1}{2}p \\ s[x := 1, y := 1] & \mapsto & \frac{1}{2}q \\ s[x := 2, y := 0] & \mapsto & \frac{1}{2}(1-p) \\ s[x := 3, y := 0] & \mapsto & \frac{1}{2}(1-q) \end{bmatrix} \;\middle|\; 0 \le p, q \le 1 \right\}$$

Linearization provides new insights about the program, which were not obvious in the pomset structure. The scheduler can make y equal to 0 or 1 with any probability, which matches our operational understanding of the program: the scheduler can choose to execute the commands in either the order $y := 0; y := 1$ or $y := 1; y := 0$, based on a coin flip with any bias. In addition, the scheduler's choice occurs *after* $x :\approx \mathbf{flip}\,(\frac{1}{2})$, so the scheduler picks one probability when a 0 is flipped (p) and a second probability when a 1 is flipped (q). However, regardless of the scheduler's choice of p and q, x is even (or odd) with probability exactly $\frac{1}{2}$. This formal semantics matches the intuition for the program, since the parity of x is fixed by the initial coin flip.

We now contrast the semantics above with pomset language semantics to show that the program C cannot be meaningfully interpreted using prior techniques. Recall that in $[\![C]\!]_{\mathsf{PL}}$ each pomset corresponds to a particular resolution of the test, *i.e.*, $y = 1$ and $y \neq 1$. So, in the first pomset we will always have $x \in \{0, 1\}$ whereas in the second pomset $x \in \{2, 3\}$. Let us now try to obtain a meaningful state transformer semantics from C by linearizing each structure and then merging the results. To do so, we assume an evaluation function for boolean tests of the form $[\![\cdot]\!]_{\mathit{test}} : \mathit{test} \to \mathcal{S} \to \{0, 1\}$ and we use the following semantics for **assume**, where failed tests

evaluate to a special **?** symbol, which halts the program execution:

$$[\![\textbf{assume } b]\!]_{act}(s) = \begin{cases} \{[s \mapsto 1]\} & \text{if } [\![b]\!]_{test}(s) = 1 \\ \{[? \mapsto 1]\} & \text{if } [\![b]\!]_{test}(s) = 0 \end{cases}$$

Now, letting $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ be the left and right pomsets in $[\![C]\!]_{\mathsf{PL}}$, we apply linearization to both structures to obtain:

$$\mathcal{L}(\boldsymbol{\alpha}_1)(s) = \left\{ \left[\begin{array}{lcl} s[x := 0, y := 1] & \mapsto & \frac{1}{2}p_1 \\ s[x := 1, y := 1] & \mapsto & \frac{1}{2}q_1 \\ ? & \mapsto & 1 - \frac{1}{2}(p_1 + q_1) \end{array} \right] \;\middle|\; 0 \le p_1, q_1 \le 1 \right\}$$

$$\mathcal{L}(\boldsymbol{\alpha}_2)(s) = \left\{ \left[\begin{array}{lcl} s[x := 2, y := 0] & \mapsto & \frac{1}{2}p_2 \\ s[x := 3, y := 0] & \mapsto & \frac{1}{2}q_2 \\ ? & \mapsto & 1 - \frac{1}{2}(p_2 + q_2) \end{array} \right] \;\middle|\; 0 \le p_2, q_2 \le 1 \right\}$$

Already, we can begin to see a problem; in $\mathcal{L}([\![C]\!])(s)$ the scheduler picks two probabilities, but here the scheduler picks four probabilities, giving it a higher degree of freedom to influence the outcome of the program. We could try to define an operation $\bowtie$ to merge the two semantics. As a first attempt, let us try to generate a new set of distributions by summing the non-**?** probability mass in every pair of distributions from the two sets such that the mass of **?** in one is equal to the non-**?** mass in the other:

$$S \bowtie T \triangleq \{\mu \bowtie \nu \mid \mu \in S, \nu \in T, \mu(?) = 1 - \nu(?)\} \qquad (\mu \bowtie \nu)(s) \triangleq \begin{cases} \mu(s) + \nu(s) & \text{if } s \neq ? \\ 0 & \text{if } s = ? \end{cases}$$

Indeed, this gives us $\mathcal{L}([\![C]\!])(s) \subseteq \mathcal{L}(\boldsymbol{\alpha}_1)(s) \bowtie \mathcal{L}(\boldsymbol{\alpha}_2)(s)$: take any $\mu \in \mathcal{L}([\![C]\!])(s)$, which is generated by fixing some probabilities $0 \le p, q \le 1$. Picking $p_1 = p$, $q_1 = q$, $p_2 = 1 - p$, and $q_2 = 1 - q$, we see that $\mu \in \mathcal{L}(\boldsymbol{\alpha}_1)(s) \bowtie \mathcal{L}(\boldsymbol{\alpha}_2)(s)$. However, $\mathcal{L}(\boldsymbol{\alpha}_1)(s) \bowtie \mathcal{L}(\boldsymbol{\alpha}_2)(s)$ contains extra distributions that are not in $\mathcal{L}([\![C]\!])(s)$, and are not correct outcomes of the program. For example, letting $p_1 = p_2 = 1$ and $q_1 = q_2 = 0$, we get:

$$\left[\begin{array}{ll} s[x := 0, y := 1] & \mapsto \frac{1}{2} \\ s[x := 1, y := 1] & \mapsto 0 \\ s[x := 2, y := 0] & \mapsto \frac{1}{2} \\ s[x := 3, y := 0] & \mapsto 0 \end{array} \right] \in \mathcal{L}(\boldsymbol{\alpha}_1)(s) \bowtie \mathcal{L}(\boldsymbol{\alpha}_2)(s)$$

But the distribution above cannot be an outcome of running the program, since x is even with probability 1, deviating from the expected operational behavior!

So a different implementation of $\bowtie$ is needed, which is more discerning about the compatibility of pairs of distributions from the two sets. But as it stands, there is no information in $\mathcal{L}(\boldsymbol{\alpha}_1)(s)$ and $\mathcal{L}(\boldsymbol{\alpha}_2)(s)$ to indicate compatibility. While we do not claim that it is impossible to recover compatibility information during linearization, our investigation suggests that it would not be straightforward. A proper linearization of the pomset language $[\![C]\!]_{\mathsf{PL}}$ would essentially amount to a lockstep execution of the two pomsets, so that both branches can be taken with the appropriate probabilities. That lockstep execution corresponds to a straightforward linearization of the single pomset with formulae $[\![C]\!]$, where each test already has information about both branches. Pomsets with formulae instead provide the correct semantic structure for this domain. In the remainder of the paper, we develop the formal details of our pomset with formulae model and associated linearization procedure.

3 Semantic Structures

In this section, we define the mathematical structures that will form the basis of our semantic model. As we already discussed in Section 2, these structures must account for both the deterministic and the parallel branching behavior of the language from Figure 1. We begin by recalling basic order theory definitions in Section 3.1 before defining the specifics of the structures that we will use throughout the remainder of the paper in Sections 3.2 and 3.3.

3.1 Preliminaries

We begin by giving basic definitions from order theory and the pomset literature. First, we introduce partial orders, which are used to represent causality in our semantic structures.

Definition 3.1 (Poset). A partially ordered set, or *poset*, is a set equipped with a partial order: that is, a pair $\langle X, \leq \rangle$ consisting of a set X and an order relation $\leq \subseteq X \times X$ which is reflexive, transitive and antisymmetric. When the order is instead irreflexive (*i.e.,* $x \not< x$ for all $x \in X$), the poset is called *strict* and the order relation is denoted by $<$.

Given a strict poset $\langle X, < \rangle$ and $x \in X$, we define the upward and downward closures:

$$x\uparrow \triangleq \{y \in X \mid x < y\} \qquad\qquad x\downarrow \triangleq \{y \in X \mid y < x\};$$

and its set of immediate successors and predecessors as

$$\mathsf{succ}(x) \triangleq \{y \in x\uparrow \mid \nexists z.(x < z < y)\} \qquad\qquad \mathsf{pred}(x) \triangleq \{y \in x\downarrow \mid \nexists z.(y < z < x)\}$$

All the above operations can easily be extended to sets of elements, *e.g.,* $X\uparrow \triangleq \bigcup_{x \in X} x\uparrow$, for any set X. The set of minimal and maximal elements of a poset are given by:

$$\max \triangleq \{x \in X \mid \mathsf{succ}(x) = \emptyset\} \qquad \min \triangleq \{x \in X \mid \mathsf{pred}(x) = \emptyset\}$$

To define our new structures, we need a few more definitions on posets that will play a role in proving the existence of fixed points, essential for giving a semantics to unbounded loops. First, we recall the notion of downward closed sets, which are those subsets of a given poset that contain all the predecessors of all their elements.

Definition 3.2 (Downward Closed Set). Given a poset $\langle X, < \rangle$, a set $Y \subseteq X$ is *downward closed*, written $Y \subseteq_{\downarrow} X$, iff for all $y \in Y$ it holds that $y\downarrow \subseteq Y$.

Second, we need a notion of a poset being finitely preceded.

Definition 3.3 (Finitely Preceded). Given a poset $\langle X, < \rangle$, an element $x \in X$ is *finitely preceded* if there are finitely many elements smaller than it; *i.e.,* $|x\downarrow| < \infty$. The poset $\langle X, < \rangle$ is finitely preceded if every $x \in X$ is finitely preceded.

Third, we define the level of an element x in the poset, $\mathsf{lev}(x)$, which denotes the length of the longest path from a minimal element to x (due to Meyer and de Vink [1989]).

Definition 3.4 (Level). Given a finitely preceded poset $\langle X, < \rangle$, define $\mathsf{lev} \colon X \to \mathbb{N}$:

$$\mathsf{lev}(x) \triangleq \sup \left\{ n \;\middle|\; \exists x_0, \ldots, x_n \in X.\, x_0 \in \min \,\wedge\, x_n = x \,\wedge\, \forall i < n.\, x_{i+1} \in \mathsf{succ}(x_i) \right\}$$

We also define its inverse $\mathsf{lev}^{-1} \colon \mathbb{N} \to \mathcal{P}(X)$:

$$\mathsf{lev}^{-1}(n) \triangleq \{x \in X \mid \mathsf{lev}(x) = n\}$$

Let *nodes* be a countable universe of nodes, that will be denoted by $x, y, z, \ldots$, whereas subsets of *nodes* will be denoted by $N, X, Y, \ldots$. We start by defining the class of Boolean formulae using nodes as the variables:

$$\mathit{form} \ni \psi ::= \mathsf{true} \mid \mathsf{false} \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \neg\psi \mid x$$

Formulae are interpreted by valuations $v \colon \mathit{nodes} \to \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$; the satisfaction relation, written $v \vDash \psi$, is defined in the standard way:

Definition 3.5. The *satisfaction relation* $\vDash \subseteq (\mathbb{B}^{\mathit{nodes}}) \times \mathit{form}$ for Boolean formulae is given by:

$v \vDash \mathsf{true}$	always	$v \vDash \mathsf{false}$	never
$v \vDash \psi_1 \wedge \psi_2$	iff $v \vDash \psi_1$ and $v \vDash \psi_2$	$v \vDash \psi_1 \vee \psi_2$	iff $v \vDash \psi_1$ or $v \vDash \psi_2$
$v \vDash \neg\psi$	iff $v \nvDash \psi$	$v \vDash x$	iff $v(x) = 1$

The variables of a formula ψ, written $\mathsf{vars}(\psi)$, are those nodes that appear in ψ. We write:

- $\mathsf{sat}(\psi)$ iff there exists a valuation v such that $v \vDash \psi$;
- $\psi \Rightarrow \psi'$ iff $v \vDash \psi'$, for every $v \vDash \psi$; and
- $\psi \Leftrightarrow \psi'$ iff $\psi \Rightarrow \psi'$ and $\psi' \Rightarrow \psi$.

We conclude this section with the definition of a *directed complete partial order*, a standard domain theoretic definition, which will allow us to reason about fixed points (more details in Section 4.2).

Definition 3.6 (DCPO). Given a poset $\langle X, \leq \rangle$, a subset $D \subseteq X$ is called *directed* iff it is not empty and, for every two elements $x_1, x_2 \in D$, there exists $x \in D$ such that $x_1, x_2 \leq x$. The poset $\langle X, \leq \rangle$ is a *directed complete partial order* (DCPO) iff, for every directed subset D, $\sup D$ exists; furthermore, $\langle X, \leq \rangle$ is called *pointed* if there exists an element $\bot \in X$ such that $\bot \leq x$ for all $x \in X$.

3.2 Labelled Partial Orders with Formulae

The core of our semantic structure is a new type of labelled partial order, called a *labelled partial order with formulae* (LPOF). In order to formally represent mathematical objects like the ones in Figure 2, each node must have a unique identifier, so that we can describe its relationship to the other nodes. Using the node's label (*e.g.,* the actions and tests in Figure 2) is not sufficient, since each label may appear multiple times. But since the specific identifiers used are ultimately irrelevant, we will later abstract them away by working with equivalence classes of LPOFs.

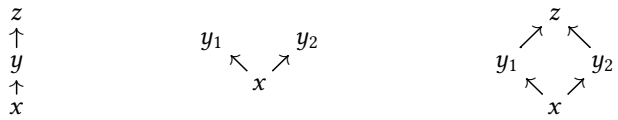
Definition 3.7 (LPOF). Let $\langle L, \leq \rangle$ be a pointed, finitely preceded DCPO with bottom element $\bot$. A *labelled partial order with formulae* (LPOF) over L is a 4-tuple $\alpha = \langle N, <, \lambda, \varphi \rangle$ where:

(1) $N \subseteq \mathit{nodes}$ is a countable set of nodes;
(2) $\langle N, < \rangle$ is a (strict) poset such that:
  (a) it is finitely preceded, that is $|x{\downarrow}| < \infty$, for every $x \in N$;
  (b) every level has a finite number of nodes, *i.e.,* $|\mathsf{lev}^{-1}(n)| < \infty$ for all $n \in \mathbb{N}$; and
  (c) it is single-rooted, that is: $|\min| = 1$.
(3) $\lambda \colon N \to L$ is a labelling function such that $\mathsf{succ}(x) = \emptyset$ whenever $\lambda(x) = \bot$.
(4) $\varphi \colon N \to \mathit{form}$ is a formula function satisfying:
  (a) $\mathsf{sat}(\varphi(x))$ and $\mathsf{vars}(\varphi(x)) \subseteq x{\downarrow}$, for all $x \in N$;
  (b) $\varphi(y) \Rightarrow \varphi(x)$, for all $x < y$.

We denote by $\mathit{lpof}(L)$ the set of all LPOFs over L.

For some LPOF $\alpha = \langle N, <, \lambda, \varphi \rangle$, we use N_α, $<_\alpha$, λ_α, and φ_α to refer to its constituent parts; similarly, we annotate all functions with the LPOF they refer to, *e.g.,* $x{\downarrow}_\alpha$, $\mathsf{succ}_\alpha(x)$, $\min_\alpha$, etc. We now explain the conditions in Definition 3.7.

As is typical in pomset semantics, the partial order in $\langle N, < \rangle$ denotes *causality*: $x < y$ iff x must be scheduled before y. If $x \not< y$ and $y \not< x$, then x and y execute concurrently and can be interleaved in any order. For example, see the diagrams below:



From left to right, the first LPOF represents a sequential (totally ordered) program, where $x < y < z$. The second LPOF has a parallel branch, so that $x < y_1$ and $x < y_2$, but y_1 and y_2 have no relation. This means that y_1 and y_2 execute concurrently: one is not required to come before the other. The last LPOF also has a branch, but the branches eventually merge, as $y_1 < z$ and $y_2 < z$. Here, y_1 and y_2 still execute concurrently, but both must be executed before z.

With this in mind, conditions (2a) and (2b) are standard; in particular, since a node represents an action of a program, having finitely many predecessors ensures that every action may be scheduled in a finite amount of time. Infinitely preceded actions are omitted from the structure, as they will never be executed. Moreover, requiring that every level has a finite number of nodes is a weakening of the usual finite branching property: we allow a node to have infinitely many successors, but they cannot be all at the same level; this suffices to carry out our proofs. We defer the discussion on condition (2c) to Remark 1 later on.

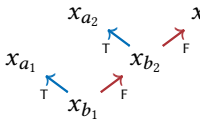
Concerning condition (3), labels correspond to *Boolean tests* and *actions*, performed during a program execution (*e.g.,* Figure 2). Unlike standard pomset models, actions can be probabilistic, and our model is consistent with known sequential probabilistic semantics (Theorem 7.2). For the moment, we leave labels unspecified, only assuming the presence of $\bot$, used to denote a nonterminating computation, which is fundamental in approximating the fixed point semantics of while-loops. Since $\bot$ denotes nontermination, nodes labelled with $\bot$ cannot be predecessors of any node, as the successors of $\bot$ would never be executed.

Condition (4) is about formulae, which are crucial for modeling conditional constructs (if-then-else and while-loops), such as

$$\texttt{if}\ b_1\ \texttt{then}\ a_1\ \texttt{else if}\ b_2\ \texttt{then}\ a_2\ \texttt{else}\ a_3$$

where b_1 and b_2 are tests and a_1, a_2, a_3 are actions. This program corresponds to the LPOF below, where $\lambda(x_\ell) = \ell$ for all $\ell \in \{b_1, b_2, a_1, a_2, a_3\}$, $w \xrightarrow{\mathsf{T}} w_1$ means that $\varphi(w_1) \Leftrightarrow \varphi(w) \wedge w$ and $w \xrightarrow{\mathsf{F}} w_2$ means that $\varphi(w_2) \Leftrightarrow \varphi(w) \wedge \neg w$ (*i.e.,* w is labelled with a test, whose outcome leads to w_1 or w_2, depending on the Boolean outcome depicted on the colored arc):

$$\varphi(x_{a_2}) = \neg x_{b_1} \wedge x_{b_2} \qquad \varphi(x_{a_3}) = \neg x_{b_1} \wedge \neg x_{b_2}$$
$$\varphi(x_{a_1}) = x_{b_1} \qquad \varphi(x_{b_2}) = \neg x_{b_1}$$
$$\varphi(x_{b_1}) = \text{true}$$



(3)

Hence, every node x labelled with a Boolean test yields a (binary) branch and its (two) successors have the same formula as x, with an extra conjunct that is either x or $\neg x$, according to the outcome of the test. With this in mind, Definition 3.7(4) is quite intuitive:

- Requiring $\mathrm{sat}(\varphi(x))$ amounts to expressing the fact that every node x is reachable, *i.e.,* there exists a truth assignment v such that $v \vDash \varphi(x)$. The truth assignment tells, for every node labeled with a Boolean test, whether that test passes or not. In our example, x_{a_3} is reachable when both the tests b_1 and b_2 fail; this is exactly what satisfiability of $\varphi(x_{a_3})$ entails (viz., that both b_1 and b_2 must be false to satisfy $\neg x_{b_1} \wedge \neg x_{b_2}$).

- This intuition justifies the other two requirements of Definition 3.7(4): reachability of a node can only depend on the values of the tests that precede it (and so $\varphi(x)$ can only use nodes that precede x) and, since along a path we shall only add conjuncts, the formula of a higher-level node is stronger than the formulae of all of its predecessors.

3.3 Pomsets with Formulae

Although LPOFs express causality and branching behavior, they are not ideal semantic structures because information about node identifiers is extraneous. We instead work with *partially ordered multisets* (pomsets), which abstract away from the specific names used. Pomsets are equivalence classes of LPOFs. To formalize equivalence classes, we first need to define isomorphism of LPOFs. Intuitively, two LPOFs are isomorphic if they are equal, modulo renaming the node identifiers.

Definition 3.8 (LPOF Isomorphism). Let α and β be LPOFs and $f : N_\alpha \to N_\beta$ be a bijection between their nodes. Define $f(\alpha) = \langle N, <, \lambda, \varphi\rangle$ as

$$N \triangleq \{f(x) \mid x \in N_\alpha\} \qquad < \triangleq \{(f(x), f(y)) \mid x <_\alpha y\} \qquad \lambda \triangleq \lambda_\alpha \circ f^{-1} \qquad \varphi \triangleq f \circ \varphi_\alpha \circ f^{-1}$$

where $f(\psi)$ syntactically renames the variables of ψ in the obvious way. Then, we say that α and β are *isomorphic*, written $\alpha \equiv \beta$, iff there is a bijection $f : N_\alpha \to N_\beta$ such that $f(\alpha) = \beta$.

Given the above definition of isomorphism, we can now define pomsets to be isomorphism classes of LPOFs.

Definition 3.9 (Pomsets with Formulae). Let $[\alpha] \triangleq \{\beta \in lpof(L) \mid \alpha \equiv \beta\}$ be the isomorphism class of α. A *pomset with formulae* (or, simply, pomset) $\boldsymbol{\alpha} \in pom(L)$ is an isomorphism class of LPOFs:

$$pom(L) \triangleq \{[\alpha] \mid \alpha \in lpof(L)\}$$

4 Domain Theoretic Properties

Although Definitions 3.7 and 3.9 do accommodate structures with infinitely many nodes, we have not yet developed the tools needed to construct, manipulate, and compactly represent them. In this section, we develop those tools, based on Domain Theory [Abramsky and Jung 1995; Scott 1970, 1972; Scott and Strachey 1971].

4.1 Orders on LPOFs and Pomsets

The first step is to define an order relation, which describes what it means for one semantic structure to *approximate* another one. For LPOFs, the intuition is that $\alpha \sqsubseteq_{lpof} \beta$ iff β can be obtained by expanding $\bot$ nodes in α into larger, more defined structures; in a sense, we can say that β has *more behaviors* than α. For example, each unfolding of a while-loop produces a larger structure (w.r.t. $\sqsubseteq_{lpof}$), as shown in Section 6.1. Since LPOFs can only be expanded from $\bot$ nodes, the label set L must be pointed (*i.e.,* $\bot \in L$); however, the order on L need not be flat: labels themselves can also become larger when passing from α to β (*e.g.,* if actions are nondeterministic assignments from a given set, this corresponds to enlarging the set of possible choices). The order on labels can be useful to model *invariant sensitive execution*, introduced in Probabilistic Concurrent Outcome Logic [Zilberstein 2026b; Zilberstein et al. 2026]. Notationally, Bot_α denotes the set of nodes labelled with $\bot$, *i.e.,* $\mathrm{Bot}_\alpha \triangleq \{x \in N_\alpha \mid \lambda_\alpha(x) = \bot\}$.

Definition 4.1 (Ordering on LPOFs). We let $\alpha \sqsubseteq_{lpof} \beta$ iff:

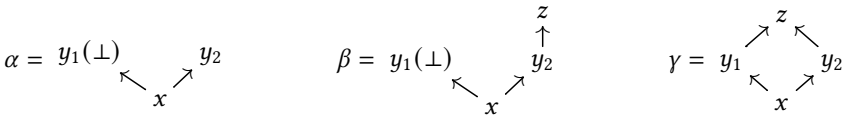
(1) $N_\alpha \subseteq_\downarrow N_\beta$, i.e., N_α is a downward-closed subset of N_β;

(2) $<_\alpha = <_\beta \cap (N_\alpha \times N_\alpha)$;

(3) For all $x \in N_\alpha$:

(a) $\lambda_\alpha(x) \leq \lambda_\beta(x)$;
(b) $\varphi_\alpha(x) = \varphi_\beta(x)$;
(c) $\mathrm{succ}_\alpha(x) = \mathrm{succ}_\beta(x) \setminus \mathrm{Bot}_\alpha{\uparrow_\beta}$.

Example 4.2. Consider the following three LPOFs:



where node y_1 is labelled with $\bot$ in α and β, whereas all other nodes have non-$\bot$ labels. Following the intuition given above, both α and β are attempts to specify that the computation in γ is truncated at y_1. However, only $\alpha \sqsubseteq_{lpof} \gamma$ because, if $\bot$ represents a diverging computation, then z will never have all the causes it needs to be executed and so it must disappear. Indeed, $\beta \not\sqsubseteq_{lpof} \gamma$ because, even though $N_\beta \subseteq_\downarrow N_\gamma$, condition (3c) of Definition 4.1 is violated: $z \in \mathrm{succ}_\beta(y_2)$ but $z \notin \mathrm{succ}_\gamma(y_2) \setminus \mathrm{Bot}_\beta{\uparrow_\gamma}$, since $y_1 \in \mathrm{Bot}_\beta$ and $y_1 <_\gamma z$.

The order on LPOFs yields a partial order (see the proof in Appendix B.1):

LEMMA 4.3. *$\langle lpof(L), \sqsubseteq_{lpof} \rangle$ is a poset.*

Moving to pomsets, we use the same order relation on the underlying LPOF structures of the pomset.

Definition 4.4 (Ordering on Pomsets). Let $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$ iff $\exists \alpha \in \boldsymbol{\alpha}.\ \exists \beta \in \boldsymbol{\beta}.\ \alpha \sqsubseteq_{lpof} \beta$.

In fact, the $\sqsubseteq_{pom}$ order can be defined equivalently by taking any $\beta \in \boldsymbol{\beta}$, or by taking any $\alpha \in \boldsymbol{\alpha}$ that has enough unused node identifiers. The proof is in Appendix C.1.

LEMMA 4.5. *The following two statements hold:*
(1) $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta} \iff \forall \beta \in \boldsymbol{\beta}.\ \exists \alpha \in \boldsymbol{\alpha}.\ \alpha \sqsubseteq_{lpof} \beta$*; and*
(2) $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta} \implies \forall \alpha \in \boldsymbol{\alpha}.\ |nodes \setminus N_\alpha| = \infty \implies \exists \beta \in \boldsymbol{\beta}.\ \alpha \sqsubseteq_{lpof} \beta$

As a consequence of Lemmas 4.3 and 4.5, $\sqsubseteq_{pom}$ also yields a partial order (see Appendix C.1).

COROLLARY 4.6. *$\langle pom(L), \sqsubseteq_{pom} \rangle$ is a poset.*

4.2 Suprema and Directed Complete Partial Orders

Infinite pomsets, particularly those resulting from iterated computations such as while-loops, are constructed most readily using fixed points. We begin by laying out some standard domain theoretic definitions and results, which will allow us to prove that fixed points over pomsets exist. For a more detailed introduction, refer to Abramsky and Jung [1995].

Definition 4.7 (Scott Continuity). Given two DCPOs $\langle X, \leq_X \rangle$ and $\langle Y, \leq_Y \rangle$, a function $f : X \to Y$ is *Scott Continuous* if

- it is monotone: $f(x) \leq_Y f(x')$, for all $x \leq_X x'$; and
- it preserves suprema of directed sets: $\sup_{x \in D} f(x) = f(\sup D)$, for all $D \subseteq X$ directed.

The well known Kleene fixed point theorem states that any Scott Continuous function $f : X \to X$ on a pointed DCPO $\langle X, \leq \rangle$ with bottom $\bot \in X$ has a least fixed point, which is given by $\sup_{n \in \mathbb{N}} f^n(\bot)$, where $f^0 = \mathrm{id}$ and $f^{n+1} = f \circ f^n$. We will now proceed to show that $\langle pom(L), \sqsubseteq_{pom} \rangle$ is a DCPO, and hence it is a suitable domain for defining fixed points. To do so, we must first show how to find suprema of directed sets of LPOFs. Suprema of LPOFs are given by a direct construction, shown in the following lemma (for the proof, see Appendix B.1).

Lemma 4.8. *For any directed set $D \subseteq lpof(L)$, $\sup D = \langle N, <, \lambda, \varphi\rangle$, where:*

$$N \triangleq \bigcup_{\beta\in D} N_\beta \qquad < \triangleq \bigcup_{\beta\in D} <_\beta \qquad \lambda(x) \triangleq \sup_{\beta\in D\,:\,x\in N_\beta} \lambda_\beta(x) \qquad \varphi(x) \triangleq \psi_x$$

and $\psi_x = \varphi_\beta(x)$, for all $\beta \in D$ such that $x \in N_\beta$.

In Lemma 4.8, φ is well-defined since, for every $x \in nodes$ and $\beta, \beta' \in D$ that contain x, it must be that $\varphi_\beta(x) = \varphi_{\beta'}(x)$. Indeed, since D is directed, there must exist a $\gamma \in D$ such that $\beta, \beta' \sqsubseteq_{lpof} \gamma$ and, by Definition 4.1(3b), we have that $\varphi_\beta(x) = \varphi_\gamma(x) = \varphi_{\beta'}(x)$. As an obvious corollary, $\langle lpof(L), \sqsubseteq_{lpof}\rangle$ is a DCPO.

Corollary 4.9. *$\langle lpof(L), \sqsubseteq_{lpof}\rangle$ is a DCPO.*

Proof. Follows immediately from Lemmas 4.3 and 4.8. □

Notice, however, that $lpof(L)$ is *not* pointed, since the node at the root can vary; hence, there is no single LPOF that is smaller (w.r.t. $\sqsubseteq_{lpof}$) than all other LPOFs. For example, two singleton LPOFs with nodes x and y (where $x \neq y$) as their roots do not have any common lower bound. So, *lpof* is incompatible with the Kleene fixed point theorem, one of several reasons why it is not a suitable basis for a program semantics. The situation will change when moving to pomsets (Lemma 4.10), which abstract away from the specific node identifiers.

However, finding suprema of directed sets of pomsets is more challenging; it cannot be done by direct construction, like in Lemma 4.8, because pomsets do not give a straightforward way to match up the node identifiers from one structure to the next. Consider an ascending chain of pomsets $(\boldsymbol{\alpha}_n)_{n\in\mathbb{N}}$, as shown below:

$$\boldsymbol{\alpha}_0 \sqsubseteq_{pom} \boldsymbol{\alpha}_1 \sqsubseteq_{pom} \boldsymbol{\alpha}_2 \sqsubseteq_{pom} \boldsymbol{\alpha}_3 \sqsubseteq_{pom} \cdots$$

It is relatively easy to inductively construct a corresponding chain of LPOFs $\alpha_0 \sqsubseteq_{lpof} \alpha_1 \sqsubseteq_{lpof} \cdots$ such that $\alpha_n \in \boldsymbol{\alpha}_n$ and $|nodes \setminus \alpha_n| = \infty$ for each $n \in \mathbb{N}$. First, fix any $\alpha_0 \in \boldsymbol{\alpha}_0$ such that $|nodes \setminus \alpha_0| = \infty$. Such an element must exist; if the selected α_0 has only finitely many unused nodes, then we can simply permute it to use only half of *nodes*, which is still countably infinite and thus has the same cardinality as *nodes* itself.

Next, for all $n \in \mathbb{N}$, we pick some $\alpha_{n+1} \in \boldsymbol{\alpha}_{n+1}$ such that $\alpha_n \sqsubseteq_{lpof} \alpha_{n+1}$ and $|nodes \setminus \alpha_{n+1}| = \infty$ as follows. By Lemma 4.5(2), some α_{n+1} must exist, but it may not have infinitely many unused nodes. If α_{n+1} only has finitely many unused nodes, then we can permute only $N_{\alpha_{n+1}} \setminus N_{\alpha_n}$ to use half of $|nodes \setminus \alpha_n|$, which is still countably infinite. Finally, Since $\langle lpof(L), \sqsubseteq_{lpof}\rangle$ is a DCPO, then $\sup_{n\in\mathbb{N}} \alpha_n$ must exist. Now, letting $\boldsymbol{\alpha} = [\sup_{n\in\mathbb{N}} \alpha_n]$, $\boldsymbol{\alpha}_n \sqsubseteq_{pom} \boldsymbol{\alpha}$ for all $n \in \mathbb{N}$ by the definition of $\sqsubseteq_{pom}$, and hence $\boldsymbol{\alpha}$ is an upper bound of the chain. However, it is more complicated to establish that $\boldsymbol{\alpha}$ is the *least* upper bound.

To do so, we take any $\boldsymbol{\beta}$ such that $\boldsymbol{\alpha}_n \sqsubseteq_{pom} \boldsymbol{\beta}$ for all $n \in \mathbb{N}$ and show that $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$. Using the chain $(\alpha_n)_{n\in\mathbb{N}}$ from before, it is easy to see that there exists a $\beta_n \in \boldsymbol{\beta}$ for each $n \in \mathbb{N}$ such that $\alpha_n \sqsubseteq_{lpof} \beta_n$, but that does not mean that there is a single $\beta \in \boldsymbol{\beta}$ that is an upper bound of all of them. To show that such a β exists, we fix any $\beta \in \boldsymbol{\beta}$ and show that there exists a sequence of bijections $(f_n)_{n\in\mathbb{N}}$ such that $f_n(\alpha_n) \sqsubseteq_{lpof} \beta$, for all $n \in \mathbb{N}$, where each f_{n+1} is obtained by *extending* f_n so that $f_n(x) = f_{n+1}(x)$ for all x in the domain of f_n. This gives us a new isomorphic chain $(f_n(\alpha_n))_{n\in\mathbb{N}}$, which still has the property that $f_n(\alpha_n) \in \boldsymbol{\alpha}_n$ for all $n \in \mathbb{N}$, but also that $\sup_{n\in\mathbb{N}} f_n(\alpha_n) \sqsubseteq_{lpof} \beta$.

Although there are potentially multiple such f_n functions for each $n \in \mathbb{N}$, under the right conditions, those f_n functions form a finitely branching tree of infinite size. Therefore, by König's Lemma, the tree must have an infinite path, which corresponds to the chain $(f_n(\alpha_n))_{n\in\mathbb{N}}$ [de Bakker and Warmerdam 1990]. Finally, since we know that $\sup_{n\in\mathbb{N}} f_n(\alpha_n)$ is the *least* upper bound of the

chain and that $f_n(\alpha_n) \sqsubseteq_{lpof} \beta$ for all $n \in \mathbb{N}$, then it must be that $\sup_{n\in\mathbb{N}} f_n(\alpha_n) \sqsubseteq_{lpof} \beta$. Since $\sup_{n\in\mathbb{N}} f_n(\alpha_n) \in \boldsymbol{\alpha}$ and $\beta \in \boldsymbol{\beta}$, then $\boldsymbol{\alpha} \sqsubseteq_{lpof} \boldsymbol{\beta}$ and so $\boldsymbol{\alpha}$ is the least upper bound.

Although the proof sketch above gives the general idea, the formal details are more complex and are given in Appendix C.3. Ultimately, we conclude that pomsets have a pointed DCPO structure, where the bottom element is a singleton pomset with label $\bot$.

LEMMA 4.10. *$\langle pom(L), \sqsubseteq_{pom}\rangle$ is a pointed DCPO.*

4.3 Compactness and Approximation

When manipulating pomsets, it is often convenient to work with finite structures. Finite structures are particularly useful for linearization (Section 6.2), which is most naturally defined recursively: at each step, we process some node and then proceed to process the remainder of the structure in the same way. But that recursion is not well-founded for infinite structures, as the number of remaining nodes to process does not decrease each iteration.

Instead, we can define linearization (and related operations) recursively on *finite approximations* of the infinite structure. To linearize the entire structure, we simply take the supremum of the linearization of the finite approximations. In this section, we develop the necessary theory to justify why that approach is correct. First, we formally define finite pomsets.

Definition 4.11 (Finite LPOFs and Pomsets). Let $lpof_{\text{fin}}(L)$ and $pom_{\text{fin}}(L)$ denote the set of LPOFs and pomsets with finitely many nodes, respectively.

$$lpof_{\text{fin}}(L) \triangleq \{\alpha \in lpof(L) \mid |N_\alpha| < \infty\} \qquad pom_{\text{fin}}(L) \triangleq \{[\alpha] \mid \alpha \in lpof_{\text{fin}}(L)\}$$

Given that, a finite approximation of a pomset $\boldsymbol{\alpha}$ is a finite pomset $\boldsymbol{\beta}$ such that $\boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\alpha}$.

Definition 4.12 (Finite Approximations). For any $\boldsymbol{\alpha} \in pom(L)$, let the set of finite approximations of $\boldsymbol{\alpha}$ be defined as follows:

$$\lfloor \boldsymbol{\alpha} \rfloor_{\text{fin}} \triangleq \{\boldsymbol{\beta} \in pom_{\text{fin}}(L) \mid \boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\alpha}\}$$

In order to define operations such as linearization on finite approximations, it is crucial that the aggregate of the finite approximations includes *all* the information about the infinite structure, which is true because every node (even in an infinite structure) must appear at a finite level, and therefore every node is represented in some finite structure. To formalize that argument, we prove that $\boldsymbol{\alpha}$ coincides with the supremum of its finite approximations (see Appendix D).

LEMMA 4.13 (FINITE APPROXIMATIONS). *For any $\boldsymbol{\alpha} \in pom(L)$, $\boldsymbol{\alpha} = \sup \lfloor \boldsymbol{\alpha} \rfloor_{\text{fin}}$.*

The notion of finite approximations intersects with the well-studied concepts in domain theory of approximation orders and compactness, whose definitions are given below [Abramsky and Jung 1995].

Definition 4.14 (Approximation and Compactness). Given a DCPO $\langle X, \leq\rangle$, the *approximation order* $\ll \subseteq X \times X$ is defined as follows: $x \ll y$ iff for any directed set $D \subseteq X$ such that $y \leq \sup D$, there exists $z \in D$ such that $x \leq z$.

An element $x \in X$ is *compact* if it approximates itself ($x \ll x$). We denote the set of compact elements of X as $\mathsf{K}(X) \triangleq \{x \in X \mid x \ll x\}$.

The approximation order is sometimes called the *way-below* relation or the *order of definite refinement* [Smyth 1986]. Intuitively, $x \ll y$ means that x is a "simpler representation" of y because x is below an approximant of y in any sequence that converges to y. So, any element that approximates itself must already be a simple representation, hence it is called compact. In the case of pomsets, the compact elements correspond exactly to the finite pomsets (see Appendix D).

Lemma 4.15 (Compactness). $\mathsf{K}(pom(L)) = pom_{\mathsf{fin}}(L)$.

In particular, that means that $pom_{\mathsf{fin}}(L)$ is the smallest basis for $pom(L)$ [Abramsky and Jung 1995, Corollary 2.2.5]—*i.e.,* it is the smallest set of structures that can approximate any pomset. In fact, the approximation order $\ll$ corresponds exactly to finite approximation (for the proof, see Appendix D); hence, we will furthermore use $\boldsymbol{\alpha} \ll \boldsymbol{\beta}$ and $\boldsymbol{\alpha} \in \lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}}$ interchangeably.

Corollary 4.16 (Approximation). $\boldsymbol{\alpha} \ll \boldsymbol{\beta}$ *iff* $\boldsymbol{\alpha} \in \lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}}$.

The final step is to show that functions defined on finite pomsets can be extended to functions on infinite pomsets in a well-behaved way. As mentioned previously, the extension of f is performed by taking the supremum of f applied to each finite approximation. For that supremum to be defined, f must be monotone, otherwise the set of results may not be directed, and hence it may not have a supremum. In that case, we prove that the extension is not only well-defined, but also Scott continuous in the following lemma inspired by Meyer and de Vink [1989] (the proof is in Appendix D).

Lemma 4.17 (Extension). *Let* $f \colon pom_{\mathsf{fin}}(L)^n \to T$ *be a monotone function on the* dcpo $\langle T, \leq \rangle$. *Then* $f^* \colon pom(L)^n \to T$, *shown below, is well-defined and Scott continuous:*

$$f^*(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n) \triangleq \sup_{\boldsymbol{\alpha}'_1 \ll \boldsymbol{\alpha}_1} \cdots \sup_{\boldsymbol{\alpha}'_n \ll \boldsymbol{\alpha}_n} f(\boldsymbol{\alpha}'_1, \ldots, \boldsymbol{\alpha}'_n)$$

Scott continuity is important because it means that $f(\boldsymbol{\alpha}) = f^*(\boldsymbol{\alpha})$ for all $\boldsymbol{\alpha} \in pom_{\mathsf{fin}}(L)$, *i.e.,* extension does not change the behavior of f when applied to finite pomsets.

5 Pomset Operations

Thus far, we have seen the formal definitions of the structure of pomsets, but we have not defined operations to build pomsets compositionally. In this section, we define those pomset operations, which mirror the program syntax of Figure 1.

We first define the singleton lpof $\langle \ell \rangle_x$ on node x with label $\ell \in L$, which generalizes to $\langle \ell \rangle$ on pomsets and will be used to give the semantics to **skip** and atomic actions:

$$\begin{aligned} \langle \ell \rangle_x &\triangleq \langle \{x\}, \emptyset, [x \mapsto \ell], [x \mapsto \mathsf{true}] \rangle && \in lpof(L) \\ \langle \ell \rangle &\triangleq \{ \langle \ell \rangle_x \mid x \in nodes \} && \in pom(L) \end{aligned}$$

Singletons are the basic building blocks for pomsets; larger pomsets will be constructed by composing singletons. In the following sections, we will see three such operators for composing pomsets.

5.1 Guarded Choice

As we saw in Section 2, the key novel functionality of pomsets with formulae is the ability to encode guarded branching using Boolean formulae to express the sequence of tests that must be passed and failed to reach each point in the structure. We now define the operation:

$$\mathrm{guard} \colon nodes \times L \times lpof(L) \times lpof(L) \to lpof(L)$$

the $\mathrm{guard}(x, \ell, \alpha, \beta)$ operation constructs a pomset with a test ℓ at the root x, and two branches corresponding to the cases where ℓ passes and fails, respectively. More specifically, the formulae in α are altered to specify that the root node x must be true and the formulae in β dictate that x is false. Assuming that $N_\alpha \cap N_\beta = \emptyset$ and $x \notin N_\alpha \cup N_\beta$, we let $\mathrm{guard}(x, \ell, \alpha, \beta) = \langle N, <, \lambda, \varphi \rangle$ where

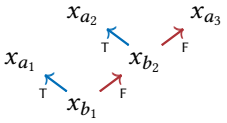
we overload $\wedge$ s.t. $\psi \wedge \mathsf{true} = \mathsf{true} \wedge \psi = \psi$, and:

$$N \triangleq \{x\} \cup N_\alpha \cup N_\beta \qquad < \triangleq <_\alpha \cup <_\beta \cup \big(\{x\} \times (N_\alpha \cup N_\beta)\big)$$

$$\lambda(y) \triangleq \begin{cases} \ell & \text{if } y = x \\ \lambda_\alpha(y) & \text{if } y \in N_\alpha \\ \lambda_\beta(y) & \text{if } y \in N_\beta \end{cases} \qquad \varphi(y) \triangleq \begin{cases} \mathsf{true} & \text{if } y = x \\ \varphi_\alpha(y) \wedge x & \text{if } y \in N_\alpha \\ \varphi_\beta(y) \wedge \neg x & \text{if } y \in N_\beta \end{cases}$$

So, $\mathsf{guard}(x, b, \alpha, \beta)$ joins α and β with a new root node x, whose label is b, and additionally, the formulae in α and β are updated to require that b passes or fails, respectively. As an example, $\mathsf{guard}(x_{b_1}, b_1, \langle a_1 \rangle_{x_{a_1}}, \mathsf{guard}(x_{b_2}, b_2, \langle a_2 \rangle_{x_{a_2}}, \langle a_3 \rangle_{x_{a_3}}))$ produces the LPOF depicted in (3), which is repeated below.

$\varphi(x_{a_2}) = \neg x_{b_1} \wedge x_{b_2}$ $\quad$ $\varphi(x_{a_1}) = x_{b_1}$ $\quad$ $\varphi(x_{b_1}) = \mathsf{true}$



$\varphi(x_{a_3}) = \neg x_{b_1} \wedge \neg x_{b_2}$ $\quad$ $\varphi(x_{b_2}) = \neg x_{b_1}$

For finite pomsets, we define $\mathsf{guard} \colon L \times \mathit{pom}_{\mathsf{fin}}(L) \times \mathit{pom}_{\mathsf{fin}}(L) \to \mathit{pom}_{\mathsf{fin}}(L)$ simply as the equivalence class of the corresponding LPOFs:

$$\mathsf{guard}(\ell, \boldsymbol{\alpha}, \boldsymbol{\beta}) \triangleq \left\{ \mathsf{guard}(x, \ell, \alpha, \beta) \;\middle|\; \alpha \in \boldsymbol{\alpha},\ \beta \in \boldsymbol{\beta}, N_\alpha \cap N_\beta = \emptyset, x \notin N_\alpha \cup N_\beta \right\}$$

Unsurprisingly, guard is monotone, as making either branch larger will make the composed structure larger too. The proof is in Appendix E.1.

Lemma 5.1 (Monotonicity of guard). *Let $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\alpha}'$ and $\boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\beta}'$; then, for every $\ell \in L$, it holds that* $\mathsf{guard}(\ell, \boldsymbol{\alpha}, \boldsymbol{\beta}) \sqsubseteq_{pom} \mathsf{guard}(\ell, \boldsymbol{\alpha}', \boldsymbol{\beta}')$.

Given that, we can define $\mathsf{guard} \colon L \times \mathit{pom}(L) \times \mathit{pom}(L) \to \mathit{pom}(L)$ for infinite structures by extension, and we get that it is Scott Continuous for free.

$$\mathsf{guard}(\ell, \boldsymbol{\alpha}, \boldsymbol{\beta}) \triangleq \sup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \sup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathsf{guard}(\ell, \boldsymbol{\alpha}', \boldsymbol{\beta}')$$

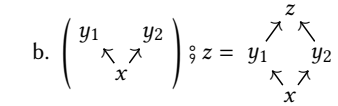
Corollary 5.2. *The operation* guard *is Scott continuous.*

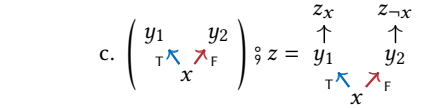
Proof. Immediate from Lemmas 5.1 and 4.17. □

5.2 Sequential Composition

The sequential composition operation $\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta}$ is meant to enforce that all actions in $\boldsymbol{\alpha}$ must occur before any of those in $\boldsymbol{\beta}$. The causality dependency interacts with guarded branching, as we will now see, which makes the formal definition of $\fatsemi$ more challenging. Consider the following LPOFs (where we denote with x the singleton $\langle \ell \rangle_x$ when the label is not relevant):

a. $x \fatsemi y =$

b. 

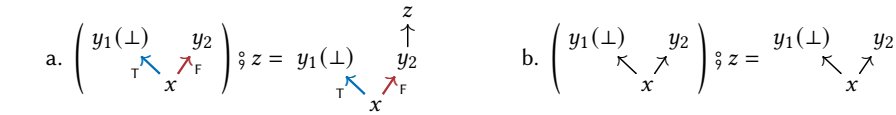
c. 

(4)

The sequential composition of two singletons (4a) simply creates a causality between them. When the program has forked into parallel (unguarded) branches (4b), $\fatsemi$ introduces a diamond structure, as is typical in pomset semantics. However, if the program has a *guarded* branch (4c), then the following actions must be copied after each branch. Without copying, the formula of a single occurrence of z after both y_1 and y_2 would include both x and $\neg x$, and thus be unsatisfiable. Moreover, copying is also essential to correctly account for loops: in the program (**while** b **do** a) ; a', if we only

created a single node labelled with a', then that node would not be finitely preceded, invalidating Lemma 4.13; for a more detailed account, see Section 6.1 and Appendix A (that however uses the semantics of our reference language, that will be provided in Figure 4).

When a structure contains $\bot$ nodes, so that some paths are *stuck*, the behavior of $\fatsemi$ is more complicated. Consider the following LPOFs, where node y_1 is labelled with $\bot$ and all other nodes have non-$\bot$ labels:



(5)

When composing the singleton z after a guarded branch with one stuck path (5a), z is only added to the non-stuck branch. In the program execution, only one branch will be taken, so z can be safely executed as long as the test x is false. However, the same is not true for parallel branching: in (5b), both y_1 and y_2 will eventually be scheduled, so z must occur after y_1—which is equivalent to not occurring at all—thus it does not appear in the final structure. This behavior ensures monotonicity of $\fatsemi$ (Lemma 5.3).

Building on the above intuitions, we first define $\fatsemi$ for finite LPOFs, and then extend our definition to infinite ones using Lemma 4.17. In the rest of this section, we assume that $\alpha, \beta \in \mathit{lpof}_{\mathsf{fin}}(L)$. We start by defining the notion of stuck computations, extensible nodes, and branches. The formula stuck_α indicates which nodes are guaranteed to encounter a $\bot$ later in their execution. Extensible nodes $x \in \mathsf{ext}_\alpha$ are those that can take some computation path without encountering a $\bot$-node.

$$\mathsf{stuck}_\alpha \triangleq \bigvee_{x \in \mathrm{Bot}_\alpha} \varphi_\alpha(x) \qquad\qquad \mathsf{ext}_\alpha \triangleq \{x \in N_\alpha \mid \varphi_\alpha(x) \not\Rightarrow \mathsf{stuck}_\alpha\}$$

In (4), all nodes are extensible, whereas in (5a) to the left of the $\fatsemi$, only x and y_2 are extensible and in (5b) none are extensible. The set of branches br_α contains all of the maximal formulae that can be obtained as the conjunction of formulae of extensible nodes:

$$\mathrm{br}_\alpha \triangleq \{\varphi_\alpha(S) \mid \emptyset \subset S \subseteq \mathsf{ext}_\alpha,\ \varphi_\alpha(S) \Rightarrow \neg\mathsf{stuck}_\alpha,\ \forall T.\ S \subset T \subseteq \mathsf{ext}_\alpha \Rightarrow \neg\mathsf{sat}(\varphi_\alpha(T))\}$$

where, for a set $S \subseteq N_\alpha$, we define $\varphi_\alpha(S) \triangleq \bigwedge_{x \in S} \varphi_\alpha(x)$. For example, in (4a,b), $\mathrm{br} = \{\mathsf{true}\}$ since there are no tests; in (4c), $\mathrm{br} = \{x, \neg x\}$ since both outcomes of the test are not stuck; in (5a), $\mathrm{br} = \{\neg x\}$ since the program is stuck if x passes; and in (5b), $\mathrm{br} = \emptyset$, since all paths are stuck.

As a more involved example containing both guarded and parallel branching in a single structure, we give three LPOFs in Figure 3 along with their stuck nodes, extensible nodes, branches, and the result of sequentially composing them with the singleton LPOF $\langle \ell \rangle_w$. To lighten notation, we assume that α_1, α_2 and α_3 have the same nodes and we only show the $\bot$ labels (associated to z_1 and z_2 in α_1 and just to z_1 in α_2). The three LPOFs form an increasing chain with $\alpha_1 \sqsubseteq_{\mathit{lpof}} \alpha_2 \sqsubseteq_{\mathit{lpof}} \alpha_3$. In addition, the stuck nodes, extensible nodes, and branches are monotone with respect to the LPOFs. As the structure becomes more defined, the formula $\mathsf{stuck}_{\alpha_i}$ becomes stronger, since a more specific condition must be true in order to encounter a $\bot$. Similarly, the set of extensible nodes grows monotonically, since more nodes become extensible as the structure becomes more defined. And finally, the set of branches grows monotonically too, since more tests are uncovered as the structure grows.

In α_1, all nodes are stuck, since the leftmost branch will always lead to $\bot$ and so there are no branches. In α_2, the only stuck node is z_1, arising when y_1 is true; all other nodes are extensible but the only branches to be considered are those in which y_1 is not true. In α_3, there are no stuck nodes, since all elements are non-$\bot$; hence, all nodes are extensible and the branches include all the possible Boolean combinations on y_1 and y_2.

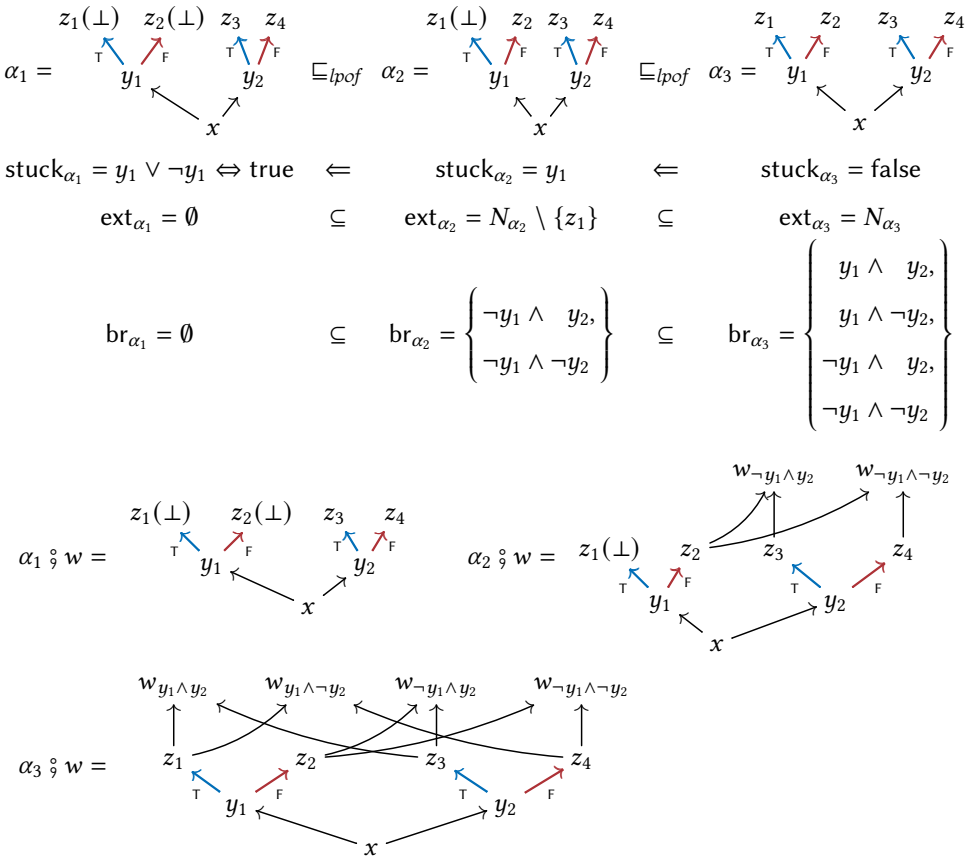


Fig. 3. An example of sequential composition and some of the functions used to define $\fatsemi$.

As we saw in (4c), we will need an isomorphic copy $\beta_\psi \equiv \beta$ for each branch $\psi \in \mathsf{br}_\alpha$. These copies are generated by a function $f\colon \mathsf{br}_\alpha \to [\beta]$ such that the nodes of $f(\psi)$ are disjoint from those of α and of every $f(\psi')$ where $\psi \neq \psi'$. The function f is drawn from the following set:

$$\mathsf{copy}_{\alpha,\beta} \triangleq \left\{ f\colon \mathsf{br}_\alpha \to [\beta] \;\middle|\; \forall \psi \in \mathsf{br}_\alpha.\, (N_{f(\psi)} \cap N_\alpha = \emptyset \;\wedge\; \forall \psi' \neq \psi. N_{f(\psi)} \cap N_{f(\psi')} = \emptyset) \right\}$$

Given $\alpha, \beta \in \mathit{lpof}_{\mathsf{fin}}(L)$ and $f \in \mathsf{copy}_{\alpha,\beta}$, we define sequential composition $\alpha \fatsemi_f \beta = \langle N, <, \lambda, \varphi \rangle$ where the components are defined as follows (notationally, we let $\beta_\psi \triangleq f(\psi)$):

$$N \triangleq N_\alpha \cup \bigcup_{\psi \in \mathsf{br}_\alpha} N_{\beta_\psi} \qquad < \triangleq <_\alpha \cup \bigcup_{\psi \in \mathsf{br}_\alpha} \left(<_{\beta_\psi} \cup (\{x \in N_\alpha \mid \psi \Rightarrow \varphi_\alpha(x)\} \times N_{\beta_\psi}) \right)$$

$$\lambda(x) \triangleq \begin{cases} \lambda_\alpha(x) & \text{if } x \in N_\alpha \\ \lambda_{\beta_\psi}(x) & \text{if } x \in N_{\beta_\psi} \end{cases} \qquad \varphi(x) \triangleq \begin{cases} \varphi_\alpha(x) & \text{if } x \in N_\alpha \\ \varphi_{\beta_\psi}(x) \wedge \psi & \text{if } x \in N_{\beta_\psi} \end{cases}$$

(where, again, $\psi \wedge \mathsf{true} = \mathsf{true} \wedge \psi = \psi$). So, the nodes of $\alpha \fatsemi_f \beta$ are the nodes of α, and all the nodes of the isomorphic copies β_ψ. The new order preserves the causalities in α and in each β_ψ, and additionally requires that β_ψ occurs after all the nodes in the branch ψ. All labels are preserved, and formulae in β_ψ are updated to also include ψ.

Returning to Figure 3, let $f(\psi) = \langle \ell \rangle_{w_\psi}$ be a copying function. We see that $\alpha_1 \fatsemi_f \langle \ell \rangle_w = \alpha_1$; the sequential composition does nothing since there are no branches to extend and all nodes are stuck. On the other hand, $\alpha_2 \fatsemi_f \langle \ell \rangle_w$ generates two copies of w, since there are two branches and $\alpha_3 \fatsemi \langle \ell \rangle_w$ requires four copies of w, since there are four branches.

Sequential composition for finite pomsets is defined by taking the equivalence class of all underlying LPOFs. Note that $\mathsf{copy}_{\alpha,\beta}$ is clearly nonempty for any $\alpha, \beta \in \mathit{lpof}_{\mathsf{fin}}(L)$, since α and β contain only finitely many nodes and br_α is also a finite set.

$$\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \triangleq \{\alpha \fatsemi_f \beta \mid \alpha \in \boldsymbol{\alpha},\ \beta \in \boldsymbol{\beta},\ f \in \mathsf{copy}_{\alpha,\beta}\}$$

Following from the monotonicity properties of stuck, extensible nodes, and branches, sequential composition is also monotone (the proof is in Appendix E.2):

Lemma 5.3 (Monotonicity of $\fatsemi$). *If $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\alpha}'$ and $\boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\beta}'$, then $\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\alpha}' \fatsemi \boldsymbol{\beta}'$*

We see monotonicity at play in Figure 3, where new copies of w appear as more branches become un-stuck. Given that, we define sequential composition of infinite pomsets by extension, and we get for free that it is Scott continuous:

$$\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \triangleq \sup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \sup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \boldsymbol{\alpha}' \fatsemi \boldsymbol{\beta}'$$

Corollary 5.4. *$\fatsemi$ is Scott continuous.*

Proof. Immediate from Lemmas 5.3 and 4.17. □

Remark 1. We can now explain condition (2c) in Definition 3.7, requiring single-rootedness, which may seem strange in a framework with concurrency; indeed, the usual semantics of two commands put in parallel is obtained by taking the (disjoint) union of their pomsets (and this yields several possible minimum elements in the resulting pomset). However, having multi-rooted LPOFs would make sequential composition not monotone. For example, $y \sqsubseteq_{lpof} y\ z$, but:

$$x \fatsemi y \triangleq \begin{array}{c} y \\ \uparrow \\ x \end{array} \not\sqsubseteq_{lpof} \begin{array}{ccc} & y & \\ \nwarrow & & \nearrow \\ & x & \end{array} \begin{array}{c} z \\ \\ \end{array} \triangleq x \fatsemi (y\ z)$$

This is undesirable and easily fixed with the mild requirement of single-rootedness in Definition 3.7.

5.3 Parallel Composition

We finally define parallel composition. We assume that L contains a special (non-$\bot$) label fork that denotes thread forking and let $\mathsf{nFork}_\alpha \triangleq N_\alpha \setminus \{x \in \min_\alpha \mid \lambda_\alpha(x) = \mathsf{fork}\}$ denote the set of nodes of α without its fork-minimal elements (if any). Then, for both finite and infinite LPOFs α and β, and for any $x \notin N_\alpha \cup N_\beta$, we let $\alpha \parallel_x \beta \triangleq \langle N, <, \lambda, \varphi \rangle$, where

$$N \triangleq \{x\} \cup \mathsf{nFork}_\alpha \cup \mathsf{nFork}_\beta$$

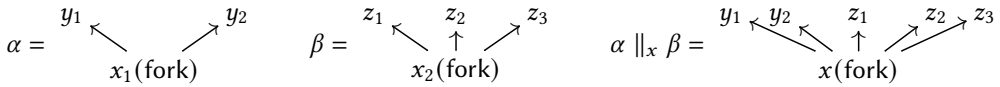
$$< \triangleq (\{x\} \times (\mathsf{nFork}_\alpha \cup \mathsf{nFork}_\beta)) \cup (<_\alpha \cap (\mathsf{nFork}_\alpha \times \mathsf{nFork}_\alpha)) \cup (<_\beta \cap (\mathsf{nFork}_\beta \times \mathsf{nFork}_\beta))$$

$$\lambda(y) \triangleq \begin{cases} \mathsf{fork} & \text{if } y = x \\ \lambda_\alpha(y) & \text{if } y \in \mathsf{nFork}_\alpha \\ \lambda_\beta(y) & \text{if } y \in \mathsf{nFork}_\beta \end{cases} \qquad \varphi(y) \triangleq \begin{cases} \mathsf{true} & \text{if } y = x \\ \varphi_\alpha(y) & \text{if } y \in \mathsf{nFork}_\alpha \\ \varphi_\beta(y) & \text{if } y \in \mathsf{nFork}_\beta \end{cases}$$

Notice that the branching that arises from $\parallel$ is conceptually (and practically) different from the branching that arises from a test: the branching due to $\parallel$ does not exclude any branch, whereas the two branches due to a test are mutually exclusive. That mutual exclusion is apparent in how the formulae of α and β are handled. The formulae remain the same under parallel composition,

whereas they are extended with a new conjunct (specifying the value of the test at the branch) under a guard. The complication arising in our handling of parallel composition through fork is that we want one single node labelled with fork, even if we put many LPOFs in parallel.

For example, by putting in parallel the singleton LPOFs made up, respectively, by nodes y_1 and y_2 and by nodes z_1, z_2 and z_3 (and where fork is the label of the root nodes, say x_1 and x_2 respectively), we obtain the LPOFs α, β, and their parallel composition as follows:

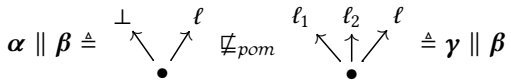


where x_1 and x_2 disappear and the new root is x, labelled fork. From here on, we will use the $\bullet$ symbol to depict fork labels in pomset diagrams.

As a side note, observe that the way in which we defined parallel composition entails that $\alpha \sqsubseteq_{lpof} \alpha \parallel_x \beta$ if and only if $\alpha = \langle\bot\rangle_x$: the LPOF α has just a node x labelled with $\bot$; the second LPOF has node x labelled with fork that precedes a node labelled with $\bot$ and all the nodes of β. Also, a somewhat unusual notion of associativity for parallel of LPOFs holds, viz. $(\alpha \parallel_x \beta) \parallel_y \gamma = \alpha \parallel_y (\beta \parallel_z \gamma)$; the expectable notion of associativity is recovered for pomsets, where we abstract away the specific node set[4]. Like before, we define parallel composition for (both finite and infinite) pomsets on top of that for LPOFs:

$$\boldsymbol{\alpha} \parallel \boldsymbol{\beta} \triangleq \{\alpha \parallel_x \beta \mid \alpha \in \boldsymbol{\alpha},\ \beta \in \boldsymbol{\beta},\ x \notin N_\alpha \cup N_\beta\}$$

Remark 2. Unlike the previous pomset operations, $\parallel$ is *not* monotone (not for LPOFs nor for pomsets). Indeed, consider $\boldsymbol{\alpha} = \langle\bot\rangle$, $\boldsymbol{\beta} = \langle\ell\rangle$, and $\boldsymbol{\gamma} = \langle\ell_1\rangle \parallel \langle\ell_2\rangle$; although $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\gamma}$, we get that



This is not surprising: if $\parallel$ were monotone, then the function $f(\boldsymbol{\alpha}) = \langle\ell\rangle \parallel \boldsymbol{\alpha}$ would have a fixed point, but that fixed point would be an infinitely branching structure, which is invalid.

In fact, it is impossible for $\parallel$ to be *both* associative *and* Scott continuous. If it were continuous, then $f(\boldsymbol{\alpha}) \triangleq \langle\ell\rangle \parallel \boldsymbol{\alpha}$ would have a least fixed point $\mathsf{lfp}(f) = \langle\ell\rangle \parallel (\langle\ell\rangle \parallel \cdots)$, and, due to associativity, this pomset would be infinitely branching, which invalidates Lemma 4.13.

Since $\parallel$ is not monotone, we need a different way to approximate infinite pomsets composed in parallel, which we now describe. Let $\mathsf{root}(\boldsymbol{\gamma})$ be the label of the root node of the pomset $\boldsymbol{\gamma}$; then,

$$\boldsymbol{\gamma} \ll_1 \boldsymbol{\gamma}' \quad \triangleq \quad \boldsymbol{\gamma} \ll \boldsymbol{\gamma}' \quad \text{and} \quad \mathsf{root}(\boldsymbol{\gamma}) = \mathsf{root}(\boldsymbol{\gamma}') = \mathsf{fork} \text{ or } \mathsf{root}(\boldsymbol{\gamma}') \neq \mathsf{fork}$$

i.e., $\boldsymbol{\gamma} \ll_1 \boldsymbol{\gamma}'$ is similar to the standard approximation order $\ll$ on pomsets, but it prevents a fork node at the root of $\boldsymbol{\gamma}'$ from being converted to $\bot$, which would cause the root of a parallel composition involving $\boldsymbol{\gamma}'$ vs $\boldsymbol{\gamma}$ to have a different number of successors (the bad case from Remark 2). Returning to the example in Remark 2, $[\boldsymbol{\alpha}] \sqsubseteq_{pom} [\boldsymbol{\gamma}]$ and also $[\boldsymbol{\alpha}] \ll [\boldsymbol{\gamma}]$, whereas $[\boldsymbol{\alpha}] \not\ll_1 [\boldsymbol{\gamma}]$, since the root of $\boldsymbol{\gamma}$ is fork but the root of $\boldsymbol{\alpha}$ is not fork. We need two properties of $\ll_1$. First:

$$\sup_{\boldsymbol{\gamma} \ll_1 \boldsymbol{\gamma}'} \boldsymbol{\gamma} = \sup_{\boldsymbol{\gamma} \ll \boldsymbol{\gamma}'} \boldsymbol{\gamma} = \boldsymbol{\gamma}' \tag{6}$$

By Lemma 4.13 and Corollary 4.16, $\sup_{\boldsymbol{\gamma} \ll \boldsymbol{\gamma}'} \boldsymbol{\gamma} = \boldsymbol{\gamma}'$. If $\mathsf{root}(\boldsymbol{\gamma}') = \mathsf{fork}$, then $\{\boldsymbol{\gamma} \mid \boldsymbol{\gamma} \ll_1 \boldsymbol{\gamma}'\} = \{\boldsymbol{\gamma} \mid \boldsymbol{\gamma} \ll \boldsymbol{\gamma}'\} \setminus \{\bot_{pom}\}$, which clearly has the same supremum, as we have only removed the bottom

[4]We remark that $\fatsemi$ on LPOFs also satisfies a non-straightforward associativity, that however becomes the usual one when passing to pomsets.

$$
\begin{aligned}
[\![\mathbf{skip}]\!] &\triangleq \langle \mathsf{fork} \rangle \\
[\![a]\!] &\triangleq \langle a \rangle \\
[\![C_1; C_2]\!] &\triangleq [\![C_1]\!] \fatsemi [\![C_2]\!] \\
[\![C_1 \mid C_2]\!] &\triangleq [\![C_1]\!] \parallel [\![C_2]\!] \\
[\![C_1 \,\&\, C_2]\!] &\triangleq \mathsf{guard}(\&, [\![C_1]\!], [\![C_2]\!]) \\
[\![C_1 \oplus_p C_2]\!] &\triangleq \mathsf{guard}(\oplus_p, [\![C_1]\!], [\![C_2]\!]) \\
[\![\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2]\!] &\triangleq \mathsf{guard}(b, [\![C_1]\!], [\![C_2]\!]) \\
[\![\mathbf{while}\ b\ \mathbf{do}\ C]\!] &\triangleq \mathsf{lfp}\left(\Phi_{\langle C,b\rangle}\right) \\
&\qquad \text{where } \Phi_{\langle C,b\rangle}(\boldsymbol{\alpha}) \triangleq \mathsf{guard}(b, [\![C]\!] \fatsemi \boldsymbol{\alpha}, [\![\mathbf{skip}]\!])
\end{aligned}
$$

Fig. 4. Pomset semantics of commands $[\![-]\!] : \mathit{cmd} \to \mathit{pom}$.

element from the set. If $\mathrm{root}(\boldsymbol{\gamma}') \neq \mathsf{fork}$, then the orders are equivalent. The second property that we need is

$$\lfloor \boldsymbol{\alpha} \parallel \boldsymbol{\beta} \rfloor_{\mathsf{fin}} \setminus \{\bot_{\mathit{pom}}\} = \{\boldsymbol{\alpha}' \parallel \boldsymbol{\beta}' \mid \boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}, \boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}\} \tag{7}$$

This can be easily seen by observing that $\bot_{\mathit{pom}}$ is not included in the right set, since all parallel compositions have fork at the root therein. Moreover, the successors nodes (but not their labels) of the root are also fixed in both cases: on the left they are fixed because the root is fixed, and on the right they are fixed due to the $\ll_1$ order, since any fork node at the root would be removed by the parallel composition. Above level 2, both sets allow for any approximation of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

6 Denotational Semantics

We now define the denotational semantics for commands $C \in \mathit{cmd}$ (see Figure 1) based on pomsets with formulae. Although actions and tests at this stage are uninterpreted, our pomsets allow for actions and tests with a variety of different *computational effects*, which we will see in Section 6.2. Actions $a \in \mathit{act}$ are drawn from a DCPO $\langle \mathit{act}, \leq_{\mathit{act}} \rangle$ and tests $b \in \mathit{test}$ are drawn from a DCPO $\langle \mathit{test}, \leq_{\mathit{test}} \rangle$. In order to interpret $C_1 \,\&\, C_2$ and $C_1 \oplus_p C_2$, we assume that $\& \in \mathit{test}$ and $\oplus_p \in \mathit{test}$ for all $p \in [0, 1]$. We first define a compositional denotational model for this language in Section 6.1, and then in Section 6.2 we show how to interpret pomsets as state transformers, to learn their behavior on particular inputs.

6.1 Pomset Semantics

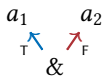
We will now discuss the semantics of programs $C \in \mathit{cmd}$ as pomsets, which record both the causality between atomic actions and the branching behavior of tests. Given two finitely preceded DCPOs $\langle \mathit{act}, \leq_{\mathit{act}} \rangle$ and $\langle \mathit{test}, \leq_{\mathit{test}} \rangle$, the label set $\mathit{label} \triangleq \mathit{act} \cup \mathit{test} \cup \{\mathsf{fork}, \bot\}$ consists of actions, tests, fork, and bottom nodes, which form a pointed DCPO $\langle \mathit{label}, \leq \rangle$, with $\bot$ as the bottom and $\leq$ defined as follows:

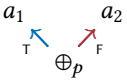
$$\leq \triangleq (\{\bot\} \times \mathit{label}) \cup \{(\mathsf{fork}, \mathsf{fork})\} \cup \leq_{\mathit{act}} \cup \leq_{\mathit{test}}$$

We will also henceforth use $\mathit{pom} \triangleq \mathit{pom}(\mathit{label})$ to refer to pomsets over this label set.

The semantic function $[\![-]\!] : \mathit{cmd} \to \mathit{pom}$, mapping commands to pomsets, is shown in Figure 4. Its definition is straightforward using the pomset operations. In particular, **skip** and basic actions are singletons, sequential composition uses $\fatsemi$, and if-statements use guard. Nondeterministic and

probabilistic choice also use guard, but with the special test symbols & and $\oplus_p$, giving us:

$[\![a_1 \,\&\, a_2]\!] =$ 

$[\![a_1 \oplus_p a_2]\!] =$ 

Notice that the two pomsets above are very similar, and similar to

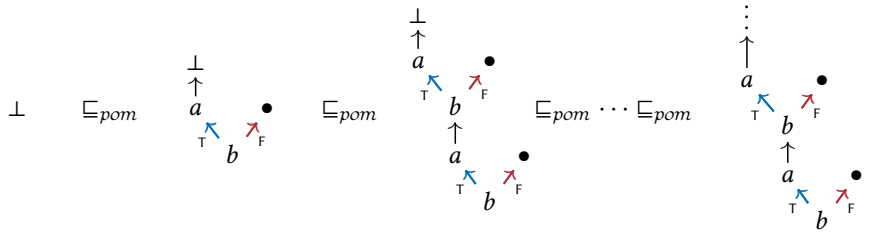
$[\![\texttt{if}\ b\ \texttt{then}\ a_1\ \texttt{else}\ a_2]\!] =$ 

The three will be distinguished when linearizing the structures in Section 6.2. In particular: when linearizing the first one, we will nondeterministically select one of the two branches; when linearizing the second one, we will consider both branches, the leftmost with probability p and the rightmost with probability $1 - p$; and when linearizing the third one, we will select one of the two branches, according to the evaluation of the guard b.

While-loops are interpreted as the least fixed point of the characteristic function $\Phi_{\langle C,b\rangle} : pom \to pom$. Clearly $\Phi_{\langle C,b\rangle}$ is Scott continuous, as it is defined by the Scott continuous operations guard and $\fatsemi$ (see Corollaries 5.2 and 5.4). So, by Kleene's fixed point theorem, $\Phi_{\langle C,b\rangle}$ has a least fixed point given by:

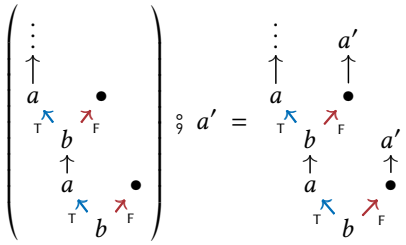
$$[\![\texttt{while}\ b\ \texttt{do}\ C]\!] = \mathsf{lfp}\left(\Phi_{\langle C,b\rangle}\right) = \sup_{n\in\mathbb{N}} \Phi^n_{\langle C,b\rangle}\left(\bot_{pom}\right)$$

where $f^0 \triangleq \mathsf{id}$ and $f^{n+1} \triangleq f \circ f^n$. Considering a loop $[\![\texttt{while}\ b\ \texttt{do}\ a]\!]$ with a single action a as its body, we can unroll the definition several times to get the following infinite chain:



The supremum of this chain is clearly an infinite structure with a terminating branch for each $n \in \mathbb{N}$, and an infinitely ascending spine. The $\bot$ node is pushed to a progressively higher level after each unrolling, so in the supremum it does not appear at all.

Below, we show $[\![(\texttt{while}\ b\ \texttt{do}\ a)\ ;\ a']\!]$, containing a loop with a single action in the body followed by another action:



As discussed in Section 5.2, sequentially composing another action a' after a loop produces a copy of a' for each branch, resulting in countably many copies of a'. This approach is necessary to make

a' finitely preceded. More discussion on how to unroll loops, and on the interplay between this and sequential composition, can be found in Appendix A.

The semantics obeys a binary branching property, guaranteeing that test nodes branch into exactly two successors, one where the test passes and another where it fails. Further, variables can only be introduced into formulae after a test. From now on, we assume that all pomsets have this property, which is clearly preserved by the operations of Section 5.

Definition 6.1 (Binary Branching). A pomset $\boldsymbol{\alpha} \in \mathit{pom}$ has the *binary branching property* if, for every $\alpha \in \boldsymbol{\alpha}$ and every $x \in N_\alpha$ such that $\lambda_\alpha(x) \in \mathit{test}$, $\mathrm{succ}_\alpha(x) = \{y_1, y_2\}$ such that:

(1) $\varphi_\alpha(y_1) \Leftrightarrow \varphi_\alpha(x) \wedge x$;
(2) $\varphi_\alpha(y_2) \Leftrightarrow \varphi_\alpha(x) \wedge \neg x$; and
(3) $\mathrm{pred}_\alpha(y_1) = \mathrm{pred}_\alpha(y_2) = \{x\}$.

Furthermore, if x is not the successor of a test, then $\varphi_\alpha(x) \Leftrightarrow \bigwedge_{y \in \mathrm{pred}_\alpha(x)} \varphi_\alpha(y)$.

6.2 Linearization

We now discuss how to produce a state transformer from the pomset semantics of Section 6.1, which is useful for understanding the input-output behavior of programs resulting from scheduling concurrent threads. We thus fix set of states $\mathcal{S}$, ranged over by s, and let the state transformer be obtained via *linearization*, which considers all interleavings of the threads in order to interpret the semantics of a *complete* program as a function from inputs to collections of outputs. Once the program is linearized, more threads cannot be composed in parallel, as the causality information is gone. Nevertheless, linearization is useful to understand the program's behavior, and relate the behavior to other semantic models, as shown in Sections 2 and 8.

When performing linearization, we will interpret programs in a computational domain D, which must support nondeterminism—in order to model the different ways that threads can be interleaved—but may support additional computational effects too, which we will model using monads. Hence, we assume some familiarity with basic notions of monads and Kleisli categories; for a detailed introduction, we refer to [Awodey 2006]. More precisely, we need the following:

(1) A domain D and set of states $\mathcal{S}$, such that $\langle D(\mathcal{S}), \sqsubseteq \rangle$ is a pointed DCPO with bottom $\bot_D$.
(2) A nondeterminism operator $\&\colon D(\mathcal{S})^+ \to D(\mathcal{S})$ that is monotone, i.e., $\&_{i\in I}\, d_i \sqsubseteq \&_{i\in I}\, d'_i$ if $d_i \sqsubseteq d'_i$ for all $i \in I$, where I is a nonempty, finite index set.
(3) A probabilistic choice operator $\oplus_p\colon D(\mathcal{S})^2 \to D(\mathcal{S})$ ($\oplus_p$ is only used to interpret commands of the form $C_1 \oplus_p C_2$. If those commands do not appear, then $\oplus_p$ can be left undefined, as we will see in Section 8).
(4) An additive monad in the category **DCPO** $\langle D, \eta, (-)^\dagger \rangle$, meaning that:
   (a) the operations $\eta\colon X \to D(X)$ and $(-)^\dagger\colon (X \to D(Y)) \to D(X) \to D(Y)$ obey the monad laws $f^\dagger \circ \eta = f$, $\eta^\dagger = \mathrm{id}$, and $(g^\dagger \circ f)^\dagger = g^\dagger \circ f^\dagger$.
   (b) Kleisli extension is
      - Scott Continuous: $\sup_{f\in D} \sup_{d\in D'} f^\dagger(d) = (\sup D)^\dagger(\sup D')$;
      - strict: $f^\dagger(\bot_D) = \bot_D$; and
      - additive: $f^\dagger(\&_{i\in I}\, d_i) = \&_{i\in I}\, f^\dagger(d_i)$ and $f^\dagger(d_1 \oplus_p d_2) = f^\dagger(d_1) \oplus_p f^\dagger(d_2)$
(5) Semantics for actions $[\![-]\!]_{act} : \mathit{act} \to \mathcal{S} \to D(\mathcal{S})$ and tests $[\![-]\!]_{test} : \mathit{test} \to \mathcal{S} \to D(\mathbb{B})$ such that:
   (a) $[\![-]\!]_{act}$ is monotone: if $a \leq_{act} a'$, then $[\![a]\!]_{act}(s) \sqsubseteq [\![a']\!]_{act}(s)$.
   (b) $[\![-]\!]_{test}$ is monotone: if $b \leq_{test} b'$, then $[\![b]\!]_{test}(b) \sqsubseteq [\![b']\!]_{test}(s)$.
   (c) $[\![\&]\!]_{test}(s) = \&_{x\in\{0,1\}}\, \eta(x)$ and $[\![\oplus_p]\!]_{test}(s) = \eta(1) \oplus_p \eta(0)$.

$$\mathcal{L}_{lpof}(\alpha,\psi,S)(s) \triangleq \begin{cases} \eta(s) & \text{if } \mathsf{next}(\alpha,\psi,S) = \emptyset \\ \&_{x\in\mathsf{next}(\alpha,\psi,S)}\ \mathcal{L}_{node}(\alpha,\psi,S,x)(s) & \text{if } \mathsf{next}(\alpha,\psi,S) \neq \emptyset \end{cases}$$

$$\mathcal{L}_{node}(\alpha,\psi,S,x)(s) \triangleq \begin{cases} \mathcal{L}_{lpof}(\alpha,\psi,S\cup\{x\})^\dagger\left(\llbracket\lambda_\alpha(x)\rrbracket_{act}(s)\right) & \text{if } \lambda_\alpha(x) \in act \\ \mathcal{L}_{test}(\alpha,\psi,S,x)(s)^\dagger(\llbracket\lambda_\alpha(x)\rrbracket_{test}(s)) & \text{if } \lambda_\alpha(x) \in test \\ \bot_D & \text{if } \lambda_\alpha(x) = \bot \\ \mathcal{L}_{lpof}(\alpha,\psi,S\cup\{x\})(s) & \text{if } \lambda_\alpha(x) = \mathsf{fork} \end{cases}$$

$$\mathcal{L}_{test}(\alpha,\psi,S,x)(s)(b) \triangleq \begin{cases} \mathcal{L}_{lpof}(\alpha,\psi \wedge\ \ x, S\cup\{x\})(s) & \text{if } b = 1 \\ \mathcal{L}_{lpof}(\alpha,\psi\wedge\neg x, S\cup\{x\})(s) & \text{if } b = 0 \end{cases}$$

$$\mathsf{next}(\alpha,\psi,S) \triangleq \{x \in N_\alpha \setminus S \mid x{\downarrow} \subseteq S, \psi \Rightarrow \varphi_\alpha(x)\}$$

Fig. 5. Linearization for finite LPOFs $\mathcal{L}_{lpof}\colon lpof_{\mathsf{fin}}(label)\times form\times\mathcal{P}(nodes)\to\mathcal{S}\to D(\mathcal{S})$.

There are many domains that obey these properties, depending on which *computational effects* are present. These include the Hoare, Smyth, and Plotkin powerdomains for nondeterministic computation [Abramsky and Jung 1995; Plotkin 1976; Smyth 1978] (where the latter two additionally deal with nontermination), the convex powerdomain [He et al. 1997; Morgan et al. 1996b] and the powerdomain of indexed valuations [Varacca 2002, 2003; Varacca and Winskel 2006] (for mixed probabilistic and nondeterministic computation), and other domains for nondeterminism with exceptions [Zilberstein et al. 2023, 2024]. In Theorems 7.2 and 8.2, we will show that standard semantics in two of those domains are recovered from our pomset semantics.

We will start by defining a linearization operation on finite LPOFs, which is shown in Figure 5. Linearization is defined recursively, until there are no more nodes to schedule. It must be defined on finite structures, otherwise the recursion is not well-founded, but it can be extended to infinite structures using the extension lemma. In $\mathcal{L}_{lpof}(\alpha,\psi,S)$, the set $S \subseteq N_\alpha$ contains all the nodes that have already been processed and ψ is a path condition, indicating the outcomes of the tests associated to the nodes in S.

The function $\mathsf{next}(\alpha,\psi,S)$ gives the nodes that are ready to be scheduled, which consists of all nodes that obey the path condition ψ, and whose predecessors are already in S. If $\mathsf{next}(\alpha,\psi,S)$ is empty, then $\mathcal{L}_{lpof}(\alpha,\psi,S)$ simply returns the current state s using the monad unit η, since there are no more nodes to schedule. If not, then it nondeterministically selects a next node using $\&$. If the next node is an action, then it interprets the action using $\llbracket-\rrbracket_{act}$ and uses Kleisli composition to compose the result with the linearization of the remaining LPOF. If the next node is a test, then that test is evaluated and the path condition is updated to account for the outcome of the test. If the next node is $\bot$, then the linearization is $\bot_D$, and fork is treated like a no-op.

We next use $\mathcal{L}_{\mathsf{fin}}\colon pom_{\mathsf{fin}}(label)\to\mathcal{S}\to D(\mathcal{S})$ to define linearization on finite pomsets. Clearly $\mathcal{L}_{lpof}(\alpha,\mathsf{true},\emptyset) = \mathcal{L}_{lpof}(\beta,\mathsf{true},\emptyset)$ if $\alpha\equiv\beta$, so $\mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha})$ can be defined for any arbitrary representative LPOF $\alpha\in\boldsymbol{\alpha}$ using the trivial path condition true and an empty set of visited nodes:

$$\mathcal{L}_{\mathsf{fin}}([\alpha]) \triangleq \mathcal{L}_{lpof}(\alpha,\mathsf{true},\emptyset)$$

Linearization of finite pomsets is monotone, which relies on the fact that action and test evaluation is monotone; the proof is in Appendix F.1.

Lemma 6.2 (Monotonicity of $\mathcal{L}_{\mathsf{fin}}$). *If $\boldsymbol{\alpha}\sqsubseteq_{pom}\boldsymbol{\beta}$, then $\mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha})(s)\sqsubseteq\mathcal{L}_{\mathsf{fin}}(\boldsymbol{\beta})(s)$.*

Therefore, linearization can be extended to infinite pomsets $\mathcal{L}\colon pom \to \mathcal{S} \to D(\mathcal{S})$ by taking the supremum over all of the finite approximations, and the extension is well-defined and Scott continuous.

$$\mathcal{L}(\alpha) \triangleq \mathcal{L}^*_{\mathsf{fin}}(\alpha)$$

Corollary 6.3. *$\mathcal{L}\colon pom \to \mathcal{S} \to D(\mathcal{S})$ is Scott continuous.*

Proof. Immediate from Lemmas 6.2 and 4.17. □

To ensure that linearization acts as desired, we provide a sanity check lemma to relate the linearized semantics to standard monadic denotational semantics. Linearizing **skip** gives the monad unit; linearizing a singleton action has the same behavior as interpreting the action; linearizing a sequential composition is equal to Kleisli composition of the individual linearizations; linearizing an if-statement is equal to linearizing one of the two branches, depending on the truth of the guard; and the linearization of a while-loop is equal to the least fixed point over a different characteristic function $\Psi_{\langle f,b\rangle}$. The function $\Psi_{\langle f,b\rangle}$ inherits Scott continuity from the Kleisli composition operator, therefore the fixed point exists. The proof is in Appendix F.6.

Lemma 6.4 (Linearization). *The following properties hold:*

$$\mathcal{L}(\llbracket \mathbf{skip} \rrbracket) = \eta \qquad \mathcal{L}(\llbracket a \rrbracket) = \llbracket a \rrbracket_{act} \qquad \mathcal{L}(\llbracket C_1; C_2 \rrbracket) = \mathcal{L}(\llbracket C_2 \rrbracket)^\dagger \circ \mathcal{L}(\llbracket C_1 \rrbracket)$$

$$\mathcal{L}(\llbracket C_1 \,\&\, C_2 \rrbracket)(s) = \mathcal{L}(\llbracket C_1 \rrbracket)(s) \,\&\, \mathcal{L}(\llbracket C_2 \rrbracket)(s) \qquad \mathcal{L}(\llbracket C_1 \oplus_p C_2 \rrbracket)(s) = \mathcal{L}(\llbracket C_1 \rrbracket)(s) \oplus_p \mathcal{L}(\llbracket C_2 \rrbracket)(s)$$

$$\mathcal{L}(\llbracket \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2 \rrbracket)(s) = \left[b' \mapsto \begin{cases} \mathcal{L}(\llbracket C_1 \rrbracket)(s) & \text{if } b' = 1 \\ \mathcal{L}(\llbracket C_2 \rrbracket)(s) & \text{if } b' = 0 \end{cases} \right]^\dagger \left(\llbracket b \rrbracket_{test}(s) \right)$$

$$\mathcal{L}(\llbracket \mathbf{while}\ b\ \mathbf{do}\ C \rrbracket) = \mathsf{lfp}\left(\Psi_{\langle \mathcal{L}(\llbracket C \rrbracket), b\rangle} \right)$$

where $\Psi_{\langle f,b\rangle}(g)(s) \triangleq \left[b' \mapsto \begin{cases} g^\dagger(f(s)) & \text{if } b' = 1 \\ \eta(s) & \text{if } b' = 0 \end{cases} \right]^\dagger \left(\llbracket b \rrbracket_{test}(s) \right)$.

Lemma 6.4 is quite significant; it implies that $\mathcal{L}(\llbracket C \rrbracket)$ corresponds to standard monadic semantics in a variety of domains. We will see two such examples in Sections 7 and 8. Properties about parallel programs are notably missing from Lemma 6.4. In fact, no such compositional properties exist, since parallel composition can completely change the behavior of each thread. As a simple example, consider the following programs:

$$C_1 \triangleq x := 1; \mathbf{if}\ x = 1\ \mathbf{then}\ y := 1\ \mathbf{else}\ y := 0 \qquad C_2 \triangleq x := 0$$

Using Lemma 6.4, we can derive the linearization of C_1 as follows.

$$\begin{aligned} \mathcal{L}\llbracket C_1 \rrbracket(s) &= \mathcal{L}(\llbracket \mathbf{if}\ x = 1\ \mathbf{then}\ y := 1\ \mathbf{else}\ y := 0 \rrbracket)^\dagger(\mathcal{L}(\llbracket x := 1 \rrbracket)(s)) \\ &= \mathcal{L}(\llbracket \mathbf{if}\ x = 1\ \mathbf{then}\ y := 1\ \mathbf{else}\ y := 0 \rrbracket)(s[x := 1]) \\ &= \begin{cases} \mathcal{L}(\llbracket y := 1 \rrbracket)(s[x := 1]) & \text{if } s[x := 1](x) = 1 \\ \mathcal{L}(\llbracket y := 0 \rrbracket)(s[x := 1]) & \text{if } s[x := 1](x) = 0 \end{cases} \\ &= \eta(s[x := 1][y := 1]) \end{aligned}$$

So, by linearizing C_1 on its own, we get that y will always be set to 1. In addition, $\mathcal{L}(\llbracket C_2 \rrbracket)(s) = \eta(s[x := 0])$. However, it is not hard to see that $C_1 \,|\, C_2$ has an execution where y is set to 0, even though neither $\mathcal{L}(\llbracket C_1 \rrbracket)$ nor $\mathcal{L}(\llbracket C_2 \rrbracket)$ display that behavior. So, it is impossible to determine

$\mathcal{L}(\llbracket C_1 \mid C_2 \rrbracket)$ compositionally from $\mathcal{L}(\llbracket C_1 \rrbracket)$ and $\mathcal{L}(\llbracket C_2 \rrbracket)$. Deriving properties about parallel executions is a significant area of research, and requires more advanced logical methods such as Rely-Guarantee [Jones 1983] or Separation Logic [Brookes 2004; O'Hearn 2004], which are out of scope for this paper; such techniques, based on pomsets with formulae, have been explored by [Zilberstein 2026b; Zilberstein et al. 2026].

7 Applications to Probabilistic Concurrency

We now revisit the convex powerdomain, which was first introduced in Section 2. First, we give the full formal definition of the computational domain, and show that it is suitable for linearization given the constraints of Section 6.2. We will then show that our linearized semantics corresponds exactly to the standard convex powerdomain semantics of non-parallel languages. Finally, we will show additional examples of the semantics generated for probabilistic concurrent programs.

7.1 Formal Definitions

We begin by providing the omitted definitions from Section 2, regarding the DCPO structure of the convex powerset. We start by defining the order on probability distributions. For any set X, let $X_\perp = X \cup \{\perp\}$ and the order over distributions $\sqsubseteq_{\mathcal{D}} \subseteq \mathcal{D}(X_\perp) \times \mathcal{D}(X_\perp)$ be defined as follows:

$$\mu \sqsubseteq_{\mathcal{D}} \nu \quad \text{iff} \quad \forall x \in X.\ \mu(x) \leq \nu(x)$$

Note that $\mu \sqsubseteq_{\mathcal{D}} \nu$ means that $\mu(\perp) \geq \nu(\perp)$, since as distributions become larger, the leftover mass of nontermination shrinks. The bottom of the order is $\perp_{\mathcal{D}} = [\perp \mapsto 1]$. A set of distributions $S \subseteq \mathcal{D}(X_\perp)$ is *upward closed* if, for all $\mu \in S$, it holds that $\mu \sqsubseteq_{\mathcal{D}} \nu$ implies $\nu \in S$. In addition, S is *Cauchy closed* if it is closed in the product of Euclidean topologies [McIver and Morgan 2005, Definition 5.4.3]. Now, the convex powerset is defined as follows:

Definition 7.1 (The Convex Powerdomain).

$$C(X) \triangleq \{S \subseteq \mathcal{D}(X_\perp) \mid S \text{ is nonempty, convex, upward closed, and Cauchy closed}\}$$

Convex powersets are ordered via the Smyth order [Smyth 1978] $\sqsubseteq_C \subseteq C(X) \times C(X)$, defined below:

$$S \sqsubseteq_C T \quad \text{iff} \quad \forall \nu \in T.\ \exists \mu \in S.\ \mu \sqsubseteq_{\mathcal{D}} \nu$$

Since the sets above are upward closed, the Smyth order collapses to reverse subset inclusion: $S \sqsubseteq_C T$ iff $S \supseteq T$, which makes the *convex powerdomain* $\langle C(X), \sqsubseteq_C \rangle$ a pointed DCPO with suprema given by set intersection and bottom $\perp_C = \mathcal{D}(X_\perp)$ being the set of all distributions. For the full details, refer to [Zilberstein et al. 2025b].

Finally, we give the definitions for the monad operations [Jacobs 2008], namely unit $\eta : X \to C(X)$ and Kleisli extension $(-)^\dagger : (X \to C(Y)) \to C(X) \to C(Y)$. We let

$$\eta(x) \triangleq \{[x \mapsto 1]\} \qquad f^\dagger(S) \triangleq \left\{ \sum_{x \in \mathsf{supp}(\mu)} \mu(x) \cdot \nu_x \;\middle|\; \mu \in S, \forall x \in \mathsf{supp}(\mu).\ \nu_x \in f_\perp(x) \right\}$$

where $(-)_\perp : (X \to C(Y)) \to X_\perp \to C(Y)$ is defined as follows: $f_\perp(x) \triangleq f(x)$ if $x \in X$ and $f_\perp(\perp) \triangleq \perp_C$ otherwise. As is required in Section 6.2, it was proven in Zilberstein et al. [2025b] that the Kleisli extension for C defined above is Scott continuous and additive:

$$\sup_{f \in D, S \in D'} f^\dagger(S) = (\sup D)^\dagger(\sup D') \qquad f^\dagger\left(\bigotimes_{i \in I} S_i\right) = \bigotimes_{i \in I} f^\dagger(S_i) \qquad f^\dagger(S \oplus_p T) = f^\dagger(S) \oplus_p f^\dagger(T)$$

$$
\begin{aligned}
[\![\mathbf{skip}]\!]_C(s) &\triangleq \eta(s) \\
[\![C_1; C_2]\!]_C(s) &\triangleq [\![C_2]\!]_C^{\dagger}\left([\![C_1]\!]_C(s)\right) \\
[\![C_1 \,\&\, C_2]\!]_C(s) &\triangleq [\![C_1]\!]_C(s) \,\&\, [\![C_2]\!]_C(s) \\
[\![C_1 \oplus_p C_2]\!]_C(s) &\triangleq [\![C_1]\!]_C(s) \oplus_p [\![C_2]\!]_C(s) \\
[\![\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2]\!]_C(s) &\triangleq \begin{cases} [\![C_1]\!]_C(s) & \text{if } [\![b]\!]_{test}(s) = \eta(1) \\ [\![C_2]\!]_C(s) & \text{if } [\![b]\!]_{test}(s) = \eta(0) \end{cases} \\
[\![\mathbf{while}\ b\ \mathbf{do}\ C]\!]_C(s) &\triangleq \mathsf{lfp}(\Psi_{\langle [\![C]\!]_C, b\rangle})(s) \\
[\![a]\!]_C(s) &\triangleq [\![a]\!]_{act}(s)
\end{aligned}
$$

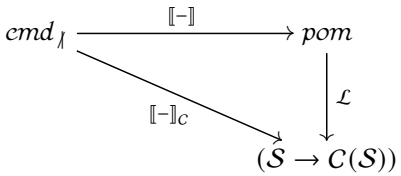
Fig. 6. Convex powerset semantics $[\![-]\!]_C : cmd_{/\!/} \to \mathcal{S} \to C(\mathcal{S})$ due to [He et al. 1997; McIver and Morgan 2005; Zilberstein et al. 2025b], where Ψ is defined in Lemma 6.4 and b is a pure test, meaning that for every s there exists $b_s \in \{0, 1\}$ such that $[\![b]\!]_{test}(s) = \eta(b_s)$.

7.2 Parallel-Free Semantics and Adequacy

Let $cmd_{/\!/}$ be the subset of commands from Figure 1 that do not contain parallel composition. Based on these operations, Figure 6 shows the convex powerset semantics for the parallel-free fragment of the language, $[\![-]\!]_C : cmd_{/\!/} \to \mathcal{S} \to C(\mathcal{S})$, which s is precisely the same semantics used in prior work for reasoning about programs that are both nondeterministic and probabilistic [He et al. 1997; McIver and Morgan 2005; Zilberstein et al. 2025b]. However, $[\![-]\!]_C$ does not capture parallel computation, which requires the more sophisticated semantic domains that we develop in this paper. Nonetheless, it is an important point of comparison to ensure that our pomset with formulae model does indeed give the desired meaning. The Scott continuity property above ensures that $\Psi_{\langle [\![C]\!]_C, b\rangle} : (\mathcal{S} \to C(\mathcal{S})) \to \mathcal{S} \to C(\mathcal{S})$ is Scott continuous, and therefore the semantics of loops is well defined.

We now arrive at our first adequacy theorem, showing that our pomset model recovers the standard probabilistic semantics $[\![C]\!]_C : \mathcal{S} \to C(\mathcal{S})$ shown in Figure 6, based on the convex powerset for a parallel-free fragment of our programming language.

Theorem 7.2 (Probabilistic Adequacy). *The following diagram commutes:*



In other words, $\mathcal{L}([\![C]\!]) = [\![C]\!]_C$ *for all* $C \in cmd_{/\!/}$.

Proof. By induction on the structure of C. The cases follow immediately from Lemma 6.4. □

7.3 Examples

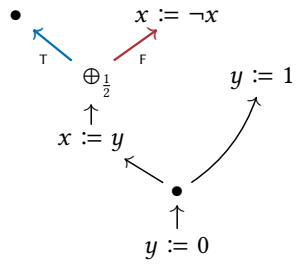
In this section, we provide two simple examples showing our semantics at work.

Example 7.3 (Entropy Mixer). Consider the entropy mixer program shown below:

$$\mathbf{EM} \quad \triangleq \quad y := 0; \left(x := y; (\mathbf{skip} \oplus_{\frac{1}{2}} x := \neg x) \;\middle|\; y := 1\right)$$

The **EM** program models a system where multiple sources of entropy are mixed, with the goal of obtaining a high quality source of randomness as output [Zilberstein et al. 2026]. The value of y is

influenced by the nondeterministic behavior of the scheduler and, although x is derived from y, we want to ensure that x is ultimately distributed uniformly. First, we give the pomset semantics of the program:

$[\![\mathbf{EM}]\!] =$ 

Next, we will derive its linearized meaning. Clearly, the $y := 0$ action must come first, so the linearization of the program applied to state s is equal to the linearization without $y := 0$ in state $s[y := 0]$:

$$\mathcal{L}\left([\![\mathbf{EM}]\!]\right)(s) = \mathcal{L}\left(\left[\!\left[x := y; (\mathbf{skip} \oplus_{\frac{1}{2}} x := \neg x) \;\middle|\; y := 1\right]\!\right]\right)(s[y := 0])$$

Next, we come to the fork node, and the scheduler chooses whether to execute $x := y$ or $y := 1$ as the next action. If $x := y$ is executed next, then the value of x will be 0, and $y := 1$ can be scheduled at any later time without affecting the outcome, as none of the remaining commands depend on y. If instead $y := 1$ is executed first, then there is only one possible schedule for the remaining commands, and x will be set to 1. So, we can reduce the linearization to the following nondeterministic choice:

$$= \mathcal{L}\left(\left[\!\left[\mathbf{skip} \oplus_{\frac{1}{2}} x := \neg x\right]\!\right]\right)\left(s\begin{bmatrix} x := 0 \\ y := 1 \end{bmatrix}\right) \,\&\, \mathcal{L}\left(\left[\!\left[\mathbf{skip} \oplus_{\frac{1}{2}} x := \neg x\right]\!\right]\right)\left(s\begin{bmatrix} x := 1 \\ y := 1 \end{bmatrix}\right)$$

Using Lemma 6.4, we can further reduce the sub-expressions to be simple probabilistic choices between two states.

$$= \left(\eta\left(s\begin{bmatrix} x := 0 \\ y := 1 \end{bmatrix}\right) \oplus_{\frac{1}{2}} \eta\left(s\begin{bmatrix} x := 1 \\ y := 1 \end{bmatrix}\right)\right) \,\&\, \left(\eta\left(s\begin{bmatrix} x := 1 \\ y := 1 \end{bmatrix}\right) \oplus_{\frac{1}{2}} \eta\left(s\begin{bmatrix} x := 0 \\ y := 1 \end{bmatrix}\right)\right)$$

Finally, since the expressions on either side of & are equivalent, and since the convex union is idempotent, then the entire linearization reduces to a single probabilistic choice:

$$= \eta\left(s\begin{bmatrix} x := 0 \\ y := 1 \end{bmatrix}\right) \oplus_{\frac{1}{2}} \eta\left(s\begin{bmatrix} x := 1 \\ y := 1 \end{bmatrix}\right)$$

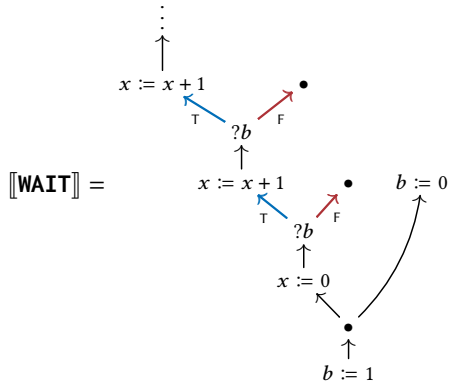
So, regardless of the scheduler's behavior, x will be uniformly distributed after running the program. The effects of nondeterminism introduced by parallel composition are completely annihilated by the later random choice.

Example 7.4 (Busy-Waiting and Fairness). In the following program, the left thread busy-waits (and counts its number of iterations) until the right thread releases a resource:

$$\mathbf{WAIT} \quad \triangleq \quad b := 1; \left(x := 0; \mathbf{while}\ b\ \mathbf{do}\ x := x + 1 \;\middle|\; b := 0\right)$$

We can use pomsets with formulae to determine the probability that the first thread will wait for a certain number of iterations n. First, we give the pomset semantics of the program, which is the

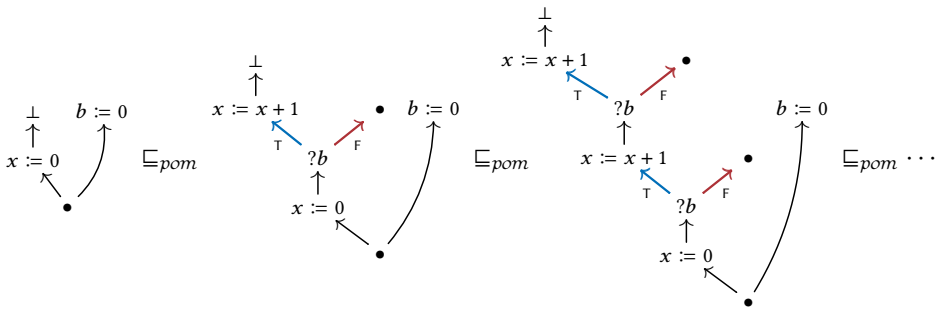
infinite structure below:



Now, we begin to derive the linearization as follows:

$$\begin{aligned}\mathcal{L}(\llbracket \textbf{WAIT} \rrbracket)(s) &= \mathcal{L}(\llbracket x := 0; \textbf{while } b \textbf{ do } x := x+1 \rrbracket \parallel \langle b := 0 \rangle)(s[b := 1]) \\ &= \mathcal{L}\left(\left(\langle x := 0 \rangle \fatsemi \sup_{n \in \mathbb{N}} \Phi^n_{\langle x := x+1, b \rangle}(\bot)\right) \parallel \langle b := 0 \rangle\right)(s[b := 1]) \\ &= \sup_{n \in \mathbb{N}} \mathcal{L}\left(\left(\langle x := 0 \rangle \fatsemi \Phi^n_{\langle x := x+1, b \rangle}(\bot)\right) \parallel \langle b := 0 \rangle\right)(s[b := 1])\end{aligned}$$

In the last step, we are able to move the supremum to the outside because $\mathcal{L}$ is Scott continuous, and because the supremum only affects nodes above level 2, so it commutes with $\parallel$ in the same way that $\ll_1$ does in Equation (7). So, the behavior of the program is equivalent to the supremum of linearizations of the following chain of finite approximations:



Let $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \ldots$ be the structures above. In each $\boldsymbol{\alpha}_n$, the scheduler could choose to always advance the first thread—and not the second one—until it reaches the $\bot$ node. So, $\bot_C \in \mathcal{L}(\boldsymbol{\alpha}_n)(s[b := 1])$ for all $n \in \mathbb{N}$; nontermination is a possible outcome of each finite approximation. Since supremum in the convex powerset corresponds to set intersection, then nontermination being an outcome of each finite approximation means that the scheduler can cause the full program to not terminate:

$$(\forall n \in \mathbb{N}.\ \bot_C \in \mathcal{L}(\boldsymbol{\alpha}_n)(s[b := 1])) \implies \bot_C \in \bigcap_{n \in \mathbb{N}} \mathcal{L}(\boldsymbol{\alpha}_n)(s[b := 1]) = \mathcal{L}(\llbracket \textbf{WAIT} \rrbracket)(s)$$

However, that defies standard intuitions about the **WAIT** program, which tells us that a reasonable scheduler will not only run the busy-wait thread. As a remedy, we need to ensure that the scheduler is *fair*, meaning that each enabled thread is never delayed for an infinite amount of time.

However fairness is generally not a compositional property, and it is well-known that it breaks Scott continuity of program operators [Apt and Plotkin 1986]. Intuitively, we can see the non-continuity in the chain above. In each $\boldsymbol{\alpha}_n$, nontermination occurs by delaying $b := 0$ for only a finite number of steps, therefore nontermination is a possible outcome in *every* finite approximation. But in $\sup_{n\in\mathbb{N}} \boldsymbol{\alpha}_n$, nontermination can only occur by delaying $b := 0$ for an *infinite* number of steps, which is not allowed due to fairness. That would mean that $\mathcal{L}(\sup_{n\in\mathbb{N}} \boldsymbol{\alpha}_n) \neq \sup_{n\in\mathbb{N}} \mathcal{L}(\boldsymbol{\alpha}_n)$, *i.e.,* $\mathcal{L}$ is not Scott continuous subject to a fair scheduler.

We will instead use a concept called *probabilistic fairness*, due to de Alfaro [1999]. Given a program with N threads, each of the N threads must be scheduled with some minimal probability $0 < \varepsilon \leq \frac{1}{N}$ on each step. Rather than being a global property on infinite traces, probabilistic fairness is a local property at each step of the program, and it does not break Scott continuity of $\mathcal{L}$. To make our model probabilistically fair, we simply use a new definition of nondeterministic choice, where we now additionally constrain $|I| \leq N$.

$$\mathop{\&}_{i\in I} S_i \triangleq \left\{ \sum_{i\in I} \mu(i) \cdot v_i \;\middle|\; \mu \in \mathcal{D}(I),\ \forall i \in I.\ \mu(i) \geq \varepsilon \wedge v_i \in S_i \right\}$$

That is, a nondeterministic choice over a finite index set I is the set of all convex combinations of elements from each S_i, where each $i \in I$ must occur with probability at least ε. We can now proceed to analyze the **WAIT** program, and determine bounds on the probability that the program terminates with $x = n$ for each $n \in \mathbb{N}$.

Following the fork, x is set to n on level $2n + 1$ of the first thread for every $n \in \mathbb{N}$. That means that **WAIT** will terminate with $x = n$ if the first thread was scheduled for the first $2n$ steps, and the second thread is scheduled on either step $2n + 1$ (immediately before x is set to n) or step $2n + 2$ (immediately after x is set to n), in which case the loop will exit before incrementing x again.

Let p_i be the probability that the first thread is scheduled at step i given that the second thread has not yet been scheduled. Following the reasoning above, the probability that $x = n$ is given by the following equation.

$$\mathbb{P}[x = n] = \left(\prod_{i=1}^{2n} p_i\right) \cdot (p_{2n+1}(1 - p_{2n+2}) + (1 - p_{2n+1})) = \left(\prod_{i=1}^{2n} p_i\right) \cdot (1 - p_{2n+1}p_{2n+2})$$

Given the probabilistic fairness assumption, each p_i is bounded between ε and $1 - \varepsilon$, so we can minimize $\mathbb{P}[x = n]$ by letting $p_i = \varepsilon$ for all $i \leq 2n$ and $p_{2n+1} = p_{2n+2} = (1 - \varepsilon)$. Similarly, we maximize $\mathbb{P}[x = n]$ by letting $p_i = 1 - \varepsilon$ for all $i \leq 2n$ and $p_{2n+1} = p_{2n+2} = \varepsilon$. That gives us:

$$\varepsilon^{2n}(1 - (1 - \varepsilon)^2) \leq \mathbb{P}[x = n] \leq (1 - \varepsilon)^{2n}(1 - \varepsilon^2)$$

To maximize the probability of nontermination, the scheduler should minimize the probability of ever running the second thread by setting $p_i = 1 - \varepsilon$ for all $i \in \mathbb{N}$. That allows us to bound the

termination probability as follows:

$$\begin{aligned}
\mathbb{P}[\mathsf{term}] &= \sum_{n\in\mathbb{N}} \mathbb{P}[x = n] \\
&= \sum_{n\in\mathbb{N}} \left(\prod_{i=1}^{2n} p_i\right) \cdot (1 - p_{2n+1}p_{2n+2}) \\
&\geq \sum_{n\in\mathbb{N}} (1-\varepsilon)^{2n} \cdot (1 - (1-\varepsilon)^2) \\
&= (1 - (1-\varepsilon)^2) \cdot \sum_{n\in\mathbb{N}} \left((1-\varepsilon)^2\right)^n \\
&= (1 - (1-\varepsilon)^2) \cdot \frac{1}{1 - (1-\varepsilon)^2} \quad = \quad 1
\end{aligned}$$

But since the probability of termination cannot be strictly greater than 1, then the program must terminate with probability exactly 1, regardless of how the scheduler behaves.

8 Purely Nondeterministic Concurrency

Pomset semantics for purely nondeterministic languages, without additional effects such as randomization, are well studied [Gischer 1988; Grabowski 1981; Pratt 1986]. As we saw in Section 2, such semantic models use pomsets that only record one form of branching: parallel branching and not guarded branching. To account for deterministic control flow (*i.e.,* if-statements and while-loops) sets of pomsets are used, where each structure in the set corresponds to a particular resolution of the tests in the program [Kappé et al. 2020; Laurence and Struth 2014] (see Figure 2 for an example). That approach relies on the problematic equation:

$$C; (C_1 \,\&\, C_2) \quad \equiv \quad (C; C_1) \,\&\, (C; C_2) \tag{8}$$

Although Equation (8) is generally accepted in purely nondeterminstic domains, it is invalid in domains that mix in probabilistic computation, such as the convex powerdomain. To see why, consider the case where C performs the probabilistic action $x :\approx \mathtt{flip}\left(\frac{1}{2}\right)$; then, the left side of the equation states that x is resolved *before* the nondeterministic choice is made, whereas the right side states that that the nondeterministic choice is made first. On the left, the scheduler has more power, as it can make its choice to schedule C_1 or C_2 based on the outcome of the coin flip; on the right, it cannot see the outcome of the coin flip until after the choice is made.

In terms of the semantic domains that we introduced in Section 6.2, Equation (8) corresponds to a second kind of additivity:

$$f^\dagger(d) = \bigamp_{i\in I} f_i^\dagger(d) \qquad \text{where} \qquad f(x) = \bigamp_{i\in I} f_i(x) \tag{9}$$

This kind of additivity is valid when the domain D does not include a secondary type of branching (like probabilistic choice), as in the case for *powerdomains*, which give semantics to nondeterministic programs. In this section, we will show how to recover pomset language semantics from our pomset with formulae model, and that the two models are equivalent when Equation (9) holds.

8.1 Pomset Language Semantics

We will emulate pomset language semantics using pomsets with formulae. Since in this section we are concerned with purely nondeterministic concurrency, we work in a restricted version of the language from Figure 1, called $cmd_{\not\oplus}$, that does not contain the $C_1 \oplus_p C_2$ construct; accordingly, we presume that $\oplus_p \notin \mathit{test}$. First, we define a label set $\mathit{label}_{\mathsf{PL}}$, which does not contain $\bot$ or tests,

$$
\begin{aligned}
[\![\mathbf{skip}]\!]_{\mathsf{PL}} &\triangleq \{\langle \mathsf{fork} \rangle\} \\
[\![a]\!]_{\mathsf{PL}} &\triangleq \{\langle a \rangle\} \\
[\![C_1; C_2]\!]_{\mathsf{PL}} &\triangleq [\![C_1]\!]_{\mathsf{PL}} \fatsemi [\![C_2]\!]_{\mathsf{PL}} \\
[\![C_1 \mid C_2]\!]_{\mathsf{PL}} &\triangleq [\![C_1]\!]_{\mathsf{PL}} \parallel [\![C_2]\!]_{\mathsf{PL}} \\
[\![C_1 \,\&\, C_2]\!]_{\mathsf{PL}} &\triangleq [\![C_1]\!]_{\mathsf{PL}} \cup [\![C_2]\!]_{\mathsf{PL}} \\
[\![\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2]\!]_{\mathsf{PL}} &\triangleq \left(\{\langle \mathbf{assume}\ b \rangle\} \fatsemi [\![C_1]\!]_{\mathsf{PL}}\right) \cup \left(\{\langle \mathbf{assume}\ \neg b \rangle\} \fatsemi [\![C_2]\!]_{\mathsf{PL}}\right) \\
[\![\mathbf{while}\ b\ \mathbf{do}\ C]\!]_{\mathsf{PL}} &\triangleq \mathsf{lfp}\left(\Xi_{\langle C,b \rangle}\right)
\end{aligned}
$$

$$
\begin{aligned}
\text{where} \quad \Xi_{\langle C,b \rangle}(S) &\triangleq \left(\{\langle \mathbf{assume}\ b \rangle\} \fatsemi [\![C]\!]_{\mathsf{PL}} \fatsemi S\right) \cup \left(\{\langle \mathbf{assume}\ \neg b \rangle\} \fatsemi [\![\mathbf{skip}]\!]_{\mathsf{PL}}\right) \\
S \fatsemi T &\triangleq \{\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \mid \boldsymbol{\alpha} \in S, \boldsymbol{\beta} \in T\} \\
S \parallel T &\triangleq \{\boldsymbol{\alpha} \parallel \boldsymbol{\beta} \mid \boldsymbol{\alpha} \in S, \boldsymbol{\beta} \in T\}
\end{aligned}
$$

Fig. 7. Standard pomset language semantics $[\![-]\!]_{\mathsf{PL}} : cmd_{\not\Phi} \to pom\mathcal{L}ang$.

but now act_{PL} includes an action $\mathbf{assume}\ b$ for every test $b \in test \setminus \{\&\}$, which guarantees that b holds in the current computation branch:

$$label_{\mathsf{PL}} \triangleq act_{\mathsf{PL}} \cup \{\mathsf{fork}\} \qquad \text{where} \qquad act_{\mathsf{PL}} \triangleq act \cup \{\mathbf{assume}\ b \mid b \in test \setminus \{\&\}\}$$

We omit $\bot$ because pomset languages are sets of finite pomsets, and therefore we can construct each structure directly without the need to approximate larger structures with $\bot$ nodes.

$$pom\mathcal{L}ang \triangleq \mathcal{P}(pom_{\mathsf{fin}}(label_{\mathsf{PL}}))$$

We now define standard pomset language semantics $[\![-]\!]_{\mathsf{PL}} : cmd_{\not\Phi} \to pom\mathcal{L}ang$ in Figure 7.

Although the semantics is technically defined using pomsets with tests, our restrictions to the label set make $pom\mathcal{L}ang$ isomorphic to the standard definition [de Bakker and Warmerdam 1990; Gischer 1988; Meyer and de Vink 1989; Pratt 1986]. The label set $label_{\mathsf{PL}}$ does not contain any tests; so, by the binary branching property (Definition 6.1), the formula of every node must be true, essentially meaning that the formulae in the LPOF structure are completely unused. Going further, the definition of sequential composition collapses to the standard one (for the proof, see Appendix G):

Lemma 8.1. *If* $\mathrm{Bot}_\alpha = \emptyset$, $\varphi_\alpha(x) = \mathsf{true}$ *for all* $x \in N_\alpha$, *and* $f = [\mathsf{true} \mapsto \beta]$, *then:*

$$N_{\alpha \fatsemi_f \beta} = N_\alpha \cup N_\beta \quad \text{and} \quad <_{\alpha \fatsemi_f \beta} = <_\alpha \cup <_\beta \cup (N_\alpha \times N_\beta)$$

In the pomset language model, each $\boldsymbol{\alpha} \in [\![C]\!]_{\mathsf{PL}}$ corresponds to a particular sequence of test resolutions. When an $\mathbf{assume}$ fails, the trace is eliminated. When a branch occurs, *i.e.,* in an if-statement or while-loop, the set of traces is duplicated to account for both the "true" and "false" branches, as we saw in Figure 2.

Loops are interpreted as the (possibly infinite) union of all finite traces. Infinite pomsets are not needed, since we instead use infinite sets. The semantics of loops is given by the least fixed point of $\Xi_{\langle C,b \rangle}$, which is Scott continuous in the DCPO $\langle pom\mathcal{L}ang, \subseteq \rangle$, since suprema (given by $\cup$) always exist, and are associative and commutative. The $\mathbf{skip}$ appears in the false branch to correspond to $\Phi_{\langle C,b \rangle}$ (defined in Figure 4).

8.2 The Hoare Powerdomain

Powerdomains give semantics to programs that combine nondeterminism with looping or recursion [Plotkin 1976; Smyth 1978]. In this section, we create a powerdomain instantiation of our semantics

from Section 6 and prove that the resulting model corresponds to well-known models of non-probabilistic concurrent programs [Kappé et al. 2020; Laurence and Struth 2014]. We work in the *Hoare* or *lower* powerdomain, named as such for its connection to partial correctness and Hoare Logic [Floyd 1967; Hoare 1969]. Indeed, the Hoare powerdomain identifies the terminating traces of a program, but not that the program *always* terminates; from a concurrency perspective, it corresponds to a *safety* property [Lamport 1977].

For simplicity, we presume that the orders over actions $\leq_{act}$ and tests $\leq_{test}$ in this section are flat. Then, the Hoare powerdomain consists of sets of states, ordered by subset inclusion $\langle \mathcal{P}(\mathcal{S}), \subseteq \rangle$, which is known to be a pointed DCPO with supremum given by union $\cup$ and $\emptyset$ as bottom. Let the remaining operators be defined as follows:

$$\mathop{\&}_{i \in I} S_i \triangleq \bigcup_{i \in I} S_i \qquad \eta(s) \triangleq \{s\} \qquad f^\dagger(S) \triangleq \bigcup_{s \in S} f(s)$$

Since $\oplus_p \notin \mathit{test}$, we do not define a $\oplus_p \colon \mathcal{P}(\mathcal{S})^2 \to \mathcal{P}(\mathcal{S})$ operation It is also well known that $\langle \mathcal{P}, \eta, (-)^\dagger \rangle$ is a Scott continuous, strict, and additive monad [Zilberstein et al. 2023]. In addition, $\mathcal{P}$ upholds Equation (9) since the set unions in $\&$ and $(-)^\dagger$ are commutative and associative. Now, action evaluation $[\![-]\!]_{act_{\mathsf{PL}}} : act_{\mathsf{PL}} \to \mathcal{S} \to \mathcal{P}(\mathcal{S})$ is given below, where **assume** actions result in $\{s\}$, if the test could nondeterministically be true in state s, and $\emptyset$, otherwise:

$$[\![a]\!]_{act_{\mathsf{PL}}}(s) \triangleq \begin{cases} [\![a]\!]_{act}(s) & \text{if } a \in act \\ \{s \mid 1 \in [\![b]\!]_{test}(s)\} & \text{if } a = \textbf{assume } b \end{cases}$$

We also define a specialized linearization operation $\mathcal{L}_{\mathsf{PL}} \colon \mathit{pom}\mathcal{L}\mathit{ang} \to \mathcal{S} \to \mathcal{P}(\mathcal{S})$, which is simply a union over the linearization of all traces:

$$\mathcal{L}_{\mathsf{PL}}(S)(s) \triangleq \bigcup_{\boldsymbol{\alpha} \in S} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha})(s)$$

Since all formulae are true, the next nodes to schedule are exactly those whose predecessors have all been processed, *i.e.,* $\mathrm{next}(\alpha, \mathsf{true}, S) = \{x \in N_\alpha \setminus S \mid x{\downarrow_\alpha} \subseteq S\}$, which corresponds to standard interleaving definitions of linearization [Pratt 1986].

8.3 Translation and Adequacy

We will now prove that pomset language semantics $[\![-]\!]_{\mathsf{PL}}$ (Figure 7) corresponds exactly to the Hoare powerdomain instance of our semantic model from Section 6. To do so, we first need to define a translation $\mathrm{tr} \colon \mathit{pom} \to \mathit{pom}\mathcal{L}\mathit{ang}$ from pomsets with formulae to pomset languages. To define tr, we start with a translation on finite LPOFS, which additionally takes a formula ψ to encode the resolution of tests in the current trace:

$$\mathrm{tr}_{lpof}(\alpha, \psi) \triangleq \langle N_\psi, <_\psi, \lambda_\psi, [- \mapsto \mathsf{true}] \rangle$$

where $[- \mapsto \mathsf{true}]$ is the constant true function and

$$N_\psi \triangleq \{x \in N_\alpha \mid \psi \Rightarrow \varphi_\alpha(x), \lambda_\alpha(x) \neq \&\} \qquad <_\psi \triangleq <_\alpha \cap (N_\psi \times N_\psi)$$

$$\lambda_\psi(x) \triangleq \begin{cases} \lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \notin \mathit{test} \\ \textbf{assume } \lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \in \mathit{test} \text{ and } \psi \Rightarrow x \\ \textbf{assume } \neg\lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \in \mathit{test} \text{ and } \psi \Rightarrow \neg x \end{cases}$$

The nodes of $\mathrm{tr}_{lpof}(\alpha, \psi)$ are those nodes $x \in N_\alpha$ such that $\psi \Rightarrow \varphi_\alpha(x)$ and $\lambda_\alpha(x) \neq \&$. The first condition, $\psi \Rightarrow \varphi_\alpha(x)$, indicates that x agrees with the outcomes of the tests described by the path condition ψ. Nodes with label $\&$ are deleted to avoid superfluous no-op nodes in the translated structure. Test nodes with label b are converted into either **assume** b or **assume** $\neg b$, depending on

whether that node is true or false in ψ. We define translation on finite pomsets by taking a union over all branches $\psi \in \mathsf{br}_\alpha$:

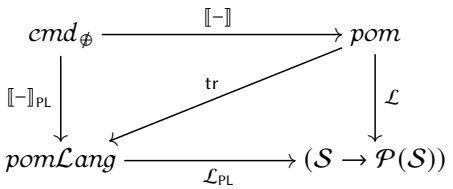
$$\mathsf{tr}_{\mathsf{fin}}([\alpha]) \triangleq \{[\mathsf{tr}_{lpof}(\alpha, \psi)] \mid \psi \in \mathsf{br}_\alpha\}$$

Finite translation is monotonic (Lemma G.2)—$\mathsf{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}) \subseteq \mathsf{tr}_{\mathsf{fin}}(\boldsymbol{\beta})$ if $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$—therefore we can extend to infinite pomsets with the extension lemma:

$$\mathsf{tr}(\boldsymbol{\alpha}) \triangleq \mathsf{tr}^*_{\mathsf{fin}}(\boldsymbol{\alpha})$$

Now, with the definition of tr, we can state our next adequacy theorem, showing that our pomset with formulae model corresponds exactly to the usual concurrency semantics in the Hoare powerdomain; the proof is in Appendix G.5.

Theorem 8.2 (Nondeterministic Adequacy). *The following diagram commutes:*



The upper commuting triangle does not depend on the Hoare powerdomain, so it is tempting to say that pomset languages give an adequate model in other domains too. However, as we showed in Section 2, just because a semantic structure exists does not mean that it conveys the desired meaning. Indeed, the fact that $\mathcal{L} = \mathcal{L}_{\mathsf{PL}} \circ \mathsf{tr}$ relies on two particular properties of the Hoare powerdomain. First, Equation (9), which is invalid in probabilistic domains. Second, $\mathcal{L}_{\mathsf{PL}}$ is computed as a union over a possibly infinite set (the union is infinite whenever the program contains a while-loop). Unbounded nondeterminism is known to cause problems in many domains—including the Smyth and Plotkin powerdomains [Apt and Plotkin 1986; Back 1980; Søndergaard and Sestoft 1992]—thus no infinitary version of & exists.

So, although programs in any domain *can* be interpreted as pomset languages, those semantic objects may not give an adequate meaning to the program. Particularly in probabilistic domains—*i.e.,* the convex powerset—prior pomset language models [Kappé et al. 2020; Laurence and Struth 2014] do not give us a way to find the relative probabilities of the different outcomes after actually running the program. Our new approach, using a single structure that records both causality and control flow, is more suitable for interpreting the semantics of concurrent programs in new domains, such as probabilistic computation. As we saw in Section 2 and Theorem 7.2, pomsets with formulae accurately capture the concurrent behavior of these programs, while also allowing us to recover the precise probabilities of each outcome.

9 The Randomized Dining Philosophers

As a more complex case study, we show how pomsets with formulae can be used to reason about Dijkstra's Dining Philosophers problem [Dijkstra 1971]. In a nutshell, n philosophers sit around a circular table and alternately think and eat; each philosopher has a plate and there is a chopstick between each adjacent pair of plates. Two chopsticks are needed to eat, but the philosophers can only take the two chopsticks next to their plates and chopsticks can only be taken one at a time. So, the philosophers can eat only when their two nearest neighbors are not eating. After eating, the philosopher will put down both chopsticks and either think further, or die.

The problem is to design a symmetric algorithm such that any philosopher will not starve (*i.e.,* try to acquire chopsticks forever) or, at the very least, no deadlock arises. Although the Dining

$$\mathbf{Phil}_i \triangleq \left\{\begin{array}{ll}
1 & \textbf{while } \& \textbf{ do} \\
2 & \quad \textbf{think}(); \\
3 & \quad got_i := 0; \\
4 & \quad \textbf{while } \neg got_i \textbf{ do} \\
5 & \quad\quad x_i :\approx \textbf{flip}\left(\frac{1}{2}\right); \\
6 & \quad\quad \textbf{while } \neg got_i \textbf{ do} \\
7 & \quad\quad\quad got_i := \textbf{cas}(ch[i +_n x_i], 1, 0); \\
8 & \quad\quad got_i := \textbf{cas}(ch[i +_n (1 - x_i)], 1, 0); \\
9 & \quad\quad \textbf{cas}(ch[i +_n x_i], \neg got_i, 1); \\
10 & \quad \textbf{eat}(); \\
11 & \quad ch[i] := 1; \\
12 & \quad ch[i +_n 1] := 1; \\
13 & \textbf{die}()
\end{array}\right.$$

Fig. 8. The randomized solution to the Dining Philosophers due to Lehmann and Rabin [1981], implemented in the language of Figure 1. Program $\mathbf{Phil}_i$ implements the protocol of philosopher P_i.

Philosophers may seem like a silly thought exercise, the paradigm of symmetric choice is applicable to a large number of real world synchronization problems. As a result, the Dining Philosophers has been studied for decades, and continues to be relevant today.

Dijkstra [1971] showed that no symmetric, deterministic, and fully distributed solution to the Dining Philosophers problem exists. However, the problem has a simple *probabilistic* solution due to Lehmann and Rabin [1981], where random choices are used to break symmetries and the probability of deadlock converges to 0 as time goes on. Lehmann and Rabin's [1981] solution is represented as pseudocode, and their proof is not based on a formal semantics. In fact, due to the interplay between concurrent scheduling, random choice, and unbounded looping, no semantic model prior to pomsets with formulae could adequately capture the randomized Dining Philosophers.

9.1 The Dining Philosophers Program

In Figure 8, we show the randomized solution to the Dining Philosophers problem, adapted from Lehmann and Rabin [1981]. A few aspects deserve discussion. First, philosophers are numbered from P_0 to P_{n-1} and so are the chopsticks: we let the chopstick at the left and at the right of philosopher i be $ch[i]$ and $ch[i +_n 1]$, respectively (where the notation '$+_n$' to denotes addition modulo n). Every chopstick is modeled as a boolean variable, which holds 1 if the chopstick is available and 0 if it is taken. Every philosopher has two variables in its local state, x_i and got_i, where i is the philosopher's index. The variable x_i is a boolean variable used to indicate which chopstick P_i will try to grab first, with 0 indicating left and 1 indicating right. The variable got_i is also Boolean-valued, and indicates whether P_i successfully obtained a chopstick.

The philosophers also perform the actions **think**(), **eat**(), and **die**(). For the purposes of our analysis, we let those actions have the same behavior as **skip** and we include them in the program only to make it clear how our implementation relates to the classic problem.

The program $\mathbf{Phil}_i$, implementing P_i's protocol, begins on line 1 of Figure 8 with a top-level while loop over the test &, meaning that, at every iteration, P_i will arbitrarily choose whether to keep going or die. Once inside the loop, P_i begins to think (which has no observable effect), then becomes hungry and tries to acquire a chopstick. First, got_i is set to 0 on line 3, indicating that

P_i has no chopsticks. Then, on line 5, $\mathbf{flip}\left(\frac{1}{2}\right)$ equiprobably selects a target chopstick (either the left or right one). Once a target $ch[i +_n x_i]$ has been set, P_i loops on line 6 until it acquires that chopstick. The acquisition must be done atomically, so that no other philosopher can take the chopstick in between P_i's check that $ch[i +_n x_i]$ is available and the state update to claim it. To achieve atomicity, we introduce a new compare-and-set action $b \coloneqq \mathbf{cas}(x, old, new)$, which sets in one atomic step x to new if its current value is old. The variable b indicates whether the operation was successful, but it can be omitted if we do not need to know whether the operation succeeded:

$$[\![b \coloneqq \mathbf{cas}(x, old, new)]\!]_{act}(s) \triangleq \begin{cases} \eta(s[b \coloneqq 1][x \coloneqq new]) & \text{if } s(x) = old \\ \eta(s[b \coloneqq 0]) & \text{if } s(x) \neq old \end{cases}$$

Once P_i has acquired $ch[i+_n x_i]$, it then turns its attention to the second chopstick $ch[i+_n (1-x_i)]$ on line 8. However, this time P_i does not loop, it simply takes the chopstick if it is available. Following line 8, got_i is 1 if P_i is holding two chopsticks and 0 if it is only holding $ch[i +_n x_i]$. So, the **cas** operation on line 9 frees the first chopstick ($ch[i +_n x_i]$) only if P_i failed to acquire the second chopstick ($ch[i +_n (1 - x_i)]$). At this point, P_i will exit the loop on line 4 if it has acquired two chopsticks, otherwise it has no chopsticks and will try again. If the loop on line 4 exits, then P_i gets to eat on line 10 (again, **eat**() has no observable effect). After eating, P_i releases both of its chopsticks and once again arbitrarily chooses whether to think again or die.

9.2 Pomset Semantics and Deadlock Freedom

We now show how pomsets with formulae can be used to reason about the Dining Philosophers program. The structure corresponding to $[\![\mathbf{Phil}_i]\!]$ is shown in Figure 9. For simplicity, we use dashed back arrows $x \dashleftarrow y$ to indicate that the entire structure rooted at x is copied and placed as a successor to y. In reality, $[\![\mathbf{Phil}_i]\!]$ is an infinite structure and each of the three nested loops repeats indefinitely, as we saw in Section 6.1 and Example 7.4. We want to remark that, as this example testifies, despite the complex rules and operations for compositionally defining pomsets out of programs, the correspondence between a program (see Figure 8) and its associated pomset (see Figure 9) is quite simple and evident.

To obtain the overall system, we initialize all chopsticks to be available and then run the protocols of $P_0, \ldots P_{n-1}$ in parallel:

$$ch[0] \coloneqq 1; \ldots; ch[n-1] \coloneqq 1; (\mathbf{Phil}_0 \mid \ldots \mid \mathbf{Phil}_{n-1})$$

The fundamental property required for any solution to the Dining Philosophers is that no deadlock arises: if at some point at least one philosopher is active (has not terminated), eventually at least one of them eats. We now use the pomset presented in Figure 9 to conclude that the probability of deadlock tends to 0, provided some basic assumption on the fairness of the scheduler (that is, each enabled thread will eventually get run, which is also assumed by Lehmann and Rabin [1981]). For simplicity, we shall first consider the case with only two philosophers—P_0 and P_1—and prove that the infinite traces containing no **eat**() actions have probability 0. At the end of the section, we will discuss how the reasoning can be extended to more than two philosophers.

From Figure 9, it is clear that there are two places where a philosopher P_i can indefinitely run without ever encountering an **eat**(). The first scenario (shown on the left of Figure 10) is where P_i repeatedly acquires one chopstick, but always fails to acquire the second chopstick. The second scenario (shown on the right of Figure 10) is where P_i gets stuck trying to acquire its first chopstick, after succeeding to acquire one chopstick a finite number of times.

Thus, the only possible traces of the overall system in which no **eat**() action occurs are those obtained by interleaving the individual traces just shown, respectively performed by $\mathbf{Phil}_0$ and $\mathbf{Phil}_1$. For simplicity, let us focus on $\mathbf{Phil}_0$ (the argument holds dually for $\mathbf{Phil}_1$).

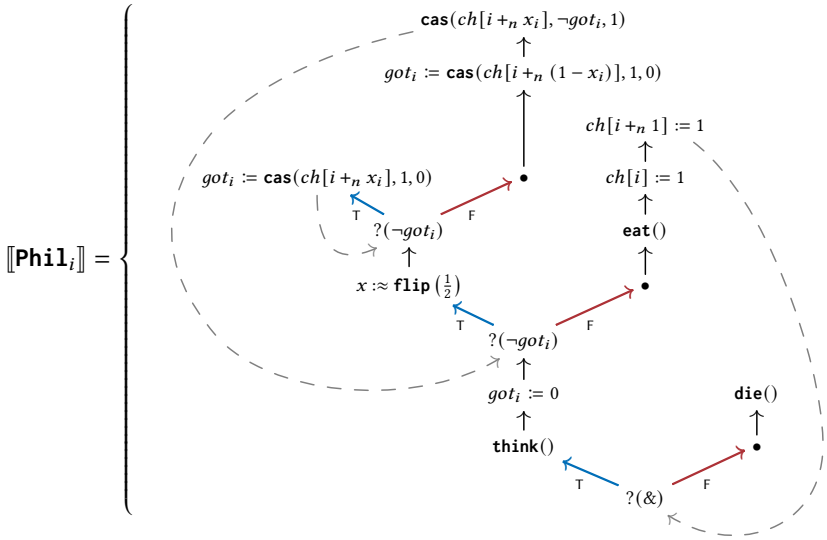


Fig. 9. Pomset corresponding to $[\![\mathbf{Phil}_i]\!]$ from Figure 8. The dashed back arrows $x \dashleftarrow y$ indicate that a copy of the entire structure rooted at x is a successor of y.

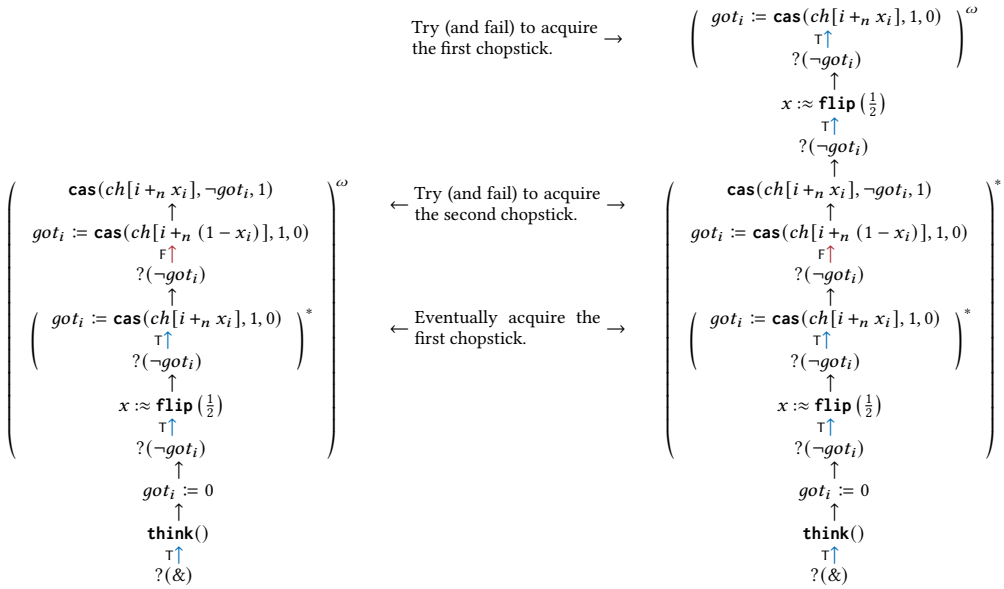


Fig. 10. Traces extracted from the pomset $[\![\mathbf{Phil}_i]\!]$ of Figure 9 where no **eat**() action occurs.

First, consider the case in which $\mathbf{Phil}_0$ contributes to the system evolution by performing the second trace (Figure 10 right), *i.e.*, at some moment t it starts looping forever on lines 6 and 7 of Figure 8. This can only happen if, from that moment on, $\mathbf{Phil}_0$ tries to access the chopstick i ($\in \{0, 1\}$) and it never finds it available; moreover, being stuck in the loop, $\mathbf{Phil}_0$ does not hold chopstick $1 - i$, that is thus permanently available from time t on. By assuming that every action terminates in a finite amount of time and that there are no process failures, this implies that $\mathbf{Phil}_1$ remains active forever (i.e., it never dies in line 13) and

- $\mathbf{Phil}_1$ got chopstick i before time t, and hence it is not performing lines 4-7; hence, it will eventually eat; and
- $\mathbf{Phil}_1$ releases chopstick i (by performing one of the actions in lines 11/12), but gets it again, by starting the **while** loop of line 4, before that $\mathbf{Phil}_0$ is able to get chopstick i.

Since the loop of $\mathbf{Phil}_0$ is infinite, this means that this scenario repeats infinitely often and, hence, $\mathbf{Phil}_1$ eats infinitely many times, thus satisfying deadlock freedom.

Second, consider the case in which $\mathbf{Phil}_0$ contributes to the system evolution by performing the first trace (Figure 10 left); that is, it remains trapped in the **while** of line 4. This means that, at some moment t, it repeatedly selects a chopstick i (uniformly chosen in the set $\{0, 1\}$), it is able to get it (possibly, after some attempts), but it then finds chopstick $1 - i$ unavailable and has to restart everything from the beginning (including the random selection of the chopstick, for a possibly different i). Again, this implies that $\mathbf{Phil}_1$ remains active forever (otherwise at some point the second chopstick would eventually become available for $\mathbf{Phil}_0$) and it performs the 'dual' trace of $\mathbf{Phil}_0$, thus repeatedly getting chopstick $1 - i$ but then finding chopstick i unavailable. However, this may only happen if, at every attempt, $\mathbf{Phil}_0$ and $\mathbf{Phil}_1$ select different chopsticks in their random choice of line 5; these choices being independent, the fact that they always choose different chopsticks in k successive iterations is $\frac{1}{2^k}$, and so tends to 0 with k going to infinity.

To conclude, Lehmann and Rabin [1981] prove something similar, i.e., that no trace can indefinitely avoid an **eat** action; however, the two proofs do not follow the same conceptual steps. Our reasoning is, in our opinion, much easier to follow and verify, since it provides in a single figure (viz., Figure 9) all the possible behaviors, making it simpler to extract the possible traces where **eat** actions are not present (see Figure 10) and reason on why they can happen with a probability tending to 0.

Our reasoning can be adapted to handle an arbitrary number of philosophers n. We should consider $\mathbf{Phil}_i$ and $\mathbf{Phil}_{(i+_n 1)}$ instead of $\mathbf{Phil}_0$ and $\mathbf{Phil}_1$. Then, we cannot directly conclude that, if $\mathbf{Phil}_i$ is blocked, then $\mathbf{Phil}_{(i+_n 1)}$ will surely eat: if it does, we're done; otherwise, if $\mathbf{Phil}_{(i+_n 1)}$ is blocked as well, we move the focus on $\mathbf{Phil}_{(i+_n 2)}$, and so on. In the worst case, we would arrive back at $\mathbf{Phil}_{(i+_n n)} = \mathbf{Phil}_i$, meaning that everyone is blocked. But that scenario can only happen if all philosophers happen to infinitely often get the specific combinations of coin flips that cause them to acquire exactly one chopstick, with the second one taken by their neighbor. The probability of that scenario repeating indefinitely clearly tends to 0.

10 Related Work

We developed a new semantic structure called *pomsets with formulae* and studied its domain-theoretic properties, which we used as basis to provide semantics for a probabilistic concurrent imperative language with unbounded loops. Our work builds on a rich history of both pomset and powerdomain semantics, and also contrasts with other approaches, as we now describe.

Concurrency Semantics. The use of partial orders and pomsets in denotational models of concurrent programs dates back to Gischer [1988]; Grabowski [1981]; Pratt [1976]. Pomsets have subsequently been used as a semantic basis for concurrent separation logic [Brookes 2004], concurrent Kleene algebras [Hoare et al. 2011; Jipsen and Moshier 2016; Kappé et al. 2020; Laurence and Struth 2014], and other applications [Edixhoven et al. 2022; Jagadeesan et al. 2020; Kavanagh and Brookes 2018]. While there are links between pomset semantics and operational models of concurrency [Brookes 2002], pomsets are often preferred for their finer notion of concurrency compared to the interleaving semantics offered by operational models, which is especially useful in the context of weak memory [Jagadeesan et al. 2020; Kavanagh and Brookes 2018].

Concurrency has previously been combined with other computational effects in limited ways. For example, semantics for probabilistic concurrent programs have been defined operationally

[Tassarotti 2018; Tassarotti and Harper 2019], but that approach is limited to bounded loops due to the lack of domain-theoretic tools to determine the behavior in the limit. The inclusion of unbounded looping adds significant complexity in the probabilistic case. Whereas finite-trace models are sufficient for many non-probabilistic scenarios, unbounded looping in probabilistic programs requires infinite traces, as the probability of termination may only become 1 in the limit. A proper theory of infinite traces requires pomsets to be enriched with a DCPO or metric structure; prior work in this area was very informative to our own development [de Bakker and Warmerdam 1990; Meyer and de Vink 1988, 1989], although the inclusion of deterministic branching in our own pomset structure added significant complexity.

Another operational model of probabilistic concurrency has been developed based on Markov Decision Processes [Fesefeldt et al. 2022]. This semantics is used to lower bound expected values, but does not straightforwardly extend to more complex domains such as the convex powerset, which give the full set of distributions over outcomes. In addition, a denotational model has been developed [Neves 2025], where the underlying semantic structure is obtained as the solution to a domain equation. Our construction is more concrete, giving a full account of the causality in the program and thus capturing non-interleaving models of concurrency too.

Branching pomsets were recently introduced to model choices in choreographic programs [Edixhoven and Jongmans 2022; Edixhoven et al. 2024, 2022]. While branching pomsets can, in theory, contain infinitely many nodes, the domain theoretic properties needed to approximate infinite structures have not been explored, which was a significant focus of this paper. Indeed, our extension lemma (Lemma 4.17) was necessary for linearization, without which we would not have been able to relate our pomset model to the known convex powerset semantics [He et al. 1997; McIver and Morgan 2005; Zilberstein et al. 2025b].

Powerdomain Semantics. Concurrency semantics must be interpreted in a domain that captures nondeterministic computation in order to account for different interleavings of the parallel threads. Powerdomains have long been used to give denotational semantics to nondeterministic programs [Plotkin 1976; Smyth 1978], and were later extended to probabilistic semantics as well [Jones and Plotkin 1989]. However, as documented by Parlant [2020]; Varacca and Winskel [2006]; Zwart [2020]; Zwart and Marsden [2019], standard powerdomains do not compose with probabilistic ones in a straightforward way, but several alternatives have been explored.

In this paper, we use the *convex powerdomain* [He et al. 1997; Mislove 2000; Morgan et al. 1996b; Tix 1999], which operationally represents a probabilistic program with nondeterminism resolved by a probabilistic scheduler [Varacca 2002, Theorem 6.12]. The convex powerdomain additionally has a well-defined monad structure [Jacobs 2008] and good domain-theoretic properties [Keimel and Plotkin 2017; McIver and Morgan 2005; Tix et al. 2009].

The *powerdomain of indexed valuations* is an alternative computational domain, which instead models probabilistic programs with nondeterminism resolved by a deterministic scheduler [Varacca 2002; Varacca and Winskel 2006]. Distributions of multisets can also represent probabilistic nondeterminism [Jacobs 2021; Kozen and Silva 2024] with an angelic interpretation of nondeterminism [Ong et al. 2025]. Further denotational models represent an *oblivious* scheduler, which makes decisions independently of random coin flips [Chen et al. 2026] or even resolve nondeterminism before seeing the program inputs [Wang et al. 2019].

Although we chose to focus on the convex powerdomain in this paper, the core pomsets with formulae model is agnostic to semantic domains. Those domains only become relevant during linearization, and our linearization procedure is defined generically so as to support a variety of models. Many of the aforementioned domains, particularly the powerdomain of indexed valuations, are already known to have the properties needed for linearization.

Alternative Approaches. The approach taken in this paper was motivated by the need to obtain a full set of possible distributions that result from running a program. Alternative approaches exist, but are focused more on conflicts and interactions between individual program actions.

A first example is the well established model of *event structures* [Nielsen et al. 1981; Winskel 1980], whose main feature is the possibility of explicitly representing causality and conflict. This makes it possible to constrain the possible executions, whereas pomsets provide a compact representation of concurrent behavior by recording causal dependencies while leaving concurrent events unordered. To model randomized computations, probabilistic event structures have been devised [Katoen 1996; Varacca et al. 2006; Varacca and Yoshida 2007; Winskel 2014]: they globally encode all of the interactions in a probabilistic system. In general, while event structures have expressive features to represent control flow and synchronization, they are less compositional than pomsets and are less amenable to linearization. In addition, event structures model process algebras, which have a more restricted syntax than the full imperative language of Figure 1. Most notably, process algebras usually do not have a general purpose sequential composition.

Another approach is the research line on *game semantics* [Abramsky et al. 2000] and its probabilistic extensions [Castellan et al. 2018]. In this setting, program behaviors are modeled as interactions between a player and an environment, with game structure capturing the possible choices. Game semantics are particularly useful for reasoning about program equivalence and full abstraction, whereas pomsets provide a natural representation of concurrent executions whose linearizations can be used to characterize observable program behaviors.

11 Conclusion and Future Work

We presented a novel denotational model for probabilistic and concurrent imperative programs. Our model solves a longstanding problem in the field of program semantics by developing a structure that correctly captures interactions between control flow, probabilistic actions, concurrent execution, and unbounded looping. Our model has good domain theoretic properties, important for semantics of unbounded loops, and it subsumes typical powerdomain semantics for concurrency as well as convex powerdomain semantics for probabilistic nondeterminism.

Ultimately, the existence of a good denotational model will be key to verification of probabilistic and concurrent imperative programs, which can be used to capture important protocols in randomized distributed systems, such as the Dining Philosophers. We have taken preliminary steps through the development of a program logic, which we call probabilistic concurrent Outcome Logic (pcOL) [Zilberstein et al. 2026]. The main innovation of pcOL is its use of a concurrent separation logic [O'Hearn 2004] style parallel composition rule to infer specifications about parallel programs in ways that are not captured by Lemma 6.4. However, pcOL has limited abilities to reason about concurrent loops whose termination are mutually dependent, making the Dining Philosophers and other similar examples out of reach [Hart et al. 1983; Lehmann and Rabin 1981; Rabin 1980, 1982]. We are interested in developing further logical methods based on pomsets with formulae in order to extend mechanized reasoning to a broader class of probabilistic concurrent programs.

Another interesting direction is to explore the applicability of pomsets with formulae to convex powerdomain constructions other than Smyth [Keimel and Plotkin 2017; Tix 1999, 2000; Tix et al. 2009], or to other domains for mixing probabilities and nondeterminism including indexed valuations [Varacca 2002, 2003; Varacca and Winskel 2006] and multisets of distributions [Jacobs 2021; Kozen and Silva 2024]. While our linearization procedure is based on an *omniscient* scheduler, we are also interested in exploring more restricted models including oblivious schedulers (which cannot see the outcomes of random sampling) [Chen et al. 2026], and fair schedulers. This will be challenging, as those models are non-compositional and therefore linearization on infinite pomsets could not be defined via Lemma 4.17. Still, pomsets with formulae are a good basis for studying

these questions, as the structure itself makes no assumptions about how scheduling of parallel branches occurs.

References

Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. 2000. Full Abstraction for PCF. *Information and Computation* 163, 2 (2000), 409–470. https://doi.org/10.1006/inco.2000.2930

Samson Abramsky and Achim Jung. 1995. *Domain theory.* Oxford University Press, Inc., USA, 1–168.

Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, Martin Michelsen, Riley Patterson, Eric Rodriguez, Laura Scharff, Vikram Shanker, Vladmir Sicca, Hari Sowrirajan, Aidan Swope, Matyas Tamas, Vlad Tenev, Jonathan Thomm, Harold Williams, and Lawrence Wu. 2025. Aristotle: IMO-level Automated Theorem Proving. arXiv:2510.01346 [cs.AI] https://arxiv.org/abs/2510.01346

Krzysztof Apt and Gordon Plotkin. 1986. Countable nondeterminism and random assignment. *J. ACM* 33, 4 (aug 1986), 724–767. https://doi.org/10.1145/6490.6494

Steve Awodey. 2006. *Category Theory.* Ebsco Publishing.

Ralph-Johan Back. 1980. Semantics of unbounded nondeterminism. In *Automata, Languages and Programming*, Jaco de Bakker and Jan van Leeuwen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 51–63. https://doi.org/10.1007/3-540-10003-2_59

Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. An Assertion-Based Program Logic for Probabilistic Programs. In *Programming Languages and Systems*. Springer International Publishing, Cham, 117–144. https://doi.org/10.1007/978-3-319-89884-1_5

Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. 2021. Presenting Convex Sets of Probability Distributions by Convex Semilattices and Unique Bases. In *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 211)*, Fabio Gadducci and Alexandra Silva (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 11:1–11:18. https://doi.org/10.4230/LIPIcs.CALCO.2021.11

Stephen Brookes. 2002. Traces, Pomsets, Fairness and Full Abstraction for Communicating Processes. In *CONCUR 2002 — Concurrency Theory*, Luboš Brim, Mojmír Křetínský, Antonín Kučera, and Petr Jančar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 466–482. https://doi.org/10.1007/3-540-45694-5_31

Stephen Brookes. 2004. A Semantics for Concurrent Separation Logic. In *CONCUR 2004 - Concurrency Theory*, Philippa Gardner and Nobuko Yoshida (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 16–34. https://doi.org/10.1007/978-3-540-28644-8_2

Simon Castellan, Pierre Clairambault, Hugo Paquet, and Glynn Winskel. 2018. The concurrent game semantics of Probabilistic PCF. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (Oxford, United Kingdom) *(LICS '18)*. Association for Computing Machinery, New York, NY, USA, 215–224. https://doi.org/10.1145/3209108.3209187

Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. 2007. Paxos made live: an engineering perspective. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing* (Portland, Oregon, USA) *(PODC '07)*. Association for Computing Machinery, New York, NY, USA, 398–407. https://doi.org/10.1145/1281100.1281103

Hanxi Chen, Noam Zilberstein, Andrew C. Myers, and Alexandra Silva. 2026. Oblivious Probabilistic Outcome Logic: Verifying Probabilistic Programs with an Oblivious Adversary. arXiv:2607.16533 [cs.PL] https://arxiv.org/abs/2607.16533

Luca de Alfaro. 1999. From Fairness to Chance. *Electronic Notes in Theoretical Computer Science* 22 (1999), 55–87. https://doi.org/10.1016/S1571-0661(05)80597-3 PROBMIV'98, First International Workshop on Probabilistic Methods in Verification.

J. W. de Bakker and J. H. A. Warmerdam. 1990. Metric pomset semantics for a concurrent language with recursion. In *Semantics of Systems of Concurrent Processes*, Irène Guessarian (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 21–49. https://doi.org/10.1007/3-540-53479-2_2

Jerry den Hartog. 1999. Verifying Probabilistic Programs Using a Hoare like Logic. In *Advances in Computing Science — ASIAN'99*, P. S. Thiagarajan and Roland Yap (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 113–125. https://doi.org/10.1007/3-540-46674-6_11

Edsger W. Dijkstra. 1971. Hierarchical ordering of sequential processes. *Acta Inf.* 1, 2 (June 1971), 115–138. https://doi.org/10.1007/BF00289519

Luc Edixhoven and Sung-Shik Jongmans. 2022. Realisability of Branching Pomsets. In *Formal Aspects of Component Software*, Silvia Lizeth Tapia Tarifa and José Proença (Eds.). Springer International Publishing, Cham, 185–204. https://doi.org/10.1007/978-3-031-20872-0_11

Luc Edixhoven, Sung-Shik Jongmans, José Proença, and Ilaria Castellani. 2024. Branching pomsets: Design, expressiveness and applications to choreographies. *Journal of Logical and Algebraic Methods in Programming* 136 (2024), 100919. https://doi.org/10.1016/j.jlamp.2023.100919

Luc Edixhoven, Sung-Shik Jongmans, José Proença, and Guillermina Cledou. 2022. Branching Pomsets for Choreographies. *Electronic Proceedings in Theoretical Computer Science* 365 (Aug. 2022), 37–52. https://doi.org/10.4204/eptcs.365.3

Ira Fesefeldt, Joost-Pieter Katoen, and Thomas Noll. 2022. Towards Concurrent Quantitative Separation Logic. In *33rd International Conference on Concurrency Theory (CONCUR 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 243)*, Bartek Klin, Sławomir Lasota, and Anca Muscholl (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 25:1–25:24. https://doi.org/10.4230/LIPIcs.CONCUR.2022.25

Robert W. Floyd. 1967. Assigning Meanings to Programs. In *Mathematical Aspects of Computer Science (Proceedings of Symposia in Applied Mathematics, Vol. 19)*. American Mathematical Society, Providence, Rhode Island, 19–32. https://doi.org/10.1007/978-94-011-1793-7_4

Jay L. Gischer. 1988. The equational theory of pomsets. *Theoretical Computer Science* 61, 2 (1988), 199–224. https://doi.org/10.1016/0304-3975(88)90124-7

Jan Grabowski. 1981. On Partial Languages. *Fundamenta Informaticae* 4, 2 (1981), 427–498. https://doi.org/10.3233/fi-1981-4210

Sergiu Hart, Micha Sharir, and Amir Pnueli. 1983. Termination of Probabilistic Concurrent Program. *ACM Trans. Program. Lang. Syst.* 5, 3 (July 1983), 356–380. https://doi.org/10.1145/2166.357214

Jifeng He, Karen Seidel, and Annabelle McIver. 1997. Probabilistic models for the guarded command language. *Science of Computer Programming* 28, 2 (1997), 171–192. https://doi.org/10.1016/S0167-6423(96)00019-6 Formal Specifications: Foundations, Methods, Tools and Applications.

Charles Antony Richard Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (Oct. 1969), 576–580. https://doi.org/10.1145/363235.363259

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

Bart Jacobs. 2008. Coalgebraic Trace Semantics for Combined Possibilitistic and Probabilistic Systems. *Electronic Notes in Theoretical Computer Science* 203, 5 (2008), 131–152. https://doi.org/10.1016/j.entcs.2008.05.023 Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science (CMCS 2008).

Bart Jacobs. 2021. From Multisets over Distributions to Distributions over Multisets. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science* (Rome, Italy) *(LICS '21)*. Association for Computing Machinery, New York, NY, USA, Article 39, 13 pages. https://doi.org/10.1109/LICS52264.2021.9470678

Radha Jagadeesan, Alan Jeffrey, and James Riely. 2020. Pomsets with preconditions: a simple model of relaxed memory. *Proc. ACM Program. Lang.* 4, Oopsla, Article 194 (Nov. 2020), 30 pages. https://doi.org/10.1145/3428262

Peter Jipsen and M. Andrew Moshier. 2016. Concurrent Kleene algebra with tests and branching automata. *Journal of Logical and Algebraic Methods in Programming* 85, 4 (2016), 637–652. https://doi.org/10.1016/j.jlamp.2015.12.005 Relational and algebraic methods in computer science.

Claire Jones and Gordon Plotkin. 1989. A Probabilistic Powerdomain of Evaluations. In *Fourth Annual Symposium on Logic in Computer Science*. 186–195. https://doi.org/10.1109/lics.1989.39173

C. B. Jones. 1983. Tentative steps toward a development method for interfering programs. *ACM Trans. Program. Lang. Syst.* 5, 4 (Oct. 1983), 596–619. https://doi.org/10.1145/69575.69577

Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. 2020. *Concurrent Kleene Algebra with Observations: From Hypotheses to Completeness*. Springer International Publishing, 381–400. https://doi.org/10.1007/978-3-030-45231-5_20

Joost-Pieter Katoen. 1996. *Quantitative and Qualitative Extensions of Event Structures*. PhD Thesis - Research UT, graduation UT. University of Twente, Netherlands. https://doi.org/10.3990/1.9789036507998

Ryan Kavanagh and Stephen Brookes. 2018. A Denotational Semantics for SPARC TSO. *Electronic Notes in Theoretical Computer Science* 336 (2018), 223–239. https://doi.org/10.1016/j.entcs.2018.03.025 The Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII).

Klaus Keimel and Gordon Plotkin. 2017. Mixed powerdomains for probability and nondeterminism. *Logical Methods in Computer Science* Volume 13, Issue 1 (Jan. 2017). https://doi.org/10.23638/LMCS-13(1:2)2017

Dexter Kozen. 1979. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science (SFCS '79)*. 101–114. https://doi.org/10.1109/SFCS.1979.38

Dexter Kozen and Alexandra Silva. 2024. Multisets and Distributions. In *Logics and Type Systems in Theory and Practice: Essays Dedicated to Herman Geuvers on The Occasion of His 60th Birthday*, Venanzio Capretta, Robbert Krebbers, and Freek Wiedijk (Eds.). Springer Nature Switzerland, Cham, 168–187. https://doi.org/10.1007/978-3-031-61716-4_11

L. Lamport. 1977. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering* SE-3, 2 (1977), 125–143. https://doi.org/10.1109/TSE.1977.229904

Michael R. Laurence and Georg Struth. 2014. Completeness Theorems for Bi-Kleene Algebras and Series-Parallel Rational Pomset Languages. In *Relational and Algebraic Methods in Computer Science*, Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller (Eds.). Springer International Publishing, Cham, 65–82. https://doi.org/10.1007/978-3-319-06251-8_5

Daniel Lehmann and Michael O. Rabin. 1981. On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Williamsburg, Virginia) *(POPL '81)*. Association for Computing Machinery, New York, NY, USA, 133–138. https://doi.org/10.1145/567532.567547

Xuezheng Liu, Wei Lin, Aimin Pan, and Zheng Zhang. 2007. WiDS checker: combating bugs in distributed systems. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation* (Cambridge, MA) *(NSDI'07)*. USENIX Association, USA, 19. https://doi.org/10.5555/1973430.1973449

George Markowsky. 1976. Chain-complete posets and directed sets with applications. *Algebra Universalis* 6 (12 1976), 53–68. https://doi.org/10.1007/BF02485815

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer. https://doi.org/10.1007/b138392

J. J. Ch. Meyer and E.P. de Vink. 1988. Applications of compactness in the Smyth powerdomain of streams. *Theoretical Computer Science* 57, 2 (1988), 251–282. https://doi.org/10.1016/0304-3975(88)90042-4

J. J. Ch. Meyer and E. P. de Vink. 1989. Pomset semantics for true concurrency with synchronization and recursion. In *Mathematical Foundations of Computer Science 1989*, Antoni Kreczmar and Grazyna Mirkowska (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 360–369. https://doi.org/10.1007/3-540-51486-4_83

Michael Mislove. 2000. Nondeterminism and Probabilistic Choice: Obeying the Laws. In *CONCUR 2000 — Concurrency Theory*, Catuscia Palamidessi (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 350–365. https://doi.org/10.1007/3-540-44618-4_26

Michael Mislove, Joël Ouaknine, and James Worrell. 2004. Axioms for Probability and Nondeterminism. *Electronic Notes in Theoretical Computer Science* 96 (2004), 7–28. https://doi.org/10.1016/j.entcs.2004.04.019 Proceedings of the 10th International Workshop on Expressiveness in Concurrency.

Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996a. Probabilistic Predicate Transformers. *ACM Trans. Program. Lang. Syst.* 18, 3 (may 1996), 325–353. https://doi.org/10.1145/229542.229547

Carroll Morgan, Annabelle McIver, Karen Seidel, and J. W. Sanders. 1996b. Refinement-oriented probability for CSP. *Form. Asp. Comput.* 8, 6 (nov 1996), 617–647. https://doi.org/10.1007/bf01213492

Robert Morris. 1978. Counting large numbers of events in small registers. *Commun. ACM* 21, 10 (Oct. 1978), 840–842. https://doi.org/10.1145/359619.359627

Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 625–635. https://doi.org/10.1007/978-3-030-79876-5_37

Renato Neves. 2025. An Adequacy Theorem Between Mixed Powerdomains and Probabilistic Concurrency. *Electronic Proceedings in Theoretical Computer Science* 433 (Oct. 2025), 23–45. https://doi.org/10.4204/eptcs.433.4

Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. 1981. Petri nets, event structures and domains. *Theoretical Computer Science* 13, 1 (1981), 85–108.

Peter W. O'Hearn. 2004. Resources, Concurrency and Local Reasoning. In *CONCUR 2004 - Concurrency Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 49–67. https://doi.org/10.1016/j.tcs.2006.12.035

Shawn Ong, Stephanie Ma, and Dexter Kozen. 2025. Probabilistic Kleene Algebra with Angelic Nondeterminism. In *Proc. 46th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2025)*. ACM, Seoul.

Louis Parlant. 2020. *Monad Composition via Preservation of Algebras*. Ph.D. Dissertation. University College London. https://discovery.ucl.ac.uk/id/eprint/10112228/

Gordon Plotkin. 1976. A Powerdomain Construction. *SIAM J. Comput.* 5, 3 (1976), 452–487. https://doi.org/10.1137/0205035

Vaughan R. Pratt. 1976. Semantical Considerations on Floyd-Hoare Logic. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*. 109–121. https://doi.org/10.1109/SFCS.1976.27

Vaughan R. Pratt. 1986. Modeling Concurrency with Partial Orders. *Int. J. Parallel Program.* 15, 1 (1986), 33–71. https://doi.org/10.1007/BF01379149

Michael O. Rabin. 1980. N-process synchronization by 4.log2N-valued shared variable. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*. 407–410. https://doi.org/10.1109/SFCS.1980.26

Michael O. Rabin. 1982. The choice coordination problem. *Acta Inf.* 17, 2 (June 1982), 121–134. https://doi.org/10.1007/BF00288965

Wojciech Różowski, Tobias Kappé, Dexter Kozen, Todd Schmid, and Alexandra Silva. 2023. Probabilistic Guarded KAT Modulo Bisimilarity: Completeness and Complexity. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 261)*, Kousha Etessami, Uriel Feige, and Gabriele Puppis (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 136:1–136:20. https://doi.org/10.4230/LIPIcs.ICALP.2023.136

Dana Scott. 1970. *Outline of a Mathematical Theory of Computation*. Technical Report PRG02. OUCL. 30 pages.

Dana Scott. 1972. Continuous lattices. In *Toposes, Algebraic Geometry and Logic*, F. W. Lawvere (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 97–136. https://doi.org/10.1007/BFb0073967

Dana Scott and Christopher Strachey. 1971. *Toward A Mathematical Semantics For Computer Languages.* Technical Report Prg06. OUCL. 49 pages.

Michael Smyth. 1978. Power domains. *J. Comput. System Sci.* 16, 1 (1978), 23–36. https://doi.org/10.1016/0022-0000(78)90048-X

Michael Smyth. 1986. Finite approximation of spaces. In *Proceedings of a Tutorial and Workshop on Category Theory and Computer Programming* (Guildford, United Kingdom). Springer-Verlag, Berlin, Heidelberg, 225–241. https://doi.org/10.1007/3-540-17162-2_125

Harald Søndergaard and Peter Sestoft. 1992. Non-determinism in Functional Languages. *Comput. J.* 35, 5 (10 1992), 514–523. https://doi.org/10.1093/comjnl/35.5.514

Joseph Tassarotti. 2018. *Verifying Concurrent Randomized Algorithms.* Ph.D. Dissertation. Carnegie Mellon University. https://csd.cmu.edu/academics/doctoral/degrees-conferred/joseph-tassarotti

Joseph Tassarotti and Robert Harper. 2019. A Separation Logic for Concurrent Randomized Programs. *Proc. ACM Program. Lang.* 3, POPL, Article 64 (Jan 2019), 30 pages. https://doi.org/10.1145/3290377

Regina Tix. 1999. *Continuous D-cones: convexity and powerdomain constructions.* Ph.D. Dissertation. Darmstadt University of Technology, Germany. https://d-nb.info/957239157

Regina Tix. 2000. Convex Power Constructions for Continuous D-Cones. *Electronic Notes in Theoretical Computer Science* 35 (2000), 206–229. https://doi.org/10.1016/S1571-0661(05)80746-7 Workshop on Domains IV.

Regina Tix, Klaus Keimel, and Gordon Plotkin. 2009. Semantic Domains for Combining Probability and Non-Determinism. *Electronic Notes in Theoretical Computer Science* 222 (2009), 3–99. https://doi.org/10.1016/j.entcs.2009.01.002

Daniele Varacca. 2002. The powerdomain of indexed valuations. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science.* 299–308. https://doi.org/10.1109/LICS.2002.1029838

Daniele Varacca. 2003. *Probability, Nondeterminism and Concurrency: Two Denotational Models for Probabilistic Computation.* Ph.D. Dissertation. University of Aarhus. https://www.brics.dk/DS/03/14/

Daniele Varacca, Hagen Völzer, and Glynn Winskel. 2006. Probabilistic event structures and domains. *Theoretical Computer Science* 358, 2 (2006), 173–199. https://doi.org/10.1016/j.tcs.2006.01.015 Concurrency Theory (CONCUR 2004).

Daniele Varacca and Glynn Winskel. 2006. Distributing probability over non-determinism. *Mathematical Structures in Computer Science* 16, 1 (2006), 87–113. https://doi.org/10.1017/S0960129505005074

Daniele Varacca and Nobuko Yoshida. 2007. Probabilistic π-Calculus and Event Structures. *Electronic Notes in Theoretical Computer Science* 190, 3 (2007), 147–166. https://doi.org/10.1016/j.entcs.2007.07.009 Proceedings of the Fifth Workshop on Quantitative Aspects of Programming Languages (QAPL 2007).

Di Wang, Jan Hoffmann, and Thomas Reps. 2019. A Denotational Semantics for Low-Level Probabilistic Programs with Nondeterminism. *Electr. Notes Theor. Comp. Sci.* 347 (November 2019). https://doi.org/10.1016/j.entcs.2019.09.016 Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics.

Glynn Winskel. 1980. *Events in Computation.* Ph.D. Dissertation. University of Edinburgh, UK.

Glynn Winskel. 2014. Probabilistic and Quantum Event Structures. In *Horizons of the Mind. A Tribute to Prakash Panangaden: Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*, Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten (Eds.). Springer International Publishing, Cham, 476–497. https://doi.org/10.1007/978-3-319-06880-0_25

Noam Zilberstein. 2025. Outcome Logic: A Unified Approach to the Metatheory of Program Logics with Branching Effects. *ACM Trans. Program. Lang. Syst.* 47, 3, Article 14 (Sept. 2025), 71 pages. https://doi.org/10.1145/3743131

Noam Zilberstein. 2026a. Pomsets with Formulae (Lean Formalization). https://github.com/zilberstein/pomsets-with-formulae.

Noam Zilberstein. 2026b. *Semantic and Logical Foundations for Randomized Concurrent Programs.* Ph.D. Dissertation. Cornell University. https://www.proquest.com/dissertations-theses/semantic-logical-foundations-randomized/docview/3385352698/se-2?accountid=10267

Noam Zilberstein, Derek Dreyer, and Alexandra Silva. 2023. Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 93 (Apr 2023), 29 pages. https://doi.org/10.1145/3586045

Noam Zilberstein, Daniele Gorla, and Alexandra Silva. 2025a. Denotational Semantics for Probabilistic and Concurrent Programs. In *36th International Conference on Concurrency Theory (CONCUR 2025) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 348)*, Patricia Bouyer and Jaco van de Pol (Eds.). Schloss Dagstuhl – Leibniz-Zentrum f"ur Informatik, Dagstuhl, Germany, 39:1–39:24. https://doi.org/10.4230/LIPIcs.CONCUR.2025.39

Noam Zilberstein, Dexter Kozen, Alexandra Silva, and Joseph Tassarotti. 2025b. A Demonic Outcome Logic for Randomized Nondeterminism. *Proc. ACM Program. Lang.* 9, POPL, Article 19 (Jan 2025), 30 pages. https://doi.org/10.1145/3704855

Noam Zilberstein, Angelina Saliling, and Alexandra Silva. 2024. Outcome Separation Logic: Local Reasoning for Correctness and Incorrectness with Computational Effects. *Proc. ACM Program. Lang.* 8, OOPSLA1 (Apr 2024). https://doi.org/10.1145/3649821

Noam Zilberstein, Alexandra Silva, and Joseph Tassarotti. 2026. Probabilistic Concurrent Reasoning in Outcome Logic: Independence, Conditioning, and Invariants. *Proc. ACM Program. Lang.* 10, POPL (Jan 2026), 30 pages. https://doi.org/10.1145/3776651

Maaike Zwart. 2020. *On the Non-Compositionality of Monads via Distributive Laws.* Ph.D. Dissertation. University of Oxford. https://ora.ox.ac.uk/objects/uuid:b2222b14-3895-4c87-91f4-13a8d046febb

Maaike Zwart and Dan Marsden. 2019. No-Go Theorems for Distributive Laws. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS).* 1–13. https://doi.org/10.1109/lics.2019.8785707

A On the interplay between sequential composition and loop unrolling

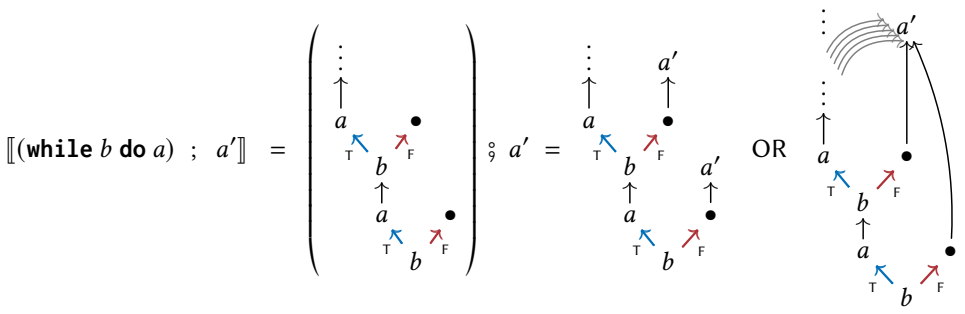
In this section, we discuss the rationale for the copying behavior of sequential composition by demonstrating the semantic structures that are generated as the result of looping programs. We start with a simple program, which is a loop containing a single action as the body:

$$\textbf{while } b \textbf{ do } a$$

As defined in Figure 4, the semantics of this program is given by the following fixed point:

$$[\![\textbf{while } b \textbf{ do } a]\!] = \mathrm{lfp}\left(\Phi_{\langle a,b\rangle}\right) \qquad \text{where} \qquad \Phi_{\langle a,b\rangle}(\alpha) = \mathrm{guard}(b, \langle a\rangle \fatsemi \alpha, [\![\textbf{skip}]\!])$$

By the Kleene fixed point theorem, the least fixed point above is given by the supremum over all the finite iterations of $\Phi_{\langle a,b\rangle}$ applied to $\bot_{pom}$. Unrolling this definition several times, we get the chains shown in Section 6.1. Now suppose that we would like to sequentially compose another action after the loops, to obtain the program: $(\textbf{while } b \textbf{ do } a) \;;\; a'$. According to the definition of sequential composition from Section 5.2, the semantics of this program is obtained by composing a copy of a' after each branch of $[\![\textbf{while } b \textbf{ do } a]\!]$. As we just saw, the branches of $[\![\textbf{while } b \textbf{ do } a]\!]$ correspond to all of the terminating traces of the loop, and therefore we make countably many copies of a', as shown in the first option below. We investigated the possibility of defining sequential composition without such a copying, similar to what was done for finite structures in [**?**]. However, this approach does not translate to the infinite case. To see why not, consider the second option below structure, obtained by composing a' after $\textbf{while } b \textbf{ do } a$, without copying.

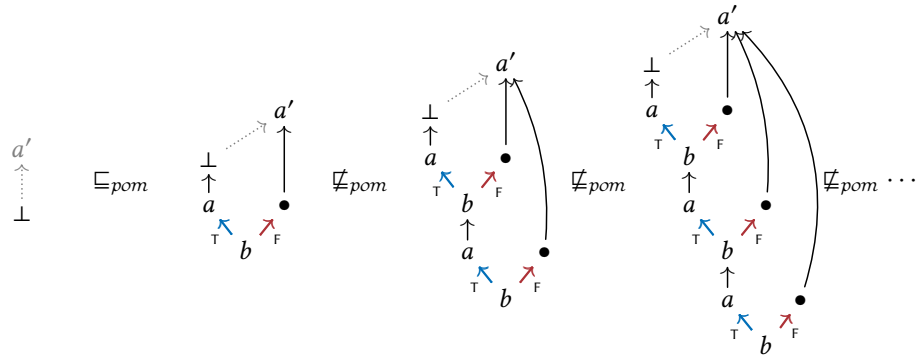


The node labelled as a' is not finitely preceded; it has countably many predecessors, one for each terminating branch of the loop. Relaxing the finitely preceded requirement for LPOF s is not an option; it breaks many of the proofs from Section 4, preventing us from approximating a pomset using finite structures and invalidating the extension lemma. Without the extension lemma, it is unclear how linearization could be defined, as well-founded recursion relies on the structure being finite. Scott continuity of sequential composition comes into question as well. Scott continuity of $\fatsemi$

means that the following equation holds:

$$\left(\sup_{n\in\mathbb{N}} \Phi^n_{\langle a,b\rangle}(\bot_{pom})\right) \fatsemi \langle a'\rangle \qquad \stackrel{?}{=} \qquad \sup_{n\in\mathbb{N}} \left(\Phi^n_{\langle a,b\rangle}(\bot_{pom}) \fatsemi \langle a'\rangle\right)$$

We investigated two possible expansions for the latter expression. The first possibility retained the requirement that nothing can proceed $\bot$; it is shown as the sequence below, ignoring the grey dotted arrows. The second possibility allowed $\bot$ nodes to have successors, and is shown below, where the grey dotted arrows indicate standard causality.



$\Phi^0_{\langle a,b\rangle}(\bot_{pom}) \fatsemi \langle a'\rangle$ $\Phi^1_{\langle a,b\rangle}(\bot_{pom}) \fatsemi \langle a'\rangle$ $\Phi^2_{\langle a,b\rangle}(\bot_{pom}) \fatsemi \langle a'\rangle$ $\Phi^3_{\langle a,b\rangle}(\bot_{pom}) \fatsemi \langle a'\rangle$

The causality from $\bot$ to a' is problematic, since $\bot$ could be expanded into an infinite structure, again causing a' to not be finitely preceded. But the lack of causality from $\bot$ to a' is also problematic, as in the next approximation, $\bot$ will be replaced by the test b, which is supposed to be scheduled before a'. This complicates monotonicity of linearization, as the allowable interleavings can change as the structure grows.

In any case, the sequence above does not form a chain—at least, not according to our current definition of $\sqsubseteq_{pom}$. The reason why this is not a chain is because the node labelled a' gains more predecessors with each successive structure. We investigated several alternative definitions of $\sqsubseteq_{pom}$, which did allow for this sort of expansion, however each one had significant problems—*e.g.,* some were not transitive, and thus not partial orders; some were not DCPO s; and some caused $\Phi_{\langle C,b\rangle}$ not to be monotone, meaning that loops could not be interpreted as fixed points using the Kleene fixed point theorem or the Knaster-Tarski theorem. In the course of our investigation, we found that the copying of structures in sequential composition was the best way to obtain a sound structure, obeying all the necessary laws.

Interestingly, this copying behavior corresponds to the following equational law of Probabilistic Guarded Kleene Algebra with Tests [Różowski et al. 2023], where sequencing a command C after a guarded choice is equivalent to including C in both branches:

$$(\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2)\ ;\ C \quad \equiv \quad \mathbf{if}\ b\ \mathbf{then}\ C_1;C\ \mathbf{else}\ C_2;C$$

B Proofs for the results on LPOFs (from Section 4)

B.1 Basic properties

Let nBot_α denote the set of nodes not labelled with $\bot$, *i.e.,* $\mathsf{nBot}_\alpha \triangleq \{x \in N_\alpha \mid \lambda_\alpha(x) \neq \bot\}$.

Lemma B.1. *Let $\alpha \sqsubseteq_{lpof} \beta$; then,* $\mathsf{nBot}_\alpha \subseteq \mathsf{nBot}_\beta$.

Proof. By Def. 4.1 (Item 1 and 3a), we have that $\lambda_\alpha(x) \leq \lambda_\beta(x)$, for all $x \in N_\alpha \subseteq N_\beta$. Hence, if $x \in \mathsf{nBot}_\alpha$, then $\bot < \lambda_\alpha(x) \leq \lambda_\beta(x)$ and so $x \in \mathsf{nBot}_\beta$. □

Lemma B.2. *If $\alpha \sqsubseteq_{lpof} \beta$, then $N_\beta \setminus N_\alpha = \mathrm{Bot}_\alpha{\uparrow_\beta}$.*

Proof. We show the set inclusion in both ways.

First, take any $x \in N_\beta \setminus N_\alpha$. Since x is finitely preceded, there must be a positive integer n and nodes $x_1, \ldots, x_n \in N_\beta$ such that x_1 is the root of β, $x_n = x$, and $x_{i+1} \in \mathrm{succ}_\beta(x_i)$ for all $1 \leq i < n$. Since $\alpha \sqsubseteq_{lpof} \beta$, then x_1 must also be the root of α. Now, let i be smallest index such that $x_i \in N_\alpha$ and $x_{i+1} \notin N_\alpha$ (note that $1 < i < n$ since $x_1 \in N_\alpha$ but $x = x_n \notin N_\alpha$). From $\alpha \sqsubseteq_{lpof} \beta$, we know that $\mathrm{succ}_\alpha(x_i) = \mathrm{succ}_\beta(x_i) \setminus \mathrm{Bot}_\alpha{\uparrow_\beta}$; so, $x_{i+1} \in \mathrm{Bot}_\alpha{\uparrow_\beta}$ that allows us to conclude $x \in \mathrm{Bot}_\alpha{\uparrow_\beta}$, being $x_{i+1} <_\beta x_n = x$.

Now, take $x \in \mathrm{Bot}_\alpha{\uparrow_\beta}$, i.e. there exists $z \in \mathrm{Bot}_\alpha$ such that $z <_\beta x$. If $x \in N_\alpha$, being $<_\alpha = <_\beta \cap (N_\alpha \times N_\alpha)$, we would have that $z <_\alpha x$, in contradiction with $z \in \mathrm{Bot}_\alpha$. □

Lemma 4.3. *$\langle lpof(L), \sqsubseteq_{lpof} \rangle$ is a poset.*

Proof. Reflexivity is trivial. For antisymmetry, we have that $\alpha \sqsubseteq_{lpof} \beta$ and $\beta \sqsubseteq_{lpof} \alpha$ imply that $N_\alpha = N_\beta$; hence:

$$<_\alpha = <_\beta \cap (N_\alpha \times N_\alpha) = <_\beta \cap (N_\beta \times N_\beta) = <_\beta$$

Then, let $x \in N_\alpha (= N_\beta)$. Trivially, $\varphi_\alpha(x) = \varphi_\beta(x)$; furthermore, $\lambda_\alpha(x) \leq \lambda_\beta(x)$ and $\lambda_\beta(x) \leq \lambda_\alpha(x)$, so, $\lambda_\alpha(x) = \lambda_\beta(x)$, by antisimmetry of $\leq$.

For transitivity, let us assume that $\alpha \sqsubseteq_{lpof} \beta$ and $\beta \sqsubseteq_{lpof} \gamma$, and prove that $\alpha \sqsubseteq_{lpof} \gamma$. By hypothesis, $N_\alpha \subseteq_\downarrow N_\beta \subseteq_\downarrow N_\gamma$, so $N_\alpha \subseteq N_\gamma$; we have to prove that N_α is downwards closed (w.r.t. to γ). Take any $x \in N_\alpha$ and $y <_\gamma x$. Since $N_\alpha \subseteq N_\beta$, $x \in N_\beta$ and, being $N_\beta \subseteq_\downarrow N_\gamma$, then $y \in N_\beta$. Further, since $<_\beta = <_\gamma \cap (N_\beta \times N_\beta)$, then $y <_\beta x$. Therefore $y \in N_\alpha$, since $N_\alpha \subseteq_\downarrow N_\beta$.

For the second item of Definition 4.1,

$$\begin{aligned} <_\alpha &= <_\beta \cap (N_\alpha \times N_\alpha) \\ &= (<_\gamma \cap (N_\beta \times N_\beta)) \cap (N_\alpha \times N_\alpha) \\ &= <_\gamma \cap ((N_\beta \cap N_\alpha) \times (N_\beta \cap N_\alpha)) \\ &= <_\gamma \cap (N_\alpha \times N_\alpha) \end{aligned}$$

Finally, let $x \in N_\alpha$. Trivially, $\varphi_\alpha(x) = \varphi_\beta(x) = \varphi_\gamma(x)$ and $\lambda_\alpha(x) \leq \lambda_\beta(x) \leq \lambda_\gamma(x)$; so, $\lambda_\alpha(x) \leq \lambda_\gamma(x)$, by transitivity of $\leq$. Moreover, we have:

$$\begin{aligned} \mathrm{succ}_\alpha(x) &= \mathrm{succ}_\beta(x) \setminus \mathrm{Bot}_\alpha{\uparrow_\beta} \\ &= \left(\mathrm{succ}_\gamma(x) \setminus \mathrm{Bot}_\beta{\uparrow_\gamma}\right) \setminus \mathrm{Bot}_\alpha{\uparrow_\beta} \\ &= \mathrm{succ}_\gamma(x) \setminus \left(\mathrm{Bot}_\beta{\uparrow_\gamma} \cup \mathrm{Bot}_\alpha{\uparrow_\beta}\right) \\ &= \mathrm{succ}_\gamma(x) \setminus \mathrm{Bot}_\alpha{\uparrow_\gamma} \end{aligned}$$

where the last equality holds thanks to Lemma B.2: indeed, $\mathrm{Bot}_\alpha{\uparrow_\beta} = N_\beta \setminus N_\alpha$, $\mathrm{Bot}_\beta{\uparrow_\gamma} = N_\gamma \setminus N_\beta$ and $\mathrm{Bot}_\alpha{\uparrow_\gamma} = N_\gamma \setminus N_\alpha$; we conclude since $(N_\gamma \setminus N_\beta) \cup (N_\beta \setminus N_\alpha) = (N_\gamma \setminus N_\alpha)$. □

Lemma B.3. *Let $N_\beta \subseteq_\downarrow N_\alpha$ and $<_\beta = <_\alpha \cap (N_\beta \times N_\beta)$. Then:*

(1) if $y \in \mathrm{succ}_\alpha(x)$ and $x <_\beta y$, then $y \in \mathrm{succ}_\beta(x)$;
(2) if $y \in \mathrm{succ}_\beta(x)$, then $y \in \mathrm{succ}_\alpha(x)$;
(3) $\mathrm{succ}_\beta(x) \subseteq \mathrm{succ}_\alpha(x) \setminus \mathrm{Bot}_\beta{\uparrow_\alpha}$, for every $x \in N_\beta$.

Proof. For the first claim, if it was $x <_\beta z <_\beta y$, then $x <_\alpha z <_\alpha y$ (being $<_\beta \subseteq <_\alpha$), in contradiction with $y \in \mathrm{succ}_\alpha(x)$.

For the second claim, $x <_\beta y$ and $<_\beta \subseteq <_\alpha$ imply that $x <_\alpha y$. Now, by contradiction, assume that $x <_\alpha z <_\alpha y$. Since $y \in N_\beta \subseteq_\downarrow N_\alpha$, then $z \in N_\beta$; hence, being $<_\beta = <_\alpha \cap (N_\beta \times N_\beta)$, we would have that $x <_\beta z <_\beta y$, in contradiction with $y \in \mathrm{succ}_\beta(x)$. So, $y \in \mathrm{succ}_\alpha(x)$.

For the third claim, let $y \in \mathrm{succ}_\beta(x)$; by the second claim, $y \in \mathrm{succ}_\alpha(x)$. If it was $y \in \mathrm{Bot}_\beta{\uparrow_\alpha}$, then there would exist $z \in \mathrm{Bot}_\beta$ such that $z <_\alpha y$; being $<_\beta = <_\alpha \cap (N_\beta \times N_\beta)$, we would have that $z <_\beta y$, i.e. $z \notin \mathrm{Bot}_\beta$. □

Lemma B.4. *If $D \subseteq \mathit{lpof}(L)$ is a directed set, then $x{\downarrow_\beta} = x{\downarrow_{\beta'}}$ for any $\beta, \beta' \in D$ such that $x \in N_\beta \cap N_{\beta'}$.*

Proof. Take any $\beta, \beta' \in D$ such that $x \in N_\beta$ and $x \in N_{\beta'}$. Since D is directed, there must be a $\gamma \in D$ such that $\beta \sqsubseteq_{\mathit{lpof}} \gamma$ and $\beta' \sqsubseteq_{\mathit{lpof}} \gamma$. Now, take any $y <_\beta x$, clearly this means that $y <_\gamma x$, and since $N_{\beta'} \subseteq_\downarrow N_\gamma$, then $y \in N_{\beta'}$, and so $y \in x{\downarrow_{\beta'}}$, so we have showed that $x{\downarrow_\beta} \subseteq x{\downarrow_{\beta'}}$. By an equivalent argument, we get that $x{\downarrow_{\beta'}} \subseteq x{\downarrow_\beta}$, therefore $x{\downarrow_\beta} = x{\downarrow_{\beta'}}$. □

Lemma B.5. *For any directed set $D \subseteq \mathit{lpof}(L)$ and $n \in \mathbb{N}$, there exists a finite set X such that $\{x \mid \mathrm{lev}_\beta(x) = n\} \subseteq X$ for all $\beta \in D$.*

Proof. The proof is by induction on n. Let $n = 1$, take any $\beta \in D$, and let x be the root of β. Since D is directed, then x is the root of every $\beta' \in D$, therefore letting $X = \{x\}$, we have that $\{y \mid \mathrm{lev}_\beta(y) = 1\} = X$ for all $\beta \in D$.

Now, suppose that the claim holds for all $k < n$, and let X_k be the finite set such that $\{x \mid \mathrm{lev}_\beta(x) = k\} \subseteq X_k$ for each $k < n$. Also, let $X = \bigcup_{k<n} X_k$, so $\{x \mid \mathrm{lev}_\beta(x) < n\} \subseteq X$, which is also finite. This means that only finitely many $\bot$ nodes can appear before level n. Now construct A as follows: for each $x \in X$, select a $\beta \in D$ such that $\lambda_\beta(x) \neq \bot$, if such a β exists. So $|A| \leq |X|$, which is clearly finite. Since A is a finite subset of D, which is directed, then there must be some $\gamma \in D$ that is an upper bound of all the elements in A. This means that for all $x \in X$, $x \notin \mathrm{Bot}_\beta$ for some $\beta \in D$ iff $x \notin \mathrm{Bot}_\gamma$.

Now let $Y = \{x \mid \mathrm{lev}_\gamma(x) = n\}$. Take any $\beta \in D$, so there is a $\beta' \in D$ that is an upper bound of β and γ. By construction, β' can only extend γ after level n, and so the elements of γ at level n must be the same as the elements of β' and since $\beta \sqsubseteq_{\mathit{lpof}} \beta'$, then $\{x \mid \mathrm{lev}_{\beta'}(x) = n\} \subseteq Y$. □

Lemma 4.8. *For any directed set $D \subseteq \mathit{lpof}(L)$, $\sup D = \langle N, <, \lambda, \varphi\rangle$, where:*

$$N \triangleq \bigcup_{\beta\in D} N_\beta \qquad < \triangleq \bigcup_{\beta\in D} <_\beta \qquad \lambda(x) \triangleq \sup_{\beta\in D\,:\,x\in N_\beta} \lambda_\beta(x) \qquad \varphi(x) \triangleq \psi_x$$

and $\psi_x = \varphi_\beta(x)$, for all $\beta \in D$ such that $x \in N_\beta$.

Proof. We need to show that:

(1) $\alpha \in \mathit{lpof}(L)$;
(2) $\beta \sqsubseteq_{\mathit{lpof}} \alpha$, for all $\beta \in D$; and
(3) $\alpha \sqsubseteq_{\mathit{lpof}} \gamma$, for all γ s.t. $\beta \sqsubseteq_{\mathit{lpof}} \gamma$ (for all $\beta \in D$).

We start with property (1). First, clearly α is single-rooted, since D is a directed set and therefore every $\beta \in D$ must have the same root. We now show that every element of N_α is finitely preceded. Take any $x \in N_\alpha$, so $x \in N_\beta$ for some $\beta \in D$. By Lemma B.4, we know that $x{\downarrow_\beta} = x{\downarrow_{\beta'}}$ for all $\beta' \in D$, therefore $x{\downarrow_\alpha} = \bigcup_{\beta'\in A} x{\downarrow_{\beta'}} = x{\downarrow_\beta}$, which much be finite.

Next, we show that finitely many elements appear at level n. By Lemma B.5, we know that there is a finite set X such that $\{x \mid \mathrm{lev}_\beta(x) = n\} \subseteq X$ for all $\beta \in D$. Following from Lemma B.4, $\mathrm{lev}_\beta(x) = \mathrm{lev}_{\beta'}(x) = \mathrm{lev}_\alpha(x)$ for all $\beta, \beta' \in D$ and $x \in N_\alpha$, so the elements at level n in α are the elements at level n from all $\beta \in D$. This gives us:

$$\{x \mid \mathrm{lev}_\alpha(x) = n\} = \bigcup_{\beta\in D} \{x \mid \mathrm{lev}_\beta(x) = n\} \subseteq X$$

Now, take any $x \in N_\alpha$. If $\lambda_\alpha(x) = \bot$, then $\lambda_\beta(x) = \bot$ for all $\beta \in D$ such that $x \in N_\beta$. This means that x has no successors in any $\beta \in D$, therefore it has no successors in α.

If $x <_\alpha y$, then $x <_\beta y$ for some $\beta \in D$. Since D is directed, $\varphi_\beta(x) = \varphi_{\beta'}(x)$ for all $\beta' \in D$ that contain x (and similarly for y); therefore, $\varphi_\alpha(y) = \varphi_\beta(y) \Rightarrow \varphi_\beta(x) = \varphi_\alpha(x)$. Finally, take any $x \in N_\alpha$, so $x \in N_\beta$ for some $\beta \in D$. By construction, $\varphi_\alpha(x) = \varphi_\beta(x) = \psi_x$ and, by Lemma B.4, $x{\downarrow_\alpha} = x{\downarrow_\beta}$; so, since $\mathrm{vars}(\varphi_\beta(x)) \subseteq x{\downarrow_\beta}$, then $\mathrm{vars}(\varphi_\alpha(x)) \subseteq x{\downarrow_\alpha}$.

We next prove property (2): we choose any $\beta \in D$ and show all conditions of $\sqsubseteq_{lpof}$.

First, we prove that $N_\beta \subseteq_\downarrow N_\alpha$; the inclusion is by construction. Then, take $x \in N_\beta$ and $y <_\alpha x$; by construction, there must exist a $\beta' \in D$ such that $y <_{\beta'} x$. Take $\gamma \in D$ to be an upper bound of β and β', which must exist since D is a directed set. So $\beta' \sqsubseteq_{lpof} \gamma$ and therefore $y <_\gamma x$. Since N_β is a downward closed subset of N_γ (being $\beta \sqsubseteq_{lpof} \gamma$), it must be that $y \in N_\beta$.

For the order, we proceed as follows:

$$\begin{aligned} <_\alpha \cap (N_\beta \times N_\beta) &= \left(\bigcup_{\beta' \in D} <_{\beta'}\right) \cap \left(N_\beta \times N_\beta\right) \\ &= \bigcup_{\beta' \in D} \left(<_{\beta'} \cap \left(N_\beta \times N_\beta\right)\right) \\ &\overset{\ddagger}{=} <_\beta \end{aligned}$$

For the step (‡), we prove that, for every $\beta' \in D$ and every $(x, y) \in <_{\beta'} \cap (N_\beta \times N_\beta)$, we have that $(x, y) \in <_\beta$. Let γ be the upper bound of β and β'; then, $x <_\gamma y$ (being $x <_{\beta'} y$ and $\beta' \sqsubseteq_{lpof} \gamma$). Since $<_\beta = <_\gamma \cap (N_\beta \times N_\beta)$ (because $\beta \sqsubseteq_{lpof} \gamma$), we conclude $x <_\beta y$.

Now, fix $x \in N_\beta$; since D is directed, $\varphi_\alpha(x) = \psi_x = \varphi_\beta(x)$ and, by definition of sup, $\lambda_\beta(x) \leq \lambda_\alpha(x)$. Finally, by Lemma B.3(3), $\mathrm{succ}_\beta(x) \subseteq \mathrm{succ}_\alpha(x) \setminus \mathrm{Bot}_\beta{\uparrow_\alpha}$. We now show the reverse inclusion. Suppose that $y \in \mathrm{succ}_\alpha(x)$ and $y \notin \mathrm{Bot}_\beta{\uparrow_\alpha}$; by construction, there must be some $\beta' \in D$ such that $x <_{\beta'} y$. By Lemma B.3(1), $y \in \mathrm{succ}_{\beta'}(x)$. Since D is directed, there is a $\gamma \in D$ that is an upper bound of β and β'. Since $\beta' \sqsubseteq_{lpof} \gamma$, then $\mathrm{succ}_{\beta'}(x) = \mathrm{succ}_\gamma(x) \setminus \mathrm{Bot}_{\beta'}{\uparrow_\gamma}$; so, $y \in \mathrm{succ}_\gamma(x)$. Since $\beta \sqsubseteq_{lpof} \gamma$, then we also have $\mathrm{succ}_\beta(x) = \mathrm{succ}_\gamma(x) \setminus \mathrm{Bot}_\beta{\uparrow_\gamma}$. Since $y \notin \mathrm{Bot}_\beta{\uparrow_\alpha}$, then y also cannot be in $\mathrm{Bot}_\beta{\uparrow_\gamma}$ since $<_\gamma \subseteq <_\alpha$ by construction; therefore, it must be that $y \in \mathrm{succ}_\beta(x)$.

Let us now move to property (3): we fix γ satisfying $\beta \sqsubseteq_{lpof} \gamma$ (for all $\beta \in D$) and show that $\alpha \sqsubseteq_{lpof} \gamma$.

We know that $N_\beta \subseteq N_\gamma$ for all $\beta \in D$, therefore $N_\alpha = \bigcup_{\beta \in D} N_\beta \subseteq N_\gamma$. We now need to show that N_α is downwards closed. Take any $x \in N_\alpha$ and $y <_\gamma x$. Since $x \in N_\alpha$, then $x \in N_\beta$ for some $\beta \in D$. Since $N_\beta \subseteq_\downarrow N_\gamma$, then $y \in N_\beta$, and therefore $y \in N_\alpha$.

Second,

$$\begin{aligned} <_\gamma \cap (N_\alpha \times N_\alpha) &= <_\gamma \cap \left(\bigcup_{\beta \in D} N_\beta \times \bigcup_{\beta \in D} N_\beta\right) \\ &\overset{\dagger}{=} <_\gamma \cap \left(\bigcup_{\beta \in D} (N_\beta \times N_\beta)\right) \\ &= \bigcup_{\beta \in D} (<_\gamma \cap (N_\beta \times N_\beta)) \\ &= \bigcup_{\beta \in D} <_\beta \;=\; <_\alpha \end{aligned}$$

where the step marked with (†) follows using two set inclusions. The first is $\bigcup_{\beta\in D}(N_\beta \times N_\beta) \subseteq \bigcup_{\beta\in D} N_\beta \times \bigcup_{\beta\in D} N_\beta$, which is immediate by the properties of cartesian product. For the reverse inclusion, take any $(x, y) \in \bigcup_{\beta\in D} N_\beta \times \bigcup_{\beta\in D} N_\beta$, which means that there exist $\beta', \beta'' \in D$ such that $x \in N_{\beta'}$ and $y \in N_{\beta''}$. Since D is a directed set, there exists $\hat{\beta} \in D$ that is an upper bound of β' and β''; thus, $N_{\beta'} \subseteq N_{\hat{\beta}}$ and $N_{\beta''} \subseteq N_{\hat{\beta}}$ and so $N_{\beta'} \times N_{\beta''} \subseteq N_{\hat{\beta}} \times N_{\hat{\beta}}$. The latter implies that $(x, y) \in N_{\hat{\beta}} \times N_{\hat{\beta}} \subseteq \bigcup_{\beta\in D}(N_\beta \times N_\beta)$, as desired.

To conclude, take any $x \in N_\alpha$; since $\beta \sqsubseteq_{lpof} \gamma$ (for all β), we know that $\varphi_\beta(x) = \varphi_\gamma(x)$ and $\lambda_\beta(x) \leq \lambda_\gamma(x)$, for all β that contain x. This allows us to obtain that $\varphi_\alpha(x) = \varphi_\gamma(x)$ and $\lambda_\alpha(x) \leq \lambda_\gamma(x)$.

Finally, by Lemma B.3(3) (with α in place of β and γ in place of α therein), $\mathrm{succ}_\alpha(x) \subseteq \mathrm{succ}_\gamma(x) \setminus \mathrm{Bot}_\alpha{\uparrow_\gamma}$. We now show the reverse inclusion too. Let $y \in \mathrm{succ}_\gamma(x)$ and $y \notin \mathrm{Bot}_\alpha{\uparrow_\gamma}$. Since y is finitely preceded, we know that $y{\downarrow_\gamma}$ is finite, therefore $y{\downarrow_\gamma} \cap N_\alpha$ is finite and $\lambda_\alpha(z) \neq \bot$ for each $z \in y{\downarrow_\gamma} \cap N_\alpha$ since $y \notin \mathrm{Bot}_\alpha{\uparrow_\gamma}$. We now construct a set A as follows: for each $z \in y{\downarrow_\gamma} \cap N_\alpha$, select a $\beta_z \in D$ such that $z \in N_{\beta_z}$ and $\lambda_{\beta_z}(z) \neq \bot$, which must exist since $\lambda_\alpha(z) \neq \bot$. Since A is a finite subset of D, which is directed, there is an element $\beta \in D$ which is an upper bound of all the elements in A; therefore, $y{\downarrow_\gamma} \cap N_\alpha \subseteq N_\beta$ and $\lambda_\beta(z) \neq \bot$ for each $z \in y{\downarrow_\gamma} \cap N_\alpha$. Since $\beta \in D$, we also know that $\beta \sqsubseteq_{lpof} \gamma$ and, since $x <_\gamma y$ and $x \in N_\alpha$, then $x \in N_\beta$; hence, $\mathrm{succ}_\beta(x) = \mathrm{succ}_\gamma(x) \setminus \mathrm{Bot}_\beta{\uparrow_\gamma}$, so either $y \in \mathrm{succ}_\beta(x)$ or $y \in \mathrm{Bot}_\beta{\uparrow_\gamma}$. In the first case, we are done since $\beta \sqsubseteq_{lpof} \alpha$ and therefore $\mathrm{succ}_\beta(x) \subseteq \mathrm{succ}_\alpha(x)$ by Lemma B.3(3). The second case is a contradiction, since it implies that there is some $z \in \mathrm{Bot}_\beta$ such that $z <_\gamma y$; hence, $z \in y{\downarrow_\gamma} \cap N_\beta \subseteq y{\downarrow_\gamma} \cap N_\alpha$, where the last inclusion holds since $\beta \sqsubseteq_{lpof} \alpha$. This implies that $\lambda_\alpha(z) \neq \bot$, i.e. $z \notin \mathrm{Bot}_\beta$. □

B.2 Truncation of LPOFs

First, we define some notation for lpos, similar to that of pomsets:

$$lpof_{\mathsf{fin}}(L) \triangleq \{\alpha \in lpof(L) \mid |N_\alpha| < \infty\} \qquad \lfloor\alpha\rfloor_{\mathsf{fin}} \triangleq \{\beta \in lpof_{\mathsf{fin}}(L) \mid \beta \sqsubseteq_{lpof} \alpha\}$$

Definition B.6 (Truncation). Let $\alpha \in lpof(L)$ and $n \in \mathbb{N}$. Then, $\lfloor\alpha\rfloor_n = \langle N, <, \lambda, \varphi\rangle$ where

- $N \triangleq \{x \in N_\alpha \mid \mathsf{lev}_\alpha(x) \leq n\}$;
- $< \triangleq <_\alpha \cap (N \times N)$;
- for all $x \in N$, let $\lambda(x) \triangleq \lambda_\alpha(x)$, if $\mathsf{lev}_\alpha(x) < n$, and $\lambda(x) \triangleq \bot$, otherwise;
- for all $x \in N$, let $\varphi(x) \triangleq \varphi_\alpha(x)$.

Essentially, the truncation of α with n consists in keeping α unchanged for all nodes $x \in N_\alpha$ such that $\mathsf{lev}_\alpha(x) < n$ and changing the labels of all nodes $x \in \mathsf{lev}^{-1}(n)$ to $\bot$. For any $\alpha \in lpof(L)$, $\lfloor\alpha\rfloor_0$ is a singleton LPOF with $\bot$ at the root.

For a set of LPOFs $S \subseteq lpof(L)$, we also let $\lfloor S\rfloor_n = \{\lfloor\alpha\rfloor_n \mid \alpha \in S\}$. Note that since a pomset is a set of LPOFs, this definition applies to pomsets too.

LEMMA B.7. *For any $\alpha \in lpof(L)$ and $n > 0$, it holds that $\lfloor\alpha\rfloor_n \in \lfloor\alpha\rfloor_{\mathsf{fin}}$.*

PROOF. To lighten notation, let $\beta = \lfloor\alpha\rfloor_n$. First, note that N_β is finite, since all nodes occur at level at most n, and there can only be finitely many nodes up to that level.

Now, we prove that $\beta \in lpof(L)$. First, β is single rooted, is finitely preceded and has finitely many nodes at every finite level, because it inherits these properties from α. For the same reason, the conditions on φ_β are satisfied. Finally, for any $x \in N_\beta$ such that $\lambda_\beta(x) = \bot$, we consider two cases:

(1) $\mathsf{lev}_\alpha(x) < n$: by construction, $\lambda_\beta(x) = \lambda_\alpha(x)$ and so $\mathrm{succ}_\alpha(x) = \emptyset$. By the definition of $<_\beta$, we have that $\mathrm{succ}_\beta(x) \subseteq \mathrm{succ}_\alpha(x)$ and so also $\mathrm{succ}_\beta(x) = \emptyset$.

(2) $\mathrm{lev}_\alpha(x) = n$: obviously x has no successors, since everything above level n has been deleted.

Finally, we show that $\beta \sqsubseteq_{lpof} \alpha$. First, we show that $N_\beta \subseteq_\downarrow N_\alpha$. Clearly $N_\beta \subseteq N_\alpha$ by construction. Now, take any $y \in N_\beta$ and $z <_\alpha y$. Since $\mathrm{lev}_\alpha(y) \leq n$, then $\mathrm{lev}_\alpha(z) < n$ and therefore $z \in N_\beta$. By construction, we also have that $<_\beta = <_\alpha \cap (N_\beta \times N_\beta)$, $\lambda_\beta(y) \leq \lambda_\alpha(y)$, and $\varphi_\beta(y) = \varphi_\alpha(y)$ for all $y \in N_\beta$. It remains only to show the condition on successors. Take any $y \in N_\beta$. By Lemma B.3(3), $\mathrm{succ}_\beta(y) \subseteq \mathrm{succ}_\alpha(y) \setminus \mathrm{Bot}_\beta{\uparrow_\alpha}$; we now show the reverse inclusion. Suppose that $z \in \mathrm{succ}_\alpha(y)$ and $z \notin \mathrm{Bot}_\beta{\uparrow_\alpha}$. Since $y \in N_\beta$, then $\mathrm{lev}_\alpha(z) \leq n+1$. However, it cannot be that $\mathrm{lev}_\alpha(z) = n+1$, since that would imply that there exists $w \in \mathrm{pred}_\alpha(z)$ such that $\mathrm{lev}_\alpha(w) = n$, but then $\lambda_\beta(w) = \bot$, which is a contradiction. So, $\mathrm{lev}_\alpha(z) \leq n$, which means that $z \in N_\beta$. □

An immediate consequence of Lemma B.7 is that $\lfloor\alpha\rfloor_n \in lpof(L)$ and $\lfloor\alpha\rfloor_n \sqsubseteq_{lpof} \alpha$.

Lemma B.8 (Monotonicity of truncation). *If $\alpha \sqsubseteq_{lpof} \beta$, then $\lfloor\alpha\rfloor_n \sqsubseteq_{lpof} \lfloor\beta\rfloor_n$, for every $n \in \mathbb{N}$.*

Proof. By Lemma B.4, we have that $\mathrm{lev}_\alpha(x) = \mathrm{lev}_\beta(x)$, for every $x \in N_\alpha \subseteq N_\beta$; hence, $N_{\lfloor\alpha\rfloor_n} \subseteq N_{\lfloor\beta\rfloor_n}$. For downward closure, let $x \in N_{\lfloor\alpha\rfloor_n}$ and $y <_{\lfloor\beta\rfloor_n} x$; thus, $y \in x{\downarrow_{\lfloor\beta\rfloor_n}}$ and again by Lemma B.4 we easily conclude.

For the order relation, we have that

$$\begin{aligned}
<_{\lfloor\beta\rfloor_n} \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n}) &= (<_\beta \cap (N_{\lfloor\beta\rfloor_n} \times N_{\lfloor\beta\rfloor_n})) \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n}) \\
&= <_\beta \cap ((N_{\lfloor\beta\rfloor_n} \times N_{\lfloor\beta\rfloor_n}) \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n})) \\
&= <_\beta \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n}) \\
&= <_\beta \cap ((N_\alpha \times N_\alpha) \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n})) \\
&= (<_\beta \cap (N_\alpha \times N_\alpha)) \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n}) \\
&= <_\alpha \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n}) = <_{\lfloor\alpha\rfloor_n}
\end{aligned}$$

where all steps follow by definition of truncation, associativity of $\cap$, $\alpha \sqsubseteq_{lpof} \beta$, $N_{\lfloor\alpha\rfloor_n} \subseteq N_{\lfloor\beta\rfloor_n}$ and $N_{\lfloor\alpha\rfloor_n} \subseteq N_\alpha$.

Then, let $x \in N_{\lfloor\alpha\rfloor_n}$. By definition and $\alpha \sqsubseteq_{lpof} \beta$, we have that $\lambda_{\lfloor\alpha\rfloor_n}(x) = \lambda_\alpha(x) \leq \lambda_\beta(x) = \lambda_{\lfloor\beta\rfloor_n}(x)$ and $\varphi_{\lfloor\alpha\rfloor_n}(x) = \varphi_\alpha(x) = \varphi_\beta(x) = \varphi_{\lfloor\beta\rfloor_n}(x)$. By Lemma B.3(3), $\mathrm{succ}_{\lfloor\alpha\rfloor_n}(x) \subseteq \mathrm{succ}_{\lfloor\beta\rfloor_n}(x) \setminus \mathrm{Bot}_{\lfloor\alpha\rfloor_n}{\uparrow_{\lfloor\beta\rfloor_n}}$; let us prove the reverse inclusion. So, let $y \in \mathrm{succ}_{\lfloor\beta\rfloor_n}(x)$; this means that $\mathrm{lev}_\beta(x) < n$ and $\mathrm{lev}_\beta(y) \leq n$. By assumption, $x \in N_\alpha$. If also $y \in N_\alpha$, then $y \in \mathrm{succ}_{\lfloor\alpha\rfloor_n}(x)$, because $\mathrm{lev}_\beta(y) = \mathrm{lev}_\alpha(y)$ and by Lemma B.3(1). If $y \notin N_\alpha$, then $y \in \mathrm{Bot}_\alpha{\uparrow_\beta}$, because $\alpha \sqsubseteq_{lpof} \beta$; this means that there exists $z \in \mathrm{Bot}_\alpha$ such that $z <_\beta y$. Hence, $\mathrm{lev}_\alpha(z) = \mathrm{lev}_\beta(z) < \mathrm{lev}_\beta(y) = \mathrm{lev}_\alpha(y) \leq n$; thus, $z \in N_{\lfloor\alpha\rfloor_n} \subseteq N_{\lfloor\beta\rfloor_n}$ and so $z <_{\lfloor\beta\rfloor_n} y$. By definition of truncation, $\lambda_{\lfloor\alpha\rfloor_n}(z) = \lambda_\alpha(z) = \bot$; hence, we can conclude that $y \in \mathrm{Bot}_{\lfloor\alpha\rfloor_n}{\uparrow_{\lfloor\beta\rfloor_n}}$. □

We extend the truncation to a set of lpos in the obvious way: $\lfloor D\rfloor_n \triangleq \{\lfloor\beta\rfloor_n \mid \beta \in D\}$.

Lemma B.9 (Scott continuity of truncation). *If D is a directed set of lpof s, then $\sup\lfloor D\rfloor_n$ exists and it is $\lfloor \sup D\rfloor_n$, for every $n \in \mathbb{N}$.*

Proof. By Lemma 4.8, $\sup D = \alpha$, where:

$$N_\alpha \triangleq \bigcup_{\beta\in D} N_\beta \qquad <_\alpha \triangleq \bigcup_{\beta\in D} <_\beta \qquad \lambda_\alpha(x) \triangleq \sup_{\beta\in D\,:\,x\in N_\beta} \lambda_\beta(x) \qquad \varphi_\alpha(x) \triangleq \psi_x$$

We now want to prove that γ, given by

$$N_\gamma \triangleq \bigcup_{\beta\in D} N_{\lfloor\beta\rfloor_n} \qquad <_\gamma \triangleq \bigcup_{\beta\in D} <_{\lfloor\beta\rfloor_n} \qquad \lambda_\gamma(x) \triangleq \sup_{\beta\in D\,:\,x\in N_{\lfloor\beta\rfloor_n}} \lambda_{\lfloor\beta\rfloor_n}(x) \qquad \varphi_\gamma(x) \triangleq \psi_x$$

is the sup of $\lfloor D\rfloor_n$ and that it coincides with $\lfloor\alpha\rfloor_n$.

The fact that $\gamma = \sup\lfloor D\rfloor_n$ follows by Lemma 4.8, once we prove that $\lfloor D\rfloor_n$ is directed; so, take $\beta_1, \beta_2 \in \lfloor D\rfloor_n$ and prove that they admit an upper bound in $\lfloor D\rfloor_n$. By definition, there exist $\beta_1', \beta_2' \in D$ such that $\beta_1 = \lfloor\beta_1'\rfloor_n$ and $\beta_2 = \lfloor\beta_2'\rfloor_n$. Since D is directed, there exist a $\beta \in D$ such that $\beta_1', \beta_2' \sqsubseteq_{lpof} \beta$. By Lemma B.8, $\beta_1, \beta_2 \sqsubseteq_{lpof} \lfloor\beta\rfloor_n$ and, by definition, $\lfloor\beta\rfloor_n \in \lfloor D\rfloor_n$.

We are left with proving that $\gamma = \lfloor\alpha\rfloor_n$. First, $x \in N_\gamma$ iff there exists $\beta \in D$ such that $x \in N_{\lfloor\beta\rfloor_n}$, i.e. $x \in N_\beta$ and $\mathrm{lev}_\beta(x) = n$; this happens iff $x \in \bigcup_{\beta\in D} N_\beta$ and $\mathrm{lev}_\alpha(x) = n$, being $\mathrm{lev}_\alpha(x) = \mathrm{lev}_\beta(x)$ (since $\beta \sqsubseteq_{lpof} \alpha$), i.e. $x \in N_{\lfloor\alpha\rfloor_n}$. Then,

$$
\begin{aligned}
<_{\lfloor\alpha\rfloor_n} &= <_\alpha \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n}) \\
&= \left(\bigcup_{\beta\in D} <_\beta\right) \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n}) \\
&= \bigcup_{\beta\in D} (<_\beta \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n})) \\
&= \bigcup_{\beta\in D} ((<_\alpha \cap (N_\beta \times N_\beta)) \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n})) \\
&= \bigcup_{\beta\in D} (<_\alpha \cap ((N_\beta \times N_\beta) \cap (N_{\lfloor\alpha\rfloor_n} \times N_{\lfloor\alpha\rfloor_n})) \\
&\overset{\dagger}{=} \bigcup_{\beta\in D} (<_\alpha \cap (N_{\lfloor\beta\rfloor_n} \times N_{\lfloor\beta\rfloor_n})) \\
&\overset{\ddagger}{=} \bigcup_{\beta\in D} <_{\lfloor\beta\rfloor_n} = <_\gamma
\end{aligned}
$$

Step (†) holds because $N_{\lfloor\alpha\rfloor_n} = N_\gamma = \bigcup_{\beta'\in D} N_{\lfloor\beta'\rfloor_n}$ and so $N_{\lfloor\alpha\rfloor_n} \cap N_\beta = (\bigcup_{\beta'\in D} N_{\lfloor\beta'\rfloor_n}) \cap N_\beta = \bigcup_{\beta'\in D}(N_{\lfloor\beta'\rfloor_n} \cap N_\beta) = N_{\lfloor\beta\rfloor_n}$, once we prove that $N_{\lfloor\beta'\rfloor_n} \cap N_\beta \subseteq N_{\lfloor\beta\rfloor_n}$, for every $\beta' \in D$ (indeed, obviously $N_{\lfloor\beta\rfloor_n} \subseteq \bigcup_{\beta'\in D}(N_{\lfloor\beta'\rfloor_n} \cap N_\beta)$). Fix $\beta' \neq \beta$ and $x \in N_{\lfloor\beta'\rfloor_n} \cap N_\beta$; this means that $\mathrm{lev}\beta'(x) \leq n$, and so $\mathrm{lev}_\beta(x) = \mathrm{lev}_\alpha(x) = \mathrm{lev}\beta'(x) \leq n$, since $\beta, \beta' \sqsubseteq_{lpof} \alpha$. This allows us to conclude that $x \in N_{\lfloor\beta\rfloor_n}$, as desired. Step (‡) is proved by double inclusion. Trivially, $x <_{\lfloor\beta\rfloor_n} y$ means that $x <_\beta y$ and $\mathrm{lev}_\beta(x), \mathrm{lev}_\beta(y) \leq n$; hence, $x, y \in N_{\lfloor\beta\rfloor_n}$ and, since $\beta \sqsubseteq_{lpof} \alpha$, we also have $x <_\alpha y$, as desired. Conversely, if $x <_\alpha y$ and $x, y \in N_{\lfloor\beta\rfloor_n}$, then $x, y \in N_\beta$ and, since $\beta \sqsubseteq_{lpof} \alpha$, also $x <_\beta y$; thus, by definition, $x <_{\lfloor\beta\rfloor_n} y$.

Finally, fix $x \in N_\gamma = N_{\lfloor\alpha\rfloor_n}$. By definition, $\varphi_\gamma(x) = \varphi_{\lfloor\alpha\rfloor_n}(x)$. For the labeling, first observe that $\mathrm{lev}_\gamma(x) = \mathrm{lev}_{\lfloor\beta\rfloor_n}(x) = \mathrm{lev}_\alpha(x)$, for every $\beta \in D$ such that $x \in N_\beta$. If $\mathrm{lev}_\gamma(x) < n$, then by construction $\lambda_\gamma(x) = \sup_{\beta\in D\,:\,x\in N_{\lfloor\beta\rfloor_n}} \lambda_{\lfloor\beta\rfloor_n}(x) = \sup_{\beta\in D\,:\,x\in N_\beta} \lambda_\beta(x) = \lambda_\alpha(x) = \lambda_{\lfloor\alpha\rfloor_n}(x)$. Otherwise, $x \in N_{\lfloor\beta\rfloor_n}$ and $\lambda_{\lfloor\beta\rfloor_n}(x) = \bot$, for all β such that $x \in N_\beta$; hence, $\lambda_\gamma(x) = \sup_{\beta\in D\,:\,x\in N_{\lfloor\beta\rfloor_n}} \lambda_{\lfloor\beta\rfloor_n}(x) = \bot = \lambda_{\lfloor\alpha\rfloor_n}(x)$. □

C Proofs of the results on Pomsets

C.1 Basic properties

The following lemma provides alternative characterizations of the pomset order that will be convenient in some proofs and definitions later on.

Lemma 4.5. *The following two statements hold:*

(1) $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta} \iff \forall \beta \in \boldsymbol{\beta}.\ \exists \alpha \in \boldsymbol{\alpha}.\ \alpha \sqsubseteq_{lpof} \beta$*; and*
(2) $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta} \implies \forall \alpha \in \boldsymbol{\alpha}.\ |nodes \setminus N_\alpha| = \infty \implies \exists \beta \in \boldsymbol{\beta}.\ \alpha \sqsubseteq_{lpof} \beta$

Proof. Starting with (1), the reverse direction is immediate since $\boldsymbol{\beta}$ is nonempty. For the forward direction, suppose that $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$ and therefore there exist $\alpha \in \boldsymbol{\alpha}$ and $\beta \in \boldsymbol{\beta}$ such that $\alpha \sqsubseteq_{lpof} \beta$. Now take any $\beta' \in \boldsymbol{\beta}$, so clearly $\beta \equiv \beta'$ and thus there is a bijection $f\colon N_\beta \to N_{\beta'}$. Applying that same bijection to α, we get that $f(\alpha) \equiv \alpha \in \boldsymbol{\alpha}$ and $f(\alpha) \sqsubseteq_{lpof} \beta'$.

We now show (2). Suppose that $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$ and therefore there exist $\alpha \in \boldsymbol{\alpha}$ and $\beta \in \boldsymbol{\beta}$ such that $\alpha \sqsubseteq_{lpof} \beta$. Now take any $\alpha' \in \boldsymbol{\alpha}$ such that $|\mathit{nodes} \setminus N_{\alpha'}| = \infty$. We know that $\alpha \equiv \alpha'$ and thus there is a bijection $f\colon N_\alpha \to N_{\alpha'}$. Since $|\mathit{nodes} \setminus N_{\alpha'}| = \infty$, there are enough leftover nodes to extend f to $g\colon N_\beta \to X$ where $X \supset N_{\alpha'}$ is some set of nodes and $f(x) = g(x)$ for all $x \in N_\alpha$. Applying g to β, we therefore get $g(\beta) \equiv \beta \in \boldsymbol{\beta}$ and $\alpha' \sqsubseteq_{lpof} g(\beta)$. □

Corollary 4.6. *$\langle pom(L), \sqsubseteq_{pom}\rangle$ is a poset.*

Proof. For reflexivity, take any $\boldsymbol{\alpha} \in pom(L)$ and any $\alpha \in \boldsymbol{\alpha}$. By Lemma 4.3, we know that $\alpha \sqsubseteq_{lpof} \alpha$, therefore $[\alpha] \sqsubseteq_{pom} [\alpha]$ by definition, *i.e.,* $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\alpha}$.

For transitivity, suppose that $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$ and $\boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\gamma}$. By $\boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\gamma}$, we know that there exist $\beta \in \boldsymbol{\beta}$ and $\gamma \in \boldsymbol{\gamma}$ such that $\beta \sqsubseteq_{lpof} \gamma$. By $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$ and Lemma 4.5(1), we know there exists an $\alpha \in \boldsymbol{\beta}$ such that $\alpha \sqsubseteq_{lpof} \beta$. Since $\sqsubseteq_{lpof}$ is a partial order, $\alpha \sqsubseteq_{lpof} \gamma$, therefore $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\gamma}$.

For antisymmetry, suppose that $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$ and $\boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\alpha}$. By $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$, we know that there exist $\alpha \in \boldsymbol{\alpha}$ and $\beta \in \boldsymbol{\beta}$ such that $\alpha \sqsubseteq_{lpof} \beta$. By $\boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\alpha}$ and Lemma 4.5(1), we get that there exists $\beta' \in \boldsymbol{\beta}$ such that $\beta' \sqsubseteq_{lpof} \alpha$. So, we have that $\beta' \sqsubseteq_{lpof} \alpha \sqsubseteq_{lpof} \beta$ where $\beta \equiv \beta'$. In other words, there exists a bijection f such that $f(\beta) \sqsubseteq_{lpof} \beta$. By the definition of $\sqsubseteq_{lpof}$, f must map each node to one with the same label, and identical predecessors and successors, thus it must be that $\beta = f(\beta)$, and therefore $\alpha \sqsubseteq_{lpof} \beta$ and $\beta \sqsubseteq_{lpof} \alpha$, so by antisymmetry of $\sqsubseteq_{lpof}$, $\alpha = \beta$, and therefore $\boldsymbol{\alpha} = [\alpha] = [\beta] = \boldsymbol{\beta}$. □

C.2 Truncation

As mentioned in Appendix B.2, we extend the truncation operation to pomsets in the expected way: $\lfloor \boldsymbol{\alpha} \rfloor_n \triangleq \{\lfloor \alpha \rfloor_n \mid \alpha \in \boldsymbol{\alpha}\}$.

Lemma C.1 (Monotonicity of Truncation). *If $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$, then $\forall n \in \mathbb{N}.\ \lfloor \boldsymbol{\alpha} \rfloor_n \sqsubseteq_{pom} \lfloor \boldsymbol{\beta} \rfloor_n$.*

Proof. We know that there exist $\alpha \in \boldsymbol{\alpha}$ and $\beta \in \boldsymbol{\beta}$ such that $\alpha \sqsubseteq_{lpof} \beta$; by Lemma B.8, this entails that $\lfloor \alpha \rfloor_n \sqsubseteq_{lpof} \lfloor \beta \rfloor_n$ and so, $\lfloor \boldsymbol{\alpha} \rfloor_n \sqsubseteq_{pom} \lfloor \boldsymbol{\beta} \rfloor_n$. □

Lemma C.2. *If $\lfloor \boldsymbol{\alpha} \rfloor_n \sqsubseteq_{pom} \boldsymbol{\beta}$ for all $n \in \mathbb{N}$, then $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$.*

Proof. (*Adapted from [de Bakker and Warmerdam 1990, Proposition 3.1.10]*) Fix any $\alpha \in \boldsymbol{\alpha}$ and $\beta \in \boldsymbol{\beta}$. We will show that there exists $N \subseteq_\downarrow N_\beta$ and a bijection $f\colon N_\alpha \to N$ such that $f(\alpha) \sqsubseteq_{lpof} \beta$; and thus $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$.

We construct a tree of bijections as follows. Each node in the tree is a triple (f, X, n) where $n \in \mathbb{N}$, $X \subseteq_\downarrow N_\beta$, and $f\colon N_{\lfloor \alpha \rfloor_n} \to X$ is a bijection such that $f(\lfloor \alpha \rfloor_n) \sqsubseteq_{lpof} \beta$. We add an edge from (f, X, n) to $(g, Y, n+1)$ if $X \subseteq Y$ and $g(x) = f(x)$ for all $x \in N_{\lfloor \alpha \rfloor_n}$ (*i.e.,* f is g restricted to $N_{\lfloor \alpha \rfloor_n}$ or, equivalently, g is an extension of f). Let x_α and x_β be the roots of α and β respectively, and $f_0\colon \{x_\alpha\} \to \{x_\beta\}$ be the obvious bijection. We will now show that the structure we described above defines an infinite tree with $(f_0, \{x_\beta\}, 0)$ as the root.

We first show that, for each $n \in \mathbb{N}$, there exists a possible node with n as the third element; hence, there are infinitely many nodes (but we do not yet know that they can be all reached from the root). Fix any $n \in \mathbb{N}$. Since $\lfloor \boldsymbol{\alpha} \rfloor_n \sqsubseteq_{pom} \boldsymbol{\beta} = [\beta]$, then there exists an $\alpha' \in \boldsymbol{\alpha}$ such that $\lfloor \alpha' \rfloor_n \sqsubseteq_{lpof} \beta$. Since $\alpha \equiv \alpha'$, then there must be a bijection $g\colon N_\alpha \to N_{\alpha'}$. Now, let $X = \{g(x) \mid x \in N_{\lfloor \alpha \rfloor_n}\}$ and $f\colon N_{\lfloor \alpha \rfloor_n} \to X$ be defined as $f(x) = g(x)$, which is clearly a bijection since g is. Clearly

$$f(\lfloor \alpha \rfloor_n) = \lfloor g(\alpha) \rfloor_n = \lfloor \alpha' \rfloor_n \sqsubseteq_{lpof} \beta$$

and so (f, X, n) is a node.

We now show that any node (f, X, n) as described above can be reached from the root. The proof is by induction on n. If $n = 0$, then the node must be $(f_0, \{x_\beta\}, 0)$, hence it is the root. If $(f, X, n+1)$ is a node, then $f\colon N_{\lfloor\alpha\rfloor_{n+1}} \to X$ is a bijection. Now, let $Y = \{f(x) \mid x \in N_{\lfloor\alpha\rfloor_n}\}$. So, $g\colon N_{\lfloor\alpha\rfloor_n} \to Y$ defined as $g(x) = f(x)$ is also a bijection. We also know that $f(\lfloor\alpha\rfloor_{n+1}) \sqsubseteq_{lpof} \beta$; since $g(\lfloor\alpha\rfloor_n) \sqsubseteq_{lpof} f(\lfloor\alpha\rfloor_{n+1})$, we have that $g(\lfloor\alpha\rfloor_n) \sqsubseteq_{lpof} \beta$ by transitivity. Therefore, (g, Y, n) is a valid node and there is an edge from (g, Y, n) to $(f, X, n+1)$. By the induction hypothesis, (g, Y, n) is reachable from the root, therefore so is $(f, X, n+1)$.

Now, let $k = |N_{\lfloor\alpha\rfloor_n}|$, which must be finite since only finitely many nodes can occur at each level. Therefore, there can only be finitely many elements in each level of the tree, since there are only only finitely many downward closed subsets of N_β of size k; so, the tree must be finitely branching and, by König's Lemma, there exists an infinite path:

$$(f_n, X_n, n)_{n\in\mathbb{N}} \quad \text{where} \quad X_n \subseteq X_{n+1} \quad \text{and} \quad \forall x \in X_n.\, f_n(x) = f_{n+1}(x) \quad \text{for all } n \in \mathbb{N}$$

Now let $X = \bigcup_{n\in\mathbb{N}} X_n$ and let $f\colon N_\alpha \to X$ be defined as $f(x) = f_{\mathsf{lev}_\alpha(x)}(x)$. We now show that $f(\alpha) \sqsubseteq_{lpof} \beta$. First, clearly $N_{f(\alpha)} = X \subseteq_\downarrow N_\beta$ since it is the union of downward closed sets. Now, for the order, we have:

$$\begin{aligned}
<_{f(\alpha)} &= \bigcup_{n\in\mathbb{N}} <_{f_n(\lfloor\alpha\rfloor_n)} \\
&= \bigcup_{n\in\mathbb{N}} \left(<_\beta \cap \left(N_{f_n(\lfloor\alpha\rfloor_n)} \times N_{f_n(\lfloor\alpha\rfloor_n)}\right)\right) \\
&= <_\beta \cap \bigcup_{n\in\mathbb{N}} \left(N_{f_n(\lfloor\alpha\rfloor_n)} \times N_{f_n(\lfloor\alpha\rfloor_n)}\right) \\
&= <_\beta \cap \left(N_{f(\alpha)} \times N_{f(\alpha)}\right)
\end{aligned}$$

Now, for each $x \in N_{f(\alpha)}$, let $n = \mathsf{lev}_\alpha(x)$. We have that:

$$\begin{aligned}
\lambda_{f(\alpha)}(x) &= \lambda_\alpha\left(f^{-1}(x)\right) \\
&= \lambda_\alpha\left(f_n^{-1}(x)\right) \\
&= \lambda_\alpha\left(f_{n+1}^{-1}(x)\right) \\
&= \lambda_{\lfloor\alpha\rfloor_{n+1}}\left(f_{n+1}^{-1}(x)\right) \\
&= \lambda_{f_{n+1}(\lfloor\alpha\rfloor_{n+1})}(x) \\
&\leq \lambda_\beta(x)
\end{aligned}$$

By an analogous argument, $\varphi_{f(\alpha)}(x) = \varphi_\beta(x)$. Now, by Lemma B.3, we know that $\mathrm{succ}_{f(\alpha)}(x) \subseteq \mathrm{succ}_\beta(x) \setminus \mathrm{Bot}_{f(\alpha)}{\uparrow_\beta}$, it just remains to show the reverse inclusion. Take any $y \in \mathrm{succ}_\beta(x)$ such that $y \notin \mathrm{Bot}_{f(\alpha)}{\uparrow_\beta}$. Let $m = \mathsf{lev}_\beta(y)$ and note that $y \notin \mathrm{Bot}_{f_m(\lfloor\alpha\rfloor_m)}$ since $f(\alpha)$ and $f_m(\lfloor\alpha\rfloor_m)$ are identical up to level m. Since $f_m(\lfloor\alpha\rfloor_m) \sqsubseteq_{lpof} \beta$, then $y \in \mathrm{succ}_{f_m(\lfloor\alpha\rfloor_m)}(x)$, and therefore $y \in \mathrm{succ}_{f(\alpha)}(x)$. □

C.3 The Pomset Dcpo Structure

Lemma C.3. *Let $D = \{\alpha_i \mid i \in I\} \subseteq lpof(L)$ be a directed set and let $\boldsymbol{D} = \{[\alpha_i] \mid i \in I\} \subseteq pom(L)$; then, $\sup \boldsymbol{D}$ exists and $\sup \boldsymbol{D} = [\sup D]$.*

Proof. By Lemma 4.8, we know that D admits a sup, let us call it α and let $\boldsymbol{\alpha} = [\alpha]$; we have to prove that $\boldsymbol{\alpha} = \sup \boldsymbol{D}$, *i.e.*,

(1) for all $i \in I$ it holds that $[\alpha_i] \sqsubseteq_{pom} [\alpha]$, and
(2) for every $\boldsymbol{\beta}$, if $[\alpha_i] \sqsubseteq_{pom} \boldsymbol{\beta}$ for all $i \in I$, then $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$.

The first property is trivial, by the definition of $\sqsubseteq_{lpof}$, since $\alpha_i \sqsubseteq_{lpof} \alpha$ for all $i \in I$.

For the second property, take some $\boldsymbol{\beta}$ such that $[\alpha_i] \sqsubseteq_{pom} \boldsymbol{\beta}$ for all $i \in I$ and let $A_n = \{\beta \in \boldsymbol{\beta} \mid \lfloor\alpha\rfloor_n \sqsubseteq_{lpof} \beta\}$.

We first prove that $A_n \neq \emptyset$ for all $n \in \mathbb{N}$. Let $N = \{x \in N_\alpha \mid \mathsf{lev}_\alpha(x) \leq n\}$, which by definition is a finite set. By the definition of supremum for lpos, for each $x \in N$ there exists an $i_x \in I$ such that $x \in N_{\alpha_{i_x}}$ and $\lambda_{\alpha_{i_x}}(x) = \lambda_\alpha(x)$. Since $\{\alpha_{i_x} \mid x \in N\}$ is a finite subset of D, which is directed, there must be a $\alpha_j \in D$ such that $\alpha_{i_x} \sqsubseteq_{lpof} \alpha_j$ for all $x \in N$. By construction, $\lfloor\alpha_j\rfloor_n = \lfloor\alpha\rfloor_n$. Since $\alpha_j \in D$, we also know that $[\alpha_j] \sqsubseteq_{pom} \boldsymbol{\beta}$, so there must be a $\beta \in \boldsymbol{\beta}$ such that $\alpha_j \sqsubseteq_{lpof} \beta$. This gives us $\lfloor\alpha\rfloor_n = \lfloor\alpha_j\rfloor_n \sqsubseteq_{lpof} \alpha_j \sqsubseteq_{lpof} \beta$, so $\beta \in A_n$ and therefore $A_n \neq \emptyset$.

Now, for each $n \in \mathbb{N}$, since $A_n \neq \emptyset$, we know that there exists a $\beta_n \in \boldsymbol{\beta}$ such $\lfloor\alpha\rfloor_n \sqsubseteq_{lpof} \beta_n$; therefore we get that $\lfloor\boldsymbol{\alpha}\rfloor_n = [\lfloor\alpha\rfloor_n] \sqsubseteq_{pom} [\beta_n] = \boldsymbol{\beta}$ so by Lemma C.2, we get that $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\beta}$. □

Lemma C.4. *For every transfinite chain of pomsets $(\boldsymbol{\alpha}_\delta)_{\delta<\zeta}$ (such that $\boldsymbol{\alpha}_\delta \sqsubseteq_{pom} \boldsymbol{\alpha}_{\delta'}$ if $\delta < \delta'$), there exists a corresponding chain of LPOF s $(\alpha_\delta)_{\delta<\zeta}$ such that $\alpha_\delta \in \boldsymbol{\alpha}_\delta$, for all $\delta < \zeta$.*

Proof. First, note that since *nodes* is countably infinite, then by Cantor's zig-zag pairing argument, there exists a bijection $f : \mathbb{N} \times \mathit{nodes} \to \mathit{nodes}$. For each $n \in \mathbb{N}$, let $X_n \triangleq \{x \in \mathit{nodes} | \exists y.\ x = f(n, y)\}$, so the X_n sets are countably infinite and pairwise disjoint.

We will construct the chain $(\alpha_\delta)_{\delta<\zeta}$ by transfinite induction, with the additional constraint that for all $\delta < \zeta$, and $x \in N_{\alpha_\delta}$, $x \in X_n$ where $n = \mathsf{lev}_{\alpha_\delta}(x)$, *i.e.,* every node at level n must be drawn from the set X_n. Note that each X_n is countably infinite, ensuring that there are infinitely many node identifiers available for use at each level. However, since each level of an LPOF must be finite, the constraint that nodes at level n are drawn from X_n ensures that $|\mathit{nodes} \setminus N_{\alpha_\delta}| = \infty$ for all $\delta < \zeta$.

Now, for any β, define $f(\beta)$ to permute β by sending x to $f(\mathsf{lev}_\beta(x), x)$, and note that by definition $f(\beta)$ fulfills the invariant that each node at level n is drawn from X_n.

To construct the chain, we start with the base case. Take an arbitrary $\alpha_0 \in \boldsymbol{\alpha}_0$ and fix $f(\alpha_0)$ as the choice. For the successor case, take any ordinal $\delta < \zeta$ such that α_δ is already fixed. By Lemma 4.5(2) and the fact that $\boldsymbol{\alpha}_\delta \sqsubseteq_{pom} \boldsymbol{\alpha}_{\delta+1}$ and $|\mathit{nodes} \setminus N_{\alpha_\delta}| = \infty$, we know that there exists an $\alpha_{\delta+1} \in \boldsymbol{\alpha}_{\delta+1}$ such that $\alpha_\delta \sqsubseteq_{lpof} \alpha_{\delta+1}$. Now, apply f to permute only the nodes in $N_{\alpha_{\delta+1}} \setminus N_{\alpha_\delta}$, and fix the result as the choice of $\alpha_{\delta+1}$.

Finally, for any limit ordinal ξ, consider $\{\alpha_\delta \mid \delta < \xi\}$ that is a chain (thus, also a directed set); we have to choose an $\alpha_\xi \in \boldsymbol{\alpha}_\xi$ that is bigger (w.r.t. $\sqsubseteq_{lpof}$) than any α_δ. By Lemma C.3, we know that $\sup_{\delta<\xi} \boldsymbol{\alpha}_\delta = [\sup_{\delta<\xi} \alpha_\delta]$. Note also that each node of $\sup_{\delta<\xi} \alpha_\delta$ appears in some α_δ where $\delta < \xi$, and thus $\sup_{\delta<\xi} \alpha_\delta$ obeys the invariant that all nodes at level n are drawn from X_n. Since the supremum is the *least* upper bound, and $\boldsymbol{\alpha}_\xi$ is an upper bound of $\{\boldsymbol{\alpha}_\delta \mid \delta < \xi\}$, then $\sup_{\delta<\xi} \boldsymbol{\alpha}_\delta \sqsubseteq_{pom} \boldsymbol{\alpha}_\xi$. By Lemma 4.5(2), there exists an $\alpha_\xi \in \boldsymbol{\alpha}_\xi$ such that $\sup_{\delta<\xi} \alpha_\delta \sqsubseteq_{lpof} \alpha_\xi$, which also means that $\alpha_\delta \sqsubseteq_{lpof} \alpha_\xi$ for all $\delta < \xi$. All that remains is to apply f to permute the nodes in $N_{\alpha_\xi} \setminus \bigcup_{\delta<\xi} N_{\alpha_\delta}$. □

Lemma 4.10. *$\langle pom(L), \sqsubseteq_{pom}\rangle$ is a pointed DCPO.*

Proof. By [Abramsky and Jung 1995, Proposition 2.1.15], $\langle pom(L), \sqsubseteq_{pom}\rangle$ is a dcpo if any transfinite chain $(\boldsymbol{\alpha}_\delta)_{\delta<\zeta}$ has a supremum. By Lemma C.4, there is a corresponding chain of lpos $(\alpha_\delta)_{\delta<\zeta}$ such that $\alpha_\delta \in \boldsymbol{\alpha}_\delta$ for each $\delta < \zeta$. Therefore, by Lemma C.3, we get that $\sup_{\delta<\zeta} \boldsymbol{\alpha}_\delta = [\sup_{\delta<\zeta} \alpha_\delta]$, so the chain has a supremum.

Finally, pointedness follows from the fact that $\langle\bot\rangle \sqsubseteq_{pom} \boldsymbol{\alpha}$, for every $\boldsymbol{\alpha}$, where we let $\langle\bot\rangle$ to denote the isomorphism class of the singleton LPOs whose node is labelled with $\bot$. □

D Approximation and Extension

Lemma D.1. *For any $\alpha \in lpof(L)$, the set $\lfloor\alpha\rfloor_{\mathsf{fin}}$ is directed.*

Proof. Take any $\beta, \gamma \in \lfloor \alpha \rfloor_{\mathsf{fin}}$, so $\beta, \gamma \in \mathit{lpof}_{\mathsf{fin}}(L)$ and $\beta, \gamma \sqsubseteq_{\mathit{lpof}} \alpha$. Let n be the maximum level of any node in β or γ, and $\alpha' = \lfloor \alpha \rfloor_{n+1}$. Clearly, $\alpha' \in \lfloor \alpha \rfloor_{\mathsf{fin}}$ by Lemma B.7.

We conclude the proof by showing that α' is an upper bound for β and γ; we will only show $\beta \sqsubseteq_{\mathit{lpof}} \alpha'$, as the case for γ is identical. Clearly $N_\beta \subseteq N_{\alpha'}$ since β contains a subset of the nodes from α at level at most n and α' contains all nodes from α up to level $n+1$. We will now show that it is downward closed. Take any $x \in N_\beta$ and $y <_{\alpha'} x$; this means that $y <_\alpha x$. Being $N_\beta \subseteq_\downarrow N_\alpha$, we have that $y \in N_\beta$.

Next, since $\alpha' \sqsubseteq_{\mathit{lpof}} \alpha$ and $N_\beta \subseteq N_{\alpha'}$, we have that:

$$\begin{aligned} <_{\alpha'} \cap (N_\beta \times N_\beta) &= (<_\alpha \cap (N_{\alpha'} \times N_{\alpha'})) \cap (N_\beta \times N_\beta) \\ &= <_\alpha \cap ((N_{\alpha'} \cap N_\beta) \times (N_{\alpha'} \cap N_\beta)) \\ &= <_\alpha \cap (N_\beta \times N_\beta) \\ &= <_\beta \end{aligned}$$

Finally, let $x \in N_\beta$. By construction, $\varphi_\beta(x) = \varphi_\alpha(x) = \varphi_{\alpha'}(x)$ and $\lambda_\beta(x) \leq \lambda_\alpha(x) = \lambda_{\alpha'}(x)$, since $\mathsf{lev}_\alpha(x) \leq n$. For the last requirement, by Lemma B.3(3) we have that $\mathrm{succ}_\beta(x) \subseteq \mathrm{succ}_{\alpha'}(x) \setminus \mathrm{Bot}_\beta{\uparrow_{\alpha'}}$; we need to prove the reverse inclusion. Let $y \in \mathrm{succ}_{\alpha'}(x)$ and $y \notin \mathrm{Bot}_\beta{\uparrow_{\alpha'}}$. We have two cases:

- $y \in N_\beta$: in this case, $y \in \mathrm{succ}_\beta(x)$ by Lemma B.3(1).
- $y \notin N_\beta$: Because y is finitely preceded in α', there exist $y_1, \ldots, y_n$ such that $y_1 = \min_{\alpha'}$, $y_n = y$ and $y_{i+1} \in \mathrm{succ}_{\alpha'}(y_i)$. Because of Lemma B.3(2), $y_{i+1} \in \mathrm{succ}_\alpha(y_i)$, for all i. Being $y_1 = \min_\beta$, there exists an i ($1 < i < n$) such that $y_i \in N_\beta$ and $y_{i+1} \notin N_\beta$. Since $\beta \sqsubseteq_{\mathit{lpof}} \alpha$, we have that $\mathrm{succ}_\beta(y_i) \subseteq \mathrm{succ}_\alpha(y_i) \setminus \mathrm{Bot}_\beta{\uparrow_\alpha}$; so, $y_{i+1} \in \mathrm{Bot}_\beta{\uparrow_\alpha}$ and, consequently, $y \in \mathrm{Bot}_\beta{\uparrow_\alpha}$. This latter fact, together with $y \in N_{\alpha'}$, implies that $y \in \mathrm{Bot}_\beta{\uparrow_{\alpha'}}$: contradiction. □

Lemma D.2. *For any $\boldsymbol{\alpha} \in \mathit{pom}(L)$, the set $\lfloor \boldsymbol{\alpha} \rfloor_{\mathsf{fin}}$ is directed.*

Proof. Take any $\boldsymbol{\beta}, \boldsymbol{\beta'} \in \lfloor \boldsymbol{\alpha} \rfloor_{\mathsf{fin}}$, so $\boldsymbol{\beta}, \boldsymbol{\beta'} \in \mathit{pom}_{\mathsf{fin}}(L)$ and $\boldsymbol{\beta} \sqsubseteq_{\mathit{pom}} \boldsymbol{\alpha}$ and $\boldsymbol{\beta'} \sqsubseteq_{\mathit{pom}} \boldsymbol{\alpha}$. Now take any $\alpha \in \boldsymbol{\alpha}$, by Lemma 4.5(1) we know that there is a $\beta \in \boldsymbol{\beta}$ and $\beta' \in \boldsymbol{\beta'}$ such that $\beta \sqsubseteq_{\mathit{lpof}} \alpha$ and $\beta' \sqsubseteq_{\mathit{lpof}} \alpha$. Therefore, $\beta, \beta' \in \lfloor \alpha \rfloor_{\mathsf{fin}}$, so by Lemma D.1, there exists a $\gamma \in \lfloor \alpha \rfloor_{\mathsf{fin}}$ such that $\beta \sqsubseteq_{\mathit{lpof}} \gamma$ and $\beta' \sqsubseteq_{\mathit{lpof}} \gamma$. Since $\gamma \in \lfloor \alpha \rfloor_{\mathsf{fin}}$, then $[\gamma] \in \lfloor \boldsymbol{\alpha} \rfloor_{\mathsf{fin}}$, and clearly $\boldsymbol{\beta} \sqsubseteq_{\mathit{pom}} [\gamma]$ and $\boldsymbol{\beta'} \sqsubseteq_{\mathit{pom}} [\gamma]$. □

Lemma D.3. *For any $\alpha \in \mathit{lpof}(L)$, $\alpha = \sup \lfloor \alpha \rfloor_{\mathsf{fin}}$.*

Proof. By Lemma D.1, $\lfloor \alpha \rfloor_{\mathsf{fin}}$ is a directed set, therefore its supremum is given by the construction in Lemma 4.8. We now show that this gives us exactly α. First, we show this for the nodes. We know that $N_{\sup \lfloor \alpha \rfloor_{\mathsf{fin}}} = \bigcup_{\beta \in \lfloor \alpha \rfloor_{\mathsf{fin}}} N_\beta$. Clearly, $\bigcup_{\beta \in \lfloor \alpha \rfloor_{\mathsf{fin}}} N_\beta \subseteq N_\alpha$ since each $N_\beta \subseteq N_\alpha$ by definition. For the reverse inclusion, take any $x \in N_\alpha$ and let $n = \mathsf{lev}_\alpha(x)$. Obviously, $x \in N_{\lfloor \alpha \rfloor_n}$, and by Lemma B.7, we get that $\lfloor \alpha \rfloor_n \in \lfloor \alpha \rfloor_{\mathsf{fin}}$. Therefore, $x \in \bigcup_{\beta \in \lfloor \alpha \rfloor_{\mathsf{fin}}} N_\beta$, so $N_\alpha \subseteq \bigcup_{\beta \in \lfloor \alpha \rfloor_{\mathsf{fin}}} N_\beta$ and thus $\bigcup_{\beta \in \lfloor \alpha \rfloor_{\mathsf{fin}}} N_\beta = N_\alpha$

By definition, $\beta \sqsubseteq_{\mathit{lpof}} \alpha$, for each $\beta \in \lfloor \alpha \rfloor_{\mathsf{fin}}$; so clearly α is an upper bound for $\lfloor \alpha \rfloor_{\mathsf{fin}}$ and therefore $\sup \lfloor \alpha \rfloor_{\mathsf{fin}} \sqsubseteq_{\mathit{lpof}} \alpha$ (since the supremum is the least upper bound). Hence, $<_{\sup \lfloor \alpha \rfloor_{\mathsf{fin}}} \subseteq <_\alpha$; we show the reverse inclusion as follows. Take any $x <_\alpha y$ and let $\gamma = \lfloor \alpha \rfloor_{\mathsf{lev}_\alpha(y)}$. Clearly $y \in N_\gamma$ and by Lemma B.7, $\gamma \in \lfloor \alpha \rfloor_{\mathsf{fin}}$. Since $N_\gamma \subseteq_\downarrow N_\alpha$, then $x \in N_\gamma$ too, which means that $x <_\gamma y$. This gives us:

$$(x, y) \quad \in \quad <_\gamma \quad \subseteq \quad \bigcup_{\beta \in \lfloor \alpha \rfloor_{\mathsf{fin}}} <_\beta \quad = \quad <_{\sup \lfloor \alpha \rfloor_{\mathsf{fin}}}$$

Finally, for any $x \in N_\alpha$, we have:

- $\varphi_{\sup \lfloor \alpha \rfloor_{\mathsf{fin}}}(x) = \varphi_\alpha(x)$: Take any $\beta \in \lfloor \alpha \rfloor_{\mathsf{fin}}$, so $\beta \sqsubseteq_{\mathit{lpof}} \sup \lfloor \alpha \rfloor_{\mathsf{fin}}$ and $\beta \sqsubseteq_{\mathit{lpof}} \alpha$, therefore $\varphi_{\lfloor \alpha \rfloor_{\mathsf{fin}}} = \varphi_\beta = \varphi_\alpha$.

- $$\begin{aligned}\lambda_{\sup \lfloor\alpha\rfloor_{\mathsf{fin}}}(x) &= \sup_{\beta \in \lfloor\alpha\rfloor_{\mathsf{fin}} : x \in N_\beta} \lambda_\beta(x) \\ &= \sup\{\lambda_\beta(x) \mid \beta \in \mathit{lpof}_{\mathsf{fin}}(L), \beta \sqsubseteq_{\mathit{lpof}} \alpha, x \in N_\beta\} \\ &\overset{1}{=} \sup\{\ell \mid \ell \leq \lambda_\alpha(x)\} \\ &\overset{2}{=} \lambda_\alpha(x)\end{aligned}$$

  where (1) holds by the definition of $\sqsubseteq_{\mathit{lpof}}$, that implies that the set above must include all the labels that are less than $\lambda_\alpha(x)$, and (2) holds since the set contains $\lambda_\alpha(x)$ itself. □

Lemma 4.13 (Finite Approximations). *For any $\boldsymbol{\alpha} \in \mathit{pom}(L)$, $\boldsymbol{\alpha} = \sup \lfloor\boldsymbol{\alpha}\rfloor_{\mathsf{fin}}$.*

Proof. Fix any $\alpha \in \boldsymbol{\alpha}$ and consider the set $\lfloor\alpha\rfloor_{\mathsf{fin}}$: by Lemma D.1 it is directed and by Lemma D.3 its sup is α. Then, by Lemma C.3, $\sup \lfloor\boldsymbol{\alpha}\rfloor_{\mathsf{fin}}$ exists and it is $[\alpha]$, i.e. $\boldsymbol{\alpha}$. □

Lemma D.4. *If $\alpha \in \mathit{lpof}_{\mathsf{fin}}(L)$, then there are only finitely many β such that $\beta \sqsubseteq_{\mathit{lpof}} \alpha$.*

Proof. Any $\beta \sqsubseteq_{\mathit{lpof}} \alpha$ is such that $N_\beta \subseteq N_\alpha$; in other words, $N_\beta \in \mathcal{P}(N_\alpha)$. If $|N_\alpha| = n$. then $|\mathcal{P}(N_\alpha)| = 2^n$ (actually, there are fewer possibilities, since N_β is downward closed, but 2^n is a sufficient upper bound). The order and formula of β are deterministically determined by N_β, since $<_\beta = <_\alpha \cap (N_\beta \times N_\beta)$ and $\varphi_\beta(x) = \varphi_\alpha(x)$. Let $k = \max_{x \in N_\alpha} |\lambda_\alpha(x)\!\downarrow_L|$, which must be a finite integer since it is the maximum of a finite set of integers. So, there are at most 2^n subsets of N_α and for each of those subsets, there are at most k choices for the labels of each node. Hence, there are at most $2^n \cdot n \cdot k$ possible $\beta \sqsubseteq_{\mathit{lpof}} \alpha$. □

Lemma D.5. *If $\boldsymbol{D} \subseteq \mathit{pom}(L)$ is a directed set, then $\lfloor\boldsymbol{D}\rfloor_n$ is directed and finite, for any $n \in \mathbb{N}$.*

Proof. We first show that $\lfloor\boldsymbol{D}\rfloor_n$ is directed. Take any $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2 \in \lfloor\boldsymbol{D}\rfloor_n$. This means that there is $\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2 \in \boldsymbol{D}$ such that $\boldsymbol{\beta}_i = \lfloor\boldsymbol{\gamma}_i\rfloor_n$ for each $i \in \{1, 2\}$, and since $\boldsymbol{D}$ is directed, then there is a $\boldsymbol{\gamma} \in \boldsymbol{D}$ such that each $\boldsymbol{\gamma}_i \sqsubseteq_{\mathit{pom}} \boldsymbol{\gamma}$. By monotonicity of truncation, we have that $\boldsymbol{\beta}_i = \lfloor\boldsymbol{\gamma}_i\rfloor_n \sqsubseteq_{\mathit{pom}} \lfloor\boldsymbol{\gamma}\rfloor_n$, and clearly $\lfloor\boldsymbol{\gamma}\rfloor_n \in \lfloor\boldsymbol{D}\rfloor_n$, therefore $\lfloor\boldsymbol{D}\rfloor_n$ is directed.

Because of Lemma C.1, we know that $\boldsymbol{\beta} \sqsubseteq_{\mathit{pom}} \lfloor\boldsymbol{\alpha}\rfloor_n$ for all $\boldsymbol{\beta} \in \lfloor\boldsymbol{D}\rfloor_n$. Fix any $\alpha \in \lfloor\boldsymbol{\alpha}\rfloor_n$, so clearly α is finite. By Lemma 4.5, we know that, for every $\boldsymbol{\beta} \in \lfloor\boldsymbol{D}\rfloor_n$, there is a $\beta \in \boldsymbol{\beta}$ such that $\beta \sqsubseteq_{\mathit{lpof}} \alpha$. By Lemma D.4, there can only be finitely many such elements. Thus we have that $\{\beta \mid \beta \sqsubseteq_{\mathit{lpof}} \alpha\}$ is finite, and also clearly $\lfloor\boldsymbol{D}\rfloor_n \subseteq \{[\beta] \mid \beta \sqsubseteq_{\mathit{lpof}} \alpha\}$; therefore, $\lfloor\boldsymbol{D}\rfloor_n$ is also finite. □

Lemma D.6. *If $\boldsymbol{D}$ is a directed set of pomsets, then $\sup\lfloor\boldsymbol{D}\rfloor_n = \lfloor\sup \boldsymbol{D}\rfloor_n$ for all $n \in \mathbb{N}$.*

Proof. We show that $\lfloor-\rfloor_n$ is chain continuous; indeed, by [Markowsky 1976, Corollary 3], any chain continuous function on a DCPO is also Scott continuous, and thus $\sup\lfloor\boldsymbol{D}\rfloor_n = \lfloor\sup \boldsymbol{D}\rfloor_n$.

Take any pomset chain $(\boldsymbol{\alpha}_\delta)_{\delta<\zeta}$; by Lemma C.4, there is a corresponding chain of LPOFs $(\alpha_\delta)_{\delta<\zeta}$ such that $\alpha_\delta \in \boldsymbol{\alpha}_\delta$ for all $\delta < \zeta$. Since truncation is monotonic (Lemma C.1), then $(\lfloor\boldsymbol{\alpha}_\delta\rfloor_n)_{\delta<\zeta}$ is also a chain and, by Lemma B.9, we know that $\sup_{\delta<\zeta}\lfloor\alpha_\delta\rfloor_n = \lfloor\sup_{\delta<\zeta}\alpha_\delta\rfloor_n$. Clearly $\lfloor\alpha_\delta\rfloor_n \in \lfloor\boldsymbol{\alpha}_\delta\rfloor_n$ for all $\delta < \zeta$, therefore by Lemma C.3, we get both $\sup_{\delta<\zeta}\boldsymbol{\alpha}_\delta = [\sup_{\delta<\zeta}\alpha_\delta]$ and $\sup_{\delta<\zeta}\lfloor\boldsymbol{\alpha}_\delta\rfloor_n = [\sup_{\delta<\zeta}\lfloor\alpha_\delta\rfloor_n]$. Putting all these facts together, we get:

$$\sup_{\delta<\zeta}\lfloor\boldsymbol{\alpha}_\delta\rfloor_n = [\sup_{\delta<\zeta}\lfloor\alpha_\delta\rfloor_n] = [\lfloor\sup_{\delta<\zeta}\alpha_\delta\rfloor_n] = \lfloor[\sup_{\delta<\zeta}\alpha_\delta]\rfloor_n = \lfloor\sup_{\delta<\zeta}\boldsymbol{\alpha}_\delta\rfloor_n$$ □

Lemma 4.15 (Compactness). $\mathsf{K}(\mathit{pom}(L)) = \mathit{pom}_{\mathsf{fin}}(L)$.

Proof. We first show that any compact pomset is finite, *i.e.*, $\mathsf{K}(\mathit{pom}(L)) \subseteq \mathit{pom}_{\mathsf{fin}}(L)$. Take any $\boldsymbol{\alpha} \in \mathsf{K}(\mathit{pom}(L))$, so for any directed set A, if $\boldsymbol{\alpha} \sqsubseteq_{\mathit{lpof}} \sup A$ then $\boldsymbol{\alpha} \sqsubseteq_{\mathit{lpof}} \boldsymbol{\gamma}$ for some $\boldsymbol{\gamma} \in A$. Now, let $A = \lfloor\boldsymbol{\alpha}\rfloor_{\mathsf{fin}}$, which is directed by Lemma D.2. By Lemma 4.13, we also know that $\boldsymbol{\alpha} = \sup A$, so

$\boldsymbol{\alpha} \sqsubseteq_{pom} \sup A$. Since $\boldsymbol{\alpha}$ is compact, there must be some $\boldsymbol{\gamma} \in A$ such that $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\gamma}$. Since $\boldsymbol{\gamma} \in A$, then $\boldsymbol{\gamma} \in pom_{\mathsf{fin}}(L)$; therefore, since $\boldsymbol{\alpha} \sqsubseteq_{lpof} \boldsymbol{\gamma}$, then $\boldsymbol{\alpha} \in pom_{\mathsf{fin}}(L)$ too.

We now show that any finite pomset is compact, *i.e.,* $pom_{\mathsf{fin}}(L) \subseteq \mathsf{K}(pom(L))$. Taking any $\boldsymbol{\alpha} \in pom_{\mathsf{fin}}(L)$, we must show that $\boldsymbol{\alpha} \ll \boldsymbol{\alpha}$. That is, for any directed set A, if $\boldsymbol{\alpha} \sqsubseteq_{pom} \sup A$, then there exists $\boldsymbol{\gamma} \in A$ such that $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\gamma}$.

To this aim, let A be a directed set such that $\boldsymbol{\alpha} \sqsubseteq_{pom} \sup A$. Since $\boldsymbol{\alpha}$ is finite, let $\alpha \in \boldsymbol{\alpha}$ be an arbitrary representative lpo, then $n = \max_{x \in N_\alpha} \mathsf{lev}(x) + 1$ must be finite. Moreover, $\lfloor \boldsymbol{\alpha} \rfloor_n = \boldsymbol{\alpha}$, since n is one level above the maximum level of $\boldsymbol{\alpha}$ and therefore the truncation only converts labels to $\bot$ above the maximum level of $\boldsymbol{\alpha}$ and removes nodes even above that. Therefore, by montonicity (Lemma C.1) and Scott continuity (Lemma D.6) of truncation, we have:

$$\boldsymbol{\alpha} = \lfloor \boldsymbol{\alpha} \rfloor_n \sqsubseteq_{pom} \lfloor \sup A \rfloor_n = \sup\{\lfloor \boldsymbol{\beta} \rfloor_n \mid \boldsymbol{\beta} \in A\}$$

By Lemma D.5, we know that $\{\lfloor \boldsymbol{\beta} \rfloor_n \mid \boldsymbol{\beta} \in A\}$ is a finite directed set, and therefore it must contain its own supremum. That is, there exists $\boldsymbol{\gamma} \in \{\lfloor \boldsymbol{\beta} \rfloor_n \mid \boldsymbol{\beta} \in A\}$ such that $\boldsymbol{\gamma} = \sup\{\lfloor \boldsymbol{\beta} \rfloor_n \mid \boldsymbol{\beta} \in A\}$. This means that there exists a $\boldsymbol{\gamma}' \in A$ such that $\boldsymbol{\gamma} = \lfloor \boldsymbol{\gamma}' \rfloor_n$, and so clearly $\boldsymbol{\gamma} \sqsubseteq_{pom} \boldsymbol{\gamma}'$, and therefore by transitivity, $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\gamma}'$. □

Corollary 4.16 (Approximation). *$\boldsymbol{\alpha} \ll \boldsymbol{\beta}$ iff $\boldsymbol{\alpha} \in \lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}}$.*

Proof. Suppose that $\boldsymbol{\alpha} \ll \boldsymbol{\beta}$. Since $\lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}}$ is a directed set and $\sup \lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}} = \boldsymbol{\beta}$ (Lemmas 4.13 and D.2), then there must be some $\boldsymbol{\gamma} \in \lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}}$ such that $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\gamma}$. Since $\lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}}$ is downward closed by definition, then $\boldsymbol{\alpha} \in \lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}}$ too.

Now suppose that $\boldsymbol{\alpha} \in \lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}}$. By Lemma 4.15, $\boldsymbol{\alpha}$ is compact, so $\boldsymbol{\alpha} \ll \boldsymbol{\alpha}$. By the definition of $\lfloor \boldsymbol{\beta} \rfloor_{\mathsf{fin}}$, $\boldsymbol{\alpha} \sqsubseteq_{lpof} \boldsymbol{\beta}$ too. Therefore, by [Abramsky and Jung 1995, Proposition 2.2.2(2)], we get that $\boldsymbol{\alpha} \ll \boldsymbol{\beta}$. □

For any dcpo s $\langle X, \leq_X \rangle$ and $\langle Y, \leq_Y \rangle$, the product poset $\langle X \times Y, \leq \rangle$ where $(x, y) \leq (x', y')$ iff $x \leq_X x'$ and $y \leq_Y y'$ is also a dcpo [Abramsky and Jung 1995, Proposition 3.2.2]. We will use this fact in the next lemma. We write $\vec{x} \in X_1 \times \cdots \times X_n$ to denote elements of Cartesian products. Let $\vec{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n)$ and $\vec{\boldsymbol{\beta}} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n)$. We write $\vec{\boldsymbol{\alpha}} \ll \vec{\boldsymbol{\beta}}$ to mean that $\boldsymbol{\alpha}_i \ll \boldsymbol{\beta}_i$ for all $1 \leq i \leq n$.

Lemma D.7. *For any directed set $D \subseteq pom(X)^n$:*

$$\left\{\vec{\boldsymbol{\beta}} \mid \exists \vec{\boldsymbol{\alpha}} \in D. \vec{\boldsymbol{\beta}} \ll \vec{\boldsymbol{\alpha}}\right\} = \left\{\vec{\boldsymbol{\beta}} \mid \vec{\boldsymbol{\beta}} \ll \sup D\right\}$$

Proof. We show the set inclusion in both directions. First, take any element $\vec{\boldsymbol{\beta}} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n)$ from the first set, so we know that $\boldsymbol{\beta}_i \ll \boldsymbol{\alpha}_i$ for all $1 \leq i \leq n$ and some $\vec{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n) \in D$. Clearly $\vec{\boldsymbol{\alpha}} \sqsubseteq_{pom} \sup D$, so $\vec{\boldsymbol{\beta}} \ll \sup D$ [Abramsky and Jung 1995, Proposition 2.2.2(2)] and therefore the forward inclusion holds.

Now take any element $\vec{\boldsymbol{\beta}} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n)$ from the second set and let $D_1, \ldots, D_n$ be the projections of D. Since the supremum on the product order is coordinatewise [Abramsky and Jung 1995, Proposition 2.2.2(2)], then $\sup D = (\sup D_1, \ldots, \sup D_n)$. So, $\boldsymbol{\beta}_i \ll \sup D_i$ for all $1 \leq i \leq n$. Therefore, by the definition of $\ll$, there must be some $\boldsymbol{\alpha}_i \in D_i$ such that $\boldsymbol{\beta}_i \sqsubseteq_{pom} \boldsymbol{\alpha}_i$. Now, for each i, let $\vec{\boldsymbol{\alpha}}_i \in D$ be the tuple whose i–th projection is $\boldsymbol{\alpha}_i$. Since D is directed, then there is some $\vec{\boldsymbol{\alpha}} \in D$, which is an upper bound of all the $\vec{\boldsymbol{\alpha}}_i$s. From $\boldsymbol{\beta}_i \ll \sup D_i$ and Corollary 4.16, we know that $\boldsymbol{\beta}_i$ is finite, therefore $\boldsymbol{\beta}_i \in \lfloor \boldsymbol{\alpha}_i \rfloor_{\mathsf{fin}}$. By Corollary 4.16 again, $\boldsymbol{\beta}_i \ll \boldsymbol{\alpha}_i$, and so $\vec{\boldsymbol{\beta}} \ll \boldsymbol{\alpha}$ and the reverse inclusion holds too. □

Lemma 4.17 (Extension). *Let $f\colon pom_{\mathsf{fin}}(L)^n \to T$ be a monotone function on the DCPO $\langle T, \leq\rangle$. Then $f^*\colon pom(L)^n \to T$, shown below, is well-defined and Scott continuous:*

$$f^*(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n) \triangleq \sup_{\boldsymbol{\alpha}'_1 \ll \boldsymbol{\alpha}_1} \cdots \sup_{\boldsymbol{\alpha}'_n \ll \boldsymbol{\alpha}_n} f(\boldsymbol{\alpha}'_1, \ldots, \boldsymbol{\alpha}'_n)$$

Proof. By Corollary 4.16, $\{(\boldsymbol{\alpha}'_1, \ldots, \boldsymbol{\alpha}'_n) \mid \forall i.\ \boldsymbol{\alpha}'_i \ll \boldsymbol{\alpha}_i\} = \lfloor \boldsymbol{\alpha}_1 \rfloor_{\mathsf{fin}} \times \cdots \times \lfloor \boldsymbol{\alpha}_n \rfloor_{\mathsf{fin}}$, that is a directed set by Lemma D.2. Since f is monotone, then $\{f(\boldsymbol{\alpha}'_1, \ldots, \boldsymbol{\alpha}'_n) \mid \forall i.\ \boldsymbol{\alpha}'_i \ll \boldsymbol{\alpha}_i\}$ is also a directed set, and so clearly the supremum exists.

We now establish that f^* is Scott continuous. Let $D \subseteq pom(L)^n$ be a directed set and:

$$\begin{aligned}
\sup_{\vec{\boldsymbol{\alpha}} \in D} f^*(\vec{\boldsymbol{\alpha}}) &= \sup_{\vec{\boldsymbol{\alpha}} \in D} \sup_{\vec{\boldsymbol{\beta}} \ll \vec{\boldsymbol{\alpha}}} f(\vec{\boldsymbol{\beta}}) \\
&= \sup \left\{ f(\vec{\boldsymbol{\beta}}) \mid \exists \vec{\boldsymbol{\alpha}} \in D.\ \vec{\boldsymbol{\beta}} \ll \vec{\boldsymbol{\alpha}} \right\} && \text{By [Abramsky and Jung 1995, Proposition 2.1.4(3)].} \\
&= \sup \left\{ f(\vec{\boldsymbol{\beta}}) \mid \vec{\boldsymbol{\beta}} \ll \sup D \right\} && \text{By Lemma D.7.} \\
&= f^*(\sup D)
\end{aligned}$$

□

E Operations

E.1 Guarded Branching

Lemma E.1 (Monotonicity of guard for lpof s). *For every $\ell \in L$, $x \in$ nodes and $\alpha, \alpha', \beta, \beta' \in lpof_{\mathsf{fin}}(L)$ such that $\alpha \sqsubseteq_{lpof} \alpha'$, $\beta \sqsubseteq_{lpof} \beta'$, $N_{\alpha'} \cap N_{\beta'} = \emptyset$ and $x \notin N_{\alpha'} \cup N_{\beta'}$, it holds that* $\mathrm{guard}(x, \ell, \alpha, \beta) \sqsubseteq_{lpof} \mathrm{guard}(x, \ell, \alpha', \beta')$.

Proof. To lighten notations, let us fix ℓ, $\alpha, \alpha', \beta, \beta'$ and x, and denote $\gamma \triangleq \mathrm{guard}(x, \ell, \alpha, \beta)$ and $\gamma' \triangleq \mathrm{guard}(x, \ell, \alpha', \beta')$.

First, being $N_\alpha \subseteq N_\beta$ and $N_{\alpha'} \subseteq N_{\beta'}$, we also have that $N_\gamma = \{x\} \cup N_\alpha \cup N_\beta \subseteq \{x\} \cup N_{\alpha'} \cup N_{\beta'} = N_{\gamma'}$; we have to prove that N_γ is downward closed. Let $y \in N_\gamma$ and $z <_{\gamma'} y$; clearly, $y \neq x$, because $\mathrm{pred}_{\gamma'}(x) = \emptyset$. Assume that $y \in N_\alpha$ ($\subseteq N_{\alpha'}$); the case for $y \in N_\beta$ is identical. If $z = x$, we trivially conclude. If $z \in N_{\alpha'}$, we have $z <_{\alpha'} y$, which leads to $z \in N_\alpha$ (because $N_\alpha \subseteq_\downarrow N_{\alpha'}$); hence, $z \in N_\gamma$.

Second,

$$\begin{aligned}
&<_{\gamma'} \cap (N_\gamma \times N_\gamma) \\
&\quad = \quad (<_{\alpha'} \cup <_{\beta'} \cup (\{x\} \times (N_{\alpha'} \cup N_{\beta'}))) \ \cap \ (N_\gamma \times N_\gamma) \\
&\quad = \quad (<_{\alpha'} \cap (((\{x\} \cup N_\alpha \cup N_\beta) \times (\{x\} \cup N_\alpha \cup N_\beta))) \\
&\qquad\quad \cup (<_{\beta'} \cap (((\{x\} \cup N_\alpha \cup N_\beta) \times (\{x\} \cup N_\alpha \cup N_\beta))) \\
&\qquad\quad \cup (((\{x\} \times (N_{\alpha'} \cup N_{\beta'})) \cap ((\{x\} \cup N_\alpha \cup N_\beta) \times (\{x\} \cup N_\alpha \cup N_\beta))) \\
&\quad = \quad (<_{\alpha'} \cap (N_\alpha \times N_\alpha)) \ \cup \ (<_{\beta'} \cap (N_\beta \times N_\beta)) \ \cup \ (\{x\} \times (N_\alpha \cup N_\beta)) \\
&\quad = \quad <_\alpha \cup <_\beta \cup (\{x\} \times (N_\alpha \cup N_\beta)) \ = \ <_\gamma
\end{aligned}$$

Fix $y \in N_\gamma$; we have three subcases to consider:

(1) $y = x$: by construction, $\lambda_\gamma(y) = \lambda_{\gamma'}(y) = \ell$ and $\varphi_\gamma(y) = \varphi_{\gamma'} = \mathrm{true}$.
(2) $y \in N_\alpha$: then, $\lambda_\gamma(y) = \lambda_\alpha(y) \leq \lambda_{\alpha'}(y) = \lambda_{\gamma'}(y)$ and $\varphi_\gamma(y) = \varphi_\alpha(y) \wedge x = \varphi_{\alpha'}(y) \wedge x = \lambda_{\gamma'}(y)$.
(3) $y \in N_\beta$: this case is similar to previous one, with $\neg x$ in place of x for the φ case.

To conclude, we need to prove that $\mathrm{succ}_\gamma(y) \supseteq \mathrm{succ}_{\gamma'}(y) \setminus \mathrm{Bot}_\gamma{\uparrow_{\gamma'}}$ (since the other inclusion holds for Lemma B.3). Thus, take $z \in \mathrm{succ}_{\gamma'}(y)$; we have three cases to consider:

(1) $y = x$: by construction, $z \in \{\min_{\alpha'}, \min_{\beta'}\}$ and so we immediately conclude $z \in \mathrm{succ}_\gamma(y)$, being $\min_\alpha = \min_{\alpha'}$ and $\min_\beta = \min_{\beta'}$.
(2) $y \in N_\alpha$: by construction, $z \in \mathrm{succ}_{\alpha'}(y)$. Since $\alpha \sqsubseteq_{lpof} \alpha'$, we know that $\mathrm{succ}_\alpha(y) = \mathrm{succ}_{\alpha'}(y) \setminus \mathrm{Bot}_\alpha{\uparrow_{\alpha'}}$. So, either $z \in \mathrm{succ}_\alpha(y) = \mathrm{succ}_\gamma(y)$ or $z \in \mathrm{Bot}_\alpha{\uparrow_{\alpha'}} \subseteq \mathrm{Bot}_\gamma{\uparrow_{\gamma'}}$.
(3) $y \in N_\beta$: this case is similar to previous one. □

Lemma 5.1 (Monotonicity of guard). *Let $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\alpha'}$ and $\boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\beta'}$; then, for every $\ell \in L$, it holds that* $\mathrm{guard}(\ell, \boldsymbol{\alpha}, \boldsymbol{\beta}) \sqsubseteq_{pom} \mathrm{guard}(\ell, \boldsymbol{\alpha'}, \boldsymbol{\beta'})$.

Proof. Take any $\gamma \in \mathrm{guard}(\ell, \boldsymbol{\alpha'}, \boldsymbol{\beta'})$; by construction, $\gamma = \mathrm{guard}(x, \ell, \alpha', \beta')$, for some $\alpha' \in \boldsymbol{\alpha'}$ and $\beta' \in \boldsymbol{\beta'}$ such that $N_{\alpha'} \cap N_{\beta'} = \emptyset$ and $x \notin N_{\alpha'} \cup N_{\beta'}$. By Lemma 4.5(3), there exist $\alpha \in \boldsymbol{\alpha}$ and $\beta \in \boldsymbol{\beta}$ such that $\alpha \sqsubseteq_{lpof} \alpha'$ and $\beta \sqsubseteq_{lpof} \beta'$. So, by Lemma E.1 $\mathrm{guard}(x, \ell, \alpha, \beta) \sqsubseteq_{lpof} \mathrm{guard}(x, \ell, \alpha', \beta')$. So, $\mathrm{guard}(\ell, \boldsymbol{\alpha}, \boldsymbol{\beta}) \sqsubseteq_{pom} \mathrm{guard}(\ell, \boldsymbol{\alpha'}, \boldsymbol{\beta'})$. □

E.2 Sequential Composition

Lemma E.2 (Antitonicity of stuck). *If $\alpha \sqsubseteq_{lpof} \beta$, then* $\mathsf{stuck}_\beta \Rightarrow \mathsf{stuck}_\alpha$.

Proof. For any $z \in \mathrm{Bot}_\beta$, either $z \in \mathrm{Bot}_\alpha$ or $z \notin N_\alpha$. In the first case, $\varphi_\beta(z) = \varphi_\alpha(z)$, being $\alpha \sqsubseteq_{lpof} \beta$; in the second case, by Lemma B.2, there is some $w \in \mathrm{Bot}_\alpha$ such that $w <_\beta z$, which implies that $\varphi_\beta(z) \Rightarrow \varphi_\alpha(w)$. This means that every term in $\bigvee_{z \in \mathrm{Bot}_\beta} \varphi_\beta(z)$ implies some term in $\bigvee_{y \in \mathrm{Bot}_\alpha} \varphi_\alpha(y)$, therefore $\bigvee_{z \in \mathrm{Bot}_\beta} \varphi_\beta(z) \Rightarrow \bigvee_{y \in \mathrm{Bot}_\alpha} \varphi_\alpha(y)$. □

Lemma E.3 (Monotonicity of ext). *If $\alpha \sqsubseteq_{lpof} \beta$, then* $\mathsf{ext}_\alpha = \{x \in \mathsf{ext}_\beta \mid \varphi_\beta(x) \nRightarrow \mathsf{stuck}_\alpha\}$.

Proof. We show the inclusion in both directions.

Take any $x \in \mathsf{ext}_\alpha$, this means that $x \in N_\alpha$ and $\varphi_\alpha(x) \nRightarrow \mathsf{stuck}_\alpha$; so, there is a valuation v such that $v \vDash \varphi_\alpha(x)$ and $v \nvDash \mathsf{stuck}_\alpha$. By Lemma E.2, $\mathsf{stuck}_\beta \Rightarrow \mathsf{stuck}_\alpha$, therefore v cannot satisfy stuck_β either. Since $\varphi_\alpha(x) = \varphi_\beta(x)$ (because $\alpha \sqsubseteq_{lpof} \beta$), we have that $\varphi_\beta(x) \nRightarrow \mathsf{stuck}_\alpha$ and that $v \vDash \varphi_\beta(x)$ and $v \nvDash \mathsf{stuck}_\beta$; therefore, $\varphi_\beta(x) \nRightarrow \mathsf{stuck}_\beta$ and so $x \in \mathsf{ext}_\beta$.

For the reverse inclusion, take any $x \in \mathsf{ext}_\beta$ such that $\varphi_\beta(x) \nRightarrow \mathsf{stuck}_\alpha$. By Lemma B.2, we know that either $x \in N_\alpha$ or $x \in \mathrm{Bot}_\alpha{\uparrow_\beta}$. In the former case, we know that $\varphi_\alpha(x) = \varphi_\beta(x) \nRightarrow \mathsf{stuck}_\alpha$, therefore $x \in \mathsf{ext}_\alpha$. The latter case is a contradiction, since $x \in \mathrm{Bot}_\alpha{\uparrow_\beta}$ means that there is some $y \in \mathrm{Bot}_\alpha$ such that $y <_\beta x$, which entails that $\varphi_\beta(x) \Rightarrow \varphi_\beta(y)$; however, $\varphi_\beta(y) = \varphi_\alpha(y) \Rightarrow \mathsf{stuck}_\alpha$ and, by combining these facts, we get that $\varphi_\beta(x) \Rightarrow \mathsf{stuck}_\alpha$. □

Lemma E.4 (Monotonicity of br). *If $\alpha \sqsubseteq_{lpof} \beta$, then* $\mathsf{br}_\alpha = \{\psi \in \mathsf{br}_\beta \mid \psi \Rightarrow \neg\mathsf{stuck}_\alpha\}$.

Proof. We show the inclusion in both directions.

For the forward inclusion, take $\psi \in \mathsf{br}_\alpha$, i.e. $\psi = \varphi_\alpha(S)$, for some $S \subseteq_{\mathsf{fin}} \mathsf{ext}_\alpha$ maximally satisfiable and that does not imply stuck_α; we want to show that $\psi = \varphi_\beta(S) \in \mathsf{br}_\beta$. By Lemma E.3, we know that $\mathsf{ext}_\alpha \subseteq \mathsf{ext}_\beta$, therefore $S \subseteq_{\mathsf{fin}} \mathsf{ext}_\beta$. Further, since $\varphi_\alpha(x) = \varphi_\beta(x)$ for all $x \in N_\alpha$, then $\varphi_\beta(S) = \varphi_\alpha(S) \Rightarrow \neg\mathsf{stuck}_\alpha$; by using the contrapositive of Lemma E.2, we get that $\varphi_\beta(S) \Rightarrow \neg\mathsf{stuck}_\beta$. Last, take any T such that $S \subset T \subseteq \mathsf{ext}_\beta$. If $T \subseteq \mathsf{ext}_\alpha$, we already know that $\varphi_\alpha(T)$ cannot be satisfiable; so suppose that there is some node $x \in T$ such that $x \notin \mathsf{ext}_\alpha$, meaning that either $x \notin N_\alpha$ or $\varphi_\alpha(x) \Rightarrow \mathsf{stuck}_\alpha$. In the first case, by Lemma B.2 $x \in \mathrm{Bot}_\alpha{\uparrow_\beta}$, so there is a $z \in \mathrm{Bot}_\alpha$ such that $z <_\beta x$, which also means that $\varphi_\beta(x) \Rightarrow \varphi_\beta(z) \Rightarrow \mathsf{stuck}_\alpha$. So, in either case, we have $\varphi_\beta(x) \Rightarrow \mathsf{stuck}_\alpha$, while, since $\varphi_\alpha(S) \in \mathsf{br}_\alpha$, we know that $\varphi_\alpha(S) \Rightarrow \neg\mathsf{stuck}_\alpha$. These give us:

$$\begin{aligned}\varphi_\beta(T) &\Rightarrow \varphi_\beta(S) \wedge \varphi_\beta(x)\\ &= \varphi_\alpha(S) \wedge \varphi_\beta(x)\\ &\Rightarrow \neg\mathsf{stuck}_\alpha \wedge \mathsf{stuck}_\alpha\end{aligned}$$

$$\Rightarrow \mathsf{false}$$

So, $\varphi_\beta(T)$ is not satisfiable, and therefore we have shown that $\varphi_\beta(S) \in \mathrm{br}_\beta$ (and, clearly, $\varphi_\beta(S) \Rightarrow \neg\mathsf{stuck}_\alpha$).

For the reverse inclusion, take any $\psi \in \mathrm{br}_\beta$ such that $\psi \Rightarrow \neg\mathsf{stuck}_\alpha$. We know that $\psi = \varphi_\beta(S)$ where $S \subseteq \mathrm{ext}_\beta$. By Lemma E.3, for each $x \in S$, either $x \in \mathrm{ext}_\alpha$ or $\varphi_\beta(x) \Rightarrow \mathsf{stuck}_\alpha$. But the latter case is not possible, otherwise we would be able to contradict $\varphi_\beta(S) = \psi \Rightarrow \neg\mathsf{stuck}_\alpha$. Therefore, $S \subseteq \mathrm{ext}_\alpha$. Now take any T such that $S \subset T \subseteq_{\mathsf{fin}} \mathrm{ext}_\alpha$. Since $\psi \in \mathrm{br}_\beta$, we know that $\varphi_\beta(T)$ is not satisfiable and, since $\varphi_\alpha(T) = \varphi_\beta(T)$, then $\varphi_\alpha(T)$ is not satisfiable either. Therefore, $\psi \in \mathrm{br}_\alpha$. □

Lemma E.5 (Monotonicity of $\fatsemi$ for lpof s). *For all $\alpha, \alpha', \beta, \beta' \in lpof_{\mathsf{fin}}(L)$ and $f \in \mathrm{copy}_{\alpha',\beta'}$ such that $\alpha \sqsubseteq_{lpof} \alpha'$ and $\beta \sqsubseteq_{lpof} \beta'$, there exists $g \in \mathrm{copy}_{\alpha,\beta}$ such that $\alpha \fatsemi_g \beta \sqsubseteq_{lpof} \alpha' \fatsemi_f \beta'$.*

Proof. Let $\beta'_\psi = f(\psi)$ for each $\psi \in \mathrm{br}_{\alpha'}$. Since $\beta' \equiv f(\psi)$ for all ψ, then for each $\psi \in \mathrm{br}_{\alpha'}$ there exists a bijection $f_\psi : N_{\beta'} \to N_{\beta'_\psi}$. Since $\beta \sqsubseteq_{lpof} \beta'$, then $N_\beta \subseteq N_{\beta'}$; so, we can restrict f_ψ to N_β and obtain an lpo-isomorphism g_ψ (that clearly satisfies $g_\psi(x) = f_\psi(x)$ for all $x \in N_\beta$). Now, let β_ψ be an isomorphic copy of β generated by the bijection g_ψ. Clearly, $N_{\beta_\psi} \cap N_\alpha = \emptyset$, since $N_\alpha \subseteq N_{\alpha'}$ and $N_{\beta_\psi} \subseteq N_{\beta'_\psi}$; similarly, all the β_ψs have pairwise disjoint nodes. By Lemma E.4, $\mathrm{br}_\alpha \subseteq \mathrm{br}_{\alpha'}$; so, we have such a β_ψ for all $\psi \in \mathrm{br}_\alpha$, and we can define $g \in \mathrm{copy}_{\alpha,\beta}$ as $g(\psi) = \beta_\psi$ for all $\psi \in \mathrm{br}_\alpha$. Also, $\beta_\psi = g_\psi(\beta) \sqsubseteq_{lpof} f_\psi(\beta') = \beta'_\psi$ for all $\psi \in \mathrm{br}_\alpha$, since $\beta \sqsubseteq_{lpof} \beta'$ and $g_\psi \subseteq f_\psi$.

We start with proving that $N_{\alpha\fatsemi_g\beta} \subseteq_\downarrow N_{\alpha'\fatsemi_f\beta'}$. Take any $x \in N_{\alpha\fatsemi_g\beta}$; so, either $x \in N_\alpha (\subseteq N_{\alpha'})$ or $x \in N_{\beta_\psi} (\subseteq N_{\beta'_\psi})$, for some $\psi \in \mathrm{br}_\alpha$. Clearly $x \in N_{\alpha'\fatsemi_f\beta'}$, so the subset inclusion holds; we now show that it is downward closed. Take any $y <_{\alpha'\fatsemi_f\beta'} x$. If $x \in N_\alpha$, then clearly $y <_{\alpha'} x$, since $\fatsemi$ does not add any causality into elements of $N_{\alpha'}$, and we therefore conclude that $y \in N_\alpha \subseteq N_{\alpha\fatsemi_g\beta}$, since $N_\alpha \subseteq_\downarrow N_{\alpha'}$. If instead $x \in N_{\beta_\psi}$ for some $\psi \in \mathrm{br}_\alpha$, then we know by Lemma E.4 that $\psi \in \mathrm{br}_{\alpha'}$ and therefore $x \in N_{\beta'_\psi} \subseteq N_{\alpha'\fatsemi_f\beta'}$. Since $y <_{\alpha'\fatsemi_f\beta'} x$, then either $y <_{\beta'_\psi} x$, or $\psi \Rightarrow \varphi_{\alpha'}(y)$. In the former case, then clearly $y \in N_{\beta_\psi} \subseteq N_{\alpha\fatsemi_g\beta}$ since $N_{\beta_\psi} \subseteq_\downarrow N_{\beta'_\psi}$. In the latter case, suppose for the sake of contradiction that $y \notin N_\alpha$; then, by Lemma B.2 there is some $z \in \mathrm{Bot}_\alpha$ such that $z <_{\alpha'} y$, which also means that $\varphi_{\alpha'}(y) \Rightarrow \varphi_{\alpha'}(z)$. Since $\alpha \sqsubseteq_{lpof} \alpha'$ and $z \in \mathrm{Bot}_\alpha$, we have that $\psi \Rightarrow \varphi_{\alpha'}(y) \Rightarrow \varphi_{\alpha'}(z) \Leftrightarrow \varphi_\alpha(z) \Rightarrow \mathsf{stuck}_\alpha$; this contradicts $\psi \Rightarrow \neg\mathsf{stuck}_\alpha$, that holds since $\psi \in \mathrm{br}_\alpha$.

Next, we show the condition on the order:

$$\begin{aligned}
&<_{\alpha'\fatsemi_f\beta'} \cap (N_{\alpha\fatsemi_g\beta} \times N_{\alpha\fatsemi_g\beta}) \\
&\quad= \left(<_{\alpha'} \cup \bigcup_{\psi\in\mathrm{br}_{\alpha'}} \left(<_{\beta'_\psi} \cup (\{x \mid \psi \Rightarrow \varphi_{\alpha'}(x)\} \times N_{\beta'_\psi})\right)\right) \cap (N_{\alpha\fatsemi_g\beta} \times N_{\alpha\fatsemi_g\beta}) \\
&\quad= \left(<_{\alpha'} \cap (N_{\alpha\fatsemi_g\beta} \times N_{\alpha\fatsemi_g\beta})\right) \\
&\quad\cup \bigcup_{\psi\in\mathrm{br}_{\alpha'}} \left(<_{\beta'_\psi} \cap (N_{\alpha\fatsemi_g\beta} \times N_{\alpha\fatsemi_g\beta})\right) \cup \left((\{x \mid \psi \Rightarrow \varphi_{\alpha'}(x)\} \times N_{\beta'_\psi}) \cap (N_{\alpha\fatsemi_g\beta} \times N_{\alpha\fatsemi_g\beta})\right) \qquad (\dagger)
\end{aligned}$$

Now, N_α is a subset of $N_{\alpha'}$ and it is disjoint from all the N_{β_ψ}; moreover, $\alpha \sqsubseteq_{lpof} \alpha'$. So,

$$<_{\alpha'} \cap (N_{\alpha\fatsemi_g\beta} \times N_{\alpha\fatsemi_g\beta}) = <_{\alpha'} \cap (N_\alpha \times N_\alpha) = <_\alpha$$

Then, by Lemma E.4 $\mathrm{br}_\alpha \subseteq \mathrm{br}_{\alpha'}$, and, for all $\psi \in \mathrm{br}_{\alpha'} \setminus \mathrm{br}_\alpha$, the nodes of β'_ψ do not appear in $N_{\alpha\fatsemi_g\beta}$; hence, we can reduce the bounds of the union in (†) and obtain:

$$\bigcup_{\psi\in\mathrm{br}_{\alpha'}} \left(<_{\beta'_\psi} \cap (N_{\alpha\fatsemi_g\beta} \times N_{\alpha\fatsemi_g\beta})\right) \cup \left((\{x \mid \psi \Rightarrow \varphi_{\alpha'}(x)\} \times N_{\beta'_\psi}) \cap (N_{\alpha\fatsemi_g\beta} \times N_{\alpha\fatsemi_g\beta})\right)$$

$$= \bigcup_{\psi \in \mathrm{br}_\alpha} \left(<_{\beta'_\psi} \cap (N_{\alpha \fatsemi_g \beta} \times N_{\alpha \fatsemi_g \beta}) \right) \cup \left((\{x \mid \psi \Rightarrow \varphi_{\alpha'}(x)\} \times N_{\beta'_\psi}) \cap (N_{\alpha \fatsemi_g \beta} \times N_{\alpha \fatsemi_g \beta}) \right)$$

$$= \bigcup_{\psi \in \mathrm{br}_\alpha} \left(<_{\beta'_\psi} \cap (N_{\beta_\psi} \times N_{\beta_\psi}) \right) \cup (\{x \mid \psi \Rightarrow \varphi_{\alpha'}(x)\} \times N_{\beta_\psi})$$

$$= \bigcup_{\psi \in \mathrm{br}_\alpha} \left(<_{\beta_\psi} \cup (\{x \mid \psi \Rightarrow \varphi_{\alpha}(x)\} \times N_{\beta_\psi}) \right)$$

where the last two equalities hold because $\beta_\psi \sqsubseteq_{lpof} \beta'_\psi$ and because the nodes of β'_ψ are disjoint from those of α and of any other isomorphic copy of β. Thus, (†) is equal to:

$$<_\alpha \cup \bigcup_{\psi \in \mathrm{br}_\alpha} \left(<_{\beta_\psi} \cup (\{x \mid \psi \Rightarrow \varphi_{\alpha}(x)\} \times N_{\beta_\psi}) \right)$$

that, by definition, is $<_{\alpha \fatsemi \beta}$.

Now, take any $x \in N_{\alpha \fatsemi_g \beta}$. Since $\alpha \sqsubseteq_{lpof} \alpha'$ and $\beta_\psi \sqsubseteq_{lpof} \beta'_\psi$, we have that $\lambda_\alpha(x) \leq \lambda_{\alpha'}(x)$, if $x \in N_\alpha$, and $\lambda_{\beta_\psi}(x) \leq \lambda_{\beta'_\psi}(x)$, if $x \in N_{\beta_\psi}$. This gives us:

$$\lambda_{\alpha \fatsemi_g \beta}(x) = \left\{ \begin{array}{ll} \lambda_\alpha(x) & \text{if } x \in N_\alpha \\ \lambda_{\beta_\psi}(x) & \text{if } x \in N_{\beta_\psi} \end{array} \right\} \leq \left\{ \begin{array}{ll} \lambda_{\alpha'}(x) & \text{if } x \in N_\alpha \\ \lambda_{\beta'_\psi}(x) & \text{if } x \in N_{\beta_\psi} \end{array} \right\} = \lambda_{\alpha' \fatsemi_f \beta'}(x)$$

Similarly, we have that $\varphi_\alpha(x) = \varphi_{\alpha'}(x)$, if $x \in N_\alpha$, and $\varphi_{\beta_\psi}(x) = \varphi_{\beta'_\psi}(x)$, if $x \in N_{\beta_{psi}}$; so

$$\varphi_{\alpha \fatsemi_g \beta}(x) = \left\{ \begin{array}{ll} \varphi_\alpha(x) & \text{if } x \in N_\alpha \\ \varphi_{\beta_\psi}(x) \wedge \psi & \text{if } x \in N_{\beta_\psi} \end{array} \right\} = \left\{ \begin{array}{ll} \varphi_{\alpha'}(x) & \text{if } x \in N_\alpha \\ \varphi_{\beta'_\psi}(x) \wedge \psi & \text{if } x \in N_{\beta_\psi} \end{array} \right\} = \varphi_{\alpha' \fatsemi_f \beta'}(x)$$

For successors, we know by Lemma B.3(3) that $\mathrm{succ}_{\alpha \fatsemi_g \beta}(x) \subseteq \mathrm{succ}_{\alpha' \fatsemi_f \beta'}(x) \setminus \mathrm{Bot}_{\alpha \fatsemi_g \beta} \uparrow_{\alpha' \fatsemi_f \beta'}$, we now must show the reverse inclusion. Take any $y \in \mathrm{succ}_{\alpha' \fatsemi_f \beta'}(x)$ such that $y \notin \mathrm{Bot}_{\alpha \fatsemi_g \beta} \uparrow_{\alpha' \fatsemi_f \beta'}$. Given the way that $<_{\alpha' \fatsemi_f \beta'}$ is constructed, there are three cases to consider:

(1) $y \in \mathrm{succ}_{\alpha'}(x)$. Since $\alpha \sqsubseteq_{lpof} \alpha'$, we know that $\mathrm{succ}_\alpha(x) = \mathrm{succ}_{\alpha'}(x) \setminus \mathrm{Bot}_\alpha \uparrow_{\alpha'}$. However, $y \notin \mathrm{Bot}_\alpha \uparrow_{\alpha'}$ (since $\mathrm{Bot}_\alpha \uparrow_{\alpha'} \subseteq \mathrm{Bot}_{\alpha \fatsemi_g \beta} \uparrow_{\alpha' \fatsemi_f \beta'}$); therefore, it must be that $y \in \mathrm{succ}_\alpha(x)$, and so $y \in \mathrm{succ}_{\alpha \fatsemi_g \beta}(x)$.

(2) $y \in \mathrm{succ}_{\beta'_\psi}(x)$ for some $\psi \in \mathrm{br}_\alpha$. This means that $x, y \in N_{\beta'_\psi}$. Let $x' = f_\psi^{-1}(x)$ and $y' = f_\psi^{-1}(y)$, so $x', y' \in N_{\beta'}$. Since $\beta' \equiv \beta'_\psi$, we know that $y' \in \mathrm{succ}_{\beta'}(x')$. Since $\beta \sqsubseteq_{lpof} \beta'$, then $\mathrm{succ}_\beta(x') = \mathrm{succ}_{\beta'}(x') \setminus \mathrm{Bot}_\beta \uparrow_{\beta'}$. However, since $y \notin \mathrm{Bot}_{\alpha \fatsemi \beta} \uparrow_{\alpha' \fatsemi \beta'}$, then it must be that $y' \notin \mathrm{Bot}_\beta \uparrow_{\beta'}$; so we must have that $y' \in \mathrm{succ}_\beta(x')$. Since $x', y' \in N_\beta$, then $g_\psi(x') = f_\psi(x') = x$ and $g_\psi(y') = f_\psi(y') = y$; so, $y \in \mathrm{succ}_{\beta_\psi}(x)$ and therefore $y \in \mathrm{succ}_{\alpha \fatsemi_g \beta}(x)$.

(3) $x \in N_\alpha$ and $y \in N_{\beta'_\psi}$ for some $\psi \in \mathrm{br}_{\alpha'}$. By Lemma E.4, either $\psi \in \mathrm{br}_\alpha$ or $\psi \nRightarrow \neg \mathrm{stuck}_\alpha$.
For the first case, $x <_{\alpha' \fatsemi_f \beta'} y$, $x \in N_{\alpha'}$ and $y \in N_{\beta'_\psi}$ imply that $\psi \Rightarrow \varphi_{\alpha'}(x) = \varphi_\alpha(x)$. Furthermore, y must be the root of β'_ψ, otherwise it could not be a successor of $x \in N_{\alpha'}$. Since β'_ψ is single-rooted and $\beta_\psi \sqsubseteq_{lpof} \beta'_\psi$, then y must also be the root of β_ψ; so, $x <_{\alpha \fatsemi_g \beta} y$ and, by Lemma B.3(1), $y \in \mathrm{succ}_{\alpha \fatsemi_g \beta}(x)$.
To conclude, we show that the second case is impossible. Suppose that $\psi \nRightarrow \neg \mathrm{stuck}_\alpha$. From $\psi \in \mathrm{br}_{\alpha'}$ we also know that $\psi \Rightarrow \neg \mathrm{stuck}_{\alpha'}$. Therefore, there is some v such that $v \vDash \psi$, $v \vDash \mathrm{stuck}_\alpha$ and $v \nvDash \mathrm{stuck}_{\alpha'}$. From $v \vDash \mathrm{stuck}_\alpha$, we get that $v \vDash \varphi_\alpha(z)$, for some $z \in \mathrm{Bot}_\alpha$; so $v \vDash \varphi_{\alpha'}(z)$ too (since $\varphi_\alpha(z) = \varphi_{\alpha'}(z)$). Since $v \vDash \varphi_{\alpha'}(z)$ and $v \nvDash \mathrm{stuck}_{\alpha'}$, then $\varphi_{\alpha'}(z) \nRightarrow \mathrm{stuck}_{\alpha'}$; therefore $z \in \mathrm{ext}_{\alpha'}$. Moreover, since $v \vDash \psi$ and $v \vDash \varphi_{\alpha'}(z)$, we know that

$\psi \wedge \varphi_{\alpha'}(z)$ is satisfiable; since S is maximal, then $z \in S$. Therefore, $\psi \Rightarrow \varphi_{\alpha'}(z)$, and so $z <_{\alpha' \fatsemi_f \beta'} y$ (by the definition of $<_{\alpha' \fatsemi_f \beta'}$), in contradiction with $y \notin \mathrm{Bot}_{\alpha \fatsemi_g \beta}\uparrow_{\alpha' \fatsemi_f \beta'}$. □

Lemma 5.3 (Monotonicity of $\fatsemi$). *If $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\alpha'}$ and $\boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\beta'}$, then $\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\alpha'} \fatsemi \boldsymbol{\beta'}$*

Proof. Take any $\gamma \in \boldsymbol{\alpha'} \fatsemi \boldsymbol{\beta'}$; by construction, $\gamma = \alpha' \fatsemi_f \beta'$, for some $\alpha' \in \boldsymbol{\alpha'}$, $\beta' \in \boldsymbol{\beta'}$, and $f \in \mathrm{copy}_{\alpha',\beta'}$. By Lemma 4.5(1), there exist $\alpha \in \boldsymbol{\alpha}$ and $\beta \in \boldsymbol{\beta}$ such that $\alpha \sqsubseteq_{lpof} \alpha'$ and $\beta \sqsubseteq_{lpof} \beta'$. So, by Lemma E.5 there exists a $g \in \mathrm{copy}_{\alpha,\beta}$ such that $\alpha \fatsemi_g \beta \sqsubseteq_{lpof} \alpha' \fatsemi_f \beta'$. Therefore, $\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \sqsubseteq_{pom} \boldsymbol{\alpha'} \fatsemi \boldsymbol{\beta'}$. □

F Denotational Semantics

F.1 Linearization

Lemma F.1. *If $\alpha \sqsubseteq_{lpof} \beta$ and $S \cap \mathrm{Bot}_\alpha = \emptyset$, then $\mathsf{next}(\alpha, \psi, S) = \mathsf{next}(\beta, \psi, S)$.*

Proof. We show the set inclusion in both directions. Take any $y \in \mathsf{next}(\alpha, \psi, S)$, so $y \in N_\alpha$, $y \notin S$, $y{\downarrow_\alpha} \subseteq S$, and $\psi \Rightarrow \varphi_\alpha(y)$. Since $\alpha \sqsubseteq_{lpof} \beta$, we get that $y \in N_\beta$, $y{\downarrow_\alpha} = y{\downarrow_\beta}$, and $\varphi_\alpha(y) = \varphi_\beta(y)$, so $y \in \mathsf{next}(\beta, \psi, S)$.

For the reverse inclusion, take any $y \in \mathsf{next}(\beta, \psi, S)$, so $y \in N_\beta$, $y \notin S$, $y{\downarrow_\beta} \subseteq S$, and $\psi \Rightarrow \varphi_\beta(y)$. If $y \in N_\alpha$, then the proof follows from a similar argument as above. If not, we will show that there is a contradiction. Suppose that $y \notin N_\alpha$, so by Lemma B.2, we get that $y \in \mathrm{Bot}_\alpha{\uparrow_\beta}$. So there is a $z \in \mathrm{Bot}_\alpha$ such that $z <_\beta y$. Since $y{\downarrow_\beta} \subseteq S$, then $z \in S$. However, this is a contradiction since we assumed that $S \cap \mathrm{Bot}_\alpha = \emptyset$. □

For any set X, the pointwise extension $\langle X \to D(\mathcal{S}), \sqsubseteq^\bullet \rangle$ is also a pointed DCPO, where $f \sqsubseteq^\bullet g$ iff $f(x) \sqsubseteq g(x)$ for all $x \in X$. We use this fact below. In addition, let next^* denote the set of nodes that could still be scheduled:

$$\mathsf{next}^*(\alpha, \psi, S) \triangleq \{x \in N_\alpha \setminus S \mid \mathsf{sat}(\psi \wedge \varphi_\alpha(x))\}$$

Lemma F.2 (Monotonicity of $\mathcal{L}_{lpof}$). *For any $\alpha, \beta \in lpof_{\mathsf{fin}}(L)$, $\psi \in form$, $S \subseteq N_\alpha$, and $s \in \mathcal{S}$ such that $\alpha \sqsubseteq_{lpof} \beta$ and $S \cap \mathrm{Bot}_\alpha = \emptyset$:*

$$\mathcal{L}_{lpof}(\alpha, \psi, S) \sqsubseteq^\bullet \mathcal{L}_{lpof}(\beta, \psi, S)$$

Proof. The proof is by induction on the size of the set $\mathsf{next}^*(\alpha, \psi, S)$.

If $\mathsf{next}^*(\alpha, \psi, S)$ is empty, then $x \in S$ or $\psi \wedge \varphi_\alpha(x)$ is unsatisfiable for all $x \in N_\alpha$. In either case, clearly $x \notin \mathsf{next}(\alpha, \psi, S)$ for all $x \in N_\alpha$, therefore $\mathsf{next}(\alpha, \psi, S) = \emptyset$. Moreover, by Lemma F.1, $\mathsf{next}(\beta, \psi, S) = \emptyset$ too and so $\mathcal{L}_{lpof}(\alpha, \psi, S)(s) = \mathcal{L}_{lpof}(\beta, \psi, S)(s) = \eta(s)$.

Now suppose that $\mathsf{next}^*(\alpha, \psi, S)$ is not empty, and that the claim holds for all $n < |\mathsf{next}^*(\alpha, \psi, S)|$. We start by showing that $\mathcal{L}_{node}(\alpha, \psi, S, x)(s) \sqsubseteq \mathcal{L}_{node}(\beta, \psi, S, x)(s)$. First, we remark that $S \cap \mathrm{Bot}_\alpha = \emptyset$ and so, if $x \notin \mathrm{Bot}_\alpha$, then $(S \cup \{x\}) \cap \mathrm{Bot}_\alpha = \emptyset$, which will allow us to use the induction hypothesis in cases (1), (2), and (4) below. We proceed by case analysis on $\lambda_\alpha(x)$.

(1) If $\lambda_\alpha(x) \in act$, then $\lambda_\beta(x) \in act$ too, and $[\![\lambda_\alpha(x)]\!]_{act}(s) \sqsubseteq [\![\lambda_\beta(x)]\!]_{act}(s)$. Note that $\mathsf{next}^*(\alpha, \psi, S \cup \{x\}) \subset \mathsf{next}^*(\alpha, \psi, S)$, so by the induction hypothesis, we can conclude that:

$$\mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\}) \sqsubseteq^\bullet \mathcal{L}_{lpof}(\beta, \psi, S \cup \{x\})$$

Therefore, by monotonicity of Kleisli extension, we get that:

$$\begin{aligned}\mathcal{L}_{node}(\alpha, \psi, S, x)(s) &= \mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}(s)) \\ &\sqsubseteq \mathcal{L}_{lpof}(\beta, \psi, S \cup \{x\})^\dagger([\![\lambda_\beta(x)]\!]_{act}(s)) \\ &= \mathcal{L}_{node}(\beta, \psi, S, x)(s)\end{aligned}$$

(2) If $\lambda_\alpha(x) \in \mathit{test}$, then again $\lambda_\beta(x) \in \mathit{test}$ as well and $[\![\lambda_\alpha(x)]\!]_{\mathit{test}}(s) \sqsubseteq [\![\lambda_\beta(x)]\!]_{\mathit{test}}(s)$. Since $\mathsf{next}^*(\alpha, \psi \wedge x, S \cup \{x\}) \subset \mathsf{next}^*(\alpha, \psi, S)$ and $\mathsf{next}^*(\alpha, \psi \wedge \neg x, S \cup \{x\}) \subset \mathsf{next}^*(\alpha, \psi, S)$, then we can use the induction hypothesis to conclude that:

$$\begin{aligned}\mathcal{L}_{\mathit{test}}(\alpha, \psi, S, x)(s)(b) &= \begin{cases} \mathcal{L}_{\mathit{lpof}}(\alpha, \psi \wedge \ \ x, S \cup \{x\})(s) & \text{if } b = 1 \\ \mathcal{L}_{\mathit{lpof}}(\alpha, \psi \wedge \neg x, S \cup \{x\})(s) & \text{if } b = 0 \end{cases} \\ &\sqsubseteq \begin{cases} \mathcal{L}_{\mathit{lpof}}(\beta, \psi \wedge \ \ x, S \cup \{x\})(s) & \text{if } b = 1 \\ \mathcal{L}_{\mathit{lpof}}(\beta, \psi \wedge \neg x, S \cup \{x\})(s) & \text{if } b = 0 \end{cases} \\ &= \mathcal{L}_{\mathit{test}}(\beta, \psi, S, x)(s)(b)\end{aligned}$$

Therefore, by monotonicity of Kleisli extension, we get that:

$$\begin{aligned}\mathcal{L}_{\mathit{node}}(\alpha, \psi, S, x)(s) &= \mathcal{L}_{\mathit{test}}(\alpha, \psi, S, x)(s)^\dagger \left([\![\lambda_\alpha(x)]\!]_{\mathit{test}}(s)\right) \\ &\sqsubseteq \mathcal{L}_{\mathit{test}}(\beta, \psi, S, x)(s)^\dagger \left([\![\lambda_\beta(x)]\!]_{\mathit{test}}(s)\right) \\ &= \mathcal{L}_{\mathit{node}}(\beta, \psi, S, x)(s)\end{aligned}$$

(3) If $\lambda_\alpha(x) = \bot$, then we have:

$$\mathcal{L}_{\mathit{node}}(\alpha, \psi, S, x)(s) \quad = \quad \bot \quad \sqsubseteq \quad \mathcal{L}_{\mathit{node}}(\beta, \psi, S, x)(s)$$

(4) If $\lambda_\alpha(x) = \mathsf{fork}$, then $\lambda_\beta(x) = \mathsf{fork}$, and so since $\mathsf{next}^*(\alpha, \psi, S \cup \{x\}) \subset \mathsf{next}^*(\alpha, \psi, S)$, the induction hypothesis gives us the following:

$$\begin{aligned}\mathcal{L}_{\mathit{node}}(\alpha, \psi, S, x)(x) &= \mathcal{L}_{\mathit{lpof}}(\alpha, \psi, S \cup \{x\})(x) \\ &\sqsubseteq \mathcal{L}_{\mathit{lpof}}(\beta, \psi, S \cup \{x\})(x) \\ &= \mathcal{L}_{\mathit{node}}(\beta, \psi, S, x)(x)\end{aligned}$$

We now complete the proof of the main claim as follows:

$$\mathcal{L}_{\mathit{lpof}}(\alpha, \psi, S)(s) = \bigotimes_{x \in \mathsf{next}(\alpha, \psi, S)} \mathcal{L}_{\mathit{node}}(\alpha, \psi, S, x)(s)$$

Since $S \cap \mathsf{Bot}_\alpha = \emptyset$, by Lemma F.1 we know that $\mathsf{next}(\alpha, \psi, S) = \mathsf{next}(\beta, \psi, S)$.

$$= \bigotimes_{x \in \mathsf{next}(\beta, \psi, S)} \mathcal{L}_{\mathit{node}}(\alpha, \psi, S, x)(s)$$

Since $\&$ is monotone, and given the claim we just proved about $\mathcal{L}_{\mathit{node}}$:

$$\begin{aligned}&\sqsubseteq \bigotimes_{x \in \mathsf{next}(\beta, \psi, S)} \mathcal{L}_{\mathit{node}}(\beta, \psi, S, x)(s) \\ &= \mathcal{L}_{\mathit{lpof}}(\beta, \psi, S)(s)\end{aligned}$$

□

Lemma 6.2 (Monotonicity of $\mathcal{L}_{\mathsf{fin}}$). *If $\boldsymbol{\alpha} \sqsubseteq_{\mathit{pom}} \boldsymbol{\beta}$, then $\mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha})(s) \sqsubseteq \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\beta})(s)$.*

Proof. Fix any $\alpha \in \boldsymbol{\alpha}$, then we know there exists a $\beta \in \boldsymbol{\beta}$ such that $\alpha \sqsubseteq_{\mathit{lpof}} \beta$:

$$\begin{aligned}\mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha}) &= \mathcal{L}_{\mathsf{fin}}([\alpha]) \\ &= \mathcal{L}_{\mathit{lpof}}(\alpha, \mathsf{true}, \emptyset) \\ &\sqsubseteq^\bullet \mathcal{L}_{\mathit{lpof}}(\beta, \mathsf{true}, \emptyset) && \text{By Lemma F.2} \\ &= \mathcal{L}_{\mathsf{fin}}([\beta]) \\ &= \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\beta})\end{aligned}$$

□

F.2 Properties of Linearization: Skip and Actions

Lemma F.3. *$\mathcal{L}([\![\mathbf{skip}]\!]) = \eta$ and $\mathcal{L}([\![a]\!]) = [\![a]\!]_{act}$.*

Proof. We show only the case for **skip**, as the case for actions is nearly identical.

$$\mathcal{L}([\![\mathbf{skip}]\!])(s) = \mathcal{L}(\langle \mathsf{fork} \rangle)(s)$$

Since a singleton pomset is already finite, $\mathcal{L} = \mathcal{L}_{\mathsf{fin}}$. Fixing an arbitrary $x \in \mathit{nodes}$, we have:

$$\begin{aligned} &= \mathcal{L}_{\mathsf{fin}}([\langle \mathsf{fork} \rangle_x])(s) \\ &= \mathcal{L}_{lpof}(\langle \mathsf{fork} \rangle_x, \mathsf{true}, \emptyset)(s) \end{aligned}$$

Obviously, $\mathsf{next}(\langle \mathsf{fork} \rangle_x, \mathsf{true}, \emptyset) = \{x\}$, so we get:

$$\begin{aligned} &= \mathcal{L}_{lpof}(\langle \mathsf{fork} \rangle_x, \mathsf{true}, \{x\})(s) \\ &= \eta(s) \end{aligned}$$

□

F.3 Properties of Linearization: Sequential Composition

We first introduce a few definitions:

$$\begin{aligned} \mathsf{tests}_\alpha(S) &= \{x \in S \mid \lambda_\alpha(x) \in \mathit{test}\} \\ \mathsf{forms}(X) &= \left\{ \bigwedge_{x \in X} (\!| x = b_x |\!) \;\middle|\; \forall x \in X.\, b_x \in \mathbb{B} \right\} \\ \mathsf{valid}_\alpha(S) &= \{\psi \mid \psi \in \mathsf{forms}(\mathsf{tests}_\alpha(S)), \forall x \in S.\, \psi \Rightarrow \varphi_\alpha(x)\} \end{aligned}$$

The set $\mathsf{next}^*(\alpha, \psi, S)$ contains all of the nodes in α that are still able to be scheduled, given that S has already been processed and ψ is the path condition. The set $\mathsf{tests}_\alpha(S)$ contains all the test nodes from S. The set $\mathsf{forms}(X)$ contains all the formulae over the variables X, which are conjunctions of literals, where a literal is either a variable x or its negation $\neg x$. Finally, $\mathsf{valid}_\alpha(S)$ contains all the formulae over $\mathsf{tests}_\alpha(S)$ that imply the formulae of all $x \in S$. As an abuse of notation, we will write $\psi \in S$ to mean that there exists a $\psi' \in S$ such that $\psi \Leftrightarrow \psi'$.

Lemma F.4. *If α is binary branching, $S \subseteq \mathsf{nBot}_\alpha$, $\mathsf{next}^*(\alpha, \psi, S) = \emptyset$ and $\psi \in \mathsf{valid}_\alpha(S)$, then $\psi \in \mathsf{br}_\alpha$ and $\mathsf{next}(\alpha \fatsemi_f \beta) = \min_{f(\psi)}$.*

Proof. Since $\mathsf{next}^*(\alpha, \psi, S) = \emptyset$, we know that for all $x \in N_\alpha$, either $x \in S$ or $\psi \wedge \varphi_\alpha(x)$ is unsatisfiable. In particular, since $S \subseteq \mathsf{nBot}_\alpha$, then $\psi \wedge \varphi_\alpha(x)$ is unsatisfiable for all $x \in \mathsf{Bot}_\alpha$. We will now establish that $\psi \in \mathsf{br}_\alpha$ by proving the following claims:

(1) $\psi \Rightarrow \neg\mathsf{stuck}_\alpha \Leftrightarrow \neg \bigvee_{x \in \mathsf{Bot}_\alpha} \varphi_\alpha(x) \Leftrightarrow \bigwedge_{x \in \mathsf{Bot}_\alpha} \neg\varphi_\alpha(x)$. It will suffice to show that $\psi \Rightarrow \neg\varphi_\alpha(x)$ for all $x \in \mathsf{Bot}_\alpha$. Take any $v \vDash \psi$. We know that $\psi \wedge \varphi_\alpha(x)$ is unsatisfiable since $x \in \mathsf{Bot}_\alpha$, therefore $v \vDash \neg\varphi_\alpha(x)$, so we are done.

(2) $\psi \Leftrightarrow \bigwedge_{x \in S} \varphi_\alpha(x)$. From $\psi \in \mathsf{valid}_\alpha(S)$, we immediately get that $\psi \Rightarrow \bigwedge_{x \in S} \varphi_\alpha(x)$. We now also show the reverse. Since $\psi \in \mathsf{valid}_\alpha(S)$, then $\psi \in \mathsf{forms}(\mathsf{tests}_\alpha(S))$. We now show that for every $y \in \mathsf{tests}_\alpha(S)$, either $\bigwedge_{x \in S} \varphi_\alpha(x) \Rightarrow y$ or $\bigwedge_{x \in S} \varphi_\alpha(x) \Rightarrow \neg y$. Take any $y \in \mathsf{tests}_\alpha(S)$. By the binary branching property, $\mathsf{succ}_\alpha(y) = \{z_1, z_2\}$ such that $\varphi_\alpha(z_1) \Leftrightarrow \varphi_\alpha(y) \wedge y$ and $\varphi_\alpha(z_2) \Leftrightarrow \varphi_\alpha(y) \wedge \neg y$. We also know that since $z_1, z_2 \in N_\alpha$, then for each $i \in \{1, 2\}$ either $z_i \in S$ or $\psi \wedge \varphi_\alpha(z_i)$ is unsatisfiable. We already know that $\psi \Rightarrow \varphi_\alpha(y)$ since $y \in S$. We also know that $\psi \Rightarrow y$ or $\psi \Rightarrow \neg y$ since $y \in \mathsf{tests}_\alpha(S)$ and $\psi \in \mathsf{valid}_\alpha(S)$. Without loss of generality, suppose that $\psi \Rightarrow y$, then clearly $\psi \wedge \varphi_\alpha(z_1) \Leftrightarrow \psi \wedge \varphi_\alpha(y) \wedge y$ is satisfiable, therefore $z_1 \in S$. This clearly means that $\bigwedge_{x \in S} \varphi_\alpha(x) \Rightarrow \varphi_\alpha(z_1) \Rightarrow y$. If instead we had that $\psi \Rightarrow \neg y$, then we would have $\bigwedge_{x \in S} \varphi_\alpha(x) \Rightarrow \neg y$.

Therefore we have shown that $\bigwedge_{x \in S} \varphi_\alpha(x)$ assigns a truth value to each $y \in \mathsf{vars}(\psi)$, and so $\bigwedge_{x \in S} \varphi_\alpha(x) \Rightarrow \psi$.

(3) We now show that $S \subseteq \mathsf{ext}_\alpha$, that is $\varphi_\alpha(x) \not\Rightarrow \mathsf{stuck}_\alpha$ for all $x \in S$. For the sake of contradiction, suppose that $\varphi_\alpha(x) \Rightarrow \mathsf{stuck}_\alpha$ for some $x \in S$. Since $\psi \in \mathsf{valid}_\alpha(S)$, we also have that $\psi \Rightarrow \varphi_\alpha(x)$, which gives us:

$$\psi \Rightarrow \varphi_\alpha(x) \Rightarrow \mathsf{stuck}_\alpha \Leftrightarrow \bigvee_{y \in \mathsf{Bot}_\alpha} \varphi_\alpha(y)$$

But this contradicts the fact that $\psi \wedge \varphi_\alpha(y)$ is unsatisfiable for all $y \in \mathsf{Bot}_\alpha$, therefore it cannot be true that $\varphi_\alpha(x) \Rightarrow \mathsf{stuck}_\alpha$, and instead we have $\varphi_\alpha(x) \not\Rightarrow \mathsf{stuck}_\alpha$.

(4) $\psi \wedge \varphi_\alpha(x)$ is unsatisfiable for any $x \in \mathsf{ext}_\alpha \setminus S$. This is immediate, since $\psi \wedge \varphi_\alpha(x)$ is unsatisfiable for any $x \notin S$.

So, we have now shown that $\psi \in \mathsf{br}_\alpha$. Next, we show that $\mathsf{next}(\alpha \fatsemi_f \beta, \psi, S) = \min_{f(\psi)}$ by showing the set inclusion in both directions.

Take any $x \in \mathsf{next}(\alpha \fatsemi_f \beta, \psi, S)$, so $x \in N_{\alpha \fatsemi_f \beta}$, $x \notin S$, $x{\downarrow_{\alpha \fatsemi_f \beta}} \subseteq S$, and $\psi \Rightarrow \varphi_{\alpha \fatsemi_f \beta}(x)$. Clearly $x \notin N_\alpha$, otherwise one of $x \notin S$ or $\psi \Rightarrow \varphi_{\alpha \fatsemi_f \beta}(x) = \varphi_\alpha(x)$ would be violated. Now, take any $y <_{\alpha \fatsemi_f \beta} x$, we know from $x{\downarrow_{\alpha \fatsemi_f \beta}} \subseteq S$ that $y \in S \subseteq N_\alpha$. So, since $\psi \Rightarrow \varphi_{\alpha \fatsemi_f \beta}(x)$, then $x \in N_{f(\psi)}$. Furthermore, $\{x\} = \min_{f(\psi)}$ since we already established that all predecessors of x are in α, and so x has no predecessors in $\beta_{f(\psi)}$.

We now show the reverse direction. Take any $\{x\} = \min_{f(\psi)}$. Clearly $x \in N_{\alpha \fatsemi_f \beta}$ and $x \notin S$. By construction, $\varphi_{\alpha \fatsemi_f \beta}(x) = \varphi_{f(\psi)}(x) \wedge \psi \Leftrightarrow \mathsf{true} \wedge \psi$ ($\varphi_{f(\psi)}(x) \Leftrightarrow \mathsf{true}$ since x is minimal and therefore no branches have occurred), so clearly $\psi \Rightarrow \varphi_{\alpha \fatsemi_f \beta}(x)$. Now take any $y <_{\alpha \fatsemi_f \beta} x$, since $\{x\} = \min_{f(\psi)}$, then $y \in N_\alpha$, so $\psi \Rightarrow \varphi_\alpha(y)$. We already know that for all $y \in N_\alpha$ either $y \in S$, or $\psi \wedge \varphi_\alpha(y)$ is unsatisfiable, but clearly the latter case is impossible, so $y \in S$. Therefore $x{\downarrow_{\alpha \fatsemi_f \beta}} \subseteq S$, and we have shown all the conditions to guarantee that $x \in \mathsf{next}(\alpha \fatsemi_f \beta, \psi, S)$. □

Lemma F.5. *If α is binary branching, $S \subseteq \mathsf{nBot}_\alpha$, $\mathsf{next}^*(\alpha, \psi, S) \neq \emptyset$, and $\psi \in \mathsf{valid}_\alpha(S)$, then* $\mathsf{next}(\alpha \fatsemi_f \beta, \psi, S) = \mathsf{next}(\alpha, \psi, S) \neq \emptyset$

Proof. Since $\mathsf{next}^*(\alpha, \psi, S) \neq \emptyset$, then there exists $x \in \mathsf{next}^*(\alpha, \psi, S)$, meaning that $x \in N_\alpha \setminus S$ and $\psi \wedge \varphi_\alpha(x)$ is satisfiable. Take a minimal such x, so that there is no $y \in \mathsf{next}^*(\alpha, \psi, S)$ such that $y <_\alpha x$.

We now show that $x{\downarrow_\alpha} \subseteq S$. Take any $y <_\alpha x$. We know that $y \notin \mathsf{next}^*(\alpha, \psi, S)$, so either $y \in S$ or $\psi \wedge \varphi_\alpha(y)$ is unsatisfiable. However, since $y <_\alpha x$, then $\varphi_\alpha(x) \Rightarrow \varphi_\alpha(y)$, so $\psi \wedge \varphi_\alpha(y)$ must be satisfiable, meaning that the only option is that $y \in S$.

Now, since α is binary branching, there are two options. The first option is that $x \in \mathsf{succ}_\alpha(y)$ where y is a test node and $\varphi_\alpha(x) \Leftrightarrow \varphi_\alpha(y) \wedge y$ or $\varphi_\alpha(x) \Leftrightarrow \varphi_\alpha(y) \wedge \neg y$. Since $\psi \in \mathsf{valid}_\alpha(S)$ and $\psi \wedge \varphi_\alpha(x)$ is satisfiable, then it must be that $\psi \Rightarrow \varphi_\alpha(x)$. If instead x is not the successor of any test nodes, then $\varphi_\alpha(x) \Leftrightarrow \bigwedge_{y \in \mathsf{pred}_\alpha(x)} \varphi_\alpha(y)$, so again since $\psi \wedge \varphi_\alpha(x)$ is satisfiable, then it must be that $\psi \Rightarrow \varphi_\alpha(x)$. In both cases, we have that $x \in \mathsf{next}(\alpha, \psi, S)$.

Finally, we show that $\mathsf{next}(\alpha, \psi, S) = \mathsf{next}(\alpha \fatsemi_f \beta, \psi, S)$ by showing the set inclusion in both directions. Take any $y \in \mathsf{next}(\alpha, \psi, S)$, so $y \in N_\alpha \setminus S$ such that $y{\downarrow_\alpha} \subseteq S$ and $\psi \Rightarrow \varphi_\alpha(y)$. Since $N_\alpha \subseteq_\downarrow N_{\alpha \fatsemi_f \beta}$, and $\varphi_\alpha(y) = \varphi_{\alpha \fatsemi_f \beta}(y)$, this immediately gives us that $y \in N_{\alpha \fatsemi_f \beta} \setminus S$, $y{\downarrow_{\alpha \fatsemi_f \beta}} \subseteq S$, and $\psi \Rightarrow \varphi_{\alpha \fatsemi_f \beta}(x)$ therefore $y \in \mathsf{next}(\alpha \fatsemi_f \beta, \psi, S)$.

Now, for the reverse inclusion, take any $y \in \mathsf{next}(\alpha \fatsemi_f \beta, \psi, S)$, so $y \in N_{\alpha \fatsemi_f \beta} \setminus S$ such that $y{\downarrow_{\alpha \fatsemi_f \beta}} \subseteq S$ and $\psi \Rightarrow \varphi_{\alpha \fatsemi_f \beta}(x)$. From previously, we know that $x \in N_\alpha \setminus S$ and $\psi \wedge \varphi_\alpha(x)$ is satisfiable. Now, suppose for the sake of contradiction that $y \notin N_\alpha$, so it must be that $y \in N_{f(\psi')}$ for some $\psi' \in \mathsf{br}_\alpha$. This implies that $x <_{\alpha \fatsemi_f \beta} y$, as we will now show. First, $\psi \Rightarrow \varphi_{\alpha \fatsemi_f \beta}(y) = \varphi_{f(\psi')}(y) \wedge \psi' \Rightarrow \psi'$, so

$\psi' \wedge \varphi_\alpha(x)$ is satisfiable. If $x \notin \mathsf{ext}_\alpha$, then $\varphi_\alpha(x) \Rightarrow \mathsf{stuck}_\alpha$, but we know that $\psi' \Rightarrow \neg\mathsf{stuck}_\alpha$, which contradicts that $\psi' \wedge \varphi_\alpha(x)$ is satisfiable. Therefore, $x \in \mathsf{ext}_\alpha$, and since ψ' is maximal, then it must be the case that $\psi' \Rightarrow \varphi_\alpha(x)$ already, which implies that $x <_{\alpha \fatsemi_f \beta} y$. However, since $x \notin S$, then $y{\downarrow_{\alpha \fatsemi_f \beta}} \not\subseteq S$, which is a contradiction. Therefore, it must be the case that $y \in N_\alpha$. We therefore have that $y \in N_\alpha \setminus S$, $y{\downarrow_\alpha} = y{\downarrow_{\alpha \fatsemi_f \beta}} \subseteq S$, and $\psi \Rightarrow \varphi_{\alpha \fatsemi_f \beta}(y) = \varphi_\alpha(y)$; therefore, $y \in \mathsf{next}(\alpha, \psi, S)$. □

Lemma F.6. *For any binary branching $\alpha, \beta \in lpof_{\mathsf{fin}}(L)$, $S \subseteq \mathsf{nBot}_\alpha$, and $\psi \in \mathsf{valid}_\alpha(S)$:*

$$\mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S) = \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger \circ \mathcal{L}_{lpof}(\alpha, \psi, S)$$

Proof. The proof is by induction on the size of the set $\mathsf{next}^*(\alpha, \psi, S)$.

In the base case, $\mathsf{next}^*(\alpha, \psi, S)$ is empty, so by Lemma F.4, we know that $\psi \in \mathsf{br}_\alpha$ and $\mathsf{next}(\alpha \fatsemi_f \beta, \psi, S) = \min_{f(\psi)}$. Since each LPOF is single-rooted, then $\min_{f(\psi)} = \{x\}$ for some $x \in N_{f(\psi)}$. This gives us:

$$\begin{aligned}
&\mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S)(s) \\
&= \&_{x \in \mathsf{next}(\alpha \fatsemi_f \beta, \psi, S)} \begin{cases} \mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S \cup \{x\})^\dagger(\left[\!\left[\lambda_{\alpha \fatsemi_f \beta}(x)\right]\!\right]_{act}(s)) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) \in act \\ \mathcal{L}_{test}(\alpha \fatsemi_f \beta, \psi, S, x)(s)^\dagger(\left[\!\left[\lambda_{\alpha \fatsemi_f \beta}(x)\right]\!\right]_{test}(s)) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) \in test \\ \bot & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) = \bot \\ \mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S \cup \{x\})(s) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) = \mathsf{fork} \end{cases} \\
&= \begin{cases} \mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S \cup \{x\})^\dagger(\left[\!\left[\lambda_{\alpha \fatsemi_f \beta}(x)\right]\!\right]_{act}(s)) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) \in act \\ \mathcal{L}_{test}(\alpha \fatsemi_f \beta, \psi, S, x)(s)^\dagger(\left[\!\left[\lambda_{\alpha \fatsemi_f \beta}(x)\right]\!\right]_{test}(s)) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) \in test \\ \bot & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) = \bot \\ \mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S \cup \{x\})(s) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) = \mathsf{fork} \end{cases} \\
&= \begin{cases} \mathcal{L}_{lpof}(\beta_\psi, \mathsf{true}, \{x\})^\dagger(\left[\!\left[\lambda_{\beta_\psi}(x)\right]\!\right]_{act}(s)) & \text{if } \lambda_{\beta_\psi}(x) \in act \\ \mathcal{L}_{test}(\alpha \fatsemi_f \beta, \psi, S, x)(s)^\dagger(\left[\!\left[\lambda_{\beta_\psi}(x)\right]\!\right]_{test}(s)) & \text{if } \lambda_{\beta_\psi}(x) \in test \\ \bot & \text{if } \lambda_{\beta_\psi}(x) = \bot \\ \mathcal{L}_{lpof}(\beta_\psi, \mathsf{true}, \{x\})(s) & \text{if } \lambda_{\beta_\psi}(x) = \mathsf{fork} \end{cases} \\
&= \&_{x \in \mathsf{next}(\beta, \mathsf{true}, \emptyset)} \begin{cases} \mathcal{L}_{lpof}(\beta, \mathsf{true}, \{x\})^\dagger(\left[\!\left[\lambda_\beta(x)\right]\!\right]_{act}(s)) & \text{if } \lambda_\beta(x) \in act \\ \mathcal{L}_{test}(\alpha \fatsemi_f \beta, \psi, S, x)(s)^\dagger(\left[\!\left[\lambda_\beta(x)\right]\!\right]_{test}(s)) & \text{if } \lambda_\beta(x) \in test \\ \bot & \text{if } \lambda_\beta(x) = \bot \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \{x\})(s) & \text{if } \lambda_\beta(x) = \mathsf{fork} \end{cases} \\
&= \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)(s) \\
&= \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\eta(s)) \\
&= \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\mathcal{L}_{lpof}(\alpha, \psi, S)(s))
\end{aligned}$$

Now suppose $\mathsf{next}^*(\alpha, \psi, S) \neq \emptyset$. By Lemma F.5, we know that $\mathsf{next}(\alpha \fatsemi_f \beta, \psi, S) = \mathsf{next}(\alpha, \psi, S) \neq \emptyset$. Given this, we have:

$$\mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S)(s)$$

$$= \bigwedge_{x \in \mathsf{next}(\alpha \fatsemi_f \beta, \psi, S)} \begin{cases} \mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S \cup \{x\})^\dagger(\left[\!\left[\lambda_{\alpha \fatsemi_f \beta}(x)\right]\!\right]_{act}(s)) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) \in act \\ \mathcal{L}_{test}(\alpha \fatsemi_f \beta, \psi, S, x)(s)^\dagger(\left[\!\left[\lambda_{\alpha \fatsemi_f \beta}(x)\right]\!\right]_{test}(s)) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) \in test \\ \bot & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) = \bot \\ \mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S \cup \{x\})(s) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) = \mathsf{fork} \end{cases}$$

$$= \bigwedge_{x \in \mathsf{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S \cup \{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}(s)) & \text{if } \lambda_\alpha(x) \in act \\ \mathcal{L}_{test}(\alpha \fatsemi_f \beta, \psi, S, x)(s)^\dagger([\![\lambda_\alpha(x)]\!]_{test}(s)) & \text{if } \lambda_\alpha(x) \in test \\ \bot & \text{if } \lambda_\alpha(x) = \bot \\ \mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \mathsf{fork} \end{cases}$$

Clearly $\mathsf{next}^*(\alpha, \psi, S \cup \{x\}) \subset \mathsf{next}^*(\alpha, \psi, S)$ and $\psi \in \mathsf{valid}_\alpha(S \cup \{x\})$, if x is not a test node and therefore $\mathsf{valid}_\alpha(S \cup \{x\}) = \mathsf{valid}_\alpha(S)$. So, by the induction hypothesis and monad laws, we get:

$$= \bigwedge_{x \in \mathsf{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}(s))) & \text{if } \lambda_\alpha(x) \in act \\ \mathcal{L}_{test}(\alpha \fatsemi_f \beta, \psi, S, x)(s)^\dagger([\![\lambda_\alpha(x)]\!]_{test}(s)) & \text{if } \lambda_\alpha(x) \in test \\ \bot & \text{if } \lambda_\alpha(x) = \bot \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})(s)) & \text{if } \lambda_\alpha(x) = \mathsf{fork} \end{cases}$$

We also have that $\mathsf{next}^*(\alpha, S \cup \{x\}, \psi \wedge (\!|x = [\![\lambda_\alpha(x)]\!]_{test}(s)|\!)) \subset \mathsf{next}^*(\alpha, S, \psi)$. In addition, $\psi \wedge x, \psi \wedge \neg x \in \mathsf{valid}_\alpha(S \cup \{x\})$. So, by the induction hypothesis:

$$= \bigwedge_{x \in \mathsf{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}(s))) & \text{if } \lambda_\alpha(x) \in act \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\mathcal{L}_{test}(\alpha, \psi, S, x)(s)^\dagger([\![\lambda_\alpha(x)]\!]_{test}(s))) & \text{if } \lambda_\alpha(x) \in test \\ \bot & \text{if } \lambda_\alpha(x) = \bot \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})(s)) & \text{if } \lambda_\alpha(x) = \mathsf{fork} \end{cases}$$

Finally since $f^\dagger(\bot) = \bot$ for any f, we get:

$$= \bigwedge_{x \in \mathsf{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}(s))) & \text{if } \lambda_\alpha(x) \in act \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\mathcal{L}_{test}(\alpha, \psi, S, x)(s)^\dagger([\![\lambda_\alpha(x)]\!]_{test}(s))) & \text{if } \lambda_\alpha(x) \in test \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\bot) & \text{if } \lambda_\alpha(x) = \bot \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger(\mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})(s)) & \text{if } \lambda_\alpha(x) = \mathsf{fork} \end{cases}$$

Since $f^\dagger(X \,\&\, Y) = f^\dagger(X) \,\&\, f^\dagger(Y)$:

$$= \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger \left(\bigwedge_{x \in \mathsf{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}(s)) & \text{if } \lambda_\alpha(x) \in act \\ \mathcal{L}_{test}(\alpha, \psi, S, x)(s)^\dagger([\![\lambda_\alpha(x)]\!]_{test}(s)) & \text{if } \lambda_\alpha(x) \in test \\ \bot & \text{if } \lambda_\alpha(x) = \bot \\ \mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \mathsf{fork} \end{cases} \right)$$

$$= \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger \left(\mathcal{L}_{lpof}(\alpha, \psi, S)(s) \right) \qquad \square$$

Lemma F.7. $\mathcal{L}([\![C_1; C_2]\!]) = \mathcal{L}([\![C_2]\!])^\dagger \circ \mathcal{L}([\![C_1]\!])$.

Proof.

$$\mathcal{L}([\![C_1; C_2]\!]) = \mathcal{L}([\![C_1]\!] \fatsemi [\![C_2]\!])$$

By the definition of $\fatsemi$ for infinite pomsets:

$$= \mathcal{L}\left(\sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \right)$$

Since $\mathcal{L}$ is Scott continuous:

$$= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}(\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta})$$
$$= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \sup_{\boldsymbol{\gamma} \ll \boldsymbol{\alpha} \fatsemi \boldsymbol{\beta}} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\gamma})$$

Since $\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta}$ is finite, then $\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \ll \boldsymbol{\alpha} \fatsemi \boldsymbol{\beta}$ and therefore since $\mathcal{L}_{\mathsf{fin}}$ is monotone, then $\sup_{\boldsymbol{\gamma} \ll \boldsymbol{\alpha} \fatsemi \boldsymbol{\beta}} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\gamma}) = \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta})$.

$$= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta})$$
$$= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{\mathsf{fin}}(\{\alpha \fatsemi_f \beta \mid \alpha \in \boldsymbol{\alpha}, \beta \in \boldsymbol{\beta}, f \in \mathsf{copy}_{\alpha,\beta}\})$$

The set above is an equivalence class, so fixing an arbitrary $\alpha \in \boldsymbol{\alpha}$, $\beta \in \boldsymbol{\beta}$, and $f \in \mathsf{copy}_{\alpha,\beta}$, we can rewrite the expression as follows:

$$= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{\mathsf{fin}}([\alpha \fatsemi_f \beta])$$
$$= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{lpof}(\alpha \fatsemi_f \beta, \mathsf{true}, \emptyset)$$

By Lemma F.6:

$$= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)^\dagger \circ \mathcal{L}_{lpof}(\alpha, \mathsf{true}, \emptyset)$$
$$= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{\mathsf{fin}}([\beta])^\dagger \circ \mathcal{L}_{\mathsf{fin}}([\alpha])$$

By continuity of Kleisli extension, and since $\boldsymbol{\alpha} = [\alpha]$ and $\boldsymbol{\beta} = [\beta]$:

$$= \left(\sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\beta}) \right)^\dagger \circ \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha})$$
$$= \mathcal{L}([\![C_2]\!])^\dagger \circ \mathcal{L}([\![C_1]\!])$$ □

F.4 Properties of Linearization: Guarded Branching

Lemma F.8. *For any $\alpha, \beta \in lpof_{\mathsf{fin}}(act)$, $b \in test$, and $x \in nodes \setminus (N_\alpha \cup N_\beta)$:*

$$\mathcal{L}_{lpof}(\mathsf{guard}(x, b, \alpha, \beta), \mathsf{true}, \emptyset)(s) = \left[b' \mapsto \begin{cases} \mathcal{L}_{lpof}(\alpha, \mathsf{true}, \emptyset)(s) & \text{if } b' = 1 \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)(s) & \text{if } b' = 0 \end{cases} \right]^\dagger \left([\![b]\!]_{test}(s) \right)$$

Proof. Let $\gamma = \mathsf{guard}(x, b, \alpha, \beta)$.

$$\mathcal{L}_{lpof}(\gamma, \mathsf{true}, \emptyset)(s) = \bigotimes_{y \in \mathsf{next}(\gamma, \mathsf{true}, \emptyset)} \begin{cases} \mathcal{L}_{lpof}(\gamma, \mathsf{true}, \{y\})^\dagger([\![\lambda_\gamma(y)]\!]_{act}(s)) & \text{if } \lambda_\gamma(y) \in act \\ \mathcal{L}_{test}(\gamma, \mathsf{true}, \emptyset, y)(s)^\dagger([\![\lambda_\gamma(y)]\!]_{test}(s)) & \text{if } \lambda_\gamma(y) \in test \\ \bot & \text{if } \lambda_\gamma(y) = \bot \\ \mathcal{L}_{lpof}(\gamma, \mathsf{true}, \{y\})(s) & \text{if } \lambda_\gamma(y) = \mathsf{fork} \end{cases}$$

Since x is the root of γ, then $\mathsf{next}(\gamma, \mathsf{true}, \emptyset) = \{x\}$ and $\lambda_\gamma(x) \in test$.

$$= \mathcal{L}_{test}(\gamma, \mathsf{true}, \emptyset, x)(s)^\dagger([\![\lambda_\gamma(x)]\!]_{test}(s))$$
$$= \left[b' \mapsto \begin{cases} \mathcal{L}_{lpof}(\gamma, x, \{x\})(s) & \text{if } b' = 1 \\ \mathcal{L}_{lpof}(\gamma, \neg x, \{x\})(s) & \text{if } b' = 0 \end{cases} \right]^\dagger \left([\![b]\!]_{test}(s) \right)$$

The formula x prevents all nodes in β from being scheduled and similarly $\neg x$ prevents the α nodes from being scheduled, so we can remove the guard call to obtain the following:

$$= \left[b' \mapsto \begin{cases} \mathcal{L}_{lpof}(\alpha, \mathsf{true}, \emptyset)(s) & \text{if } b' = 1 \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)(s) & \text{if } b' = 0 \end{cases} \right]^{\dagger} \left([\![b]\!]_{test}(s) \right) \qquad \square$$

Lemma F.9.

$$\mathcal{L}([\![\texttt{if } b \texttt{ then } C_1 \texttt{ else } C_2]\!])(s) = \left[b' \mapsto \begin{cases} \mathcal{L}([\![C_1]\!])(s)) & \text{if } b' = 1 \\ \mathcal{L}([\![C_2]\!])(s) & \text{if } b' = 0 \end{cases} \right]^{\dagger} \left([\![b]\!]_{test}(s) \right)$$

Proof.

$$\begin{aligned}
&\mathcal{L}[\![\texttt{if } b \texttt{ then } C_1 \texttt{ else } C_2]\!](s) \\
&= \mathcal{L}(\mathsf{guard}(b, [\![C_1]\!], [\![C_2]\!]))(s) \\
&= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{\mathsf{fin}}(\mathsf{guard}(b, \boldsymbol{\alpha}, \boldsymbol{\beta}))(s) \\
&= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{\mathsf{fin}}(\{\mathsf{guard}(x, b, \alpha, \beta) \mid x \in \mathit{nodes}, \alpha \in \boldsymbol{\alpha}, \beta \in \boldsymbol{\beta}\})(s) \\
&= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{\mathsf{fin}}([\mathsf{guard}(x, b, \alpha, \beta)])(s) \\
&= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{lpof}(\mathsf{guard}(x, b, \alpha, \beta), \mathsf{true}, \emptyset)(s) \\
&= \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \left[b' \mapsto \begin{cases} \mathcal{L}_{lpof}(\alpha, \mathsf{true}, \emptyset)(s) & \text{if } b' = 1 \\ \mathcal{L}_{lpof}(\beta, \mathsf{true}, \emptyset)(s) & \text{if } b' = 0 \end{cases} \right]^{\dagger} \left([\![b]\!]_{test}(s) \right) \qquad \text{(By Lemma F.8)} \\
&= \left[b' \mapsto \begin{cases} \sup_{\boldsymbol{\alpha} \ll [\![C_1]\!]} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha})(s) & \text{if } b' = 1 \\ \sup_{\boldsymbol{\beta} \ll [\![C_2]\!]} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\beta})(s) & \text{if } b' = 0 \end{cases} \right]^{\dagger} \left([\![b]\!]_{test}(s) \right) \\
&= \left[b' \mapsto \begin{cases} \mathcal{L}([\![C_1]\!])(s) & \text{if } b' = 1 \\ \mathcal{L}([\![C_2]\!])(s) & \text{if } b' = 0 \end{cases} \right]^{\dagger} \left([\![b]\!]_{test}(s) \right) \qquad \square
\end{aligned}$$

Lemma F.10.

$$\mathcal{L}([\![C_1 \,\&\, C_2]\!])(s) = \mathcal{L}([\![C_1]\!])(s) \,\&\, \mathcal{L}([\![C_2]\!])(s)$$

Proof.

$$\begin{aligned}
\mathcal{L}[\![C_1 \,\&\, C_2]\!](s) &= \mathcal{L}(\mathsf{guard}(\&, [\![C_1]\!], [\![C_2]\!]))(s) \\
&= \left[b \mapsto \begin{cases} \mathcal{L}([\![C_1]\!])(s) & \text{if } b = 1 \\ \mathcal{L}([\![C_2]\!])(s) & \text{if } b = 0 \end{cases} \right]^{\dagger} \left([\![\&]\!]_{test}(s) \right) \qquad \text{By Lemma F.9} \\
&= \left[b \mapsto \begin{cases} \mathcal{L}([\![C_1]\!])(s) & \text{if } b = 1 \\ \mathcal{L}([\![C_2]\!])(s) & \text{if } b = 0 \end{cases} \right]^{\dagger} \left(\bigotimes_{x \in \{0,1\}} \eta(x) \right) \\
&= \bigotimes_{x \in \{0,1\}} \left[b \mapsto \begin{cases} \mathcal{L}([\![C_1]\!])(s) & \text{if } b = 1 \\ \mathcal{L}([\![C_2]\!])(s) & \text{if } b = 0 \end{cases} \right]^{\dagger} (\eta(x)) \\
&= \bigotimes_{x \in \{0,1\}} \begin{cases} \mathcal{L}([\![C_1]\!])(s) & \text{if } x = 1 \\ \mathcal{L}([\![C_2]\!])(s) & \text{if } x = 0 \end{cases} \\
&= \mathcal{L}([\![C_1]\!])(s) \,\&\, \mathcal{L}([\![C_2]\!])(s)
\end{aligned}$$

$\square$

Lemma F.11.

$$\mathcal{L}(\llbracket C_1 \oplus_p C_2 \rrbracket)(s) = \mathcal{L}(\llbracket C_1 \rrbracket)(s) \oplus_p \mathcal{L}(\llbracket C_2 \rrbracket)(s)$$

Proof.

$$\begin{aligned}
\mathcal{L}\llbracket C_1 \oplus_p C_2 \rrbracket(s) &= \mathcal{L}(\mathrm{guard}(\oplus_p, \llbracket C_1 \rrbracket, \llbracket C_2 \rrbracket))(s) \\
&= \left[b \mapsto \begin{cases} \mathcal{L}(\llbracket C_1 \rrbracket)(s) & \text{if } b = 1 \\ \mathcal{L}(\llbracket C_2 \rrbracket)(s) & \text{if } b = 0 \end{cases} \right]^\dagger \left(\llbracket \oplus_p \rrbracket_{test}(s) \right) \qquad \text{By Lemma F.9} \\
&= \left[b \mapsto \begin{cases} \mathcal{L}(\llbracket C_1 \rrbracket)(s) & \text{if } b = 1 \\ \mathcal{L}(\llbracket C_2 \rrbracket)(s) & \text{if } b = 0 \end{cases} \right]^\dagger \left(\eta(1) \oplus_p \eta(0) \right) \\
&= \mathcal{L}(\llbracket C_1 \rrbracket)(s) \oplus_p \mathcal{L}(\llbracket C_2 \rrbracket)(s)
\end{aligned}$$

□

F.5 Properties of Linearization: While Loops

Lemma F.12. *Let $D(S)$ be the dcpo such that $\mathcal{L} : pom \to S \to D(S)$, and let $\bot_{pom} \in pom$ be the bottom of the pomset order, $\bot_D \in D(S)$ be the bottom of $D(S)$, and $\bot_D^\bullet : S \to D(S)$ be the bottom of the pointwise order on $D(S)$. Then*

$$\mathcal{L}\left(\Phi^n_{\langle C,b\rangle}(\bot_{pom})\right) = \Psi^n_{\langle \mathcal{L}(\llbracket C \rrbracket),b\rangle}(\bot_D^\bullet)$$

Proof. The proof is by induction on n. If $n = 0$, then we have:

$$\mathcal{L}\left(\Phi^0_{\langle C,b\rangle}(\bot_{pom})\right)(s) = \mathcal{L}(\bot_{pom})(s) = \bot_D = \bot_D^\bullet(s) = \Psi^0_{\langle \mathcal{L}(\llbracket C \rrbracket),b\rangle}(\bot_D^\bullet)(s)$$

Now, suppose the claim holds for n, then we have:

$$\begin{aligned}
\mathcal{L}\left(\Phi^{n+1}_{\langle C,b\rangle}(\bot_{pom})\right)(s) &= \mathcal{L}\left(\Phi_{\langle C,b\rangle}\left(\Phi^n_{\langle C,b\rangle}(\bot_{pom})\right)\right)(s) \\
&= \mathcal{L}\left(\mathrm{guard}\left(b, \llbracket C \rrbracket \fatsemi \Phi^n_{\langle C,b\rangle}(\bot_{pom}), \llbracket \mathtt{skip} \rrbracket\right)\right)(s)
\end{aligned}$$

By Lemma F.8.

$$= \left[b' \mapsto \begin{cases} \mathcal{L}\left(\llbracket C \rrbracket \fatsemi \Phi^n_{\langle C,b\rangle}(\bot_{pom})\right)(s) & \text{if } b' = 1 \\ \mathcal{L}(\llbracket \mathtt{skip} \rrbracket)(s) & \text{if } b' = 0 \end{cases} \right]^\dagger \left(\llbracket b \rrbracket_{test}(s) \right)$$

By Lemmas F.3 and F.7.

$$= \left[b' \mapsto \begin{cases} \mathcal{L}\left(\Phi^n_{\langle C,b\rangle}(\bot_{pom})\right)^\dagger (\mathcal{L}(\llbracket C \rrbracket)(s)) & \text{if } b' = 1 \\ \eta(s) & \text{if } b' = 0 \end{cases} \right]^\dagger \left(\llbracket b \rrbracket_{test}(s) \right)$$

By the induction hypothesis.

$$\begin{aligned}
&= \left[b' \mapsto \begin{cases} \left(\Psi^n_{\langle \mathcal{L}\llbracket C \rrbracket,b\rangle}(\bot_D^\bullet)\right)^\dagger (\mathcal{L}(\llbracket C \rrbracket)(s)) & \text{if } b' = 1 \\ \eta(s) & \text{if } b' = 0 \end{cases} \right]^\dagger \left(\llbracket b \rrbracket_{test}(s) \right) \\
&= \Psi_{\langle \mathcal{L}\llbracket C \rrbracket,b\rangle}\left(\Psi^n_{\langle \mathcal{L}\llbracket C \rrbracket,b\rangle}(\bot_D^\bullet)\right)(s) \\
&= \Psi^{n+1}_{\langle \mathcal{L}\llbracket C \rrbracket,b\rangle}(\bot_D^\bullet)(s)
\end{aligned}$$

□

Lemma F.13. $\mathcal{L}(\llbracket \textbf{while } b \textbf{ do } C \rrbracket) = \mathsf{lfp}\left(\Psi_{\langle \mathcal{L}(\llbracket C \rrbracket), b\rangle}\right)$ *where*

$$\Psi_{\langle f,b\rangle}(g)(s) \triangleq \left[b' \mapsto \begin{cases} g^\dagger(f(s)) & \text{if } b' = 1 \\ \eta(s) & \text{if } b' = 0 \end{cases}\right]^\dagger \left(\llbracket b \rrbracket_{test}(s)\right)$$

Proof.

$$\begin{aligned}
\mathcal{L}(\llbracket \textbf{while } b \textbf{ do } C \rrbracket) &= \mathcal{L}\left(\mathsf{lfp}\left(\Phi_{\langle C,b\rangle}\right)\right) \\
&= \mathcal{L}\left(\sup_{n\in\mathbb{N}} \Phi^n_{\langle C,b\rangle}(\bot_{pom})\right) && \text{(By the Kleene fixed point theorem)} \\
&= \sup_{n\in\mathbb{N}} \mathcal{L}\left(\Phi^n_{\langle C,b\rangle}(\bot_{pom})\right) && \text{(By Scott continuity of } \mathcal{L}) \\
&= \sup_{n\in\mathbb{N}} \Psi^n_{\langle \mathcal{L}(\llbracket C \rrbracket), b\rangle}(\bot^\bullet_D) && \text{(By Lemma F.12)} \\
&= \mathsf{lfp}\left(\Psi_{\langle \mathcal{L}(\llbracket C \rrbracket), b\rangle}\right) && \text{(By the Kleene fixed point theorem)}
\end{aligned}$$ □

F.6 Main Lemma

Lemma 6.4 (Linearization). *The following properties hold:*

$$\mathcal{L}(\llbracket \textbf{skip} \rrbracket) = \eta \qquad \mathcal{L}(\llbracket a \rrbracket) = \llbracket a \rrbracket_{act} \qquad \mathcal{L}(\llbracket C_1; C_2 \rrbracket) = \mathcal{L}(\llbracket C_2 \rrbracket)^\dagger \circ \mathcal{L}(\llbracket C_1 \rrbracket)$$

$$\mathcal{L}(\llbracket C_1 \,\&\, C_2 \rrbracket)(s) = \mathcal{L}(\llbracket C_1 \rrbracket)(s) \,\&\, \mathcal{L}(\llbracket C_2 \rrbracket)(s) \qquad \mathcal{L}(\llbracket C_1 \oplus_p C_2 \rrbracket)(s) = \mathcal{L}(\llbracket C_1 \rrbracket)(s) \oplus_p \mathcal{L}(\llbracket C_2 \rrbracket)(s)$$

$$\mathcal{L}(\llbracket \textbf{if } b \textbf{ then } C_1 \textbf{ else } C_2 \rrbracket)(s) = \left[b' \mapsto \begin{cases} \mathcal{L}(\llbracket C_1 \rrbracket)(s) & \text{if } b' = 1 \\ \mathcal{L}(\llbracket C_2 \rrbracket)(s) & \text{if } b' = 0 \end{cases}\right]^\dagger \left(\llbracket b \rrbracket_{test}(s)\right)$$

$$\mathcal{L}(\llbracket \textbf{while } b \textbf{ do } C \rrbracket) = \mathsf{lfp}\left(\Psi_{\langle \mathcal{L}(\llbracket C \rrbracket), b\rangle}\right)$$

where $\Psi_{\langle f,b\rangle}(g)(s) \triangleq \left[b' \mapsto \begin{cases} g^\dagger(f(s)) & \text{if } b' = 1 \\ \eta(s) & \text{if } b' = 0 \end{cases}\right]^\dagger \left(\llbracket b \rrbracket_{test}(s)\right)$.

Proof. Follows immediately from Lemmas F.3, F.7, F.9 to F.11 and F.13 □

G Powerdomains

In this section, we fix our domain of computation to be the Hoare powerdomain $\langle \mathcal{P}(S), \subseteq\rangle$. In particular, this means that suprema are given by set union: $\sup_{i\in I} S_i \triangleq \bigcup_{i\in I} S_i$.

Lemma 8.1. *If* $\mathrm{Bot}_\alpha = \emptyset$, $\varphi_\alpha(x) = \mathsf{true}$ *for all* $x \in N_\alpha$, *and* $f = [\mathsf{true} \mapsto \beta]$, *then:*

$$N_{\alpha \fatsemi_f \beta} = N_\alpha \cup N_\beta \quad \textit{and} \quad <_{\alpha \fatsemi_f \beta} = <_\alpha \cup <_\beta \cup (N_\alpha \times N_\beta)$$

Proof. First note that since $\mathrm{Bot}_\alpha = \emptyset$, then $\mathsf{stuck}_\alpha = \mathsf{false}$ and so $\mathsf{ext}_\alpha = N_\alpha$. Therefore, $\mathrm{br}_\alpha = \{\bigwedge_{x\in N_\alpha} \varphi_\alpha(x)\} = \{\mathsf{true}\}$. We now show the condition on nodes.

$$N_{\alpha \fatsemi_f \beta} = N_\alpha \cup \bigcup_{\psi \in \mathrm{br}_\alpha} N_{f(\psi)} = N_\alpha \cup N_{f(\mathsf{true})} = N_\alpha \cup N_\beta$$

We now also show the condition on orders:

$$\begin{aligned}
<_{\alpha \fatsemi_f \beta} &= <_\alpha \cup \bigcup_{\psi \in \mathrm{br}_\alpha} \left(<_{f(\psi)} \cup \{x \in N_\alpha \mid \psi \Rightarrow \varphi_\alpha(x)\} \times N_{f(\psi)}\right) \\
&= <_\alpha \cup <_{f(\mathsf{true})} \cup \{x \in N_\alpha \mid \mathsf{true} \Rightarrow \mathsf{true}\} \times N_{f(\mathsf{true})} \\
&= <_\alpha \cup <_\beta \cup (N_\alpha \times N_\beta)
\end{aligned}$$ □

G.1 Properties of the Translation Function

Lemma G.1 (Monotonicity of tr_{lpof}). *If $\alpha \sqsubseteq_{lpof} \alpha'$ and $\psi \in \mathrm{br}_\alpha$, then $\mathrm{tr}_{lpof}(\alpha, \psi) = \mathrm{tr}_{lpof}(\alpha', \psi)$*

Proof. We first show that $N_{\mathrm{tr}_{lpof}(\alpha,\psi)} = N_{\mathrm{tr}_{lpof}(\alpha',\psi)}$, or in other words, that:

$$\{x \in N_\alpha \mid \psi \Rightarrow \varphi_\alpha(x), \lambda_\alpha(x) \neq \&\} = \{x \in N_{\alpha'} \mid \psi \Rightarrow \varphi_{\alpha'}(x), \lambda_{\alpha'}(x) \neq \&\}$$

For the forward inclusion, suppose that $x \in N_\alpha$ such that $\psi \Rightarrow \varphi_\alpha(x)$ and $\lambda_\alpha(x) \neq \&$. Since $\varphi_\alpha(x) = \varphi_{\alpha'}(x)$, then $\psi \Rightarrow \varphi_{\alpha'}(x)$. We know that $\lambda_\alpha(x) \neq \&$, and $\psi \in \mathrm{br}_\alpha$, so $\lambda_\alpha(x)$ cannot be $\bot$ either. From $\alpha \sqsubseteq_{lpof} \alpha'$, we know that $\lambda_\alpha(x) \leq \lambda_{\alpha'}(x)$, but the only label smaller than & is $\bot$, so $\lambda_{\alpha'}(x) \neq \&$ and therefore $x \in \{x \in N_{\alpha'} \mid \psi \Rightarrow \varphi_{\alpha'}(x), \lambda_{\alpha'(x)} \neq \&\}$.

We now show the reverse inclusion. Take any $x \in N_{\alpha'}$ such that $\psi \Rightarrow \varphi_{\alpha'}(x)$ and $\lambda_{\alpha'}(x) \neq \bot$. If $x \notin N_\alpha$, then by Lemma B.2, $x \in \mathrm{Bot}_\alpha{\uparrow_{\alpha'}}$. This means that there is some $y \in \mathrm{Bot}_\alpha$ such that $y <_{\alpha'} x$, and so $\varphi_{\alpha'}(x) \Rightarrow \varphi_{\alpha'}(y) \Rightarrow \mathsf{stuck}_\alpha$. But this contradicts the fact that $\psi \Rightarrow \varphi_{\alpha'}(x)$ and $\psi \in \mathrm{br}_\alpha$. Therefore, it must be that $x \in N_\alpha$. Since $\varphi_\alpha(x) = \varphi_{\alpha'}(x)$, then $\psi \Rightarrow \varphi_\alpha(x)$. Finally, since $\lambda_\alpha(x) \leq \lambda_{\alpha'}(x)$ and $\lambda_{\alpha'}(x) \neq \&$, then $\lambda_\alpha(x) \neq \&$ since the only label that satisfies $\& \leq \ell$ is &.

Now, for the order, we have:

$$\begin{aligned} <_{\mathrm{tr}_{lpof}(\alpha,\psi)} &= <_\alpha \cap (N_{\mathrm{tr}_{lpof}(\alpha,\psi)} \times N_{\mathrm{tr}_{lpof}(\alpha,\psi)}) \\ &= (<_{\alpha'} \cap (N_\alpha \times N_\alpha)) \cap (N_{\mathrm{tr}_{lpof}(\alpha,\psi)} \times N_{\mathrm{tr}_{lpof}(\alpha,\psi)}) \end{aligned}$$

Since $N_{\mathrm{tr}_{lpof}(\alpha,\psi)} \subseteq N_\alpha$:

$$\begin{aligned} &= <_{\alpha'} \cap (N_{\mathrm{tr}_{lpof}(\alpha,\psi)} \times N_{\mathrm{tr}_{lpof}(\alpha,\psi)}) \\ &= <_{\alpha'} \cap (N_{\mathrm{tr}_{lpof}(\alpha',\psi)} \times N_{\mathrm{tr}_{lpof}(\alpha',\psi)}) \\ &= <_{\mathrm{tr}_{lpof}(\alpha',\psi)} \end{aligned}$$

Finally, for any $x \in N_{\mathrm{tr}_{lpof}(\alpha,\psi)}$, we have:

$$\lambda_{\mathrm{tr}_{lpof}(\alpha,\psi)}(x) = \begin{cases} \lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \notin \mathit{test} \\ \mathbf{assume}\ \lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \in \mathit{test} \text{ and } \psi \Rightarrow x \\ \mathbf{assume}\ \neg\lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \in \mathit{test} \text{ and } \psi \Rightarrow \neg x \end{cases}$$

By virtue of the fact that $x \in N_{\mathrm{tr}_{lpof}(\alpha,\psi)}$, we know that $\psi \Rightarrow \varphi_\alpha(x)$, and since $\psi \in \mathrm{br}_\alpha$, then $\psi \Rightarrow \neg\mathsf{stuck}_\alpha$, therefore $\lambda_\alpha(x)$ cannot be $\bot$, and since *label* is a flat cpo, then it must be that $\lambda_\alpha(x) = \lambda_{\alpha'}(x)$.

$$\begin{aligned} &= \begin{cases} \lambda_{\alpha'}(x) & \text{if } \lambda_{\alpha'}(x) \notin \mathit{test} \\ \mathbf{assume}\ \lambda_{\alpha'}(x) & \text{if } \lambda_{\alpha'}(x) \in \mathit{test} \text{ and } \psi \Rightarrow x \\ \mathbf{assume}\ \neg\lambda_{\alpha'}(x) & \text{if } \lambda_{\alpha'}(x) \in \mathit{test} \text{ and } \psi \Rightarrow \neg x \end{cases} \\ &= \lambda_{\mathrm{tr}_{lpof}(\alpha',\psi)}(x) \end{aligned}$$

And finally, clearly we have $\varphi_{\mathrm{tr}_{lpof}(\alpha,\psi)}(x) = \varphi_{\mathrm{tr}_{lpof}(\alpha',\psi)}(x) = \mathsf{true}$. □

Lemma G.2 (Monotonicity of $\mathrm{tr}_{\mathsf{fin}}$). *If $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\alpha}'$, then $\mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}) \subseteq \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}')$.*

Proof. Since $\boldsymbol{\alpha} \sqsubseteq_{pom} \boldsymbol{\alpha}'$, there exists an $\alpha \in \boldsymbol{\alpha}$ and $\alpha' \in \boldsymbol{\alpha}'$ such that $\alpha \sqsubseteq_{lpof} \alpha'$. We now prove the claim as follows:

$$\begin{aligned} \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}) &= \mathrm{tr}_{\mathsf{fin}}([\alpha]) \\ &= \{[\beta] \mid \psi \in \mathrm{br}_\alpha, \beta \in \mathrm{tr}_{lpof}(\alpha)\} \\ &\subseteq \{[\beta] \mid \psi \in \mathrm{br}_{\alpha'}, \beta \in \mathrm{tr}_{lpof}(\alpha)\} && \text{By Lemma E.4.} \\ &= \{[\beta] \mid \psi \in \mathrm{br}_{\alpha'}, \beta \in \mathrm{tr}_{lpof}(\alpha')\} && \text{By Lemma G.1.} \\ &= \mathrm{tr}_{\mathsf{fin}}([\alpha']) \end{aligned}$$

$$= \mathsf{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}') \qquad \square$$

G.2 Translation and Pomset Operations

Lemma G.3. *For any $\ell \in act \cup \{\mathsf{fork}\}$:*

$$\mathsf{tr}(\langle \ell \rangle) = \{\langle \ell \rangle\}$$

Proof. Since $\langle \ell \rangle$ is a finite pomset:

$$\mathsf{tr}(\langle \ell \rangle) = \mathsf{tr}_{\mathsf{fin}}(\langle \ell \rangle)$$

Fixing any arbitrary $x \in nodes$, we have that $\langle \ell \rangle = [\langle \ell \rangle_x]$.

$$= \mathsf{tr}_{\mathsf{fin}}([\langle \ell \rangle_x])$$
$$= \{[\mathsf{tr}_{lpof}(\langle \ell \rangle_x, \psi)] \mid \psi \in \mathsf{br}_{\langle \ell \rangle_x}\}$$

Since $\mathsf{br}_{\langle \ell \rangle_x} = \{\mathsf{true}\}$ and $\mathsf{tr}_{lpof}(\langle \ell \rangle_x, \mathsf{true}) = \langle \ell \rangle_x$:

$$= \{[\langle \ell \rangle_x]\}$$
$$= \{\langle \ell \rangle\} \qquad \square$$

For sequential composition of lpof s, where $\mathsf{br}_\alpha = \{\mathsf{true}\}$, we will write $\alpha \fatsemi \beta$ (omitting the copy function f), to mean $\alpha \fatsemi_f \beta$ where $f \triangleq [\mathsf{true} \mapsto \beta]$, *i.e.,* f does not perform any renaming, since only a single isomorphic copy of β is needed.

Lemma G.4. *For any $\alpha, \beta \in lpof_{\mathsf{fin}}(label)$, $f \in \mathsf{copy}_{\alpha,\beta}$, $\psi \in \mathsf{br}_\alpha$, and $\psi' \in \mathsf{br}_{f(\psi)}$,*

$$\mathsf{tr}_{lpof}(\alpha \fatsemi_f \beta, \psi \wedge \psi') = \mathsf{tr}_{lpof}(\alpha, \psi) \fatsemi \mathsf{tr}_{lpof}(f(\psi), \psi')$$

Proof. We show the equality component-wise. Some of the steps use Lemma 8.1. First, for the nodes, we have:

$$N_{\mathsf{tr}_{lpof}(\alpha \fatsemi_f \beta, \psi \wedge \psi')}$$
$$= \{x \in N_{\alpha \fatsemi_f \beta} \mid \psi \wedge \psi' \Rightarrow \varphi_{\alpha \fatsemi_f \beta}(x), \lambda_{\alpha \fatsemi_f \beta}(x) \neq \&\}$$

Since $\psi \in \mathsf{br}_\alpha$, then we will only take nodes from $f(\psi)$ and no other $\psi'' \in \mathsf{br}_\alpha$

$$= \{x \in N_\alpha \mid \psi \wedge \psi' \Rightarrow \varphi_\alpha(x), \lambda_\alpha(x) \neq \&\} \cup \{x \in N_{f(\psi)} \mid \psi \wedge \psi' \Rightarrow \varphi_{f(\psi)}(x) \wedge \psi, \lambda_{f(\beta)}(x) \neq \&\}$$
$$= \{x \in N_\alpha \mid \psi \Rightarrow \varphi_\alpha(x), \lambda_\alpha(x) \neq \&\} \cup \{x \in N_{f(\psi)} \mid \psi' \Rightarrow \varphi_{f(\psi)}(x), \lambda_{f(\beta)}(x) \neq \&\}$$
$$= N_{\mathsf{tr}_{lpof}(\alpha, \psi)} \cup N_{\mathsf{tr}_{lpof}(f(\psi), \psi')}$$
$$= N_{\mathsf{tr}_{lpof}(\alpha, \psi) \fatsemi \mathsf{tr}_{lpof}(f(\psi), \psi')}$$

Now, we show that the orders are the same:

$$<_{\mathsf{tr}_{lpof}(\alpha \fatsemi_f \beta, \psi \wedge \psi')}$$
$$= <_{\alpha \fatsemi_f \beta} \cap (N_{\mathsf{tr}_{lpof}(\alpha \fatsemi_f \beta, \psi \wedge \psi')} \times N_{\mathsf{tr}_{lpof}(\alpha \fatsemi_f \beta, \psi \wedge \psi')})$$
$$= (<_\alpha \cap (N_{\mathsf{tr}_{lpof}(\alpha, \psi)} \times N_{\mathsf{tr}_{lpof}(\alpha, \psi)}) \cup (<_{f(\psi)} \cap (N_{\mathsf{tr}_{lpof}(f(\psi), \psi')} \times N_{\mathsf{tr}_{lpof}(f(\psi), \psi')}) \cup (N_{\mathsf{tr}_{lpof}(\alpha, \psi)} \times N_{\mathsf{tr}_{lpof}(f(\psi), \psi')})$$
$$= <_{\mathsf{tr}_{lpof}(\alpha, \psi) \fatsemi \mathsf{tr}_{lpof}(f(\psi), \psi')}$$

Now, for any x, clearly we have:

$$\varphi_{\mathsf{tr}_{lpof}(\alpha \fatsemi_f \beta, \psi \wedge \psi')}(x) = \varphi_{\mathsf{tr}_{lpof}(\alpha, \psi) \fatsemi \mathsf{tr}_{lpof}(f(\psi), \psi')}(x) = \mathsf{true}$$

If $x \in N_\alpha$, then:

$$\lambda_{\mathsf{tr}_{lpof}(\alpha \fatsemi_f \beta, \psi \wedge \psi')}(x) = \begin{cases} \lambda_{\alpha \fatsemi_f \beta}(x) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) \notin test \\ \mathbf{assume}\ \lambda_{\alpha \fatsemi_f \beta}(x) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) \in test \text{ and } \psi \wedge \psi' \Rightarrow x \\ \mathbf{assume}\ \neg\lambda_{\alpha \fatsemi_f \beta}(x) & \text{if } \lambda_{\alpha \fatsemi_f \beta}(x) \in test \text{ and } \psi \wedge \psi' \Rightarrow \neg x \end{cases}$$

$$= \begin{cases} \lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \notin \mathit{test} \\ \textbf{assume } \lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \in \mathit{test} \text{ and } \psi \Rightarrow x \\ \textbf{assume } \neg\lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \in \mathit{test} \text{ and } \psi \Rightarrow \neg x \end{cases}$$
$$= \lambda_{\mathrm{tr}_{lpof}(\alpha,\psi)}(x)$$
$$= \lambda_{\mathrm{tr}_{lpof}(\alpha,\psi)\fatsemi\mathrm{tr}_{lpof}(\beta,\psi')}(x)$$

The case for $x \in f(\psi)$ is nearly identical. □

Lemma G.5. $\mathrm{tr}(\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta}) = \{\boldsymbol{\alpha}' \fatsemi \boldsymbol{\beta}' \mid \boldsymbol{\alpha}' \in \mathrm{tr}(\boldsymbol{\alpha}), \boldsymbol{\beta}' \in \mathrm{tr}(\boldsymbol{\beta})\}$.

Proof.

$$\begin{aligned}
\mathrm{tr}(\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta}) &= \mathrm{tr}\Big(\sup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \sup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \boldsymbol{\alpha}' \fatsemi \boldsymbol{\beta}'\Big) \\
&= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}' \fatsemi \boldsymbol{\beta}') \\
&= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathsf{fin}}(\{\alpha \fatsemi_f \beta \mid \alpha \in \boldsymbol{\alpha}', \beta \in \boldsymbol{\beta}', f \in \mathrm{copy}_{\alpha,\beta}\})
\end{aligned}$$

Since the above set is an equivalence class, we can fix any $\alpha \in \boldsymbol{\alpha}'$, $\beta \in \boldsymbol{\beta}'$, and $f \in \mathrm{copy}_{\alpha,\beta}$.

$$\begin{aligned}
&= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathsf{fin}}([\alpha \fatsemi_f \beta]) \\
&= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{[\mathrm{tr}_{lpof}(\alpha \fatsemi_f \beta, \psi)] \mid \psi \in \mathrm{br}_{\alpha\fatsemi_f\beta}\} \\
&= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{[\mathrm{tr}_{lpof}(\alpha \fatsemi_f \beta, \psi \wedge \psi')] \mid \psi \in \mathrm{br}_\alpha, \psi' \in \mathrm{br}_{f(\psi)}\} \\
&= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{[\mathrm{tr}_{lpof}(\alpha, \psi) \fatsemi \mathrm{tr}_{lpof}(f(\psi), \psi')] \mid \psi \in \mathrm{br}_\alpha, \psi' \in \mathrm{br}_{f(\psi)}\} \quad \text{(By Lemma G.4)} \\
&= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}') \fatsemi \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\beta}') \\
&= \left(\bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}')\right) \fatsemi \left(\bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\beta}')\right) \\
&= \mathrm{tr}(\boldsymbol{\alpha}) \fatsemi \mathrm{tr}(\boldsymbol{\beta})
\end{aligned}$$

□

Lemma G.6. *If $b \neq \&$, then*

$$\mathrm{tr}_{lpof}(\mathrm{guard}(x, b, \alpha, \beta), \psi \wedge x) = \langle\textbf{assume } b\rangle_x \fatsemi \mathrm{tr}_{lpof}(\alpha, \psi)$$

and

$$\mathrm{tr}_{lpof}(\mathrm{guard}(x, b, \alpha, \beta), \psi \wedge \neg x) = \langle\textbf{assume } \neg b\rangle_x \fatsemi \mathrm{tr}_{lpof}(\beta, \psi)$$

Proof. We prove the equality for the first case. The second is entirely symmetrical. We first show equality of the node sets:

$$N_{\mathrm{tr}_{lpof}(\mathrm{guard}(x,b,\alpha,\beta),\psi\wedge x)} = \{y \in N_{\mathrm{guard}(x,b,\alpha,\beta)} \mid \psi \wedge x \Rightarrow \varphi_{\mathrm{guard}(x,b,\alpha,\beta)}(y), \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(x) \neq \&\}$$

By the construction of guard, x is true in all nodes in the α branch, and the root x has formula true and is not labeled with &.

$$= \{x\} \cup \{y \in N_\alpha \mid \psi \Rightarrow \varphi_\alpha(y), \lambda_\alpha(x) \neq \&\}$$

$$= N_{\langle \textbf{assume } b\rangle_x} \cup N_{\mathrm{tr}(\alpha,\psi)}$$
$$= N_{\langle \textbf{assume } b\rangle_x \fatsemi \mathrm{tr}(\alpha,\psi)}$$

For the order, we have:

$$<_{\mathrm{tr}_{lpof}(\mathrm{guard}(x,b,\alpha,\beta),\psi\wedge x)}$$
$$= <_{\mathrm{guard}(x,b,\alpha,\beta)} \cap \left(N_{\mathrm{tr}_{lpof}(\mathrm{guard}(x,b,\alpha,\beta),\psi\wedge x)} \times N_{\mathrm{tr}_{lpof}(\mathrm{guard}(x,b,\alpha,\beta),\psi\wedge x)}\right)$$
$$= \left(<_\alpha \cup <_\beta \cup (\{x\} \times (N_\alpha \cup N_\beta)\right) \cap \left(N_{\mathrm{tr}_{lpof}(\mathrm{guard}(x,b,\alpha,\beta),\psi\wedge x)} \times N_{\mathrm{tr}_{lpof}(\mathrm{guard}(x,b,\alpha,\beta),\psi\wedge x)}\right)$$

Using what we just showed about the node sets:

$$= \left(<_\alpha \cap \left(N_{\mathrm{tr}(\alpha,\psi)} \times N_{\mathrm{tr}(\alpha,\psi)}\right)\right) \cup \left(\{x\} \times N_{\mathrm{tr}(\alpha,\psi)}\right)$$
$$= <_{\mathrm{tr}(\alpha,\psi)} \cup \left(\{x\} \times N_{\mathrm{tr}(\alpha,\psi)}\right)$$
$$= <_{\langle \textbf{assume } b\rangle_x \fatsemi \mathrm{tr}(\alpha,\psi)}$$

For every y, clearly:

$$\varphi_{\mathrm{tr}_{lpof}(\mathrm{guard}(x,b,\alpha,\beta),\psi\wedge x)}(y) = \varphi_{\langle \textbf{assume } b\rangle_x \fatsemi \mathrm{tr}(\alpha,\psi)}(y) = \mathsf{true}$$

If $y = x$, then we have:

$$\lambda_{\mathrm{tr}_{lpof}(\mathrm{guard}(x,b,\alpha,\beta),\psi\wedge x)}(y)$$
$$= \begin{cases} \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) & \text{if } \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) \notin \mathit{test} \\ \textbf{assume } \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) & \text{if } \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) \in \mathit{test} \text{ and } \psi \wedge x \Rightarrow y \\ \textbf{assume } \neg\lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) & \text{if } \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) \in \mathit{test} \text{ and } \psi \wedge x \Rightarrow \neg y \end{cases}$$
$$= \textbf{assume } \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(x)$$
$$= \textbf{assume } b$$
$$= \lambda_{\langle \textbf{assume } b\rangle_x}(x)$$
$$= \lambda_{\langle \textbf{assume } b\rangle_x \fatsemi \mathrm{tr}(\alpha,\psi)}(x)$$

If not, then $y \in \mathrm{tr}(\alpha,\psi) \subseteq N_\alpha$:

$$\lambda_{\mathrm{tr}_{lpof}(\mathrm{guard}(x,b,\alpha,\beta),\psi\wedge x)}(y)$$
$$= \begin{cases} \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) & \text{if } \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) \notin \mathit{test} \\ \textbf{assume } \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) & \text{if } \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) \in \mathit{test} \text{ and } \psi \wedge x \Rightarrow y \\ \textbf{assume } \neg\lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) & \text{if } \lambda_{\mathrm{guard}(x,b,\alpha,\beta)}(y) \in \mathit{test} \text{ and } \psi \wedge x \Rightarrow \neg y \end{cases}$$
$$= \begin{cases} \lambda_\alpha(y) & \text{if } \lambda_\alpha(y) \notin \mathit{test} \\ \textbf{assume } \lambda_\alpha(y) & \text{if } \lambda_\alpha(y) \in \mathit{test} \text{ and } \psi \Rightarrow y \\ \textbf{assume } \neg\lambda_\alpha(y) & \text{if } \lambda_\alpha(y) \in \mathit{test} \text{ and } \psi \Rightarrow \neg y \end{cases}$$
$$= \lambda_{\mathrm{tr}(\alpha,\psi)}(y)$$
$$= \lambda_{\langle \textbf{assume } b\rangle_x \fatsemi \mathrm{tr}(\alpha,\psi)}(y)$$

□

Lemma G.7. *If $b \neq \&$, then*

$$\mathrm{tr}(\mathrm{guard}(b, \boldsymbol{\alpha}, \boldsymbol{\beta})) = \{\langle \textbf{assume } b\rangle \fatsemi \boldsymbol{\alpha}' \mid \boldsymbol{\alpha}' \in \mathrm{tr}(\boldsymbol{\alpha})\} \cup \{\langle \textbf{assume } \neg b\rangle \fatsemi \boldsymbol{\beta}' \mid \boldsymbol{\beta}' \in \mathrm{tr}(\boldsymbol{\beta})\}$$

Proof.

$$\mathrm{tr}(\mathrm{guard}(b, \boldsymbol{\alpha}, \boldsymbol{\beta}))$$
$$= \mathrm{tr}\left(\sup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \sup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{guard}(b, \boldsymbol{\alpha}', \boldsymbol{\beta}')\right)$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathrm{fin}}(\mathrm{guard}(b, \boldsymbol{\alpha}', \boldsymbol{\beta}'))$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathrm{fin}}(\{\mathrm{guard}(x, b, \alpha, \beta) \mid \alpha \in \boldsymbol{\alpha}', \beta \in \boldsymbol{\beta}', x \in \mathit{nodes}\})$$

Since the set above is an equivalence class, we can fix any x, α, and β and write:

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathrm{fin}}([\mathrm{guard}(x, b, \alpha, \beta)])$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{[\mathrm{tr}_{\mathit{lpof}}(\mathrm{guard}(x, b, \alpha, \beta), \psi)] \mid \psi \in \mathrm{br}_{\mathrm{guard}(x,b,\alpha,\beta)}\}$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{[\mathrm{tr}_{\mathit{lpof}}(\mathrm{guard}(x, b, \alpha, \beta), \psi \wedge x)] \mid \psi \in \mathrm{br}_{\alpha}\} \cup \{[\mathrm{tr}_{\mathit{lpof}}(\mathrm{guard}(x, b, \alpha, \beta), \psi \wedge \neg x)] \mid \psi \in \mathrm{br}_{\beta}\}$$

By Lemma G.6:

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{[\langle \textbf{assume } b \rangle_x \fatsemi \mathrm{tr}_{\mathit{lpof}}(\alpha, \psi)] \mid \psi \in \mathrm{br}_{\alpha}\} \cup \{[\langle \textbf{assume } \neg b \rangle_x \fatsemi \mathrm{tr}_{\mathit{lpof}}(\beta, \psi)] \mid \psi \in \mathrm{br}_{\beta}\}$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{\langle \textbf{assume } b \rangle \fatsemi \boldsymbol{\alpha}'' \mid \boldsymbol{\alpha}'' \in \mathrm{tr}_{\mathrm{fin}}(\boldsymbol{\alpha}')\} \cup \{\langle \textbf{assume } \neg b \rangle \fatsemi \boldsymbol{\beta}'' \mid \boldsymbol{\beta}'' \in \mathrm{tr}_{\mathrm{fin}}(\boldsymbol{\beta}')\}$$

$$= \{\langle \textbf{assume } b \rangle \fatsemi \boldsymbol{\alpha}'' \mid \boldsymbol{\alpha}'' \in \mathrm{tr}(\boldsymbol{\alpha})\} \cup \{\langle \textbf{assume } \neg b \rangle \fatsemi \boldsymbol{\beta}'' \mid \boldsymbol{\beta}'' \in \mathrm{tr}(\boldsymbol{\beta})\} \qquad \square$$

Lemma G.8.

$$\mathrm{tr}(\mathrm{guard}(\&, \boldsymbol{\alpha}, \boldsymbol{\beta})) = \mathrm{tr}(\boldsymbol{\alpha}) \cup \mathrm{tr}(\boldsymbol{\beta})$$

Proof.

$$\mathrm{tr}(\mathrm{guard}(\&, \boldsymbol{\alpha}, \boldsymbol{\beta}))$$

$$= \mathrm{tr}\left(\sup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \sup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{guard}(\&, \boldsymbol{\alpha}', \boldsymbol{\beta}') \right)$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathrm{fin}}(\mathrm{guard}(\&, \boldsymbol{\alpha}', \boldsymbol{\beta}'))$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathrm{fin}}(\{\mathrm{guard}(x, \&, \alpha, \beta) \mid \alpha \in \boldsymbol{\alpha}', \beta \in \boldsymbol{\beta}', x \in \mathit{nodes}\})$$

Since the set above is an equivalence class, we can fix any x, α, and β and write:

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \mathrm{tr}_{\mathrm{fin}}([\mathrm{guard}(x, \&, \alpha, \beta)])$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{[\mathrm{tr}_{\mathit{lpof}}(\mathrm{guard}(x, \&, \alpha, \beta), \psi)] \mid \psi \in \mathrm{br}_{\mathrm{guard}(x,\&,\alpha,\beta)}\}$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{[\mathrm{tr}_{\mathit{lpof}}(\mathrm{guard}(x, \&, \alpha, \beta), \psi \wedge x)] \mid \psi \in \mathrm{br}_{\alpha}\} \cup \{[\mathrm{tr}_{\mathit{lpof}}(\mathrm{guard}(x, \&, \alpha, \beta), \psi \wedge \neg x)] \mid \psi \in \mathrm{br}_{\beta}\}$$

Since $\lambda_{\mathrm{guard}(x,\&,\alpha,\beta)}(x) = \&$, then $\mathrm{tr}_{\mathit{lpof}}$ is going to delete the root node, and just linearize the two branches

$$= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll \boldsymbol{\beta}} \{[\mathrm{tr}_{\mathit{lpof}}(\alpha, \psi)] \mid \psi \in \mathrm{br}_{\alpha}\} \cup \{[\mathrm{tr}_{\mathit{lpof}}(\beta, \psi)] \mid \psi \in \mathrm{br}_{\beta}\}$$

$$= \mathrm{tr}(\boldsymbol{\alpha}) \cup \mathrm{tr}(\boldsymbol{\beta}) \qquad \square$$

Lemma G.9. *For any $\alpha, \beta \in lpof_{\mathsf{fin}}$, $\psi \in \mathsf{br}_\alpha$, and $\psi' \in \mathsf{br}_\beta$:*

$$\mathsf{tr}_{lpof}(\alpha \parallel_x \beta, \psi \wedge \psi') = \mathsf{tr}_{lpof}(\alpha, \psi) \parallel_x \mathsf{tr}_{lpof}(\beta, \psi')$$

Proof. We show the equality for each component. First, for the nodes, we have:

$$\begin{aligned} N_{\mathsf{tr}_{lpof}(\alpha\parallel_x\beta,\psi\wedge\psi')} &= \{y \in N_{\alpha\parallel_x\beta} \mid \psi \wedge \psi' \Rightarrow \varphi_{\alpha\parallel_x\beta}(y)\} \\ &= \{x\} \cup \{y \in \mathsf{nFork}_\alpha \mid \psi \Rightarrow \varphi_\alpha(y)\} \cup \{z \in \mathsf{nFork}_\beta \mid \psi' \Rightarrow \varphi_\beta(z)\} \\ &= \{x\} \cup \mathsf{nFork}_{\mathsf{tr}_{lpof}(\alpha,\psi)} \cup \mathsf{nFork}_{\mathsf{tr}_{lpof}(\beta,\psi')} \\ &= N_{\mathsf{tr}_{lpof}(\alpha,\psi)\parallel_x\mathsf{tr}_{lpof}(\beta,\psi')} \end{aligned}$$

Now, for the orders:

$$\begin{aligned} &<_{\mathsf{tr}_{lpof}(\alpha\parallel_x\beta,\psi\wedge\psi')} \\ &= <_{\alpha\parallel_x\beta} \cap \left(N_{\mathsf{tr}_{lpof}(\alpha\parallel_x\beta,\psi\wedge\psi')} \times N_{\mathsf{tr}_{lpof}(\alpha\parallel_x\beta,\psi\wedge\psi')}\right) \\ &= \left((\{x\} \times (\mathsf{nFork}_\alpha \cup \mathsf{nFork}_\beta)) \cup\ (<_\alpha \cap (\mathsf{nFork}_\alpha \times \mathsf{nFork}_\alpha)) \cup\ (<_\beta \cap (\mathsf{nFork}_\beta \times \mathsf{nFork}_\beta))\right) \\ &\qquad \cap \left(N_{\mathsf{tr}_{lpof}(\alpha\parallel_x\beta,\psi\wedge\psi')} \times N_{\mathsf{tr}_{lpof}(\alpha\parallel_x\beta,\psi\wedge\psi')}\right) \\ &= (\{x\} \times (\mathsf{nFork}_{\mathsf{tr}_{lpof}(\alpha,\psi)} \cup \mathsf{nFork}_{\mathsf{tr}_{lpof}(\beta,\psi')})) \\ &\quad \cup\ (<_\alpha \cap (\mathsf{nFork}_{\mathsf{tr}_{lpof}(\alpha,\psi)} \times \mathsf{nFork}_{\mathsf{tr}_{lpof}(\alpha,\psi)})) \\ &\quad \cup\ (<_\beta \cap (\mathsf{nFork}_{\mathsf{tr}_{lpof}(\beta,\psi')} \times \mathsf{nFork}_{\mathsf{tr}_{lpof}(\beta,\psi')}) \\ &= <_{\mathsf{tr}_{lpof}(\alpha,\psi)\parallel_x\mathsf{tr}_{lpof}(\beta,\psi')} \end{aligned}$$

Now take any y, clearly we have:

$$\varphi_{\mathsf{tr}_{lpof}(\alpha\parallel_x\beta,\psi\wedge\psi')}(y) = \varphi_{\mathsf{tr}_{lpof}(\alpha,\psi)\parallel_x\mathsf{tr}_{lpof}(\beta,\psi')}(y) = \mathsf{true}$$

Now, if $y = x$, then $\lambda_{\alpha\parallel_x\beta}(y) = \mathsf{fork}$, so:

$$\begin{aligned} \lambda_{\mathsf{tr}_{lpof}(\alpha\parallel_x\beta,\psi\wedge\psi')}(y) &= \begin{cases} \lambda_{\alpha\parallel_x\beta}(y) & \text{if } \lambda_{\alpha\parallel_x\beta}(y) \notin \mathit{test} \\ \mathbf{assume}\ \lambda_{\alpha\parallel_x\beta}(y) & \text{if } \lambda_{\alpha\parallel_x\beta}(y) \in \mathit{test} \text{ and } \psi \wedge \psi' \Rightarrow x \\ \mathbf{assume}\ \neg\lambda_{\alpha\parallel_x\beta}(y) & \text{if } \lambda_{\alpha\parallel_x\beta}(y) \in \mathit{test} \text{ and } \psi \wedge \psi' \Rightarrow \neg x \end{cases} \\ &= \lambda_{\alpha\parallel_x\beta}(y) \\ &= \mathsf{fork} \\ &= \lambda_{\mathsf{tr}_{lpof}(\alpha,\psi)\parallel_x\mathsf{tr}_{lpof}(\beta,\psi')}(y) \end{aligned}$$

Now, if $y \in \mathsf{nFork}_\alpha$, then:

$$\begin{aligned} \lambda_{\mathsf{tr}_{lpof}(\alpha\parallel_x\beta,\psi\wedge\psi')}(y) &= \begin{cases} \lambda_{\alpha\parallel_x\beta}(y) & \text{if } \lambda_{\alpha\parallel_x\beta}(y) \notin \mathit{test} \\ \mathbf{assume}\ \lambda_{\alpha\parallel_x\beta}(y) & \text{if } \lambda_{\alpha\parallel_x\beta}(y) \in \mathit{test} \text{ and } \psi \wedge \psi' \Rightarrow x \\ \mathbf{assume}\ \neg\lambda_{\alpha\parallel_x\beta}(y) & \text{if } \lambda_{\alpha\parallel_x\beta}(y) \in \mathit{test} \text{ and } \psi \wedge \psi' \Rightarrow \neg x \end{cases} \\ &= \begin{cases} \lambda_\alpha(y) & \text{if } \lambda_\alpha(y) \notin \mathit{test} \\ \mathbf{assume}\ \lambda_\alpha(y) & \text{if } \lambda_\alpha(y) \in \mathit{test} \text{ and } \psi \Rightarrow x \\ \mathbf{assume}\ \neg\lambda_\alpha(y) & \text{if } \lambda_\alpha(y) \in \mathit{test} \text{ and } \psi \Rightarrow \neg x \end{cases} \\ &= \lambda_{\mathsf{tr}(\alpha,\psi)}(y) \\ &= \lambda_{\mathsf{tr}_{lpof}(\alpha,\psi)\parallel_x\mathsf{tr}_{lpof}(\beta,\psi')}(y) \end{aligned}$$

The case where $y \in \mathsf{nBot}(\beta)$ is nearly identical. □

Lemma G.10. $\mathsf{tr}(\boldsymbol{\alpha} \parallel \boldsymbol{\beta}) = \{\boldsymbol{\alpha'} \parallel \boldsymbol{\beta'} \mid \boldsymbol{\alpha'} \in \mathsf{tr}(\boldsymbol{\alpha}), \boldsymbol{\beta'} \in \mathsf{tr}(\boldsymbol{\beta})\}$.

Proof.

$$\mathrm{tr}(\boldsymbol{\alpha} \parallel \boldsymbol{\beta}) = \mathrm{tr}\left(\sup_{\boldsymbol{\gamma} \in \lfloor \boldsymbol{\alpha} \parallel \boldsymbol{\beta} \rfloor_{\mathsf{fin}}} \boldsymbol{\gamma}\right)$$

$$= \mathrm{tr}(\langle \bot \rangle) \cup \mathrm{tr}\left(\sup_{\boldsymbol{\gamma} \in \lfloor \boldsymbol{\alpha} \parallel \boldsymbol{\beta} \rfloor_{\mathsf{fin}} \setminus \{\bot_{pom}\}} \boldsymbol{\gamma}\right)$$

By Equation (7).

$$= \emptyset \cup \mathrm{tr}\left(\sup_{\boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}} \sup_{\boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}} \boldsymbol{\alpha}' \parallel \boldsymbol{\beta}'\right)$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}} \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}' \parallel \boldsymbol{\beta}')$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}} \mathrm{tr}_{\mathsf{fin}}(\{\alpha \parallel_x \beta \mid \alpha \in \boldsymbol{\alpha}', \beta \in \boldsymbol{\beta}', x \in \mathit{nodes}\})$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}} \mathrm{tr}_{\mathsf{fin}}([\alpha \parallel_x \beta])$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}} \{[\mathrm{tr}_{lpof}(\alpha \parallel_x \beta, \psi)] \mid \psi \in \mathrm{br}_{\alpha \parallel_x \beta}\}$$

By Lemma G.9.

$$= \bigcup_{\boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}} \{[\mathrm{tr}_{lpof}(\alpha \parallel_x \beta, \psi \wedge \psi')] \mid \psi \in \mathrm{br}_\alpha, \psi' \in \mathrm{br}_\beta\}$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}} \{[\{\alpha' \parallel_x \beta' \mid \alpha' \in \mathrm{tr}_{lpof}(\alpha, \psi), \beta' \in \mathrm{tr}_{lpof}(\beta, \psi')\}] \mid \psi \in \mathrm{br}_\alpha, \psi' \in \mathrm{br}_\beta\}$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}} \{\boldsymbol{\alpha}'' \parallel \boldsymbol{\beta}'' \mid \boldsymbol{\alpha}'' \in \{[\mathrm{tr}_{lpof}(\alpha, \psi)] \mid \psi \in \mathrm{br}_\alpha\},\ \boldsymbol{\beta}'' \in \{[\mathrm{tr}_{lpof}(\beta, \psi')] \mid \psi' \in \mathrm{br}_\beta\}\}$$

$$= \bigcup_{\boldsymbol{\alpha}' \ll_1 \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta}' \ll_1 \boldsymbol{\beta}} \{\boldsymbol{\alpha}'' \parallel \boldsymbol{\beta}'' \mid \boldsymbol{\alpha}'' \in \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}'),\ \boldsymbol{\beta}'' \in \mathrm{tr}_{\mathsf{fin}}(\boldsymbol{\beta}')\}$$

By Equation (6).

$$= \{\boldsymbol{\alpha}'' \parallel \boldsymbol{\beta}'' \mid \boldsymbol{\alpha}'' \in \mathrm{tr}(\boldsymbol{\alpha}),\ \boldsymbol{\beta}'' \in \mathrm{tr}(\boldsymbol{\beta})\}$$ □

Lemma G.11. *If* $\mathrm{tr}(\llbracket C \rrbracket) = \llbracket C \rrbracket_{\mathsf{PL}}$, *for all* $n \in \mathbb{N}$:

$$\mathrm{tr}(\Phi^n_{\langle C,b \rangle}(\bot_{pom})) = \Xi^n_{\langle C,b \rangle}(\emptyset)$$

where above we use $\bot_{pom} = \langle \bot \rangle$ *to disambiguate.*

Proof. The proof is by induction on n. If $n = 0$, then we have:

$$\mathrm{tr}\left(\Phi^0_{\langle C,b \rangle}(\bot_{pom})\right) = \mathrm{tr}\left(\bot_{pom}\right) = \{[\mathrm{tr}_{lpof}(\langle \bot \rangle_x, \psi)] \mid \psi \in \mathrm{br}_{\langle \bot \rangle_x}\} = \emptyset = \Xi^0_{\langle C,b \rangle}(\emptyset)$$

Where the second to last step follows from the fact that $\mathrm{br}_{\langle \bot \rangle_x} = \emptyset$. Now, suppose the claim holds for n, we have:

$$\mathrm{tr}(\Phi^{n+1}_{\langle C,b \rangle}(\bot_{pom}))$$
$$= \mathrm{tr}\left(\Phi_{\langle C,b \rangle}\left(\Phi^n_{\langle C,b \rangle}(\bot_{pom})\right)\right)$$

$$= \mathrm{tr}\left(\mathrm{guard}\left(b, [\![C]\!] \fatsemi \Phi^n_{\langle C,b\rangle}(\emptyset), [\![\mathbf{skip}]\!]\right)\right)$$

By Lemma G.7.

$$= \left\{\langle \mathbf{assume}\ b\rangle \fatsemi \boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in \mathrm{tr}\left([\![C]\!] \fatsemi \Phi^n_{\langle C,b\rangle}(\emptyset)\right)\right\} \cup \{\langle \mathbf{assume}\ b\rangle \fatsemi \boldsymbol{\beta} \mid \boldsymbol{\beta} \in \mathrm{tr}([\![\mathbf{skip}]\!])\}$$

By Lemmas G.3 and G.5.

$$= \left\{\langle \mathbf{assume}\ b\rangle \fatsemi \boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \mid \boldsymbol{\alpha} \in \mathrm{tr}([\![C]\!]), \boldsymbol{\beta} \in \mathrm{tr}\left(\Phi^n_{\langle C,b\rangle}(\emptyset)\right)\right\} \cup \{\langle \mathbf{assume}\ b\rangle \fatsemi [\![\mathbf{skip}]\!]\}$$

By assumption, and the induction hypothesis:

$$= \left\{\langle \mathbf{assume}\ b\rangle \fatsemi \boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \mid \boldsymbol{\alpha} \in [\![C]\!]_{\mathsf{PL}}, \boldsymbol{\beta} \in \Xi^n_{\langle C,b\rangle}(\emptyset)\right\} \cup \{\langle \mathbf{assume}\ b\rangle \fatsemi [\![\mathbf{skip}]\!]\}$$
$$= \Xi_{\langle C,b\rangle}\left(\Xi^n_{\langle C,b\rangle}(\emptyset)\right)$$
$$= \Xi^{n+1}_{\langle C,b\rangle}(\emptyset)$$

□

G.3 Translations and Program Semantics

Lemma G.12. *For any program $C \in cmd$, $[\![C]\!]_{\mathsf{PL}} = \mathrm{tr}([\![C]\!])$.*

Proof. The proof is by induction on the structure of the program.

- $C = \mathbf{skip}$ or $C \in act$. Follows immediately from Lemma G.3.
- $C = C_1; C_2$.

$$\begin{aligned}
[\![C_1; C_2]\!]_{\mathsf{PL}} &= \{\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \mid \boldsymbol{\alpha} \in [\![C_1]\!]_{\mathsf{PL}}, \boldsymbol{\beta} \in [\![C_2]\!]_{\mathsf{PL}}\} \\
&= \{\boldsymbol{\alpha} \fatsemi \boldsymbol{\beta} \mid \boldsymbol{\alpha} \in \mathrm{tr}([\![C_1]\!]), \boldsymbol{\beta} \in \mathrm{tr}([\![C_2]\!])\} && \text{(By the induction hypothesis.)} \\
&= \mathrm{tr}([\![C_1]\!] \fatsemi [\![C_2]\!]) && \text{(By Lemma G.5.)} \\
&= \mathrm{tr}([\![C_1; C_2]\!])
\end{aligned}$$

- $C = C_1 \,\&\, C_2$.

$$\begin{aligned}
[\![C_1 \,\&\, C_2]\!]_{\mathsf{PL}} &= [\![C_1]\!]_{\mathsf{PL}} \cup [\![C_2]\!]_{\mathsf{PL}} \\
&= \mathrm{tr}([\![C_1]\!]) \cup \mathrm{tr}([\![C_2]\!]) && \text{(By the induction hypothesis.)} \\
&= \mathrm{tr}(\mathrm{guard}(\&, [\![C_1]\!], [\![C_2]\!])) && \text{(By Lemma G.8.)} \\
&= \mathrm{tr}([\![C_1 \,\&\, C_2]\!])
\end{aligned}$$

- $C = C_1 \mid C_2$.

$$\begin{aligned}
[\![C_1 \mid C_2]\!]_{\mathsf{PL}} &= \{\boldsymbol{\alpha} \parallel \boldsymbol{\beta} \mid \boldsymbol{\alpha} \in [\![C_1]\!]_{\mathsf{PL}}, \boldsymbol{\beta} \in [\![C_2]\!]_{\mathsf{PL}}\} \\
&= \{\boldsymbol{\alpha} \parallel \boldsymbol{\beta} \mid \boldsymbol{\alpha} \in \mathrm{tr}([\![C_1]\!]), \boldsymbol{\beta} \in \mathrm{tr}([\![C_2]\!])\} && \text{(By the induction hypothesis.)} \\
&= \mathrm{tr}([\![C_1]\!] \parallel [\![C_2]\!]) && \text{(By Lemma G.10.)} \\
&= \mathrm{tr}([\![C_1 \mid C_2]\!])
\end{aligned}$$

- $C = \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2$.

$$\begin{aligned}
&[\![\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2]\!]_{\mathsf{PL}} \\
&= \{\langle \mathbf{assume}\ b\rangle \fatsemi \boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in [\![C_1]\!]_{\mathsf{PL}}\} \cup \{\langle \mathbf{assume}\ \neg b\rangle \fatsemi \boldsymbol{\beta} \mid \boldsymbol{\beta} \in [\![C_2]\!]_{\mathsf{PL}}\}
\end{aligned}$$

By the induction hypothesis:

$$= \{\langle \mathbf{assume}\ b\rangle \fatsemi \boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in \mathrm{tr}([\![C_1]\!])\} \cup \{\langle \mathbf{assume}\ \neg b\rangle \fatsemi \boldsymbol{\beta} \mid \boldsymbol{\beta} \in \mathrm{tr}([\![C_2]\!])\}$$

By Lemma G.7.

$$= \mathsf{tr}(\mathsf{guard}(b, [\![C_1]\!], [\![C_2]\!]))$$
$$= \mathsf{tr}([\![\textbf{if } b \textbf{ then } C_1 \textbf{ else } C_2]\!])$$

- $C = \textbf{while } b \textbf{ do } C'$.

$$\begin{aligned}
&[\![\textbf{while } b \textbf{ do } C']\!]_{\mathsf{PL}} \\
&= \mathsf{lfp}\left(\Xi_{\langle C', b\rangle}\right) \\
&= \bigcup_{n\in\mathbb{N}} \Xi^n_{\langle C', b\rangle}(\emptyset) && \text{(By the Kleene fixed point theorem.)} \\
&= \bigcup_{n\in\mathbb{N}} \mathsf{tr}\left(\Phi^n_{\langle C', b\rangle}(\bot_{pom})\right) && \text{(By Lemma G.11, and the induction hypothesis.)} \\
&= \mathsf{tr}\left(\sup_{n\in\mathbb{N}} \Phi^n_{\langle C', b\rangle}(\bot_{pom})\right) && \text{(By Scott continuity of tr.)} \\
&= \mathsf{tr}\left(\mathsf{lfp}\left(\Phi^n_{\langle C', b\rangle}\right)\right) && \text{(By the Kleene fixed point theorem.)} \\
&= \mathsf{tr}\left([\![\textbf{while } b \textbf{ do } C']\!]\right) && \square
\end{aligned}$$

G.4 Translations and Linearization

Lemma G.13. *For any binary branching $\alpha \in lpof_{\mathsf{fin}}(label)$, $S \subseteq \mathsf{nBot}_\alpha$, and $\psi \in \mathsf{valid}_\alpha(S)$, then:*

$$\bigcup_{\psi' \in \mathsf{br}_\alpha : \psi' \Rightarrow \psi} \mathsf{next}(\mathsf{tr}_{lpof}(\alpha, \psi'), \mathsf{true}, S) = \mathsf{next}(\alpha, \psi, S) \setminus \{x \in N_\alpha \mid \varphi_\alpha(x) \Rightarrow \mathsf{stuck}_\alpha \vee \lambda_\alpha(x) = \&\}$$

Proof. We show the inclusion in both directions.

Take any $x \in \bigcup_{\psi' \in \mathsf{br}_\alpha : \psi' \Rightarrow \psi} \mathsf{next}(\mathsf{tr}_{lpof}(\alpha, \psi'), \mathsf{true}, S)$, so there is a $\psi' \in \mathsf{br}_\alpha$ such that $\psi' \Rightarrow \psi$ and $x \in N_{\mathsf{tr}_{lpof}(\alpha,\psi')} \setminus S$ such that $x{\downarrow}_{\mathsf{tr}_{lpof}(\alpha,\psi')} \subseteq S$. From $x \in N_{\mathsf{tr}_{lpof}(\alpha,\psi')} \setminus S$, we get that $x \in N_\alpha \setminus S$ and $\psi' \Rightarrow \varphi_\alpha(x)$ and $\lambda_\alpha(x) \neq \&$. We now show that $x{\downarrow}_\alpha \subseteq S$. Take any $y <_\alpha x$, so we know that $\psi' \Rightarrow \varphi_\alpha(x) \Rightarrow \varphi_\alpha(y)$, which means that $y \in N_{\mathsf{tr}_{lpof}(\alpha,\psi')}$, and therefore $y <_{\mathsf{tr}_{lpof}(\alpha,\psi')} x$, so it must be that $y \in S$. From $\psi' \Rightarrow \psi$ and $\psi' \Rightarrow \varphi_\alpha(x)$, we know that $\varphi_\alpha(x) \wedge \psi$ is satisfiable. Since $\psi \in \mathsf{valid}_\alpha(S)$, then ψ must assign a truth value to every test in S, and since we just showed that $x{\downarrow}_\alpha \subseteq S$, then $\mathsf{vars}(\varphi_\alpha(x)) \subseteq S$, and so we must have that $\psi' \Rightarrow \varphi_\alpha(x)$. Therefore, we have established all the conditions of $x \in \mathsf{next}(\alpha, \psi, S)$. Clearly also $\varphi_\alpha(x) \not\Rightarrow \mathsf{stuck}_\alpha$ since $\psi' \Rightarrow \varphi_\alpha(x)$ and $\psi' \Rightarrow \neg\mathsf{stuck}_\alpha$.

Now take any $x \in \mathsf{next}(\alpha, \psi, S) \setminus \{x \in N_\alpha \mid \varphi_\alpha(x) \Rightarrow \mathsf{stuck}_\alpha \vee \lambda_\alpha(x) = \&\}$, so $x \in N_\alpha \setminus S$, $\psi \Rightarrow \varphi_\alpha(x)$, $x{\downarrow}_\alpha \subseteq S$, $\varphi_\alpha(x) \not\Rightarrow \mathsf{stuck}_\alpha$, and $\lambda_\alpha(x) \neq \&$. Since $\varphi_\alpha(x) \not\Rightarrow \mathsf{stuck}_\alpha$, then $x \in \mathsf{ext}_\alpha$, and so there is clearly a $\psi' \in \mathsf{br}_\alpha$ such that $\psi' \Rightarrow \varphi_\alpha(x)$. This means that $x \in N_{\mathsf{tr}_{lpof}(\alpha,\psi')} \setminus S$. Since by definition $\varphi_{\mathsf{tr}_{lpof}(\alpha,\psi')}(x) = \mathsf{true}$, then clearly $\mathsf{true} \Rightarrow \varphi_{\mathsf{tr}_{lpof}(\alpha,\psi')}(x)$. Finally, we will show that $x{\downarrow}_{\mathsf{tr}_{lpof}(\alpha,\psi')} \subseteq S$. Take any $y <_{\mathsf{tr}_{lpof}(\alpha,\psi')} x$, this means that $y <_\alpha x$, so clearly $y \in S$. Therefore, we have that $x \in \mathsf{next}(\mathsf{tr}_{lpof}(\alpha, \psi'), \mathsf{true}, S) \subseteq \bigcup_{\psi' \in \mathsf{br}_\alpha : \psi' \Rightarrow \psi} \mathsf{next}(\mathsf{tr}_{lpof}(\alpha, \psi'), \mathsf{true}, S)$. □

Lemma G.14. *If $S \subseteq \mathsf{nBot}_\alpha$, $\psi \in \mathsf{valid}_\alpha(S)$ and $\psi \Rightarrow \mathsf{stuck}_\alpha$, then $\mathcal{L}_{lpof}(\alpha, \psi, S) = \emptyset$.*

Proof. The proof is by induction on the size of the set $\mathsf{next}^*(\alpha, \psi, S)$. If $\mathsf{next}^*(\alpha, \psi, S) = \emptyset$, then by Lemma F.4 we know that $\psi \in \mathsf{br}_\alpha$, but this is impossible since $\psi \Rightarrow \mathsf{stuck}_\alpha$. Therefore, it cannot be the case that $\mathsf{next}^*(\alpha, \psi, S) = \emptyset$.

Now suppose that $\mathsf{next}^*(\alpha,\psi,S) \neq \emptyset$. By Lemma F.5, we know that $\mathsf{next}(\alpha,\psi,S) \neq \emptyset$. So, we have:

$$\mathcal{L}_{lpof}(\alpha,\psi,S) = \bigcup_{x\in\mathsf{next}(\alpha,\psi,S)} \begin{cases} \mathcal{L}_{lpof}(\alpha,\psi,S\cup\{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}(s)) & \text{if } \lambda_\alpha(x)\in act \\ \mathcal{L}_{test}(\alpha,\psi,S,x)(s)^\dagger([\![\lambda_\alpha(x)]\!]_{test}(s)) & \text{if } \lambda_\alpha(x)\in test \\ \emptyset & \text{if } \lambda_\alpha(x)=\bot \\ \mathcal{L}_{lpof}(\alpha,\psi,S\cup\{x\})(s) & \text{if } \lambda_\alpha(x)=\mathsf{fork} \end{cases}$$

By the induction hypothesis in the first, second, and fourth cases, we get:

$$\begin{aligned} &= \bigcup_{x\in\mathsf{next}(\alpha,\psi,S)} \begin{cases} \emptyset & \text{if } \lambda_\alpha(x)\in act \\ \emptyset & \text{if } \lambda_\alpha(x)\in test \\ \emptyset & \text{if } \lambda_\alpha(x)=\bot \\ \emptyset & \text{if } \lambda_\alpha(x)=\mathsf{fork} \end{cases} \\ &= \bigcup_{x\in\mathsf{next}(\alpha,\psi,S)} \emptyset = \emptyset \end{aligned}$$

□

Lemma G.15. *For any binary branching $\alpha \in lpof_{\mathsf{fin}}(label)$, $S \subseteq \mathsf{nBot}_\alpha$, and $\psi \in \mathsf{valid}_\alpha(S)$:*

$$\bigcup_{\psi'\in\mathsf{br}_\alpha:\psi'\Rightarrow\psi} \mathcal{L}_{lpof}(\mathsf{tr}_{lpof}(\alpha,\psi'),\mathsf{true},S)(s) = \mathcal{L}_{lpof}(\alpha,\psi,S)(s)$$

Proof. The proof is by induction on the size of the set $\mathsf{next}^*(\alpha,\psi,S)$. If $\mathsf{next}^*(\alpha,\psi,S)=\emptyset$, then clearly $\mathsf{next}(\alpha,\psi,S)=\emptyset$ and by Lemma F.4, we know that $\psi\in\mathsf{br}_\alpha$. That means that the union on the left side is only over a single term, generated from ψ. Now, $\mathsf{tr}_{lpof}(\alpha,\psi)$ contains all those nodes x such that $\psi \Rightarrow \varphi_\alpha(x)$ and $\lambda_\alpha(x)\neq\&$, but we know from $\mathsf{next}^*(\alpha,\psi,S)=\emptyset$ that all such nodes are already in S, therefore by Lemma G.13, $\mathsf{next}(\mathsf{tr}_{lpof}(\alpha,\psi),\mathsf{true},S)=\emptyset$ too. This gives us

$$\bigcup_{\psi'\in\mathsf{br}_\alpha:\psi'\Rightarrow\psi} \mathcal{L}_{lpof}(\mathsf{tr}_{lpof}(\alpha,\psi'),\mathsf{true},S)(s) = \mathcal{L}_{lpof}(\mathsf{tr}_{lpof}(\alpha,\psi),\mathsf{true},S)(s) = \{s\} = \mathcal{L}_{lpof}(\alpha,\psi,S)(s)$$

Now, assume that $\mathsf{next}^*(\alpha,\psi,S)\neq\emptyset$, so by Lemma F.5, then $\mathsf{next}(\alpha,\psi,S)\neq\emptyset$.

Suppose that $\varphi_\alpha(x) \Rightarrow \mathsf{stuck}_\alpha$ for some $x\in\mathsf{next}(\alpha,\psi,S)$. This means that $\psi\Rightarrow\varphi_\alpha(x)\Rightarrow\mathsf{stuck}_\alpha$, so clearly there can be no $\psi\in\mathsf{br}_\alpha$ such that $\psi'\Rightarrow\psi$, so $\bigcup_{\psi'\in\mathsf{br}_\alpha:\psi'\Rightarrow\psi}\mathcal{L}_{lpof}(\mathsf{tr}_{lpof}(\alpha,\psi'),\mathsf{true},S)(s)=\emptyset$. In addition, by Lemma G.14, $\mathcal{L}_{lpof}(\alpha,\psi,S)(s)=\emptyset$ too.

So, we are left with the case where $\varphi_\alpha(x) \not\Rightarrow \mathsf{stuck}_\alpha$ for all $x\in\mathsf{next}(\alpha,\psi,S)$. Let $\alpha_{\psi'}=\mathsf{tr}_{lpof}(\alpha,\psi')$. We now have:

$$\begin{aligned} &\bigcup_{\psi'\in\mathsf{br}_\alpha:\psi'\Rightarrow\psi} \mathcal{L}_{lpof}(\alpha_{\psi'},\mathsf{true},S) \\ &= \bigcup_{\psi'\in\mathsf{br}_\alpha:\psi'\Rightarrow\psi}\ \bigcup_{x\in\mathsf{next}(\alpha_{\psi'},\mathsf{true},S)} \begin{cases} \mathcal{L}_{lpof}(\alpha_{\psi'},\mathsf{true},S\cup\{x\})^\dagger(\left[\!\left[\lambda_{\alpha_{\psi'}}(x)\right]\!\right]_{act}(s)) & \text{if } \lambda_{\alpha_{\psi'}}(x)\in act_{\mathsf{PL}} \\ \mathcal{L}_{test}(\alpha_{\psi'},\mathsf{true},S)(s)^\dagger(\left[\!\left[\lambda_{\alpha_{\psi'}}(x)\right]\!\right]_{test}(s)) & \text{if } \lambda_{\alpha_{\psi'}}(x)\in test_{\mathsf{PL}} \\ \emptyset & \text{if } \lambda_{\alpha_{\psi'}}(x)=\bot \\ \mathcal{L}_{lpof}(\alpha_{\psi'},\mathsf{true},S\cup\{x\})(s) & \text{if } \lambda_{\alpha_{\psi'}}(x)=\mathsf{fork} \end{cases} \end{aligned}$$

Since $\alpha_{\psi'}$ has no test or $\bot$ nodes, we can remove those two cases. In addition, we expand the case for actions into cases for whether or not the action is assume.

$$\begin{aligned} &= \bigcup_{\psi'\in\mathsf{br}_\alpha:\psi'\Rightarrow\psi}\ \bigcup_{x\in\mathsf{next}(\alpha_{\psi'},\mathsf{true},S)} \begin{cases} \mathcal{L}_{lpof}(\alpha_{\psi'},\mathsf{true},S\cup\{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}(s)) & \text{if } \lambda_\alpha(x)\in act \\ \mathcal{L}_{lpof}(\alpha_{\psi'},\mathsf{true},S\cup\{x\})^\dagger([\![\mathtt{assume}\ \lambda_\alpha(x)]\!]_{act}(s)) & \text{if } \lambda_\alpha(x)\in test \text{ and } \psi'\Rightarrow x \\ \mathcal{L}_{lpof}(\alpha_{\psi'},\mathsf{true},S\cup\{x\})^\dagger([\![\mathtt{assume}\ \neg\lambda_\alpha(x)]\!]_{act}(s)) & \text{if } \lambda_\alpha(x)\in test \text{ and } \psi'\Rightarrow \neg x \\ \mathcal{L}_{lpof}(\alpha_{\psi'},\mathsf{true},S\cup\{x\})(s) & \text{if } \lambda_\alpha(x)=\mathsf{fork} \end{cases} \end{aligned}$$

By Lemma G.13, and the fact that $\psi \Rightarrow \varphi_\alpha(x)$ for all $x \in \mathsf{next}(\alpha, \psi, S)$, we can rearrange the unions and move the inner one inside of the conditional. In the case there $\mathsf{next}(\alpha, \psi, S)$ contains a & node, evaluating $[\![\textbf{assume}\ \&]\!]\,(s)$ and $[\![\textbf{assume}\ \neg\&]\!]\,(s)$ are both no-ops, so the do not affect the outcome of linearization.

$$= \bigcup_{x \in \mathsf{next}(\alpha,\psi,S)} \begin{cases} \bigcup_{\psi' \in \mathsf{br}_\alpha : \psi' \Rightarrow \psi} \mathcal{L}_{lpof}(\alpha_{\psi'}, \mathsf{true}, S \cup \{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}\,(s)) & \text{if } \lambda_\alpha(x) \in act \\ \bigcup_{\psi' \in \mathsf{br}_\alpha : \psi' \Rightarrow \psi \wedge x} \mathcal{L}_{lpof}(\alpha_{\psi'}, \mathsf{true}, S \cup \{x\})^\dagger([\![\textbf{assume}\ \lambda_\alpha(x)]\!]_{act}\,(s)) & \text{if } \lambda_\alpha(x) \in test \\ \quad \cup \bigcup_{\psi' \in \mathsf{br}_\alpha : \psi' \Rightarrow \psi \wedge \neg x} \mathcal{L}_{lpof}(\alpha_{\psi'}, \mathsf{true}, S \cup \{x\})^\dagger([\![\textbf{assume}\ \neg\lambda_\alpha(x)]\!]_{act}\,(s)) & \\ \bigcup_{\psi' \in \mathsf{br}_\alpha : \psi' \Rightarrow \psi} \mathcal{L}_{lpof}(\alpha_{\psi'}, \mathsf{true}, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \mathsf{fork} \end{cases}$$

By the induction hypothesis.

$$= \bigcup_{x \in \mathsf{next}(\alpha,\psi,S)} \begin{cases} \mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}\,(s)) & \text{if } \lambda_\alpha(x) \in act \\ \mathcal{L}_{lpof}(\alpha, \psi \wedge x, S \cup \{x\})^\dagger([\![\textbf{assume}\ \lambda_\alpha(x)]\!]_{act}\,(s)) & \text{if } \lambda_\alpha(x) \in test \\ \quad \cup \mathcal{L}_{lpof}(\alpha, \psi \wedge \neg x, S \cup \{x\})^\dagger([\![\textbf{assume}\ \neg\lambda_\alpha(x)]\!]_{act}\,(s)) & \\ \mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \mathsf{fork} \end{cases}$$

$$= \bigcup_{x \in \mathsf{next}(\alpha,\psi,S)} \begin{cases} \mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})^\dagger([\![\lambda_\alpha(x)]\!]_{act}\,(s)) & \text{if } \lambda_\alpha(x) \in act \\ \mathcal{L}_{test}(\alpha, \psi, S, x)(s)^\dagger([\![\lambda_\alpha(x)]\!]_{test}\,(s)) & \text{if } \lambda_\alpha(x) \in test \\ \mathcal{L}_{lpof}(\alpha, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \mathsf{fork} \end{cases}$$

$$= \mathcal{L}_{lpof}(\alpha, \psi, S)$$ □

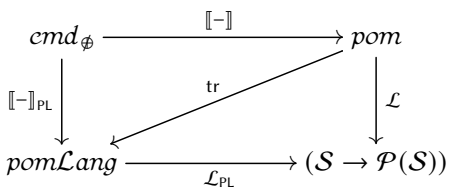
Lemma G.16. *For any $\boldsymbol{\alpha} \in pom$ and $s \in S$, $\mathcal{L}(\boldsymbol{\alpha})(s) = \mathcal{L}_{\mathsf{PL}}(\mathsf{tr}(\boldsymbol{\alpha}))(s)$.*

Proof.

$$\begin{aligned} \mathcal{L}(\boldsymbol{\alpha})(s) &= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\alpha}')(s) \\ &= \bigcup_{[\alpha] \ll \boldsymbol{\alpha}} \mathcal{L}_{lpof}(\alpha, \mathsf{true}, \emptyset)(s) \\ &= \bigcup_{[\alpha] \ll \boldsymbol{\alpha}} \bigcup_{\psi \in \mathsf{br}_\alpha} \mathcal{L}_{lpof}(\mathsf{tr}_{lpof}(\alpha, \psi), \mathsf{true}, \emptyset)(s) && \text{By Lemma G.15} \\ &= \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \bigcup_{\boldsymbol{\beta} \in \mathsf{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}')} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\beta})(s) \\ &= \bigcup_{\boldsymbol{\beta} \in \bigcup_{\boldsymbol{\alpha}' \ll \boldsymbol{\alpha}} \mathsf{tr}_{\mathsf{fin}}(\boldsymbol{\alpha}')} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\beta})(s) \\ &= \bigcup_{\boldsymbol{\beta} \in \mathsf{tr}(\boldsymbol{\alpha})} \mathcal{L}_{\mathsf{fin}}(\boldsymbol{\beta})(s) \\ &= \mathcal{L}_{\mathsf{PL}}(\mathsf{tr}(\boldsymbol{\alpha})) \end{aligned}$$ □

G.5 Main Theorem

Theorem 8.2 (Nondeterministic Adequacy). *The following diagram commutes:*



Proof. It suffices to show that $[\![-]\!]_{\mathsf{PL}} = \mathsf{tr} \circ [\![-]\!]$ and $\mathcal{L}_{\mathsf{PL}} \circ \mathsf{tr} = \mathcal{L}$, which follows immediately from Lemmas G.12 and G.16. We then have:

$$\mathcal{L}_{\mathsf{PL}} \circ [\![-]\!]_{\mathsf{PL}} = \mathcal{L}_{\mathsf{PL}} \circ (\mathsf{tr} \circ [\![-]\!]) = (\mathcal{L}_{\mathsf{PL}} \circ \mathsf{tr}) \circ [\![-]\!] = \mathcal{L} \circ [\![-]\!]$$

□