



Denotational Semantics for Probabilistic and Concurrent Programs

Noam Zilberstein  

Cornell University

Daniele Gorla  

“Sapienza” Università di Roma

Alexandra Silva  

Cornell University and University College London

Abstract

We develop a 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, their combination has not been thoroughly explored and presents unique challenges. 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 model has good domain theoretic properties, important for semantics of unbounded loops. We also prove two adequacy theorems, showing that the model subsumes typical powerdomain semantics for concurrency and convex powerdomain semantics for probabilistic nondeterminism.

2012 ACM Subject Classification Theory of computation → Parallel computing models; Theory of computation → Denotational semantics; Theory of computation → Probabilistic computation

Keywords and phrases Denotational Semantics, Pomsets, Concurrency, Convex Powerset

Digital Object Identifier [10.4230/LIPIcs.CONCUR.2025.39](https://doi.org/10.4230/LIPIcs.CONCUR.2025.39)

Related Version *Full Version:* <https://doi.org/10.48550/arXiv.2503.02768>

Funding This work was partially supported ARIA’s Safeguarded AI programme.

1 Introduction

From simple imperative languages, to functional ones, to probabilistic and quantum paradigms, the development of programming languages is often accompanied by a study of mathematical objects capturing the semantics of syntactic constructs. Program semantics underpins many static analysis and verification techniques, and also how a language can be extended.

Semantics can be presented in different styles—denotational, operational, and axiomatic—and each approach offers different insights. Denotational semantics is often chosen for expressivity and extensibility. For example, adding recursion raises questions about the underlying *domain*, motivating the development of domain theory to provide mathematical representations of iterated computations [53, 54]. Other constructs necessitate more involved domains; two prime examples are probabilistic and concurrent programs.

Probabilistic programs have been widely studied—going back to the seminal works of Kozen [31] and McIver and Morgan [37]—and their semantic domains require extra algebraic structure to capture the *distribution* of outcomes generated by operations such as sampling and coin flips. This added convex structure brings additional complexity to questions of language extensions and program analysis, but the last decade has brought success stories on taming the complexity through program logics and predicate transformer calculi [5, 10, 37, 42, 71].



© Noam Zilberstein, Daniele Gorla, and Alexandra Silva;
licensed under Creative Commons License CC-BY 4.0

36th International Conference on Concurrency Theory (CONCUR 2025).

Editors: Patricia Bouyer and Jaco van de Pol; Article No. 39; pp. 39:1–39:65

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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. There have been many approaches to concurrent semantics, some arising from process algebra research and some from more practical hardware considerations (*e.g.*, memory models). In terms of denotational semantics, *pomsets* [8, 17, 18, 48] provide a prominent model, expressing causality between actions and parallel branching as a partial order.

Semantics of concurrent and probabilistic programs, separately, are subtle and complex. Unsurprisingly, their combination introduces additional challenges. In this paper, we develop a denotational model for programs that mix probabilistic and concurrent constructs. We work with a simple concurrent imperative programming language, shown below.

$$cmd \ni C ::= \text{skip} \mid C_1; C_2 \mid C_1 \mid C_2 \mid \text{if } b \text{ then } C_1 \text{ else } C_2 \mid \text{while } b \text{ do } C \mid a \quad (1)$$

Here, the basic actions $a \in act$ can perform probabilistic operations such as random sampling. This is not only interesting from a theoretical point of view; for decades, randomization has been used to enhance the capabilities of concurrent programs [19, 44, 50, 51]. For example, Dijkstra’s famous *Dining Philosophers Problem*—a distributed synchronization scheme—has no deterministic solution [11], but it has a simple randomized one [35].

Despite the importance of mixing randomization and concurrency, semantic models for probabilistic concurrent programs with unbounded looping are not yet well understood. Such models are necessary to analyze behaviors of many concurrent randomized protocols—including the Dining Philosophers—which avoid deadlock with probability 1, but admit an infinite trace whose probability converges to 0. In this paper, we develop such a model, drawing insights from prior work on concurrent and probabilistic semantics, but solving challenges unique to their combination. In a nutshell, the contributions of this paper are:

1. We introduce *pomsets with formulae*, and prove that they have well behaved domain-theoretic properties (Section 3), including an *extension lemma* [38], for extending monotone operations on finite pomsets to continuous operations on infinite ones.
2. We define guarded, sequential, and parallel composition operations on pomsets with formulae (Section 4) and use those operations to define the denotational semantics for the concurrent imperative language (1), with uninterpreted actions (Section 5.1).
3. We define a linearization operation (via the extension lemma), parametric on the computational domain, to convert the pomset semantics into a state transformer (Section 5.2).
4. We prove two adequacy theorems showing that our model captures established models for probabilistic nondeterminism (Theorem 5.3) and pure concurrency (Theorem 6.1).

We discuss related work in Section 7; omitted proofs and details are provided in [70].

2 An Overview of Probabilistic Concurrency Semantics

In this section, we introduce the requirements for interpreting programs that are both probabilistic and concurrent. To illustrate the subtleties, consider the following program:

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

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 concurrent scheduling). The parity of x is fixed after the coin flip (returning 0 or 1 with probability $\frac{1}{2}$), but then the scheduler can influence its final value 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 this program 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 this program using a naive composition of distributions and powerset (as monads) but, as it is well known from prior work, such combination can easily lead to non-compositional semantics [46, 66, 74, 75]. While there are several domains for this combination of effects, we use the *convex powerset*, which is quite well-studied in that it is a monad [23] for sequential composition; it is a directed complete partial order (DCPO) [30, 60–62] for finding fixed points of iterated computations; and has well behaved equational laws [6, 40, 41].

We now give a basic account of the convex powerset; for more details on the DCPO structure and monad operations, refer to Appendix A.2 and [71, §3]. A discrete probability distribution $\mu \in \mathcal{D}(X) = X \rightarrow [0, 1]$ is a countable map from X to probabilities 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. Our computational domain—the *convex powerset*—consists of nonempty convex sets of distributions¹.

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

This computational domain allows us give semantics to the coin flip actions in program (2) $\llbracket a \rrbracket_{act} : \mathcal{S} \rightarrow \mathcal{C}(\mathcal{S})$ where $\mathcal{S} \triangleq \text{var} \rightarrow \text{val}$ is the set of variable valuations. The semantics is given by a set containing a single distribution, where x is set to 1 with probability p and it is set to 0 with probability $1 - p$. Being a *set of distributions*, $\mathcal{C}(\mathcal{S})$ 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 [63].

$$\llbracket x : \approx \text{flip}(p) \rrbracket_{act}(s) \triangleq \left\{ \left[\begin{array}{l} s[x := 1] \\ s[x := 0] \end{array} \right] \mapsto \begin{array}{l} p \\ 1-p \end{array} \right\} \quad S \& T \triangleq \bigcup_{p \in [0,1]} S \oplus_p T$$

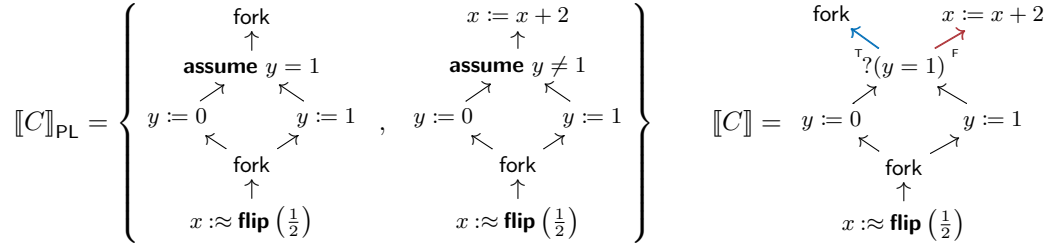
Our goal is to develop a pomset model to compositionally reason about concurrency in a manner that is compatible with the convex powerset. Existing pomset models do not track control flow, and instead rely on the following equation, stating that a command followed by an if-statement can be decomposed into two traces, with the split lifted to the top:

$$C; \text{if } b \text{ then } C_1 \text{ else } C_2 \equiv (C; \text{assume } b; C_1) \& (C; \text{assume } \neg b; C_2) \quad (3)$$

It is well known that this equation is invalid in probabilistic contexts, especially \mathcal{C} [40]. More concretely, prior denotational models for concurrency $\llbracket - \rrbracket_{PL} : \text{cmd} \rightarrow \mathcal{P}(\text{pom})$ (shown in Figure 5) map programs to *pomset languages*—sets of pomsets—where each $\alpha \in \llbracket C \rrbracket_{PL}$ corresponds to a particular resolution of the tests in the program. For program (2), this approach yields the semantics on the left of Figure 1, where the problematic Equation (3) is implicitly applied. Arrows $a_1 \rightarrow a_2$ 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. There is a significant problem in $\llbracket C \rrbracket_{PL}$; as we will soon see, there is no straightforward way to piece together the entire distribution of outcomes in the program.

Crucially, our pomset structure tracks the results of control-flow tests (which can rely on randomization). This is done symbolically by associating a Boolean formula to each node

¹ The DCPO structure relies of a few more properties, which we report in Appendix A.2.



■ **Figure 1** Pomset language model (left) vs pomset with formulae model (right) for program (2) (we use a fork with 0 branches to represent **skip**).

of the pomset, recording the resolution of tests needed to reach that point. We call these new structures *pomsets with formulae*, which capture both parallel composition and guarded branching². As such, this structure encodes many traces, avoiding the need for a set of traces and simultaneously remaining compositional in the presence of probabilistic actions.

The semantics in our new model $\llbracket - \rrbracket : \text{cmd} \rightarrow \text{pom}$ for program (2) is shown on the right of Figure 1. Compared to $\llbracket C \rrbracket_{\text{PL}}$, our new model has merged the set of structures into a single structure, where the two opposing tests are replaced by a single node labelled $?(y = 1)$. The outgoing edges are labelled \top and F , indicating that they can only be followed if the test passes or fails, respectively. Now, we may wish to know the possible probabilities for each outcome resulting from running this program. To this end, in Section 5.2 we develop a *linearization* procedure $\mathcal{L} : \text{pom} \rightarrow \mathcal{S} \rightarrow \mathcal{C}(\mathcal{S})$ for interpreting a pomset as a state transformer. Linearizing this structure, we get the following set of distributions:

$$\mathcal{L}(\llbracket C \rrbracket)(s) = \left\{ \left[\begin{array}{ll} s[x := 0, y := 0] & \mapsto p \\ s[x := 1, y := 0] & \mapsto q \\ s[x := 2, y := 1] & \mapsto \frac{1}{2} - p \\ s[x := 3, y := 1] & \mapsto \frac{1}{2} - q \end{array} \right] \mid 0 \leq p, q \leq \frac{1}{2} \right\}$$

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 $y := 1; y := 0$ (or some convex combination thereof). However, regardless of the scheduler's choice of p and q , x is even (or odd) with exactly probability $\frac{1}{2}$. 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 $\llbracket C \rrbracket_{\text{PL}}$ 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 $x \leq 1$ whereas in the second pomset $x \geq 2$. Let us now try to obtain a meaningful state transformer semantics from this program by linearizing each structure and then merging the results. To do so, we use the following semantics for **assume**, where failed tests evaluate to a special $?$ symbol, which halts the program execution:

$$\llbracket \text{assume } b \rrbracket_{\text{act}}(s) = \begin{cases} \eta(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \eta(?) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases}$$

² Formulae are more expressive than conflict relations, *e.g.*, in event structures: conflict can only exclude execution of branches, whereas formulae record the entire outcome of tests that lead to every node.

Now, letting α_1 and α_2 be the left and right pomsets in $\llbracket C \rrbracket_{\text{PL}}$, we apply linearization to both structures to obtain:

$$\mathcal{L}(\alpha_1)(s) = \left\{ \left[\begin{array}{ll} s[x := 0, y := 0] & \mapsto p_1 \\ s[x := 1, y := 0] & \mapsto q_1 \\ ? & \mapsto 1 - p_1 - q_1 \end{array} \right] \mid 0 \leq p_1, q_1 \leq \frac{1}{2} \right\}$$

$$\mathcal{L}(\alpha_2)(s) = \left\{ \left[\begin{array}{ll} s[x := 2, y := 1] & \mapsto p_2 \\ s[x := 3, y := 1] & \mapsto q_2 \\ ? & \mapsto 1 - p_2 - q_2 \end{array} \right] \mid 0 \leq p_2, q_2 \leq \frac{1}{2} \right\}$$

Already, we can begin to see a problem; in $\mathcal{L}(\llbracket C \rrbracket)(s)$ 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 make this formal by attempting to define an operation \bowtie to merge the two semantics. As a first attempt, we 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:

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

Indeed, this gives us $\mathcal{L}(\llbracket C \rrbracket)(s) \subseteq \mathcal{L}(\alpha_1)(s) \bowtie \mathcal{L}(\alpha_2)(s)$: take any $\mu \in \mathcal{L}(\llbracket C \rrbracket)(s)$, which is generated by fixing some probabilities $0 \leq p, q \leq \frac{1}{2}$. Picking $p_1 = p$, $q_1 = q$, $p_2 = \frac{1}{2} - p$, and $q_2 = \frac{1}{2} - q$, we see that $\mu \in \mathcal{L}(\alpha_1)(s) \bowtie \mathcal{L}(\alpha_2)(s)$. However, $\mathcal{L}(\alpha_1)(s) \bowtie \mathcal{L}(\alpha_2)(s)$ contains extra distributions that are not in $\mathcal{L}(\llbracket C \rrbracket)(s)$, and are not correct outcomes of the program. For example, letting $p_1 = p_2 = \frac{1}{2}$ and $q_1 = q_2 = 0$, we get:

$$\left[\begin{array}{ll} s[x := 0, y := 0] & \mapsto \frac{1}{2} \\ s[x := 1, y := 0] & \mapsto 0 \\ s[x := 2, y := 1] & \mapsto \frac{1}{2} \\ s[x := 3, y := 1] & \mapsto 0 \end{array} \right] \in \mathcal{L}(\alpha_1)(s) \bowtie \mathcal{L}(\alpha_2)(s)$$

But this distribution 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 $\mathcal{L}(\alpha_1)(s)$ and $\mathcal{L}(\alpha_2)(s)$ to indicate compatibility. While we do not claim that it is impossible to record this information during linearization, our investigation suggests that it would not be straightforward. A proper linearization of the pomset language $\llbracket C \rrbracket_{\text{PL}}$ 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 $\llbracket C \rrbracket$, where each test already has information about both branches. We therefore conclude that pomsets with formulae are 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 a particular kind of labelled partial order, which will form the core of our semantic model. We call this structure a *labelled partial order with formulae* (LPOF), as it includes a special labelling, assigning a Boolean formula to each node. We recall basic definitions from domain theory in Appendix A.1; for a complete treatment refer to [1].

3.1 Labelled Partial Orders with Formulae

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

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

Formulae are interpreted by valuations $v: \text{nodes} \rightarrow \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$; the satisfaction relation, written $v \models \psi$, is defined in the standard way (see Definition A.5 in Appendix A.1). The variables of a formula ψ , written $\text{vars}(\psi)$, are those nodes that appear in ψ . We write $\text{sat}(\psi)$ iff there exists a valuation v such that $v \models \psi$; $\psi \Rightarrow \psi'$ iff $v \models \psi' \Rightarrow v \models \psi$, for every $v \models \psi$; and $\psi \Leftrightarrow \psi'$ iff $\psi \Rightarrow \psi'$ and $\psi' \Rightarrow \psi$. In what follows, given a strict poset $\langle X, < \rangle$ and any $x \in X$, we let $x \downarrow \triangleq \{y \in X \mid y < x\}$, $x \uparrow \triangleq \{y \in X \mid x < y\}$, $\text{succ}(x) \triangleq \{y \in X \mid x < y \wedge \nexists z. (x < z < y)\}$, and $\min \triangleq \{x \in X \mid x \downarrow = \emptyset\}$. Further, the level of x , written $\text{lev}(x)$, is the length of the longest path from a minimal element to x . See Appendix A.1 for more details.

► **Definition 3.1** (LPOF). *Let $\langle L, \leq \rangle$ be a pointed, finitely preceded DCPO with bottom element \perp . A labelled partial order with formulae (LPOF) over L is a 4-tuple $\alpha = \langle N, <, \lambda, \varphi \rangle$ where:*

1. $N \subseteq \text{nodes}$ 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., $|\text{lev}^{-1}(n)| < \infty$ for all $n \in \mathbb{N}$; and
 - c. it is single-rooted, that is: $|\min| = 1$.
3. $\lambda: N \rightarrow L$ is a labelling function such that $\text{succ}(x) = \emptyset$ whenever $\lambda(x) = \perp$.
4. $\varphi: N \rightarrow \text{form}$ is a formula function satisfying:
 - a. $\text{sat}(\varphi(x))$ and $\text{vars}(\varphi(x)) \subseteq x \downarrow$, for all $x \in N$;
 - b. $\varphi(y) \Rightarrow \varphi(x)$, for all $x < y$.

We denote by $\text{lpof}(L)$ 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., $x \downarrow_\alpha$, $\text{succ}_\alpha(x)$, \min_α , etc. We now explain the conditions in Definition 3.1.

As is typical in pomset semantics, the partial order $\langle N, < \rangle$ 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. 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 happen in a finite amount of time. Moreover, requiring that every level has a finite number of nodes is a weakening of the usual finite branching property: indeed, we allow a node to have infinitely many successors, but they cannot be all at the same level. We defer the discussion on Condition (2c) to Remark 4.1 later on.

Concerning Condition (3), labels correspond to *Boolean tests* and *actions*, performed during a program execution (e.g., Figure 1). Unlike standard pomset models, actions can be probabilistic, and our model is consistent with known sequential probabilistic semantics (Theorem 5.3). For the moment, we leave labels unspecified, only assuming the presence of \perp , used to denote a nonterminating computation, which is fundamental in approximating the fixed point semantics of while-loops. Since \perp denotes nontermination, nodes labelled with \perp cannot be a predecessor of any node, as the successors of \perp would never be executed.

Condition (4) is about formulae, which are crucial for modeling conditional constructs (if-then-else and while-loops), such as **if** b_1 **then** a_1 **else if** b_2 **then** a_2 **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 $\ell \in \{b_1, b_2, a_1, a_2, a_3\}$, $w \xrightarrow{T} w_1$ means that $\varphi(w_1) \Leftrightarrow \varphi(w) \wedge w$ and $w \xrightarrow{F} w_2$ means that $\varphi(w_2) \Leftrightarrow \varphi(w) \wedge \neg w$ (i.e., w is labeled with a test, whose outcome leads to w_1 or w_2 , depending on the Boolean outcome depicted on the colored arc):

$$\begin{array}{lcl}
 \varphi(x_{a_2}) = \neg x_{b_1} \wedge x_{b_2} & \begin{array}{c} x_{a_2} \\ \swarrow \text{T} \quad \searrow \text{F} \\ x_{a_1} \quad x_{b_2} \\ \swarrow \text{T} \quad \searrow \text{F} \\ x_{b_1} \end{array} & x_{a_3} \\
 \varphi(x_{a_1}) = x_{b_1} & & \varphi(x_{a_3}) = \neg x_{b_1} \wedge \neg x_{b_2} \\
 \varphi(x_{b_1}) = \text{true} & & \varphi(x_{b_2}) = \neg x_{b_1}
 \end{array} \tag{4}$$

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 3.1(4) is quite intuitive:

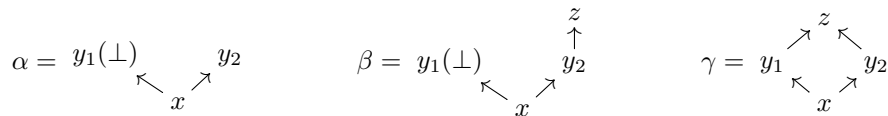
- Requiring $\text{sat}(\varphi(x))$ amounts to expressing the fact that every node x is reachable, i.e., there exists a truth assignment v such that $v \models \varphi(x)$. The truth assignment tells, for every node labeled with a Boolean test, whether that test passes or not. In our example, x_{a_3} 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 3.1(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.

Next, we define an order on LPOFs, which we will use for the construction of fixed points to give semantics to unbounded loops. The intuition is that $\alpha \sqsubseteq_{\text{lpoF}} \beta$ iff β has *more behaviors* than α , meaning that β can be obtained by expanding \perp nodes in α into larger structures (each unfolding of a while-loop is obtained in this way, as shown in Section 5.1). Since LPOFs can only be expanded from \perp nodes, the label set L must be pointed (i.e., $\perp \in L$); 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). This can also be useful to model *invariant sensitive execution*, introduced in Probabilistic Concurrent Outcome Logic [73]. Notationally, Bot_α denotes the set of nodes labelled with \perp , i.e., $\text{Bot}_\alpha \triangleq \{x \in N_\alpha \mid \lambda_\alpha(x) = \perp\}$.

► **Definition 3.2** (Ordering on LPOFs). *We let $\alpha \sqsubseteq_{\text{lpoF}} \beta$ iff:*

1. N_α is a downward-closed subset of N_β , written $N_\alpha \subseteq_\downarrow N_\beta$;
2. $<_\alpha = <_\beta \cap (N_\alpha \times N_\alpha)$;
3. $\forall x \in N_\alpha$: **a.** $\lambda_\alpha(x) \leq \lambda_\beta(x)$; **b.** $\varphi_\alpha(x) = \varphi_\beta(x)$; **c.** $\text{succ}_\alpha(x) = \text{succ}_\beta(x) \setminus \text{Bot}_\alpha \uparrow_\beta$.

► **Example 3.3.** Consider the following three LPOFs:



where node y_1 is labelled with \perp in α and β , whereas all other nodes have non- \perp 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 $\alpha \sqsubseteq_{\text{lpof}} \gamma$; this is because, if \perp represents a diverging computation, then z will never have all the causes it needs to be executed and so it must disappear. Indeed, $\beta \not\sqsubseteq_{\text{lpof}} \gamma$ because, even though $N_\beta \subseteq_\perp N_\gamma$, condition (3c) of Definition 3.2 is violated: $z \in \text{succ}_\beta(y_2)$ but $z \notin \text{succ}_\gamma(y_2) \setminus \text{Bot}_\beta \uparrow_\gamma$, since $y_1 \in \text{Bot}_\beta$ and $y_1 <_\gamma z$.

Finally, in order to find fixed points, we will need to have a directed complete partial order (DCPO) structure and a notion of continuity for functions, which we now define.

► **Definition 3.4 (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 $x_1, x_2 \leq x$.*

$\langle X, \leq \rangle$ is a directed complete partial order (DCPO) iff, for every directed set D , $\sup D$ exists; furthermore, $\langle X, \leq \rangle$ is called pointed if there exists an element $\perp \in X$ such that $\perp \leq x$ for all $x \in X$.

► **Definition 3.5 (Scott Continuity).** *Given two DCPOs $\langle X, \leq_X \rangle$ and $\langle Y, \leq_Y \rangle$, a function $f: X \rightarrow Y$ is Scott Continuous if it is monotone: $f(x) \leq_Y f(x')$ for all $x \leq_X x'$; and preserves suprema of directed sets: $\sup_{x \in D} f(x) = f(\sup D)$ for all $D \subseteq X$ directed.*

► **Lemma 3.6.** *For any directed set $D \subseteq \text{lpof}(L)$, $\sup D = \langle N, <, \lambda, \varphi \rangle$, where:*

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

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

In Lemma 3.6, φ is well defined since, for every $x \in \text{nodes}$ 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 $\beta, \beta' \sqsubseteq_{\text{lpof}} \gamma$ and, by Definition 3.2(3b), we have that $\varphi_\beta(x) = \varphi_\gamma(x) = \varphi_{\beta'}(x)$.

Notice that $\text{lpof}(L)$ is *not* pointed, since the node at the root can vary; hence, there is no single LPOF that is smaller (w.r.t. $\sqsubseteq_{\text{lpof}}$) than all other LPOFs. This situation will change when moving to pomsets (Lemma D.6), which abstract away from the specific nodes.

3.2 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.

► **Definition 3.7 (LPOF Isomorphism).** *Let α and β be LPOFs and $f: N_\alpha \rightarrow N_\beta$ be a bijection between their nodes. Define $f(\alpha) = \langle N, <, \lambda, \varphi \rangle$ as*

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

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

► **Definition 3.8 (Pomsets with Formulae).** *Let $[\alpha] \triangleq \{\beta \in \text{lpof}(L) \mid \alpha \equiv \beta\}$ be the isomorphism class of α . A pomset with formulae (or, simply, pomset) $\alpha \in \text{pom}(L)$ is an isomorphism class of LPOFs: $\text{pom}(L) \triangleq \{[\alpha] \mid \alpha \in \text{lpof}(L)\}$. Let $\text{pom}_{\text{fin}}(L)$ be the set of pomsets with finitely many nodes. Finally, let $\alpha \sqsubseteq_{\text{pom}} \beta$ iff $\forall \alpha \in \alpha. \exists \beta \in \beta. \alpha \sqsubseteq_{\text{lpof}} \beta$.*

The semantics of programs depends on a variety of pomset composition operations (sequential, guarded, and parallel composition). We will also later *linearize* pomsets into state transformer functions. Some of these operations cannot be defined inductively on infinite structures, so we need ways to extend functions on finite structures to infinite ones. We first show that infinite pomsets correspond to the suprema of their finite approximations.

► **Lemma 3.9** (Finite Approximations). *For any $\alpha \in \text{pom}(L)$, $\alpha = \sup \lfloor \alpha \rfloor_{\text{fin}}$, where $\lfloor \alpha \rfloor_{\text{fin}} \triangleq \{\beta \in \text{pom}_{\text{fin}}(L) \mid \beta \sqsubseteq_{\text{pom}} \alpha\}$ is the set of finite approximations of α .*

Finite approximations coincide with the approximation order \ll of [1] instantiated to pomsets (see Appendix E for more details); thus, from now on, we let $\alpha \ll \beta$ denote $\alpha \in \lfloor \beta \rfloor_{\text{fin}}$. We are now ready to show that every operation f on finite pomsets can be extended to infinite ones by defining the operation on α as the sup of the images through f of all the approximations of α . This will be useful in the remainder of the paper.

► **Lemma 3.10** (Extension). *Let $f: \text{pom}_{\text{fin}}(L)^n \rightarrow T$ be a monotone function on the DCPO $\langle T, \leq \rangle$. Then $f^*: \text{pom}(L)^n \rightarrow T$, shown below, is well-defined and Scott continuous:*

$$f^*(\alpha_1, \dots, \alpha_n) \triangleq \sup_{\alpha'_1 \ll \alpha_1} \dots \sup_{\alpha'_n \ll \alpha_n} f(\alpha'_1, \dots, \alpha'_n)$$

4 Pomset Operations

In this section, we define pomset operations that mirror the program syntax $C \in \text{cmd}$ from (1). We first define the singleton LPOF $\langle \ell \rangle_x$ on node x with label $\ell \in L$; this generalizes to $\langle \ell \rangle$ on pomsets and will be used to give the semantics to **skip** and atomic actions:

$$\langle \ell \rangle_x \triangleq \langle \{x\}, \emptyset, [x \mapsto \ell], [x \mapsto \text{true}] \rangle \in \text{lpoF}(L) \quad \langle \ell \rangle \triangleq \{ \langle \ell \rangle_x \mid x \in \text{nodes} \} \in \text{pom}(L)$$

4.1 Guarded Choice

We now define guard, recording the causality between a test and the two branches of computation defined on it. Assuming that $N_\alpha \cap N_\beta = \emptyset$ and $x \notin N_\alpha \cup N_\beta$, we let $\text{guard}(x, \ell, \alpha, \beta) = \langle N, <, \lambda, \varphi \rangle$ where we overload \wedge s.t. $\psi \wedge \text{true} = \text{true} \wedge \psi = \psi$, and:

$$\begin{aligned} N &\triangleq \{x\} \cup N_\alpha \cup N_\beta & < &\triangleq <_\alpha \cup <_\beta \cup (\{x\} \times (N_\alpha \cup N_\beta)) \\ \lambda(y) &\triangleq \begin{cases} \ell & \text{if } y = x \\ \lambda_\alpha(y) & \text{if } y \in N_\alpha \\ \lambda_\beta(y) & \text{if } y \in N_\beta \end{cases} & \varphi(y) &\triangleq \begin{cases} \text{true} & \text{if } y = x \\ \varphi_\alpha(y) \wedge x & \text{if } y \in N_\alpha \\ \varphi_\beta(y) \wedge \neg x & \text{if } y \in N_\beta \end{cases} \end{aligned}$$

So, $\text{guard}(x, b, \alpha, \beta)$ 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, $\text{guard}(x_{b_1}, b_1, \langle a_1 \rangle_{x_{a_1}}, \text{guard}(x_{b_2}, b_2, \langle a_2 \rangle_{x_{a_2}}, \langle a_3 \rangle_{x_{a_3}}))$ produces the LPOF depicted in (4). For finite pomsets, we define the guard operation as follows:

$$\text{guard}(\ell, \alpha, \beta) \triangleq \{ \text{guard}(x, \ell, \alpha, \beta) \mid \alpha \in \alpha, \beta \in \beta, N_\alpha \cap N_\beta = \emptyset, x \notin N_\alpha \cup N_\beta \}$$

Since guard is monotone (Lemma F.8), we can extend it to infinite pomsets:

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

4.2 Sequential Composition

The sequential composition operation $\alpha ; \beta$ is meant to enforce that all actions in α must occur before any of those in β . This causality dependency interacts with guarded branching,

as we will now see, which makes the formal definition of \circ more challenging. Consider the following LPOFs (where we denote with x the singleton $\langle \ell \rangle_x$ when the label is not relevant):

$$\begin{array}{lll} \text{a. } x \circ y = \begin{array}{c} y \\ \uparrow \\ x \end{array} & \text{b. } \left(\begin{array}{cc} y_1 & y_2 \\ \nwarrow & \nearrow \\ & x \end{array} \right) \circ z = \begin{array}{c} z \\ \nwarrow \quad \nearrow \\ y_1 \quad y_2 \\ \nwarrow \quad \nearrow \\ & x \end{array} & \text{c. } \left(\begin{array}{cc} y_1 & y_2 \\ \nwarrow \quad \nearrow \\ & x \end{array} \right) \circ z = \begin{array}{cc} z_x & z_{\neg x} \\ \uparrow & \uparrow \\ y_1 & y_2 \\ \nwarrow \quad \nearrow \\ & x \end{array} \quad (5) \end{array}$$

The sequential composition of two singletons (5a) simply creates a causality between them. When the program has forked into parallel (unguarded) branches (5b), \circ introduces a diamond structure, as is typical in pomset semantics. However, if the program has a *guarded* branch (5c), then the following actions must be copied after each branch. Copying is 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 3.9; see Figure 2 and Appendix B.2 for a more detailed account. When a structure contains \perp nodes, so that some paths are *stuck*, the behavior of \circ is more complicated. Consider the following LPOFs, where node y_1 is labelled with \perp and all other nodes have non- \perp labels:

$$\begin{array}{ll} \text{a. } \left(\begin{array}{cc} y_1(\perp) & y_2 \\ \nwarrow \quad \nearrow \\ & x \end{array} \right) \circ z = \begin{array}{c} z \\ \nwarrow \quad \nearrow \\ y_1(\perp) \quad y_2 \\ \nwarrow \quad \nearrow \\ & x \end{array} & \text{b. } \left(\begin{array}{cc} y_1(\perp) & y_2 \\ \nwarrow \quad \nearrow \\ & x \end{array} \right) \circ z = \begin{array}{c} y_1(\perp) \quad y_2 \\ \nwarrow \quad \nearrow \\ & x \end{array} \quad (6) \end{array}$$

When composing the singleton z after a guarded branch with one stuck path (6a), 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 (6b), 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 \circ (Lemma F.5).

We first define \circ for finite LPOFs, and then extend our definition to infinite ones using Lemma 3.10. In the rest of this section, we assume that $\alpha, \beta \in \text{lpof}_{\text{fin}}(L)$. We start by defining the notion of stuck computations, extensible nodes, and branches.

$$\begin{aligned} \text{stuck}_\alpha &\triangleq \bigvee_{x \in \text{Bot}_\alpha} \varphi_\alpha(x) \\ \text{ext}_\alpha &\triangleq \{x \in N_\alpha \mid \varphi_\alpha(x) \not\Rightarrow \text{stuck}_\alpha\} \\ \text{br}_\alpha &\triangleq \left\{ \varphi_\alpha(S) \mid \begin{array}{l} \emptyset \subset S \subseteq \text{ext}_\alpha, \varphi_\alpha(S) \Rightarrow \neg \text{stuck}_\alpha, \\ \forall T. S \subset T \subseteq \text{ext}_\alpha \Rightarrow \neg \text{sat}(\varphi_\alpha(T)) \end{array} \right\} \end{aligned}$$

The formula stuck_α indicates which nodes are guaranteed to encounter a \perp later in their execution. Extensible nodes $x \in \text{ext}_\alpha$ are not stuck, so there is some computation path that they can take without encountering any \perp node. In (5), all nodes are extensible, whereas in (6a) only x and y_2 are extensible and in (6b) 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: there, for a set $S \subseteq N_\alpha$, we define $\varphi_\alpha(S) \triangleq \bigwedge_{x \in S} \varphi_\alpha(x)$. For example, in (5a,b), $\text{br} = \{\text{true}\}$ since there are no tests; in (5c), $\text{br} = \{x, \neg x\}$ since both outcomes of the test are not stuck; in (6a), $\text{br} = \{\neg x\}$ since the program is stuck if x passes; and in (6b), $\text{br} = \emptyset$, since all paths are stuck. The set of branches increases monotonically with $\sqsubseteq_{\text{lpof}}$. Refer to Appendix B.1 for more examples of extensibility and branches. As we saw in (5c), we will need an isomorphic copy $\beta_\psi \equiv \beta$ for each branch $\psi \in \text{br}_\alpha$. These copies are generated by a function $f: \text{br}_\alpha \rightarrow [\beta]$ 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:

$$\text{copy}_{\alpha, \beta} \triangleq \{f: \text{br}_\alpha \rightarrow [\beta] \mid \forall \psi \in \text{br}_\alpha. (N_{f(\psi)} \cap N_\alpha = \emptyset \wedge \forall \psi' \neq \psi. N_{f(\psi)} \cap N_{f(\psi')} = \emptyset)\}$$

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

$$\begin{aligned} N &\triangleq N_\alpha \cup \bigcup_{\psi \in \text{br}_\alpha} N_{\beta_\psi} & < &\triangleq <_\alpha \cup \bigcup_{\psi \in \text{br}_\alpha} (<_{\beta_\psi} \cup (\{x \in N_\alpha \mid \psi \Rightarrow \varphi_\alpha(x)\} \times N_{\beta_\psi})) \\ \lambda(x) &\triangleq \begin{cases} \lambda_\alpha(x) & \text{if } x \in N_\alpha \\ \lambda_{\beta_\psi}(x) & \text{if } x \in N_{\beta_\psi} \end{cases} & \varphi(x) &\triangleq \begin{cases} \varphi_\alpha(x) & \text{if } x \in N_\alpha \\ \varphi_{\beta_\psi}(x) \wedge \psi & \text{if } x \in N_{\beta_\psi} \end{cases} \end{aligned}$$

(where, again, $\psi \wedge \text{true} = \text{true} \wedge \psi = \psi$). So, the nodes of $\alpha \circ_f \beta$ 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 . Sequential composition for finite pomsets is defined below (left), and is defined for infinite pomsets by extension (right).

$$\alpha \circ \beta \triangleq \{\alpha \circ_f \beta \mid \alpha \in \alpha, \beta \in \beta, f \in \text{copy}_{\alpha, \beta}\} \quad \alpha \circ \beta \triangleq \sup_{\alpha' \ll \alpha} \sup_{\beta' \ll \beta} \alpha' \circ \beta'$$

► **Remark 4.1.** We can now explain Condition (2c) in Definition 3.1, 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, $y \sqsubseteq_{\text{lpof}} yz$ but $x \circ y \triangleq \begin{smallmatrix} y \\ \uparrow \\ x \end{smallmatrix} \not\sqsubseteq_{\text{lpof}} \begin{smallmatrix} y \\ \nwarrow \nearrow \\ x \end{smallmatrix} z \triangleq x \circ (yz)$. This is undesirable and easily fixed with the mild requirement of single-rootedness in Definition 3.1.

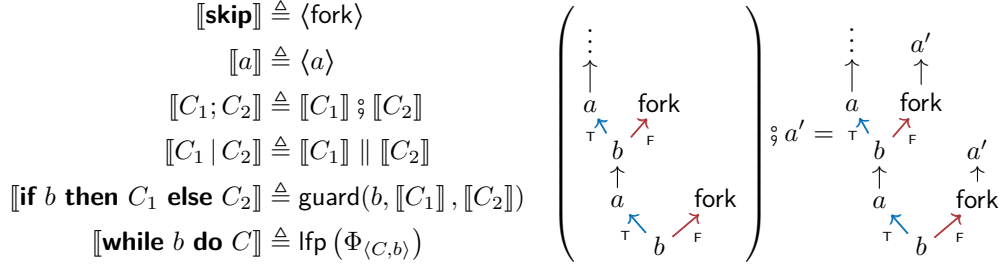
4.3 Parallel Composition

We finally define parallel composition. We assume that L contains a special (non- \perp) label **fork** that denotes thread forking and let $\text{nFork}_\alpha \triangleq N_\alpha \setminus \{x \in \min_\alpha \mid \lambda_\alpha(x) = \text{fork}\}$ 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

$$\begin{aligned} N &\triangleq \{x\} \cup \text{nFork}_\alpha \cup \text{nFork}_\beta \\ < &\triangleq (\{x\} \times (\text{nFork}_\alpha \cup \text{nFork}_\beta)) \cup (<_\alpha \cap (\text{nFork}_\alpha \times \text{nFork}_\alpha)) \cup (<_\beta \cap (\text{nFork}_\beta \times \text{nFork}_\beta)) \\ \lambda(y) &\triangleq \begin{cases} \text{fork} & \text{if } y = x \\ \lambda_\alpha(y) & \text{if } y \in \text{nFork}_\alpha \\ \lambda_\beta(y) & \text{if } y \in \text{nFork}_\beta \end{cases} & \varphi(y) &\triangleq \begin{cases} \text{true} & \text{if } y = x \\ \varphi_\alpha(y) & \text{if } y \in \text{nFork}_\alpha \\ \varphi_\beta(y) & \text{if } y \in \text{nFork}_\beta \end{cases} \end{aligned}$$

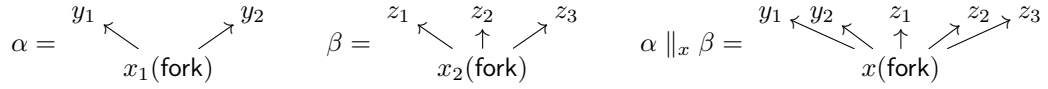
Notice that the branching that arises from \parallel is conceptually (and practically) different from the branching that arises from a test: the branching due to \parallel does not exclude any branch, whereas the two branches due to a test are mutually exclusive. This is apparent by the different way in which we handle the formulae associated to the nodes after the branch: they 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 in parallel many LPOFs.

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



■ **Figure 2** Left: pomset semantics of commands $\llbracket - \rrbracket : \text{cmd} \rightarrow \text{pom}$. Right: semantics of the program $(\text{while } b \text{ do } a) ; a'$.

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**.

As a side note, observe that the way in which we defined parallel composition entails that $\alpha \sqsubseteq_{\text{lpo}} \alpha \parallel_x \beta$ if and only if $\alpha = \langle \perp \rangle_x$: the LPOF α has just a node x labelled with \perp ; the second LPOF has node x labelled with **fork** that precedes a node labelled with \perp 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 from the specific node set³. Like before, we define parallel composition for (both finite and infinite) pomsets on top of that for LPOFs:

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

► **Remark 4.2.** It is impossible for \parallel to be *both* associative *and* Scott continuous. If it were continuous, then $f(\alpha) \triangleq \langle \ell \rangle \parallel \alpha$ would have a least fixed point $\text{lfp}(f) = \langle \ell \rangle \parallel (\langle \ell \rangle \parallel \dots)$, and, due to associativity, this pomset is infinitely branching, which invalidates Lemma 3.9.

5 Denotational Semantics

We now define denotational semantics for commands $C \in \text{cmd}$ (see (1)) based on *pom*. Although actions at this stage are uninterpreted, *pom* allows for actions with a variety of different *computational effects*, which we will see in Section 5.2. Those actions $a \in \text{act}$ are drawn from a DCPO $\langle \text{act}, \leq_{\text{act}} \rangle$ and tests $b \in \text{test}$ are drawn from a set *test* (with a flat order), which is closed under conjunction, disjunction, and negation. We first define a compositional denotational model for this language in Section 5.1, and then in Section 5.2 we show how to interpret pomsets as state transformers, to learn their behavior on particular inputs.

5.1 Pomset Semantics

We will now discuss the semantics of programs $C \in \text{cmd}$ as pomsets, which record both the causality between atomic actions and the branching behavior of tests. The label set

³ We remark that \circ on LPOFs also satisfies a non-straightforward associativity, that however becomes the usual one when passing to pomsets.

$label \triangleq act \cup test \cup \{fork, \perp\}$ consists of actions, tests, fork, and bottom nodes, which forms a pointed DCPO, with \perp as the bottom. Actions are ordered according to \leq_{act} ; for any other label $\ell \in test \cup \{fork\}$, we only have $\perp \leq \ell$ and $\ell \leq \ell$. We will also henceforth use $pom \triangleq pom(label)$ to refer to pomsets over this label set.

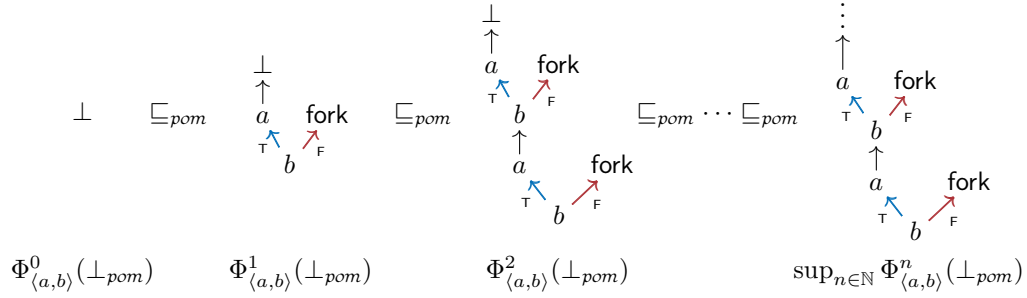
The semantic function $\llbracket - \rrbracket : cmd \rightarrow pom$, mapping commands to pomsets, is shown on the left of Figure 2. Its definition is straightforward using the pomset operations; in particular, **skip** is interpreted as a singleton fork and while-loops are interpreted as the least fixed point of the characteristic function $\Phi_{\langle C, b \rangle} : pom \rightarrow pom$ defined as follows:

$$\Phi_{\langle C, b \rangle}(\alpha) \triangleq guard(b, \llbracket C \rrbracket \circ \alpha, \llbracket skip \rrbracket)$$

Clearly $\Phi_{\langle C, b \rangle}$ is Scott continuous, as it is defined by Scott continuous operations **guard** and \circ (Corollary F.9cor:seqCont). So, by Kleene's fixed point theorem, $\Phi_{\langle C, b \rangle}$ has a least fixed point given by:

$$\llbracket \text{while } b \text{ do } C \rrbracket = \text{lfp}(\Phi_{\langle C, b \rangle}) = \sup_{n \in \mathbb{N}} \Phi_{\langle C, b \rangle}^n(\perp_{pom})$$

where $f^0 \triangleq \text{id}$ and $f^{n+1} \triangleq f \circ f^n$. Considering a loop $\llbracket \text{while } b \text{ do } a \rrbracket$ 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 \perp node is pushed to a progressively higher level after each unrolling, so in the supremum it does not appear at all.

On the right of Figure 2, we show $\llbracket (\text{while } b \text{ do } a) ; a' \rrbracket$, containing a loop with a single action in the body followed by another action. As discussed in Section 4.2, 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.

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 4.

► **Definition 5.1** (Binary Branching). *A pomset $\alpha \in pom$ has the binary branching property if, for every $\alpha \in \alpha$ and every $x \in N_\alpha$ such that $\lambda_\alpha(x) \in test$, $\text{succ}_\alpha(x) = \{y_1, y_2\}$ 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. $\text{pred}_\alpha(y_1) = \text{pred}_\alpha(y_2) = \{x\}$. Furthermore, if x is not the successor of a test, then $\varphi_\alpha(x) \Leftrightarrow \bigwedge_{y \in \text{pred}_\alpha(x)} \varphi_\alpha(y)$.*

5.2 Linearization

We now discuss how to produce a state transformer from the pomset semantics of Section 5.1, which is useful for understanding the input-output behavior of programs resulting from

$$\begin{aligned}
\text{next}(\alpha, \psi, S) &\triangleq \{x \in N_\alpha \setminus S \mid x \downarrow_\alpha \subseteq S, \psi \Rightarrow \varphi_\alpha(x)\} \\
\mathcal{L}_{\text{lpof}}(\alpha, \psi, S)(s) &\triangleq \eta(s) \quad \text{if } \text{next}(\alpha, \psi, S) = \emptyset \\
\mathcal{L}_{\text{lpof}}(\alpha, \psi, S)(s) &\triangleq \bigotimes_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})^\dagger(\llbracket a \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) = a \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi \wedge (x = \llbracket b \rrbracket_{\text{test}}(s)), S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = b \in \text{test} \\ \perp_D & \text{if } \lambda_\alpha(x) = \perp \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases}
\end{aligned}$$

■ **Figure 3** Linearization for finite LPOFs $\mathcal{L}_{\text{lpof}}: \text{lpof}_{\text{fin}}(\text{label}) \times \text{form} \times \mathcal{P}(\text{nodes}) \rightarrow \mathcal{S} \rightarrow D(\mathcal{S})$.

scheduling concurrent threads. This state transformer is obtained via *linearization*, which consists of considering 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 6.

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 [3]. 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 \perp_D .
2. An associative, commutative, and monotone nondeterminism operator $\&: D(\mathcal{S})^2 \rightarrow D(\mathcal{S})$.
3. An additive monad in the category **dcpo** $\langle D, \eta, (-)^\dagger \rangle$, meaning that:
 - a. the operations $\eta: X \rightarrow D(X)$ and $(-)^\dagger: (X \rightarrow D(Y)) \rightarrow D(X) \rightarrow D(Y)$ obey the monad laws $f^\dagger \circ \eta = f$, $\eta^\dagger = \text{id}$, and $(g^\dagger \circ f)^\dagger = g^\dagger \circ f^\dagger$.
 - b. Kleisli extension is Scott Continuous (*i.e.*, $\sup_{f \in D} \sup_{d \in D'} f^\dagger(d) = (\sup D)^\dagger(\sup D')$), strict (*i.e.*, $f^\dagger(\perp_D) = \perp_D$), and additive (*i.e.*, $f^\dagger(d \& d') = f^\dagger(d) \& f^\dagger(d')$) [69, 72].
4. An interpretation function for actions $\llbracket - \rrbracket_{\text{act}}: \text{act} \rightarrow \mathcal{S} \rightarrow D(\mathcal{S})$ that is monotone (*i.e.*, $\llbracket a \rrbracket_{\text{act}}(s) \sqsubseteq \llbracket a' \rrbracket_{\text{act}}(s)$ if $a \leq_{\text{act}} a'$) and one for tests $\llbracket - \rrbracket_{\text{test}}: \text{test} \rightarrow \mathcal{S} \rightarrow \mathbb{B}$.

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 [1, 47, 55] (where the latter two additionally deal with nontermination), the convex powerset [20, 43] and the powerdomain of indexed valuations [63, 64, 66] (for mixed probabilistic and nondeterministic computation), and other domains for nondeterminism with exceptions [69, 72]. In Theorems 5.3 and 6.1, 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 3. 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 we can extend linearization to infinite structures using the extension lemma. In $\mathcal{L}_{\text{lpof}}(\alpha, \psi, S)$, 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 $\text{next}(\alpha, \psi, S)$ gives the nodes that are ready to be scheduled, which includes all nodes that obey the path condition ψ , and whose predecessors are in S . If $\text{next}(\alpha, \psi, S)$ is empty, then $\mathcal{L}_{\text{lpof}}(\alpha, \psi, S)$ simply returns the current state s using the monad unit η . If not,

then it nondeterministically selects a next node, where $\&_{i \in I} d_i \triangleq d_{i_1} \& \dots \& d_{i_n}$, for any finite index set $I = \{i_1, \dots, i_n\}$. If the next node is an action, then it interprets the action using $\llbracket - \rrbracket_{act}$, and then 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 result is added to the path condition, where $\langle x = 1 \rangle \triangleq x$ and $\langle x = 0 \rangle \triangleq \neg x$. If the next node is \perp , then the linearization is \perp_D , and **fork** is treated like a no-op.

We next use \mathcal{L}_{lpof} to define linearization on finite pomsets $\mathcal{L}_{fin}: pom_{fin}(label) \rightarrow \mathcal{S} \rightarrow D(\mathcal{S})$ as $\mathcal{L}_{fin}([\alpha]) \triangleq \mathcal{L}_{lpof}(\alpha, \text{true}, \emptyset)$. Clearly $\mathcal{L}_{lpof}(\alpha, \text{true}, \emptyset) = \mathcal{L}_{lpof}(\beta, \text{true}, \emptyset)$ if $\alpha \equiv \beta$, so $\mathcal{L}_{fin}(\alpha)$ can be defined for any arbitrary representative LPOF $\alpha \in \alpha$. Finally, linearization is extended to infinite pomsets $\mathcal{L}: pom \rightarrow \mathcal{S} \rightarrow D(\mathcal{S})$ by taking the supremum over all of the finite approximations: $\mathcal{L}(\alpha) \triangleq \mathcal{L}_{fin}^*(\alpha)$. Linearization over finite pomsets is monotone (Lemma G.3), therefore the extension is well-defined and Scott continuous.

To ensure that linearization acts as desired, we provide a sanity check lemma to relate the linearized semantics to well-known counterparts. 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.

► **Lemma 5.2** (Linearization). *The following properties hold:*

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

$$\mathcal{L}(\llbracket \text{if } b \text{ then } C_1 \text{ else } C_2 \rrbracket)(s) = \begin{cases} \mathcal{L}(\llbracket C_1 \rrbracket)(s) & \text{if } \llbracket b \rrbracket_{test}(s) = 1 \\ \mathcal{L}(\llbracket C_2 \rrbracket)(s) & \text{if } \llbracket b \rrbracket_{test}(s) = 0 \end{cases}$$

$$\mathcal{L}(\llbracket \text{while } b \text{ do } C \rrbracket) = \text{lfp}(\Psi_{\langle \mathcal{L}(\llbracket C \rrbracket), b \rangle}) \text{ where } \Psi_{\langle f, b \rangle}(g)(s) \triangleq \begin{cases} g^\dagger(f(s)) & \text{if } \llbracket b \rrbracket_{test}(s) = 1 \\ \eta(s) & \text{if } \llbracket b \rrbracket_{test}(s) = 0 \end{cases}$$

Lemma 5.2 is quite significant; it implies that $\mathcal{L}(\llbracket C \rrbracket)$ corresponds to standard monadic semantics in a variety of domains. In particular, it brings us to our first adequacy theorem, showing that our pomset model recovers standard probabilistic semantics $\llbracket C \rrbracket_{\mathcal{C}} : \mathcal{S} \rightarrow \mathcal{C}(\mathcal{S})$ (shown in Figure 4 of Appendix A.2) [20, 37, 71] based on the convex powerset for a parallel-free fragment of our programming language.

► **Theorem 5.3.** *For any program $C \in cmd$ without parallel composition: $\mathcal{L}(\llbracket C \rrbracket) = \llbracket C \rrbracket_{\mathcal{C}}$.*

Proof. By induction on the structure of C . The cases follow immediately from Lemma 5.2. ◀

6 Pomset Languages and Purely Nondeterministic Concurrency

Powerdomains give semantics to programs that combine nondeterminism with looping or recursion [47, 55]. In this section, we create a powerdomain instantiation of our semantics from Section 5 and prove that the resulting model corresponds to well-known models of non-probabilistic concurrent programs [27, 34]. We work in the *Hoare* or *lower* powerdomain, named as such for its connection to partial correctness and Hoare Logic [16, 21]. Indeed, the Hoare powerdomain identifies the terminating traces of a program, but not that the program *always* terminates. From a concurrency perspective, this corresponds to *safety* properties [33]. For simplicity, we presume that the order over actions \leq_{act} in this section is flat.

Since typical pomsets do not contain formulae [17, 18, 49], they cannot encode control flow branching introduced by if-statements and while-loops. As we already saw in Section 2, concurrent semantics typically use *pomset languages*—sets of pomsets—where each individual pomset contains only the actions that occur in a single branch of computation [27, 34].

We emulate pomset language semantics using pomsets with formulae. First, we define a label set $label_{PL} \triangleq act_{PL} \cup test_{PL} \cup \{fork, \perp\}$, whose components are as before, but now $test_{PL} \triangleq \emptyset$, and instead $act_{PL} \triangleq act \cup \{\mathbf{assume} \ b \mid b \in test\}$ includes an action **assume** b for every test $b \in test$, which guarantees that a certain condition holds in the current branch. Pomset languages $pomLang \triangleq \mathcal{P}(pom_{fin}(label_{PL}))$ are sets of finite pomsets over $label_{PL}$.

The pomset language semantics $\llbracket - \rrbracket_{PL} : cmd \rightarrow pomLang$, which is standard [27, 34], is given in Figure 5 of Appendix A.3. In this model, each $\alpha \in \llbracket C \rrbracket_{PL}$ corresponds to a particular sequence of test resolutions. When an **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 1. Loops are interpreted as the (possibly infinite) union of all finite traces, so infinite pomsets are not needed.

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 $\& \triangleq \cup$ and define the monad operations as $\eta(s) \triangleq \{s\}$ and $f^\dagger(S) \triangleq \bigcup_{s \in S} f(s)$, which are well known to be Scott continuous, strict, and additive [69]. Accordingly, action evaluation $\llbracket - \rrbracket_{act_{PL}} : act_{PL} \rightarrow \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$ is given below, where **assume** actions result in $\{s\}$, if the test is true in state s , and \emptyset , otherwise. We also define a specialized linearization operation $\mathcal{L}_{PL} : pomLang \rightarrow \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$, which is simply a union over the linearization of all traces.

$$\llbracket a \rrbracket_{act_{PL}}(s) \triangleq \begin{cases} \llbracket a \rrbracket_{act}(s) & \text{if } a \in act \\ \{s\} & \text{if } a = \mathbf{assume} \ b, \llbracket b \rrbracket_{test}(s) = 1 \\ \emptyset & \text{if } a = \mathbf{assume} \ b, \llbracket b \rrbracket_{test}(s) = 0 \end{cases} \quad \mathcal{L}_{PL}(S)(s) \triangleq \bigcup_{\alpha \in S} \mathcal{L}_{fin}(\alpha)(s)$$

Since all formulae are true, the next nodes to schedule are exactly those whose predecessors have all been processed, *i.e.*, $next(\alpha, true, S) = \{x \in N_\alpha \setminus S \mid x \downarrow_\alpha \subseteq S\}$. This corresponds to standard interleaving definitions of linearization [49]. We now prove that pomset language semantics $\llbracket - \rrbracket_{PL}$ (Figure 5) corresponds exactly to the Hoare powerdomain instance of our semantic model from Section 5. The proof relies on a translation $tr : pom \rightarrow pomLang$, which recovers a pomset language from a pomset with formulae, and is defined in Appendix H.

► **Theorem 6.1** (Equivalence of Semantics). *The following diagram commutes:*

$$\begin{array}{ccc} cmd & \xrightarrow{\llbracket - \rrbracket} & pom \\ \llbracket - \rrbracket_{PL} \downarrow & \swarrow tr & \downarrow \mathcal{L} \\ pomLang & \xrightarrow{\mathcal{L}_{PL}} & (\mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})) \end{array}$$

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 $\mathcal{L} = \mathcal{L}_{PL} \circ tr$ relies on two particular properties of the Hoare powerdomain. First, if $h(x) = f(x) \& g(x)$, then $h^\dagger(S) = f^\dagger(S) \& g^\dagger(S)$, which is invalid in probabilistic domains. In fact, this corresponds to the problematic Equation (3) from Section 2. Second, \mathcal{L}_{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 [2, 4, 57]—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 [27, 34] 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 5.3, pomsets with formulae accurately capture the concurrent behavior of these programs, while also allowing us to recover the precise probabilities of each outcome.

7 Discussion and Related Work

We developed a new semantic structure—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 using partial orders and pomsets in denotational models of concurrent programs [17, 18, 48], which have been used as a semantic basis for concurrent separation logic [8], concurrent Kleene algebras [22, 26, 27, 34], and other applications [14, 25, 29].

While there are links between pomset semantics and operational models of concurrency [7], pomsets are often preferred for their finer notion of concurrency compared to the interleaving semantics offered by operational models. This is especially useful in the context of weak memory [25, 29]. Probabilistic event structures model probabilistic process algebras [28, 65, 67, 68], whereas we model a full imperative language with sequential composition, control flow, and unbounded loops, for which adding formulae was essential.

Concurrency has previously been combined with other computational effects in limited ways. For example, semantics for probabilistic concurrent programs have been defined in both operational [58, 59] and denotational [73] styles, but these approaches are limited to programs with bounded looping constructs. 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 [38, 39] or metric [9] structure; prior work in this area was very informative to our own development, although the inclusion of deterministic branching in our own pomset structure added significant complexity.

An operational model of probabilistic concurrency has been developed based on Markov Decision Processes [15]. 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 [45], 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 [12–14]. 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 3.10) was necessary for linearization, without which we would not have been able to relate our pomset

model to the known convex powerset semantics [20, 37, 71].

Going forward, it would be interesting to explore the applicability of pomsets with formulae to convex powerdomain constructions other than Smyth [30, 60–62], or to other domains for mixing probabilities and nondeterminism including indexed valuations [63, 64, 66] and multisets of distributions [24, 32]. 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), 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 3.10. 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

- 1 Samson Abramsky and Achim Jung. *Domain theory*, page 1–168. Oxford University Press, Inc., USA, 1995.
- 2 Krzysztof Apt and Gordon Plotkin. Countable nondeterminism and random assignment. *J. ACM*, 33(4):724–767, aug 1986. doi:10.1145/6490.6494.
- 3 Steve Awodey. *Category Theory*. Oxford Logic Guides. Ebsco Publishing, 2006.
- 4 Ralph-Johan Back. Semantics of unbounded nondeterminism. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, pages 51–63, Berlin, Heidelberg, 1980. Springer Berlin Heidelberg. doi:10.1007/3-540-10003-2_59.
- 5 Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. An Assertion-Based Program Logic for Probabilistic Programs. In *Programming Languages and Systems*, pages 117–144, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-89884-1_5.
- 6 Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. Presenting Convex Sets of Probability Distributions by Convex Semilattices and Unique Bases. In Fabio Gadducci and Alexandra Silva, editors, *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, volume 211 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CALCO.2021.11.
- 7 Stephen Brookes. Traces, Pomsets, Fairness and Full Abstraction for Communicating Processes. In Luboš Brim, Mojmir Křetínský, Antonín Kučera, and Petr Jančar, editors, *CONCUR 2002 — Concurrency Theory*, pages 466–482, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45694-5_31.
- 8 Stephen Brookes. A semantics for concurrent separation logic. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory*, pages 16–34, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-28644-8_2.
- 9 J. W. de Bakker and J. H. A. Warmerdam. Metric pomset semantics for a concurrent language with recursion. In Irène Guessarian, editor, *Semantics of Systems of Concurrent Processes*, pages 21–49, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. doi:10.1007/3-540-53479-2_2.
- 10 Jerry den Hartog. Verifying Probabilistic Programs Using a Hoare like Logic. In P. S. Thiagarajan and Roland Yap, editors, *Advances in Computing Science — ASIAN’99*, pages 113–125, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. doi:10.1007/3-540-46674-6_11.
- 11 Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1(2):115–138, June 1971. doi:10.1007/BF00289519.
- 12 Luc Edixhoven and Sung-Shik Jongmans. Realisability of Branching Pomsets. In Silvia Lizeth Tapia Tarifa and José Proença, editors, *Formal Aspects of Component Software*, pages 185–204, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-20872-0_11.

- 13 Luc Edixhoven, Sung-Shik Jongmans, José Proença, and Ilaria Castellani. Branching pomsets: Design, expressiveness and applications to choreographies. *Journal of Logical and Algebraic Methods in Programming*, 136:100919, 2024. doi:[10.1016/j.jlamp.2023.100919](https://doi.org/10.1016/j.jlamp.2023.100919).
- 14 Luc Edixhoven, Sung-Shik Jongmans, José Proença, and Guillermina Cledou. Branching Pomsets for Choreographies. *Electronic Proceedings in Theoretical Computer Science*, 365:37–52, August 2022. doi:[10.4204/eptcs.365.3](https://doi.org/10.4204/eptcs.365.3).
- 15 Ira Fesefeldt, Joost-Pieter Katoen, and Thomas Noll. Towards Concurrent Quantitative Separation Logic. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory (CONCUR 2022)*, volume 243 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:24, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:[10.4230/LIPIcs.CONCUR.2022.25](https://doi.org/10.4230/LIPIcs.CONCUR.2022.25).
- 16 Robert W. Floyd. Assigning Meanings to Programs. In *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32, Providence, Rhode Island, 1967. American Mathematical Society. doi:[10.1007/978-94-011-1793-7_4](https://doi.org/10.1007/978-94-011-1793-7_4).
- 17 Jay L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61(2):199–224, 1988. doi:[10.1016/0304-3975\(88\)90124-7](https://doi.org/10.1016/0304-3975(88)90124-7).
- 18 Jan Grabowski. On Partial Languages. *Fundamenta Informaticae*, 4(2):427–498, 1981. doi:[10.3233/fi-1981-4210](https://doi.org/10.3233/fi-1981-4210).
- 19 Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent program. *ACM Trans. Program. Lang. Syst.*, 5(3):356–380, July 1983. doi:[10.1145/2166.357214](https://doi.org/10.1145/2166.357214).
- 20 Jifeng He, Karen Seidel, and Annabelle McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2):171–192, 1997. Formal Specifications: Foundations, Methods, Tools and Applications. doi:[10.1016/S0167-6423\(96\)00019-6](https://doi.org/10.1016/S0167-6423(96)00019-6).
- 21 Charles Antony Richard Hoare. An Axiomatic Basis for Computer Programming. *Commun. ACM*, 12(10):576–580, October 1969. doi:[10.1145/363235.363259](https://doi.org/10.1145/363235.363259).
- 22 Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene Algebra and its Foundations. *J. Log. Algebraic Methods Program.*, 80(6):266–296, 2011. doi:[10.1016/J.JLAP.2011.04.005](https://doi.org/10.1016/J.JLAP.2011.04.005).
- 23 Bart Jacobs. Coalgebraic trace semantics for combined possibilistic and probabilistic systems. *Electronic Notes in Theoretical Computer Science*, 203(5):131–152, 2008. Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science (CMCS 2008). doi:[10.1016/j.entcs.2008.05.023](https://doi.org/10.1016/j.entcs.2008.05.023).
- 24 Bart Jacobs. From Multisets over Distributions to Distributions over Multisets. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '21*, New York, NY, USA, 2021. Association for Computing Machinery. doi:[10.1109/LICS52264.2021.9470678](https://doi.org/10.1109/LICS52264.2021.9470678).
- 25 Radha Jagadeesan, Alan Jeffrey, and James Riely. Pomsets with preconditions: a simple model of relaxed memory. *Proc. ACM Program. Lang.*, 4(Oopsla), November 2020. doi:[10.1145/3428262](https://doi.org/10.1145/3428262).
- 26 Peter Jipsen and M. Andrew Moshier. Concurrent Kleene algebra with tests and branching automata. *Journal of Logical and Algebraic Methods in Programming*, 85(4):637–652, 2016. Relational and algebraic methods in computer science. doi:[10.1016/j.jlamp.2015.12.005](https://doi.org/10.1016/j.jlamp.2015.12.005).
- 27 Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. *Concurrent Kleene Algebra with Observations: From Hypotheses to Completeness*, page 381–400. Springer International Publishing, 2020. doi:[10.1007/978-3-030-45231-5_20](https://doi.org/10.1007/978-3-030-45231-5_20).
- 28 Joost-Pieter Katoen. *Quantitative and Qualitative Extensions of Event Structures*. Phd thesis - research ut, graduation ut, University of Twente, Netherlands, 1996. doi:[10.3990/1.9789036507998](https://doi.org/10.3990/1.9789036507998).
- 29 Ryan Kavanagh and Stephen Brookes. A Denotational Semantics for SPARC TSO. *Electronic Notes in Theoretical Computer Science*, 336:223–239, 2018. The Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII). doi:[10.1016/j.entcs.2018.03.025](https://doi.org/10.1016/j.entcs.2018.03.025).

- 30 Klaus Keimel and Gordon Plotkin. Mixed powerdomains for probability and nondeterminism. *Logical Methods in Computer Science*, Volume 13, Issue 1, January 2017. doi:[10.23638/LMCS-13\(1:2\)2017](https://doi.org/10.23638/LMCS-13(1:2)2017).
- 31 Dexter Kozen. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science (SFCS '79)*, pages 101–114, 1979. doi:[10.1109/SFCS.1979.38](https://doi.org/10.1109/SFCS.1979.38).
- 32 Dexter Kozen and Alexandra Silva. Multisets and distributions. In Venanzio Capretta, Robbert Krebbers, and Freek Wiedijk, editors, *Logics and Type Systems in Theory and Practice: Essays Dedicated to Herman Geuvers on The Occasion of His 60th Birthday*, pages 168–187, Cham, 2024. Springer Nature Switzerland. doi:[10.1007/978-3-031-61716-4_11](https://doi.org/10.1007/978-3-031-61716-4_11).
- 33 L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977. doi:[10.1109/TSE.1977.229904](https://doi.org/10.1109/TSE.1977.229904).
- 34 Michael R. Laurence and Georg Struth. Completeness Theorems for Bi-Kleene Algebras and Series-Parallel Rational Pomset Languages. In Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller, editors, *Relational and Algebraic Methods in Computer Science*, pages 65–82, Cham, 2014. Springer International Publishing. doi:[10.1007/978-3-319-06251-8_5](https://doi.org/10.1007/978-3-319-06251-8_5).
- 35 Daniel Lehmann and Michael O. Rabin. 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*, POPL '81, page 133–138, New York, NY, USA, 1981. Association for Computing Machinery. doi:[10.1145/567532.567547](https://doi.org/10.1145/567532.567547).
- 36 George Markowsky. Chain-complete posets and directed sets with applications. *Algebra Universalis*, 6:53–68, 12 1976. doi:[10.1007/BF02485815](https://doi.org/10.1007/BF02485815).
- 37 Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005. doi:[10.1007/b138392](https://doi.org/10.1007/b138392).
- 38 J. J. Ch. Meyer and E. P. de Vink. Pomset semantics for true concurrency with synchronization and recursion. In Antoni Kreczmar and Grazyna Mirkowska, editors, *Mathematical Foundations of Computer Science 1989*, pages 360–369, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. doi:[10.1007/3-540-51486-4_83](https://doi.org/10.1007/3-540-51486-4_83).
- 39 J. J. Ch. Meyer and E.P. de Vink. Applications of compactness in the smyth powerdomain of streams. *Theoretical Computer Science*, 57(2):251–282, 1988. doi:[10.1016/0304-3975\(88\)90042-4](https://doi.org/10.1016/0304-3975(88)90042-4).
- 40 Michael Mislove. Nondeterminism and probabilistic choice: Obeying the laws. In Catuscia Palamidessi, editor, *CONCUR 2000 — Concurrency Theory*, pages 350–365, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:[10.1007/3-540-44618-4_26](https://doi.org/10.1007/3-540-44618-4_26).
- 41 Michael Mislove, Joël Ouaknine, and James Worrell. Axioms for probability and non-determinism. *Electronic Notes in Theoretical Computer Science*, 96:7–28, 2004. Proceedings of the 10th International Workshop on Expressiveness in Concurrency. doi:[10.1016/j.entcs.2004.04.019](https://doi.org/10.1016/j.entcs.2004.04.019).
- 42 Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic Predicate Transformers. *ACM Trans. Program. Lang. Syst.*, 18(3):325–353, may 1996. doi:[10.1145/229542.229547](https://doi.org/10.1145/229542.229547).
- 43 Carroll Morgan, Annabelle McIver, Karen Seidel, and J. W. Sanders. Refinement-oriented probability for CSP. *Form. Asp. Comput.*, 8(6):617–647, nov 1996. doi:[10.1007/bf01213492](https://doi.org/10.1007/bf01213492).
- 44 Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, October 1978. doi:[10.1145/359619.359627](https://doi.org/10.1145/359619.359627).
- 45 Renato Neves. An adequacy theorem between mixed powerdomains and probabilistic concurrency, 2024. [arXiv:2409.15920](https://arxiv.org/abs/2409.15920).
- 46 Louis Parlant. *Monad Composition via Preservation of Algebras*. PhD thesis, University College London, 2020. URL: <https://discovery.ucl.ac.uk/id/eprint/10112228/>.
- 47 Gordon Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976. doi:[10.1137/0205035](https://doi.org/10.1137/0205035).

- 48 Vaughan R. Pratt. Semantical Considerations on Floyd-Hoare Logic. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 109–121, 1976. doi:[10.1109/SFCS.1976.27](https://doi.org/10.1109/SFCS.1976.27).
- 49 Vaughan R. Pratt. Modeling concurrency with partial orders. *Int. J. Parallel Program.*, 15(1):33–71, 1986. doi:[10.1007/BF01379149](https://doi.org/10.1007/BF01379149).
- 50 Michael O. Rabin. N-process synchronization by $4 \cdot \log_2 n$ -valued shared variable. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 407–410, 1980. doi:[10.1109/SFCS.1980.26](https://doi.org/10.1109/SFCS.1980.26).
- 51 Michael O. Rabin. The choice coordination problem. *Acta Inf.*, 17(2):121–134, June 1982. doi:[10.1007/BF00288965](https://doi.org/10.1007/BF00288965).
- 52 Wojciech Różowski, Tobias Kappé, Dexter Kozen, Todd Schmid, and Alexandra Silva. Probabilistic Guarded KAT Modulo Bisimilarity: Completeness and Complexity. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 136:1–136:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:[10.4230/LIPIcs.ICALP.2023.136](https://doi.org/10.4230/LIPIcs.ICALP.2023.136).
- 53 Dana Scott. Outline of a Mathematical Theory of Computation. Technical Report PRG02, OUCL, November 1970.
- 54 Dana Scott and Christopher Strachey. Toward A Mathematical Semantics For Computer Languages. Technical Report Prg06, OUCL, August 1971.
- 55 Michael Smyth. Power domains. *Journal of Computer and System Sciences*, 16(1):23–36, 1978. doi:[10.1016/0022-0000\(78\)90048-X](https://doi.org/10.1016/0022-0000(78)90048-X).
- 56 Michael Smyth. Finite approximation of spaces. In *Proceedings of a Tutorial and Workshop on Category Theory and Computer Programming*, page 225–241, Berlin, Heidelberg, 1986. Springer-Verlag. doi:[10.1007/3-540-17162-2_125](https://doi.org/10.1007/3-540-17162-2_125).
- 57 Harald Søndergaard and Peter Sestoft. Non-determinism in Functional Languages. *The Computer Journal*, 35(5):514–523, 10 1992. doi:[10.1093/comjnl/35.5.514](https://doi.org/10.1093/comjnl/35.5.514).
- 58 Joseph Tassarotti. *Verifying Concurrent Randomized Algorithms*. PhD thesis, Carnegie Mellon University, Dec 2018. URL: <https://csd.cmu.edu/academics/doctoral/degrees-conferred/joseph-tassarotti>.
- 59 Joseph Tassarotti and Robert Harper. A Separation Logic for Concurrent Randomized Programs. *Proc. ACM Program. Lang.*, 3(POPL), Jan 2019. doi:[10.1145/3290377](https://doi.org/10.1145/3290377).
- 60 Regina Tix. *Continuous D-cones: convexity and powerdomain constructions*. PhD thesis, Darmstadt University of Technology, Germany, 1999. URL: <https://d-nb.info/957239157>.
- 61 Regina Tix. Convex power constructions for continuous d-cones. *Electronic Notes in Theoretical Computer Science*, 35:206–229, 2000. Workshop on Domains IV. doi:[10.1016/S1571-0661\(05\)80746-7](https://doi.org/10.1016/S1571-0661(05)80746-7).
- 62 Regina Tix, Klaus Keimel, and Gordon Plotkin. Semantic domains for combining probability and non-determinism. *Electronic Notes in Theoretical Computer Science*, 222:3–99, 2009. doi:[10.1016/j.entcs.2009.01.002](https://doi.org/10.1016/j.entcs.2009.01.002).
- 63 Daniele Varacca. The powerdomain of indexed valuations. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 299–308, 2002. doi:[10.1109/LICS.2002.1029838](https://doi.org/10.1109/LICS.2002.1029838).
- 64 Daniele Varacca. *Probability, Nondeterminism and Concurrency: Two Denotational Models for Probabilistic Computation*. PhD thesis, University of Aarhus, Nov 2003. URL: <https://www.brics.dk/DS/03/14/>.
- 65 Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic event structures and domains. *Theoretical Computer Science*, 358(2):173–199, 2006. Concurrency Theory (CONCUR 2004). doi:[10.1016/j.tcs.2006.01.015](https://doi.org/10.1016/j.tcs.2006.01.015).
- 66 Daniele Varacca and Glynn Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16(1):87–113, 2006. doi:[10.1017/S0960129505005074](https://doi.org/10.1017/S0960129505005074).

- 67 Daniele Varacca and Nobuko Yoshida. Probabilistic π -Calculus and Event Structures. *Electronic Notes in Theoretical Computer Science*, 190(3):147–166, 2007. Proceedings of the Fifth Workshop on Quantitative Aspects of Programming Languages (QAPL 2007). doi:10.1016/j.entcs.2007.07.009.
- 68 Glynn Winskel. Probabilistic and Quantum Event Structures. In Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten, editors, *Horizons of the Mind. A Tribute to Prakash Panangaden: Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*, pages 476–497, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-06880-0_25.
- 69 Noam Zilberstein, Derek Dreyer, and Alexandra Silva. Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning. *Proc. ACM Program. Lang.*, 7(OOPSLA1), Apr 2023. doi:10.1145/3586045.
- 70 Noam Zilberstein, Daniele Gorla, and Alexandra Silva. Denotational Semantics for Probabilistic and Concurrent Programs (Full Version), 2025. arXiv:2503.02768.
- 71 Noam Zilberstein, Dexter Kozen, Alexandra Silva, and Joseph Tassarotti. A demonic outcome logic for randomized nondeterminism. *Proc. ACM Program. Lang.*, 9(POPL), Jan 2025. doi:10.1145/3704855.
- 72 Noam Zilberstein, Angelina Saliling, and Alexandra Silva. Outcome separation logic: Local reasoning for correctness and incorrectness with computational effects. *Proc. ACM Program. Lang.*, 8(OOPSLA1), Apr 2024. doi:10.1145/3649821.
- 73 Noam Zilberstein, Alexandra Silva, and Joseph Tassarotti. Probabilistic Concurrent Reasoning in Outcome Logic: Independence, Conditioning, and Invariants, 2024. arXiv:2411.11662.
- 74 Maaïke Zwart. *On the Non-Compositionality of Monads via Distributive Laws*. PhD thesis, University of Oxford, 2020. URL: <https://ora.ox.ac.uk/objects/uuid:b2222b14-3895-4c87-91f4-13a8d046febb>.
- 75 Maaïke Zwart and Dan Marsden. No-Go Theorems for Distributive Laws. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2019. doi:10.1109/lics.2019.8785707.

A Preliminaries and Omitted Definitions

A.1 Order Theory and Domain Theory

► **Definition A.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., $x \not\leq x$ 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:

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

and its set of immediate successors and predecessors as

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

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 nodes of a poset is given by:

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

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.

► **Definition A.2** (Downward Closure). *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 A.3** (Finitely Preceded). *A poset $\langle X, < \rangle$ is finitely preceded if there are finitely many elements smaller than each element; that is: $|x \downarrow| < \infty$, for all $x \in X$.*

Third, we define the level of elements in the poset.

► **Definition A.4** (Level [38]). *Given a finitely preceded poset $\langle X, < \rangle$, define $\text{lev}: X \rightarrow \mathbb{N}$:*

$$\text{lev}(x) \triangleq \sup \{n \mid \exists x_0, \dots, x_n \in X. x_0 \in \min \wedge x_n = x \wedge \forall i < n. x_{i+1} \in \text{succ}(x_i)\}$$

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

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

► **Definition A.5.** *The satisfaction relation $\models \subseteq (\text{nodes} \rightarrow \mathbb{B}) \times \text{form}$ for Boolean formulae is given below:*

$v \models \text{true}$	<i>always</i>
$v \models \text{false}$	<i>never</i>
$v \models \psi_1 \wedge \psi_2$	<i>iff</i> $v \models \psi_1$ and $v \models \psi_2$
$v \models \psi_1 \vee \psi_2$	<i>iff</i> $v \models \psi_1$ or $v \models \psi_2$
$v \models \neg \psi$	<i>iff</i> $v \not\models \psi$
$v \models x$	<i>iff</i> $v(x) = 1$

A.2 The Convex Powerdomain

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 this means that $\mu(\perp) \geq \nu(\perp)$ if $\mu \sqsubseteq_{\mathcal{D}} \nu$, and that 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 [37, Definition 5.4.3]. Now, the convex powerset is defined as follows:

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

Convex powersets are ordered via the Smyth order [55] $\sqsubseteq_{\mathcal{C}} \subseteq \mathcal{C}(X) \times \mathcal{C}(X)$, defined below:

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

Since the sets above are upward closed, the Smyth order collapses to reverse subset inclusion $S \sqsubseteq_{\mathcal{C}} T$ iff $S \supseteq T$. This makes the *convex powerdomain* $\langle \mathcal{C}(X), \sqsubseteq_{\mathcal{C}} \rangle$ a pointed DCPO with suprema given by set intersection and bottom $\perp_{\mathcal{C}} = \mathcal{D}(X_{\perp})$ being the set of all distributions, for the full proof refer to [71].

$$\begin{aligned}
\llbracket \text{skip} \rrbracket_{\mathcal{C}}(s) &\triangleq \eta(s) \\
\llbracket C_1; C_2 \rrbracket_{\mathcal{C}}(s) &\triangleq \llbracket C_2 \rrbracket_{\mathcal{C}}^{\dagger}(\llbracket C_1 \rrbracket_{\mathcal{C}}(s)) \\
\llbracket \text{if } b \text{ then } C_1 \text{ else } C_2 \rrbracket_{\mathcal{C}}(s) &\triangleq \begin{cases} \llbracket C_1 \rrbracket_{\mathcal{C}}(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \llbracket C_2 \rrbracket_{\mathcal{C}}(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases} \\
\llbracket \text{while } b \text{ do } C \rrbracket_{\mathcal{C}}(s) &\triangleq \text{lfip}(\Psi_{\langle \llbracket C \rrbracket_{\mathcal{C}}, b \rangle})(s) \\
\llbracket a \rrbracket_{\mathcal{C}}(s) &\triangleq \llbracket a \rrbracket_{\text{act}}(s)
\end{aligned}$$

■ **Figure 4** Convex powerset semantics $\llbracket - \rrbracket_{\mathcal{C}} : \text{cmd} \rightarrow \mathcal{S} \rightarrow \mathcal{C}(\mathcal{S})$ due to [20, 37, 71], where Ψ is defined in Lemma 5.2.

Finally, we give the definitions for the monad operations [23]: unit $\eta : X \rightarrow \mathcal{C}(X)$ and Kleisli extension $(-)^{\dagger} : (X \rightarrow \mathcal{C}(Y)) \rightarrow \mathcal{C}(X) \rightarrow \mathcal{C}(Y)$.

$$\eta(x) \triangleq [x \mapsto 1] \quad f^{\dagger}(S) \triangleq \left\{ \sum_{x \in \text{supp}(\mu)} \mu(x) \cdot \nu_x \mid \mu \in S, \forall x \in \text{supp}(\mu). \nu_x \in f_{\perp}(x) \right\}$$

where $(-)^{\perp} : (X \rightarrow \mathcal{C}(Y)) \rightarrow X^{\perp} \rightarrow \mathcal{C}(Y)$ is defined as follows: $f_{\perp}(x) \triangleq f(x)$ if $x \in X$ and $f_{\perp}(\perp) \triangleq \perp_{\mathcal{C}}$ otherwise.

As is required in Section 5.2, it is proven in [71] that the Kleisli extension for \mathcal{C} defined above is Scott continuous and additive:

$$\sup_{f \in D, x \in D'} f^{\dagger}(S) = (\sup D)^{\dagger}(\sup D') \quad f^{\dagger}(S \& T) = f^{\dagger}(S) \& f^{\dagger}(T)$$

Based on these operations, Figure 4 shows the convex powerset semantics for the parallel-free fragment of the language in (1), $\llbracket - \rrbracket_{\mathcal{C}} : \text{cmd} \rightarrow \mathcal{S} \rightarrow \mathcal{C}(\mathcal{S})$. This is precisely the same semantics used in prior work for reasoning about programs that are both nondeterministic and probabilistic [20, 37, 71]. This semantics does not capture parallel computation, which requires the more sophisticated semantic domains that we develop in this paper. The Scott continuity property above ensures that $\Psi_{\langle \llbracket C \rrbracket_{\mathcal{C}}, b \rangle} : (\mathcal{S} \rightarrow \mathcal{C}(\mathcal{S})) \rightarrow \mathcal{S} \rightarrow \mathcal{C}(\mathcal{S})$ is Scott continuous, and therefore the semantics of loops is well defined.

A.3 Pomset Languages

As we saw in Section 6, a pomset language is a set of finite pomsets without any tests or formulae. Control flow is determined by **assume** actions, which keep or eliminate traces depending on the whether the corresponding test passes or fails.

$$\text{pomLang} \triangleq \mathcal{P}(\text{pom}_{\text{fin}}(\text{label}_{\text{PL}}))$$

We give the pomset language semantics for a nondeterministic language in Figure 5.

B Operations on Pomset with Formulae: Examples

B.1 Sequential Composition

In Figure 6 we give three LPOFs and, for each of them, we provide the stuck nodes, the extensible ones, the branches, and the result of sequentially composing them with the singleton LPOF w . To lighten notation, we assume that α_1 , α_2 and α_3 have the same nodes

$$\begin{aligned}
\llbracket \text{skip} \rrbracket_{\text{PL}} &\triangleq \{\langle \text{fork} \rangle\} \\
\llbracket a \rrbracket_{\text{PL}} &\triangleq \{\langle a \rangle\} \\
\llbracket C_1; C_2 \rrbracket_{\text{PL}} &\triangleq \llbracket C_1 \rrbracket_{\text{PL}} \circ \llbracket C_2 \rrbracket_{\text{PL}} \\
\llbracket C_1 \mid C_2 \rrbracket_{\text{PL}} &\triangleq \llbracket C_1 \rrbracket_{\text{PL}} \parallel \llbracket C_2 \rrbracket_{\text{PL}} \\
\llbracket \text{if } b \text{ then } C_1 \text{ else } C_2 \rrbracket_{\text{PL}} &\triangleq (\{\langle \text{assume } b \rangle\} \circ \llbracket C_1 \rrbracket_{\text{PL}}) \cup (\{\langle \text{assume } \neg b \rangle\} \circ \llbracket C_2 \rrbracket_{\text{PL}}) \\
\llbracket \text{while } b \text{ do } C \rrbracket_{\text{PL}} &\triangleq \text{lfp}(\Xi_{\langle C, b \rangle})
\end{aligned}$$

$$\begin{aligned}
\text{where } \Xi_{\langle C, b \rangle}(S) &\triangleq (\{\langle \text{assume } b \rangle\} \circ \llbracket C \rrbracket_{\text{PL}} \circ S) \cup (\{\langle \text{assume } \neg b \rangle\} \circ \llbracket \text{skip} \rrbracket_{\text{PL}}) \\
S \circ T &\triangleq \{\alpha \circ \beta \mid \alpha \in S, \beta \in T\} \\
S \parallel T &\triangleq \{\alpha \parallel \beta \mid \alpha \in S, \beta \in T\}
\end{aligned}$$

■ **Figure 5** Standard pomset language semantics $\llbracket - \rrbracket_{\text{PL}} : \text{cmd} \rightarrow \text{pomLang}$.

and we only show the \perp labels (associated to z_1 and z_2 in α_1 and just to z_1 in α_2). In α_3 , there are no stuck nodes, since all maximal elements are non- \perp ; hence, all nodes are extensible and the branches include all the possible Boolean combinations of y_1 and y_2 (this requires 4 copies of w when building $\alpha_3 \circ w$). 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 (thus, we only need two copies of w here). Finally, in α_1 all nodes are stuck, since the leftmost branch will always lead to \perp and so the whole LPOF cannot be sequentially composed with anything else; for this reason, α_1 has no extensible node and no branch (thus, $\alpha_1 \circ w = \alpha_1$).

B.2 Unrolling Loops

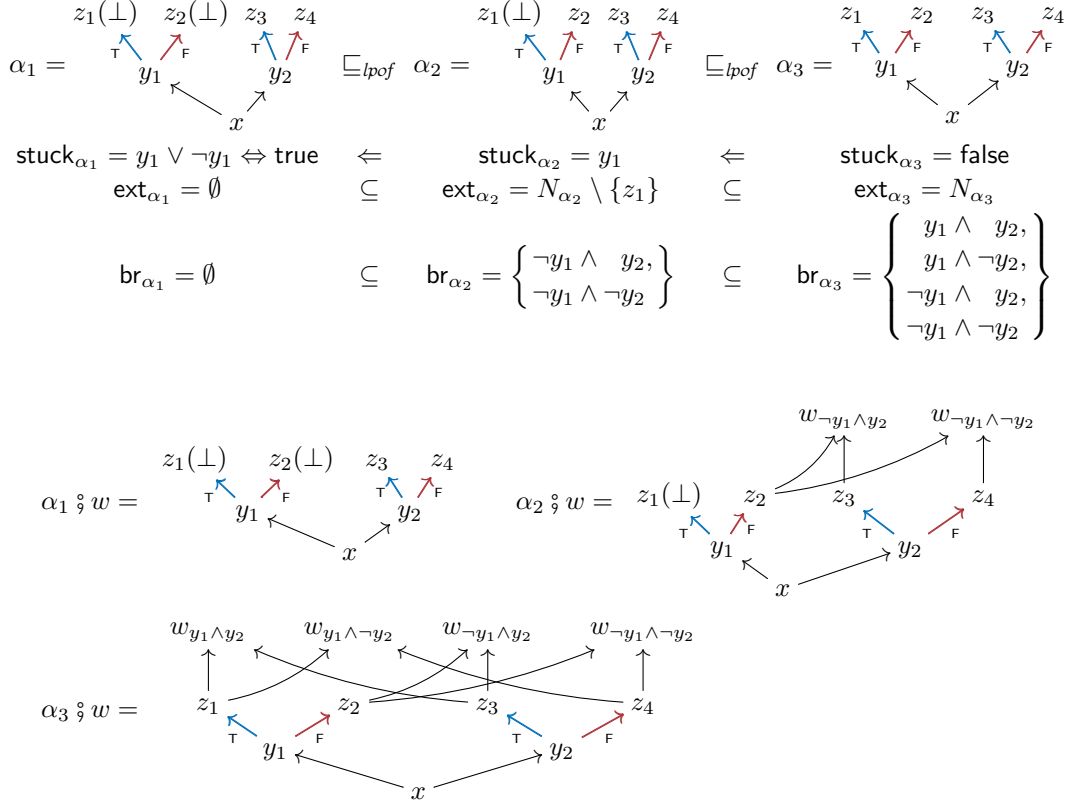
In this section, we discuss the rationale for the copying behavior of sequential composition by demonstrating the semantic structures that are generated as the result of looping programs. We start with a simple program, which is a loop containing a single action as the body:

while b **do** a

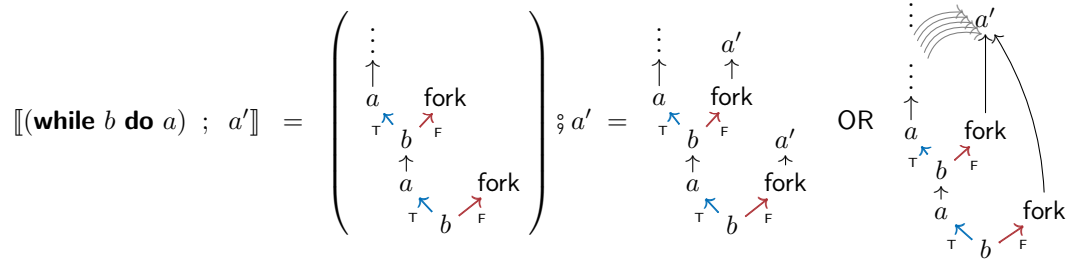
As defined in Figure 2, the semantics of this program is given by the following fixed point:

$$\llbracket \text{while } b \text{ do } a \rrbracket = \text{lfp}(\Phi_{\langle a, b \rangle}) \quad \text{where} \quad \Phi_{\langle a, b \rangle}(\alpha) = \text{guard}(b, \langle a \rangle \circ \alpha, \llbracket \text{skip} \rrbracket)$$

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 \perp_{pom} . Unrolling this definition several times, we get the chains shown in Section 5.1. Now suppose that we would like to sequentially compose another action after the loops, to obtain the program: **(while** b **do** a) ; a' . According to the definition of sequential composition from Section 4.2, the semantics of this program is obtained by composing a copy of a' after each branch of $\llbracket \text{while } b \text{ do } a \rrbracket$. As we just saw, the branches of $\llbracket \text{while } b \text{ do } a \rrbracket$ 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 [73]. However, this approach does not translate to the infinite case. To see why not, consider the second option below structure, obtained by



■ **Figure 6** An example of sequential composition and some of the functions used to define $;$.
 composing a' after **while** b **do** a , without copying.

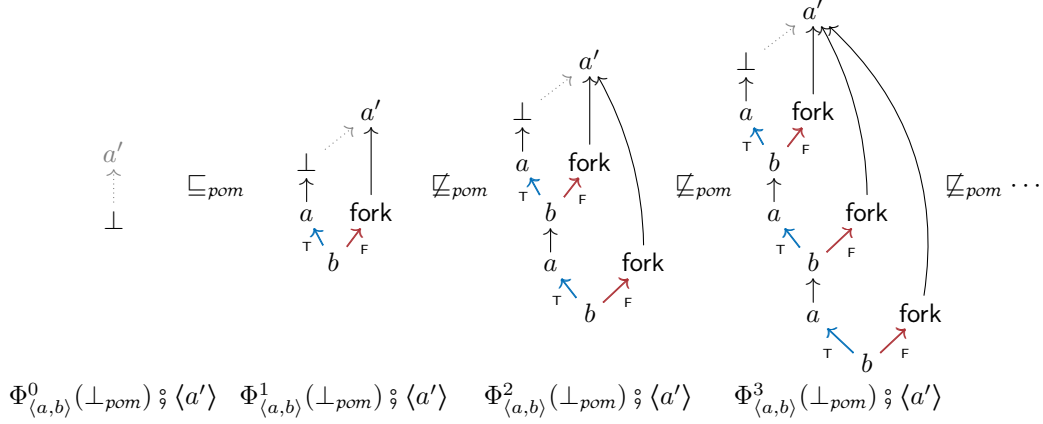


The node labelled as a' is not finitely preceded; it has countably many predecessors, one for each terminating branch of the loop. Relaxing the finitely preceded requirement for LPOFs is not an option; it breaks many of the proofs from Appendix E, preventing us from approximating a pomset using finite 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 $;$ means that the following equation holds:

$$\left(\sup_{n \in \mathbb{N}} \Phi_{\langle a, b \rangle}^n(\perp_{pom}) \right) ; \langle a' \rangle \stackrel{?}{=} \sup_{n \in \mathbb{N}} \left(\Phi_{\langle a, b \rangle}^n(\perp_{pom}) ; \langle a' \rangle \right)$$

We investigated two possible expansions for the latter expression. The first possibility retained the requirement that nothing can proceed \perp ; it is shown as the sequence below,

ignoring the grey dotted arrows. The second possibility allowed \perp nodes to have successors, and is shown below, where the grey dotted arrows indicate standard causality.



The causality from \perp to a' is problematic, since \perp could be expanded into an infinite structure, again causing a' to not be finitely preceded. But the lack of causality from \perp to a' is also problematic, as in the next approximation, \perp 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 DCPOs; 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 [52], where sequencing a command C after a guarded choice is equivalent to including C in both branches:

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

C Pomsets with Tests

C.1 Labelled Partial Orders

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

► **Lemma C.1.** *Let $\alpha \sqsubseteq_{lpof} \beta$; then, $\text{nBot}_\alpha \subseteq \text{nBot}_\beta$.*

Proof. By Def. 3.2 (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 $x \in \text{nBot}_\alpha$, then $\perp < \lambda_\alpha(x) \leq \lambda_\beta(x)$ and so $x \in \text{nBot}_\beta$. ◀

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

Proof. We show the set inclusion in both ways.

First, take any $x \in N_\beta \setminus N_\alpha$. Since x is finitely proceeded, there must be a positive integer n and nodes $x_1, \dots, x_n \in N_\beta$ such that x_1 is the root of β , $x_n = x$, and $x_{i+1} \in \text{succ}_\beta(x_i)$ for all $1 \leq i < n$. Since $\alpha \sqsubseteq_{\text{lpof}} \beta$, 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 $\alpha \sqsubseteq_{\text{lpof}} \beta$, we know that $\text{succ}_\alpha(x_i) = \text{succ}_\beta(x_i) \setminus \text{Bot}_\alpha \uparrow_\beta$; so, $x_{i+1} \in \text{Bot}_\alpha \uparrow_\beta$ that allows us to conclude $x \in \text{Bot}_\alpha \uparrow_\beta$, being $x_{i+1} <_\beta x_n = x$.

Now, take $x \in \text{Bot}_\alpha \uparrow_\beta$, i.e. there exists $z \in \text{Bot}_\alpha$ 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 $z \in \text{Bot}_\alpha$. \blacktriangleleft

► **Lemma C.3.** $\sqsubseteq_{\text{lpof}}$ is a partial order.

Proof. Reflexivity is trivial. For antisymmetry, we have that $\alpha \sqsubseteq_{\text{lpof}} \beta$ and $\beta \sqsubseteq_{\text{lpof}} \alpha$ 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 antisymmetry of \leq .

For transitivity, let us assume that $\alpha \sqsubseteq_{\text{lpof}} \beta$ and $\beta \sqsubseteq_{\text{lpof}} \gamma$, and prove that $\alpha \sqsubseteq_{\text{lpof}} \gamma$. 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 3.2,

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

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:

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

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

► **Lemma C.4.** Let $N_\beta \subseteq_\downarrow N_\alpha$ and $<_\beta = <_\alpha \cap (N_\beta \times N_\beta)$. Then:

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

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 $y \in \text{succ}_\beta(x)$. So, $y \in \text{succ}_\alpha(x)$.

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

► **Lemma C.5.** *If $D \subseteq \text{lpof}(L)$ 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 $\beta \sqsubseteq_{\text{lpof}} \gamma$ and $\beta' \sqsubseteq_{\text{lpof}} \gamma$. 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 $x \downarrow_\beta = x \downarrow_{\beta'}$. \blacktriangleleft

► **Lemma C.6.** *For any directed set $D \subseteq \text{lpof}(L)$ and $n \in \mathbb{N}$, there exists a finite set X such that $\{x \mid \text{lev}_\beta(x) = n\} \subseteq X$ 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 $\{y \mid \text{lev}_\beta(y) = 1\} = X$ 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 $\{x \mid \text{lev}_\beta(x) = k\} \subseteq X_k$ for each $k < n$. Also, let $X = \bigcup_{k < n} X_k$, so $\{x \mid \text{lev}_\beta(x) < n\} \subseteq X$, which is also finite. This means that only finitely many \perp nodes can appear before level n . Now construct A as follows: for each $x \in X$, select a $\beta \in D$ such that $\lambda_\beta(x) \neq \perp$, 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 $x \in X$, $x \notin \text{Bot}_\beta$ for some $\beta \in D$ iff $x \notin \text{Bot}_\gamma$.

Now let $Y = \{x \mid \text{lev}_\gamma(x) = n\}$. 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 $\beta \sqsubseteq_{\text{lpof}} \beta'$, then $\{x \mid \text{lev}_\beta(x) = n\} \subseteq Y$. \blacktriangleleft

► **Lemma 3.6.** *For any directed set $D \subseteq \text{lpof}(L)$, $\text{sup } D = \langle N, <, \lambda, \varphi \rangle$, where:*

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

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

Proof. We need to show that:

1. $\alpha \in \text{lpof}(L)$;
2. $\beta \sqsubseteq_{\text{lpof}} \alpha$, for all $\beta \in D$; and
3. $\alpha \sqsubseteq_{\text{lpof}} \gamma$, for all γ s.t. $\beta \sqsubseteq_{\text{lpof}} \gamma$ (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 proceeded. Take any $x \in N_\alpha$, so $x \in N_\beta$ for some $\beta \in D$. By Lemma C.5, 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 must be finite.

Next, we show that finitely many elements appear at level n . By Lemma C.6, we know that there is a finite set X such that $\{x \mid \text{lev}_\beta(x) = n\} \subseteq X$ for all $\beta \in D$. Following from

Lemma C.5, $\text{lev}_\beta(x) = \text{lev}_{\beta'}(x) = \text{lev}_\alpha(x)$ 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:

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

Now, take any $x \in N_\alpha$. If $\lambda_\alpha(x) = \perp$, then $\lambda_\beta(x) = \perp$ 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 C.5, $x \downarrow_\alpha = x \downarrow_\beta$; so, since $\text{vars}(\varphi_\beta(x)) \subseteq x \downarrow_\beta$, then $\text{vars}(\varphi_\alpha(x)) \subseteq x \downarrow_\alpha$.

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

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 $\beta' \sqsubseteq_{\text{lpo}} \gamma$ and therefore $y <_\gamma x$. Since N_β is a downward closed subset of N_γ (being $\beta \sqsubseteq_{\text{lpo}} \gamma$), it must be that $y \in N_\beta$.

For the order, we proceed as follows:

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

For the step (\ddagger) , 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 $\beta' \sqsubseteq_{\text{lpo}} \gamma$). Since $<_\beta = <_\gamma \cap (N_\beta \times N_\beta)$ (because $\beta \sqsubseteq_{\text{lpo}} \gamma$), 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 C.4(3), $\text{succ}_\beta(x) \subseteq \text{succ}_\alpha(x) \setminus \text{Bot}_\beta \uparrow_\alpha$. We now show the reverse inclusion. Suppose that $y \in \text{succ}_\alpha(x)$ and $y \notin \text{Bot}_\beta \uparrow_\alpha$; by construction, there must be some $\beta' \in D$ such that $x <_{\beta'} y$. By Lemma C.4(1), $y \in \text{succ}_{\beta'}(x)$. Since D is directed, there is a $\gamma \in D$ that is an upper bound of β and β' . Since $\beta' \sqsubseteq_{\text{lpo}} \gamma$, then $\text{succ}_{\beta'}(x) = \text{succ}_\gamma(x) \setminus \text{Bot}_{\beta'} \uparrow_\gamma$; so, $y \in \text{succ}_\gamma(x)$. Since $\beta \sqsubseteq_{\text{lpo}} \gamma$, then we also have $\text{succ}_\beta(x) = \text{succ}_\gamma(x) \setminus \text{Bot}_\beta \uparrow_\gamma$. Since $y \notin \text{Bot}_\beta \uparrow_\alpha$, then y also cannot be in $\text{Bot}_\beta \uparrow_\gamma$ since $<_\gamma \subseteq <_\alpha$ by construction; therefore, it must be that $y \in \text{succ}_\beta(x)$.

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

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,

$$<_\gamma \cap (N_\alpha \times N_\alpha) = <_\gamma \cap \left(\bigcup_{\beta \in D} N_\beta \times \bigcup_{\beta \in D} N_\beta \right)$$

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

where the step marked with (\dagger) 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 $\beta \sqsubseteq_{\text{lpof}} \gamma$ (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 C.4(3) (with α in place of β and γ in place of α therein), $\text{succ}_{\alpha}(x) \subseteq \text{succ}_{\gamma}(x) \setminus \text{Bot}_{\alpha} \uparrow_{\gamma}$. We now show the reverse inclusion too. Let $y \in \text{succ}_{\gamma}(x)$ and $y \notin \text{Bot}_{\alpha} \uparrow_{\gamma}$. Since y is finitely proceeded, we know that $y \downarrow_{\gamma}$ is finite, therefore $y \downarrow_{\gamma} \cap N_{\alpha}$ is finite and $\lambda_{\alpha}(z) \neq \perp$ for each $z \in y \downarrow_{\gamma} \cap N_{\alpha}$ since $y \notin \text{Bot}_{\alpha} \uparrow_{\gamma}$. 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 $\lambda_{\beta_z}(z) \neq \perp$, which must exist since $\lambda_{\alpha}(z) \neq \perp$. 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 $\lambda_{\beta}(z) \neq \perp$ for each $z \in y \downarrow_{\gamma} \cap N_{\alpha}$. Since $\beta \in D$, we also know that $\beta \sqsubseteq_{\text{lpof}} \gamma$ and, since $x <_{\gamma} y$ and $x \in N_{\alpha}$, then $x \in N_{\beta}$; hence, $\text{succ}_{\beta}(x) = \text{succ}_{\gamma}(x) \setminus \text{Bot}_{\beta} \uparrow_{\gamma}$, so either $y \in \text{succ}_{\beta}(x)$ or $y \in \text{Bot}_{\beta} \uparrow_{\gamma}$. In the first case, we are done since $\beta \sqsubseteq_{\text{lpof}} \alpha$ and therefore $\text{succ}_{\beta}(x) \subseteq \text{succ}_{\alpha}(x)$ by Lemma C.4(3). The second case is a contradiction, since it implies that there is some $z \in \text{Bot}_{\beta}$ 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 $\beta \sqsubseteq_{\text{lpof}} \alpha$. This implies that $\lambda_{\alpha}(z) \neq \perp$, i.e. $z \notin \text{Bot}_{\beta}$. ◀

► **Corollary C.7.** $\langle \text{lpof}(L), \sqsubseteq_{\text{lpof}} \rangle$ is a DCPO.

Proof. Follows immediately from Lemmas 3.6 and C.3. ◀

C.2 Truncation of Lpos

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

$$\text{lpof}_{\text{fin}}(L) \triangleq \{\alpha \in \text{lpof}(L) \mid |N_{\alpha}| < \infty\} \quad [\alpha]_{\text{fin}} \triangleq \{\beta \in \text{lpof}_{\text{fin}}(L) \mid \beta \sqsubseteq_{\text{lpof}} \alpha\}$$

► **Definition C.8** (Truncation). Let $\alpha \in \text{lpof}(L)$ and $n \in \mathbb{N}$. Then, $[\alpha]_n = \langle N, <, \lambda, \varphi \rangle$ where

- $N \triangleq \{x \in N_{\alpha} \mid \text{lev}_{\alpha}(x) \leq n\}$;
- $< \triangleq <_{\alpha} \cap (N \times N)$;
- for all $x \in N$, let $\lambda(x) \triangleq \lambda_{\alpha}(x)$, if $\text{lev}_{\alpha}(x) < n$, and $\lambda(x) \triangleq \perp$, 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 $\text{lev}_{\alpha}(x) < n$ and changing the labels of all nodes $x \in \text{lev}_{\alpha}^{-1}(n)$ to \perp . For any $\alpha \in \text{lpof}(L)$, $[\alpha]_0$ is a singleton LPOF with \perp at the root.

For a set of LPOFs $S \subseteq \text{lpof}(L)$, 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 C.9.** *For any $\alpha \in \text{lpof}(L)$ and $n > 0$, it holds that $\lfloor \alpha \rfloor_n \in \lfloor \alpha \rfloor_{\text{fin}}$.*

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 $\beta \in \text{lpof}(L)$. First, β is single rooted, is finitely proceeded 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 $\lambda_\beta(x) = \perp$, we consider two cases:

1. $\text{lev}_\alpha(x) < n$: by construction, $\lambda_\beta(x) = \lambda_\alpha(x)$ and so $\text{succ}_\alpha(x) = \emptyset$. By the definition of $<_\beta$, we have that $\text{succ}_\beta(x) \subseteq \text{succ}_\alpha(x)$ and so also $\text{succ}_\beta(x) = \emptyset$.
2. $\text{lev}_\alpha(x) = n$: obviously x has no successors, since everything above level n has been deleted.

Finally, we show that $\beta \sqsubseteq_{\text{lpof}} \alpha$. 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 $\text{lev}_\alpha(y) \leq n$, then $\text{lev}_\alpha(z) < n$ 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 C.4(3), $\text{succ}_\beta(y) \subseteq \text{succ}_\alpha(y) \setminus \text{Bot}_\beta \uparrow_\alpha$; we now show the reverse inclusion. Suppose that $z \in \text{succ}_\alpha(y)$ and $z \notin \text{Bot}_\beta \uparrow_\alpha$. Since $y \in N_\beta$, then $\text{lev}_\alpha(z) \leq n + 1$. However, it cannot be that $\text{lev}_\alpha(z) = n + 1$, since that would imply that there exists $w \in \text{pred}_\alpha(z)$ such that $\text{lev}_\alpha(w) = n$, but then $\lambda_\beta(w) = \perp$, which is a contradiction. So, $\text{lev}_\alpha(z) \leq n$, which means that $z \in N_\beta$. ◀

An immediate consequence of Lemma C.9 is that $\lfloor \alpha \rfloor_n \in \text{lpof}(L)$ and $\lfloor \alpha \rfloor_n \sqsubseteq_{\text{lpof}} \alpha$.

► **Lemma C.10** (Monotonicity of truncation). *If $\alpha \sqsubseteq_{\text{lpof}} \beta$, then $\lfloor \alpha \rfloor_n \sqsubseteq_{\text{lpof}} \lfloor \beta \rfloor_n$, for every $n \in \mathbb{N}$.*

Proof. By Lemma C.5, we have that $\text{lev}_\alpha(x) = \text{lev}_\beta(x)$, 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 C.5 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 , $\alpha \sqsubseteq_{\text{lpof}} \beta$, $N_{\lfloor \alpha \rfloor_n} \subseteq N_{\lfloor \beta \rfloor_n}$ and $N_{\lfloor \alpha \rfloor_n} \subseteq N_\alpha$.

Then, let $x \in N_{\lfloor \alpha \rfloor_n}$. By definition and $\alpha \sqsubseteq_{\text{lpof}} \beta$, 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 C.4(3), $\text{succ}_{\lfloor \alpha \rfloor_n}(x) \subseteq \text{succ}_{\lfloor \beta \rfloor_n}(x) \setminus \text{Bot}_{\lfloor \alpha \rfloor_n} \uparrow_{\lfloor \beta \rfloor_n}$; let us prove the reverse inclusion. So, let $y \in \text{succ}_{\lfloor \beta \rfloor_n}(x)$; this means that $\text{lev}_\beta(x) < n$ and $\text{lev}_\beta(y) \leq n$. By assumption, $x \in N_\alpha$. If also $y \in N_\alpha$, then $y \in \text{succ}_{\lfloor \alpha \rfloor_n}(x)$, because $\text{lev}_\beta(y) = \text{lev}_\alpha(y)$ and by Lemma C.4(1). If

$y \notin N_\alpha$, then $y \in \text{Bot}_\alpha \uparrow_\beta$, because $\alpha \sqsubseteq_{\text{tpof}} \beta$; this means that there exists $z \in \text{Bot}_\alpha$ such that $z <_\beta y$. Hence, $\text{lev}_\alpha(z) = \text{lev}_\beta(z) < \text{lev}_\beta(y) = \text{lev}_\alpha(y) \leq n$; thus, $z \in N_{[\alpha]_n} \subseteq N_{[\beta]_n}$ and so $z <_{[\beta]_n} y$. By definition of truncation, $\lambda_{[\alpha]_n}(z) = \lambda_\alpha(z) = \perp$; hence, we can conclude that $y \in \text{Bot}_{[\alpha]_n} \uparrow_{[\beta]_n}$. \blacktriangleleft

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

► **Lemma C.11** (Scott continuity of truncation). *If D is a directed set of LPOFs, then $\sup[D]_n$ exists and it is $[\sup D]_n$, for every $n \in \mathbb{N}$.*

Proof. By Lemma 3.6, $\sup D = \alpha$, where:

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

We now want to prove that γ , given by

$$N_\gamma \triangleq \bigcup_{\beta \in D} N_{[\beta]_n} \quad <_\gamma \triangleq \bigcup_{\beta \in D} <_{[\beta]_n} \quad \lambda_\gamma(x) \triangleq \sup_{\beta \in D : x \in N_{[\beta]_n}} \lambda_{[\beta]_n}(x) \quad \varphi_\gamma(x) \triangleq \psi_x$$

is the sup of $[D]_n$ and that it coincides with $[\alpha]_n$.

The fact that $\gamma = \sup[D]_n$ follows by Lemma 3.6, once we prove that $[D]_n$ is directed; so, take $\beta_1, \beta_2 \in [D]_n$ and prove that they admit an upper bound in $[D]_n$. By definition, there exist $\beta'_1, \beta'_2 \in D$ such that $\beta_1 = [\beta'_1]_n$ and $\beta_2 = [\beta'_2]_n$. Since D is directed, there exist a $\beta \in D$ such that $\beta'_1, \beta'_2 \sqsubseteq_{\text{tpof}} \beta$. By Lemma C.10, $\beta_1, \beta_2 \sqsubseteq_{\text{tpof}} [\beta]_n$ and, by definition, $[\beta]_n \in [D]_n$.

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

$$\begin{aligned} <_{[\alpha]_n} &= <_\alpha \cap (N_{[\alpha]_n} \times N_{[\alpha]_n}) \\ &= \left(\bigcup_{\beta \in D} <_\beta \right) \cap (N_{[\alpha]_n} \times N_{[\alpha]_n}) \\ &= \bigcup_{\beta \in D} (<_\beta \cap (N_{[\alpha]_n} \times N_{[\alpha]_n})) \\ &= \bigcup_{\beta \in D} ((<_\alpha \cap (N_\beta \times N_\beta)) \cap (N_{[\alpha]_n} \times N_{[\alpha]_n})) \\ &= \bigcup_{\beta \in D} (<_\alpha \cap ((N_\beta \times N_\beta) \cap (N_{[\alpha]_n} \times N_{[\alpha]_n}))) \\ &\stackrel{\dagger}{=} \bigcup_{\beta \in D} (<_\alpha \cap (N_{[\beta]_n} \times N_{[\beta]_n})) \\ &\stackrel{\ddagger}{=} \bigcup_{\beta \in D} <_{[\beta]_n} = <_\gamma \end{aligned}$$

Step (\dagger) holds because $N_{[\alpha]_n} = N_\gamma = \bigcup_{\beta' \in D} N_{[\beta']_n}$ and so $N_{[\alpha]_n} \cap N_\beta = (\bigcup_{\beta' \in D} N_{[\beta']_n}) \cap N_\beta = \bigcup_{\beta' \in D} (N_{[\beta']_n} \cap N_\beta) = N_{[\beta]_n}$, once we prove that $N_{[\beta']_n} \cap N_\beta \subseteq N_{[\beta]_n}$, for every $\beta' \in D$ (indeed, obviously $N_{[\beta]_n} \subseteq \bigcup_{\beta' \in D} (N_{[\beta']_n} \cap N_\beta)$). Fix $\beta' \neq \beta$ and $x \in N_{[\beta']_n} \cap N_\beta$; this means that $\text{lev}_{\beta'}(x) \leq n$, and so $\text{lev}_\beta(x) = \text{lev}_\alpha(x) = \text{lev}_{\beta'}(x) \leq n$, since $\beta, \beta' \sqsubseteq_{\text{tpof}} \alpha$. This allows us to conclude that $x \in N_{[\beta]_n}$, as desired. Step (\ddagger) is proved by double inclusion.

Trivially, $x <_{[\beta]_n} y$ means that $x <_\beta y$ and $\text{lev}_\beta(x), \text{lev}_\beta(y) \leq n$; hence, $x, y \in N_{[\beta]_n}$ and, since $\beta \sqsubseteq_{\text{lpo}} \alpha$, we also have $x <_\alpha y$, as desired. Conversely, if $x <_\alpha y$ and $x, y \in N_{[\beta]_n}$, then $x, y \in N_\beta$ and, since $\beta \sqsubseteq_{\text{lpo}} \alpha$, also $x <_\beta y$; thus, by definition, $x <_{[\beta]_n} y$.

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

D Pomsets

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

► **Lemma D.1.** *The following three statements are equivalent:*

1. $\alpha \sqsubseteq_{\text{pom}} \beta$;
2. $\exists \alpha \in \alpha, \beta \in \beta. \alpha \sqsubseteq_{\text{lpo}} \beta$; and
3. $\forall \beta \in \beta. \exists \alpha \in \alpha. \alpha \sqsubseteq_{\text{lpo}} \beta$

Proof. Clearly (1) implies (2) since α is nonempty.

We now show that (2) implies (3), assume that there exist $\alpha \in \alpha$ and $\beta \in \beta$ such that $\alpha \sqsubseteq_{\text{lpo}} \beta$. Now, take any $\beta' \in \beta$, so $\beta \equiv \beta'$ and therefore there exists a bijection $f: N_\beta \rightarrow N_{\beta'}$ that satisfies the properties in Definition 3.7. Since $\alpha \sqsubseteq_{\text{lpo}} \beta$, then $N_\alpha \subseteq N_\beta$, and so we can construct a bijection g such that $g(x) = f(x)$ for all $x \in N_\alpha$. Let $\alpha' \equiv \alpha$ be the lpo generated via the bijection g , so clearly $\alpha' \sqsubseteq_{\text{lpo}} \beta'$ since both are generated by the same bijection.

Now, we show that (3) implies (1), assume that for all $\beta \in \beta$ there exists an $\alpha \in \alpha$ such that $\alpha \sqsubseteq_{\text{lpo}} \beta$. Then, take any $\alpha' \in \alpha$; by definition, there exists a bijection $f: N_\alpha \rightarrow N_{\alpha'}$ that satisfies the properties in Definition 3.7. Now, consider the LPO β' obtained from β by replacing every $x \in N_\alpha$ with $f(x)$ and every $x \in N_\beta \setminus N_\alpha$ with a node that does not occur in $N_{\alpha'}$. Clearly, this induces an extension of f that is a bijection between N_β and $N_{\beta'}$ that satisfies the properties in Definition 3.7; hence, $\beta' \in \beta$ and, trivially, $\alpha' \sqsubseteq_{\text{lpo}} \beta'$. \blacktriangleleft

D.1 Truncation

As mentioned in Appendix C.2, we extend the truncation operation to pomsets in the expected way: $[\alpha]_n \triangleq \{[\alpha]_n \mid \alpha \in \alpha\}$.

► **Lemma D.2** (Monotonicity of Truncation). *If $\alpha \sqsubseteq_{\text{pom}} \beta$, then $\forall n \in \mathbb{N}. [\alpha]_n \sqsubseteq_{\text{pom}} [\beta]_n$.*

Proof. By Lemma D.1 we know that there exist $\alpha \in \alpha$ and $\beta \in \beta$ such that $\alpha \sqsubseteq_{\text{lpo}} \beta$; by Lemma C.10, this entails that $[\alpha]_n \sqsubseteq_{\text{lpo}} [\beta]_n$ and so, again by Lemma D.1, that $[\alpha]_n \sqsubseteq_{\text{pom}} [\beta]_n$. \blacktriangleleft

► **Lemma D.3.** *If $[\alpha]_n \sqsubseteq_{\text{pom}} \beta$ for all $n \in \mathbb{N}$, then $\alpha \sqsubseteq_{\text{pom}} \beta$.*

Proof. (Adapted from [9, Proposition 3.1.10]) Fix any $\alpha \in \alpha$ and $\beta \in \beta$. We will show that there exists $N \subseteq_\downarrow N_\beta$ and a bijection $f: N_\alpha \rightarrow N$ such that $f(\alpha) \sqsubseteq_{\text{lpo}} \beta$; therefore, by Lemma D.1, $\alpha \sqsubseteq_{\text{pom}} \beta$.

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 $f: N_{[\alpha]_n} \rightarrow X$ is a bijection such that $f([\alpha]_n) \sqsubseteq_{\text{lpo}} \beta$. We add

an edge from (f, X, n) to $(g, Y, n+1)$ if $X \subseteq Y$ and $g(x) = f(x)$ for all $x \in N_{[\alpha]_n}$ (i.e., f is g restricted to $N_{[\alpha]_n}$ or, equivalently, g is an extension of f). Let x_α and x_β be the roots of α and β respectively, and $f_0: \{x_\alpha\} \rightarrow \{x_\beta\}$ 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 $[\alpha]_n \sqsubseteq_{pom} \beta = [\beta]$, then there exists an $\alpha' \in \alpha$ such that $[\alpha']_n \sqsubseteq_{lpof} \beta$. Since $\alpha \equiv \alpha'$, then there must be a bijection $g: N_\alpha \rightarrow N_{\alpha'}$. Now, let $X = \{g(x) \mid x \in N_{[\alpha]_n}\}$ and $f: N_{[\alpha]_n} \rightarrow X$ be defined as $f(x) = g(x)$, which is clearly a bijection since g is. Clearly

$$f([\alpha]_n) = [g(\alpha)]_n = [\alpha']_n \sqsubseteq_{lpof} \beta$$

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 $f: N_{[\alpha]_{n+1}} \rightarrow X$ is a bijection. Now, let $Y = \{f(x) \mid x \in N_{[\alpha]_n}\}$. So, $g: N_{[\alpha]_n} \rightarrow Y$ defined as $g(x) = f(x)$ is also a bijection. We also know that $f([\alpha]_{n+1}) \sqsubseteq_{lpof} \beta$; since $g([\alpha]_n) \sqsubseteq_{lpof} f([\alpha]_{n+1})$, we have that $g([\alpha]_n) \sqsubseteq_{lpof} \beta$ 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 $k = |N_{[\alpha]_n}|$, 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 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 $f: N_\alpha \rightarrow X$ be defined as $f(x) = f_{\text{lev}_\alpha(x)}(x)$. We now show that $f(\alpha) \sqsubseteq_{lpof} \beta$. First, clearly $N_{f(\alpha)} = X \sqsubseteq_\downarrow N_\beta$ since it is the union of downward closed sets. Now, for the order, we have:

$$\begin{aligned} <_{f(\alpha)} &= \bigcup_{n \in \mathbb{N}} <_{f_n([\alpha]_n)} \\ &= \bigcup_{n \in \mathbb{N}} (<_\beta \cap (N_{f_n([\alpha]_n)} \times N_{f_n([\alpha]_n)})) \\ &= <_\beta \cap \bigcup_{n \in \mathbb{N}} (N_{f_n([\alpha]_n)} \times N_{f_n([\alpha]_n)}) \\ &= <_\beta \cap (N_{f(\alpha)} \times N_{f(\alpha)}) \end{aligned}$$

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

$$\begin{aligned} \lambda_{f(\alpha)}(x) &= \lambda_\alpha(f^{-1}(x)) \\ &= \lambda_\alpha(f_n^{-1}(x)) \\ &= \lambda_\alpha(f_{n+1}^{-1}(x)) \\ &= \lambda_{[\alpha]_{n+1}}(f_{n+1}^{-1}(x)) \\ &= \lambda_{f_{n+1}([\alpha]_{n+1})}(x) \\ &\leq \lambda_\beta(x) \end{aligned}$$

By an analogous argument, $\varphi_{f(\alpha)}(x) = \varphi_\beta(x)$. Now, by Lemma C.4, we know that $\text{succ}_{f(\alpha)}(x) \subseteq \text{succ}_\beta(x) \setminus \text{Bot}_{f(\alpha)} \uparrow_\beta$, it just remains to show the reverse inclusion. Take any $y \in \text{succ}_\beta(x)$ such that $y \notin \text{Bot}_{f(\alpha)} \uparrow_\beta$. Let $m = \text{lev}_\beta(y)$ and note that $y \notin \text{Bot}_{f_m(\lfloor \alpha \rfloor_m)}$ since $f(\alpha)$ and $f_m(\lfloor \alpha \rfloor_m)$ are identical up to level m . Since $f_m(\lfloor \alpha \rfloor_m) \sqsubseteq_{\text{lpof}} \beta$, then $y \in \text{succ}_{f_m(\lfloor \alpha \rfloor_m)}(x)$, and therefore $y \in \text{succ}_{f(\alpha)}(x)$. \blacktriangleleft

D.2 The Pomset Dcpo Structure

► **Lemma D.4.** *Let $D = \{\alpha_i \mid i \in I\} \subseteq \text{lpof}(L)$ be a directed set and let $\mathbf{D} = \{[\alpha_i] \mid i \in I\} \subseteq \text{pom}(L)$; then, $\sup \mathbf{D}$ exists and $\sup \mathbf{D} = [\sup D]$.*

Proof. By Lemma 3.6, we know that D admits a sup, let us call it α and let $\alpha = [\alpha]$; we have to prove that $\alpha = \sup \mathbf{D}$, i.e.,

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

The first property is trivial, thanks to Lemma D.1(2).

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

We first prove that $A_n \neq \emptyset$ for all $n \in \mathbb{N}$. Let $N = \{x \in N_\alpha \mid \text{lev}_\alpha(x) \leq n\}$, 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 $\alpha_{i_x} \sqsubseteq_{\text{lpof}} \alpha_j$ for all $x \in N$. By construction, $[\alpha_j]_n = [\alpha]_n$. Since $\alpha_j \in D$, we also know that $[\alpha_j] \sqsubseteq_{\text{pom}} \beta$, so there must be a $\beta \in \beta$ such that $\alpha_j \sqsubseteq_{\text{lpof}} \beta$. This gives us $[\alpha]_n = [\alpha_j]_n \sqsubseteq_{\text{lpof}} \alpha_j \sqsubseteq_{\text{lpof}} \beta$, 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 $\beta_n \in \beta$ such that $[\alpha]_n \sqsubseteq_{\text{lpof}} \beta_n$; therefore, by Lemma D.1, we get that $[\alpha]_n = [[\alpha]_n] \sqsubseteq_{\text{pom}} [\beta_n] = \beta$ so by Lemma D.3, we get that $\alpha \sqsubseteq_{\text{pom}} \beta$. \blacktriangleleft

► **Lemma D.5.** *For every transfinite chain of pomsets $(\alpha_\delta)_{\delta < \zeta}$ (such that $\alpha_\delta \sqsubseteq_{\text{pom}} \alpha_{\delta'}$ if $\delta < \delta'$), there exists a corresponding chain of LPOFs $(\alpha_\delta)_{\delta < \zeta}$ such that $\alpha_\delta \in \alpha_\delta$, for all $\delta < \zeta$.*

Proof. We construct the chain $(\alpha_\delta)_{\delta < \zeta}$ by transfinite induction as follows. First, fix an arbitrary $\alpha_0 \in \alpha_0$. Second, for any ordinal δ such that α_δ is fixed, we know that there exists an $\alpha_{\delta+1} \in \alpha_{\delta+1}$ such that $\alpha_\delta \sqsubseteq_{\text{lpof}} \alpha_{\delta+1}$; so, fix this element. 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 $\alpha_\xi \in \alpha_\xi$ that is bigger (w.r.t. $\sqsubseteq_{\text{lpof}}$) than any α_δ . By Lemma D.4, we know that $\sup_{\delta < \xi} \alpha_\delta = [\sup_{\delta < \xi} \alpha_\delta]$. By definition, α_ξ is an upper bound of $\{\alpha_\delta \mid \delta < \xi\}$, therefore $\sup_{\delta < \xi} \alpha_\delta \sqsubseteq_{\text{pom}} \alpha_\xi$; hence, there exists an $\alpha_\xi \in \alpha_\xi$ such that $\sup_{\delta < \xi} \alpha_\delta \sqsubseteq_{\text{lpof}} \alpha_\xi$, which also means that $\alpha_\delta \sqsubseteq_{\text{lpof}} \alpha_\xi$ for all $\delta < \xi$. \blacktriangleleft

► **Lemma D.6.** *$\langle \text{pom}(L), \sqsubseteq_{\text{pom}} \rangle$ is a pointed DCPO.*

Proof. By [1, Proposition 2.1.15], $\langle \text{pom}(L), \sqsubseteq_{\text{pom}} \rangle$ is a dcpo if any transfinite chain $(\alpha_\delta)_{\delta < \zeta}$ has a supremum. By Lemma D.5, there is a corresponding chain of lpos $(\alpha_\delta)_{\delta < \zeta}$ such that $\alpha_\delta \in \alpha_\delta$ for each $\delta < \zeta$. Therefore, by Lemma D.4, we get that $\sup_{\delta < \zeta} \alpha_\delta = [\sup_{\delta < \zeta} \alpha_\delta]$, so the chain has a supremum.

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

E

 Approximation and Extension

We start by recalling the notions of approximation and compactness taken from [1].

► **Definition E.1** (Approximation and Compactness). *Given a DCPO $\langle D, \leq \rangle$, the approximation order $\ll \subseteq D \times D$ is defined as follows: $d_1 \ll d_2$ iff, for any directed set $D' \subseteq D$ such that $d_2 \leq \sup D'$, then $d_1 \leq d$, for some $d \in D'$.*

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

The approximation order is sometimes called the *way-below* relation or the *order of definite refinement* [56]. Intuitively, $d_1 \ll d_2$ means that d_1 is a “simpler representation” of d_2 . We now want to show that such an order coincides with the notion of finite approximations (see Lemma 3.9); this requires some preliminary results for LPOFs.

► **Lemma E.2.** *For any $\alpha \in \text{lpof}(L)$, the set $\lfloor \alpha \rfloor_{\text{fin}}$ is directed.*

Proof. Take any $\beta, \gamma \in \lfloor \alpha \rfloor_{\text{fin}}$, so $\beta, \gamma \in \text{lpof}_{\text{fin}}(L)$ and $\beta, \gamma \sqsubseteq_{\text{lpof}} \alpha$. Let n be the maximum level of any node in β or γ , and $\alpha' = \lfloor \alpha \rfloor_{n+1}$. Clearly, $\alpha' \in \lfloor \alpha \rfloor_{\text{fin}}$ by Lemma C.9.

We conclude the proof by showing that α' is an upper bound for β and γ ; we will only show $\beta \sqsubseteq_{\text{lpof}} \alpha'$, 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 $\alpha' \sqsubseteq_{\text{lpof}} \alpha$ 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 $\text{lev}_\alpha(x) \leq n$. For the last requirement, by Lemma C.4(3) we have that $\text{succ}_\beta(x) \subseteq \text{succ}_{\alpha'}(x) \setminus \text{Bot}_\beta \uparrow_{\alpha'}$; we need to prove the reverse inclusion. Let $y \in \text{succ}_{\alpha'}(x)$ and $y \notin \text{Bot}_\beta \uparrow_{\alpha'}$. We have two cases:

- $y \in N_\beta$: in this case, $y \in \text{succ}_\beta(x)$ by Lemma C.4(1).
- $y \notin N_\beta$: Because y is finitely proceeded in α' , there exist y_1, \dots, y_n such that $y_1 = \min_{\alpha'}$, $y_n = y$ and $y_{i+1} \in \text{succ}_{\alpha'}(y_i)$. Because of Lemma C.4(2), $y_{i+1} \in \text{succ}_\alpha(y_i)$, 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 $\beta \sqsubseteq_{\text{lpof}} \alpha$, we have that $\text{succ}_\beta(y_i) \subseteq \text{succ}_\alpha(y_i) \setminus \text{Bot}_\beta \uparrow_\alpha$; so, $y_{i+1} \in \text{Bot}_\beta \uparrow_\alpha$ and, consequently, $y \in \text{Bot}_\beta \uparrow_\alpha$. This latter fact, together with $y \in N_{\alpha'}$, implies that $y \in \text{Bot}_\beta \uparrow_{\alpha'}$: contradiction. ◀

► **Lemma E.3.** *For any $\alpha \in \text{pom}(L)$, the set $\lfloor \alpha \rfloor_{\text{fin}}$ is directed.*

Proof. Take any $\beta, \beta' \in \lfloor \alpha \rfloor_{\text{fin}}$, so $\beta, \beta' \in \text{pom}_{\text{fin}}(L)$ and $\beta \sqsubseteq_{\text{pom}} \alpha$ and $\beta' \sqsubseteq_{\text{pom}} \alpha$. Now take any $\alpha \in \alpha$, by Lemma D.1 we know that there is a $\beta \in \beta$ and $\beta' \in \beta'$ such that $\beta \sqsubseteq_{\text{lpof}} \alpha$ and $\beta' \sqsubseteq_{\text{lpof}} \alpha$. Therefore, $\beta, \beta' \in \lfloor \alpha \rfloor_{\text{fin}}$, so by Lemma E.2, there exists a $\gamma \in \lfloor \alpha \rfloor_{\text{fin}}$ such that $\beta \sqsubseteq_{\text{lpof}} \gamma$ and $\beta' \sqsubseteq_{\text{lpof}} \gamma$. Since $\gamma \in \lfloor \alpha \rfloor_{\text{fin}}$, then $[\gamma] \in \lfloor \alpha \rfloor_{\text{fin}}$, and clearly $\beta \sqsubseteq_{\text{pom}} [\gamma]$ and $\beta' \sqsubseteq_{\text{pom}} [\gamma]$. ◀

► **Lemma E.4.** *For any $\alpha \in \text{lpof}(L)$, $\alpha = \sup [\alpha]_{\text{fin}}$.*

Proof. By Lemma E.2, $[\alpha]_{\text{fin}}$ is a directed set, therefore its supremum is given by the construction in Lemma 3.6. We now show that this gives us exactly α . First, we show this for the nodes. We know that $N_{\sup [\alpha]_{\text{fin}}} = \bigcup_{\beta \in [\alpha]_{\text{fin}}} N_\beta$. Clearly, $\bigcup_{\beta \in [\alpha]_{\text{fin}}} N_\beta \subseteq N_\alpha$ since each $N_\beta \subseteq N_\alpha$ by definition. For the reverse inclusion, take any $x \in N_\alpha$ and let $n = \text{lev}_\alpha(x)$. Obviously, $x \in N_{[\alpha]_n}$, and by Lemma C.9, we get that $[\alpha]_n \in [\alpha]_{\text{fin}}$. Therefore, $x \in \bigcup_{\beta \in [\alpha]_{\text{fin}}} N_\beta$, so $N_\alpha \subseteq \bigcup_{\beta \in [\alpha]_{\text{fin}}} N_\beta$ and thus $\bigcup_{\beta \in [\alpha]_{\text{fin}}} N_\beta = N_\alpha$.

By definition, $\beta \sqsubseteq_{\text{lpof}} \alpha$, for each $\beta \in [\alpha]_{\text{fin}}$; so clearly α is an upper bound for $[\alpha]_{\text{fin}}$ and therefore $\sup [\alpha]_{\text{fin}} \sqsubseteq_{\text{lpof}} \alpha$ (since the supremum is the least upper bound). Hence, $<_{\sup [\alpha]_{\text{fin}}} \subseteq <_\alpha$; we show the reverse inclusion as follows. Take any $x <_\alpha y$ and let $\gamma = [\alpha]_{\text{lev}_\alpha(y)}$. Clearly $y \in N_\gamma$ and by Lemma C.9, $\gamma \in [\alpha]_{\text{fin}}$. Since $N_\gamma \subseteq_\downarrow N_\alpha$, then $x \in N_\gamma$ too, which means that $x <_\gamma y$. This gives us:

$$(x, y) \in <_\gamma \subseteq \bigcup_{\beta \in [\alpha]_{\text{fin}}} <_\beta = <_{\sup [\alpha]_{\text{fin}}}$$

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

- $\varphi_{\sup [\alpha]_{\text{fin}}}(x) = \varphi_\alpha(x)$: Take any $\beta \in [\alpha]_{\text{fin}}$, so $\beta \sqsubseteq_{\text{lpof}} \sup [\alpha]_{\text{fin}}$ and $\beta \sqsubseteq_{\text{lpof}} \alpha$, therefore $\varphi_{[\alpha]_{\text{fin}}} = \varphi_\beta = \varphi_\alpha$.
-

$$\begin{aligned} \lambda_{\sup [\alpha]_{\text{fin}}}(x) &= \sup_{\beta \in [\alpha]_{\text{fin}} : x \in N_\beta} \lambda_\beta(x) \\ &= \sup \{ \lambda_\beta(x) \mid \beta \in \text{lpof}_{\text{fin}}(L), \beta \sqsubseteq_{\text{lpof}} \alpha, x \in N_\beta \} \\ &\stackrel{1}{=} \sup \{ \ell \mid \ell \leq \lambda_\alpha(x) \} \\ &\stackrel{2}{=} \lambda_\alpha(x) \end{aligned}$$

where (1) holds by the definition of $\sqsubseteq_{\text{lpof}}$, 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 3.9 (Finite Approximations).** *For any $\alpha \in \text{pom}(L)$, $\alpha = \sup [\alpha]_{\text{fin}}$, where $[\alpha]_{\text{fin}} \triangleq \{ \beta \in \text{pom}_{\text{fin}}(L) \mid \beta \sqsubseteq_{\text{pom}} \alpha \}$ is the set of finite approximations of α .*

Proof. Fix any $\alpha \in \alpha$ and consider the set $[\alpha]_{\text{fin}}$: by Lemma E.2 it is directed and by Lemma E.4 its sup is α . Then, by Lemma D.4, $\sup [\alpha]_{\text{fin}}$ exists and it is $[\alpha]$, i.e. α . ◀

► **Lemma E.5.** *If $\alpha \in \text{lpof}_{\text{fin}}(L)$, then there are only finitely many β such that $\beta \sqsubseteq_{\text{lpof}} \alpha$.*

Proof. Any $\beta \sqsubseteq_{\text{lpof}} \alpha$ 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 $\beta \sqsubseteq_{\text{lpof}} \alpha$. ◀

► **Lemma E.6.** *If $D \subseteq \text{pom}(L)$ is a directed set, then $[D]_n$ is directed and finite, for any $n \in \mathbb{N}$.*

Proof. We first show that $\lfloor D \rfloor_n$ is directed. Take any $\beta_1, \beta_2 \in \lfloor D \rfloor_n$. This means that there is $\gamma_1, \gamma_2 \in D$ such that $\beta_i = \lfloor \gamma_i \rfloor_n$ for each $i \in \{1, 2\}$, and since D is directed, then there is a $\gamma \in D$ such that each $\gamma_i \sqsubseteq_{pom} \gamma$. By monotonicity of truncation, we have that $\beta_i = \lfloor \gamma_i \rfloor_n \sqsubseteq_{pom} \lfloor \gamma \rfloor_n$, and clearly $\lfloor \gamma \rfloor_n \in \lfloor D \rfloor_n$, therefore $\lfloor D \rfloor_n$ is directed.

Because of Lemma D.2, we know that $\beta \sqsubseteq_{pom} \lfloor \alpha \rfloor_n$ for all $\beta \in \lfloor D \rfloor_n$. Fix any $\alpha \in \lfloor \alpha \rfloor_n$, so clearly α is finite. By Lemma D.1, we know that, for every $\beta \in \lfloor D \rfloor_n$, there is a $\beta \in \beta$ such that $\beta \sqsubseteq_{lpof} \alpha$. By Lemma E.5, there can only be finitely many such elements. Thus we have that $\{\beta \mid \beta \sqsubseteq_{lpof} \alpha\}$ is finite, and also clearly $\lfloor D \rfloor_n \subseteq \{\beta \mid \beta \sqsubseteq_{lpof} \alpha\}$; therefore, $\lfloor D \rfloor_n$ is also finite. \blacktriangleleft

► **Lemma E.7.** *If D is a directed set of pomsets, then $\sup \lfloor D \rfloor_n = \lfloor \sup D \rfloor_n$ for all $n \in \mathbb{N}$.*

Proof. We show that $\lfloor - \rfloor_n$ is chain continuous; indeed, by [36, Corollary 3], any chain continuous function on a DCPO is also Scott continuous, and thus $\sup \lfloor D \rfloor_n = \lfloor \sup D \rfloor_n$.

Take any pomset chain $(\alpha_\delta)_{\delta < \zeta}$; by Lemma D.5, there is a corresponding chain of LPOFs $(\alpha_\delta)_{\delta < \zeta}$ such that $\alpha_\delta \in \alpha_\delta$ for all $\delta < \zeta$. Since truncation is monotonic (Lemma D.2), then $(\lfloor \alpha_\delta \rfloor_n)_{\delta < \zeta}$ is also a chain and, by Lemma C.11, we know that $\sup_{\delta < \zeta} \lfloor \alpha_\delta \rfloor_n = \lfloor \sup_{\delta < \zeta} \alpha_\delta \rfloor_n$. Clearly $\lfloor \alpha_\delta \rfloor_n \in \lfloor \alpha_\delta \rfloor_n$ for all $\delta < \zeta$, therefore by Lemma D.4, we get both $\sup_{\delta < \zeta} \alpha_\delta = \lfloor \sup_{\delta < \zeta} \alpha_\delta \rfloor_n$ and $\sup_{\delta < \zeta} \lfloor \alpha_\delta \rfloor_n = \lfloor \sup_{\delta < \zeta} \alpha_\delta \rfloor_n$. Putting all these facts together, we get:

$$\sup_{\delta < \zeta} \lfloor \alpha_\delta \rfloor_n = \lfloor \sup_{\delta < \zeta} \alpha_\delta \rfloor_n = \lfloor \lfloor \sup_{\delta < \zeta} \alpha_\delta \rfloor_n \rfloor_n = \lfloor \sup_{\delta < \zeta} \alpha_\delta \rfloor_n \quad \blacktriangleleft$$

► **Lemma E.8 (Compactness).** $K(pom(L)) = pom_{fin}(L)$.

Proof. We first show that any compact pomset is finite, *i.e.*, $K(pom(L)) \subseteq pom_{fin}(L)$. Take any $\alpha \in K(pom(L))$, so for any directed set A , if $\alpha \sqsubseteq_{lpof} \sup A$ then $\alpha \sqsubseteq_{lpof} \gamma$ for some $\gamma \in A$. Now, let $A = \lfloor \alpha \rfloor_{fin}$, which is directed by Lemma E.3. By Lemma 3.9, we also know that $\alpha = \sup A$, so $\alpha \sqsubseteq_{pom} \sup A$. Since α is compact, there must be some $\gamma \in A$ such that $\alpha \sqsubseteq_{pom} \gamma$. Since $\gamma \in A$, then $\gamma \in pom_{fin}(L)$; therefore, since $\alpha \sqsubseteq_{lpof} \gamma$, then $\alpha \in pom_{fin}(L)$ too.

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

To this aim, let A be a directed set such that $\alpha \sqsubseteq_{pom} \sup A$. Since α is finite, let $\alpha \in \alpha$ be an arbitrary representative lpo, then $n = \max_{x \in N_\alpha} \text{lev}(x) + 1$ must be finite. Moreover, $\lfloor \alpha \rfloor_n = \alpha$, since n is one level above the maximum level of α and therefore the truncation only converts labels to \perp above the maximum level of α and removes nodes even above that. Therefore, by monotonicity (Lemma D.2) and Scott continuity (Lemma E.7) of truncation, we have:

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

By Lemma E.6, we know that $\{ \lfloor \beta \rfloor_n \mid \beta \in A \}$ is a finite directed set, and therefore it must contain its own supremum. That is, there exists $\gamma \in \{ \lfloor \beta \rfloor_n \mid \beta \in A \}$ such that $\gamma = \sup \{ \lfloor \beta \rfloor_n \mid \beta \in A \}$. This means that there exists a $\gamma' \in A$ such that $\gamma = \lfloor \gamma' \rfloor_n$, and so clearly $\gamma \sqsubseteq_{pom} \gamma'$, and therefore by transitivity, $\alpha \sqsubseteq_{pom} \gamma'$. \blacktriangleleft

► **Corollary E.9 (Approximation).** $\alpha \ll \beta$ iff $\alpha \in \lfloor \beta \rfloor_{fin}$.

Proof. Suppose that $\alpha \ll \beta$. Since $\lfloor \beta \rfloor_{fin}$ is a directed set and $\sup \lfloor \beta \rfloor_{fin} = \beta$ (Lemmas 3.9 and E.3), then there must be some $\gamma \in \lfloor \beta \rfloor_{fin}$ such that $\alpha \sqsubseteq_{lpof} \gamma$. Since $\lfloor \beta \rfloor_{fin}$ is downward closed by definition, then $\alpha \in \lfloor \beta \rfloor_{fin}$ too.

Now suppose that $\alpha \in [\beta]_{\text{fin}}$. By Lemma E.8, α is compact, so $\alpha \ll \alpha$. By the definition of $[\beta]_{\text{fin}}$, $\alpha \sqsubseteq_{\text{lpof}} \beta$ too. Therefore, by [1, Proposition 2.2.2(2)], we get that $\alpha \ll \beta$. ◀

For any DCPOs $\langle X, \leq_X \rangle$ 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 [1, Proposition 3.2.2]. We will use this fact in the next lemma. We write $\vec{x} \in X_1 \times \dots \times X_n$ to denote elements of Cartesian products. Let $\vec{\alpha} = (\alpha_1, \dots, \alpha_n)$ and $\vec{\beta} = (\beta_1, \dots, \beta_n)$. We write $\vec{\alpha} \ll \vec{\beta}$ to mean that $\alpha_i \ll \beta_i$ for all $1 \leq i \leq n$.

► **Lemma E.10.** *For any directed set $D \subseteq \text{pom}(X)^n$:*

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

Proof. We show the set inclusion in both directions. First, take any element $\vec{\beta} = (\beta_1, \dots, \beta_n)$ from the first set, so we know that $\beta_i \ll \alpha_i$ for all $1 \leq i \leq n$ and some $\vec{\alpha} = (\alpha_1, \dots, \alpha_n) \in D$. Clearly $\vec{\alpha} \sqsubseteq_{\text{pom}} \sup D$, so $\vec{\beta} \ll \sup D$ [1, Proposition 2.2.2(2)] and therefore the forward inclusion holds.

Now take any element $\vec{\beta} = (\beta_1, \dots, \beta_n)$ from the second set and let D_1, \dots, D_n be the projections of D . Since the supremum on the product order is coordinatewise [1, Proposition 2.2.2(2)], then $\sup D = (\sup D_1, \dots, \sup D_n)$. So, $\beta_i \ll \sup D_i$ for all $1 \leq i \leq n$. Therefore, by the definition of \ll , there must be some $\alpha_i \in D_i$ such that $\beta_i \sqsubseteq_{\text{pom}} \alpha_i$. Now, for each i , let $\vec{\alpha}_i \in D$ be the tuple whose i -th projection is α_i . Since D is directed, then there is some $\vec{\alpha} \in D$, which is an upper bound of all the $\vec{\alpha}_i$ s. From $\beta_i \ll \sup D_i$ and Corollary E.9, we know that β_i is finite, therefore $\beta_i \in [\alpha_i]_{\text{fin}}$. By Corollary E.9 again, $\beta_i \ll \alpha_i$, and so $\vec{\beta} \ll \alpha$ and the reverse inclusion holds too. ◀

► **Lemma 3.10 (Extension).** *Let $f: \text{pom}_{\text{fin}}(L)^n \rightarrow T$ be a monotone function on the DCPO $\langle T, \leq \rangle$. Then $f^*: \text{pom}(L)^n \rightarrow T$, shown below, is well-defined and Scott continuous:*

$$f^*(\alpha_1, \dots, \alpha_n) \triangleq \sup_{\alpha'_1 \ll \alpha_1} \dots \sup_{\alpha'_n \ll \alpha_n} f(\alpha'_1, \dots, \alpha'_n)$$

Proof. By Corollary E.9, $\{(\alpha'_1, \dots, \alpha'_n) \mid \forall i. \alpha'_i \ll \alpha_i\} = [\alpha_1]_{\text{fin}} \times \dots \times [\alpha_n]_{\text{fin}}$, that is a directed set by Lemma E.3. Since f is monotone, then $\{f(\alpha'_1, \dots, \alpha'_n) \mid \forall i. \alpha'_i \ll \alpha_i\}$ is also a directed set, and so clearly the supremum exists.

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

$$\begin{aligned} \sup_{\vec{\alpha} \in D} f^*(\vec{\alpha}) &= \sup_{\vec{\alpha} \in D} \sup_{\vec{\beta} \ll \vec{\alpha}} f(\vec{\beta}) \\ &= \sup \left\{ f(\vec{\beta}) \mid \exists \vec{\alpha} \in D. \vec{\beta} \ll \vec{\alpha} \right\} && \text{By [1, Proposition 2.1.4(3)].} \\ &= \sup \left\{ f(\vec{\beta}) \mid \vec{\beta} \ll \sup D \right\} && \text{By Lemma E.10.} \\ &= f^*(\sup D) \end{aligned}$$

◀

F Operations

F.1 Sequential Composition

► **Lemma F.1 (Antitonicity of stuck).** *If $\alpha \sqsubseteq_{\text{lpof}} \beta$, then $\text{stuck}_\beta \Rightarrow \text{stuck}_\alpha$.*

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

► **Lemma F.2** (Monotonicity of ext). *If $\alpha \sqsubseteq_{\text{lpo}} \beta$, then $\text{ext}_\alpha = \{x \in \text{ext}_\beta \mid \varphi_\beta(x) \not\Rightarrow \text{stuck}_\alpha\}$.*

Proof. We show the inclusion in both directions.

Take any $x \in \text{ext}_\alpha$, this means that $x \in N_\alpha$ and $\varphi_\alpha(x) \not\Rightarrow \text{stuck}_\alpha$; so, there is a valuation v such that $v \models \varphi_\alpha(x)$ and $v \not\models \text{stuck}_\alpha$. By Lemma F.1, $\text{stuck}_\beta \Rightarrow \text{stuck}_\alpha$, therefore v cannot satisfy stuck_β either. Since $\varphi_\alpha(x) = \varphi_\beta(x)$ (because $\alpha \sqsubseteq_{\text{lpo}} \beta$), we have that $\varphi_\beta(x) \not\Rightarrow \text{stuck}_\alpha$ and that $v \models \varphi_\beta(x)$ and $v \not\models \text{stuck}_\beta$; therefore, $\varphi_\beta(x) \not\Rightarrow \text{stuck}_\beta$ and so $x \in \text{ext}_\beta$.

For the reverse inclusion, take any $x \in \text{ext}_\beta$ such that $\varphi_\beta(x) \not\Rightarrow \text{stuck}_\alpha$. By Lemma C.2, we know that either $x \in N_\alpha$ or $x \in \text{Bot}_\alpha \uparrow_\beta$. In the former case, we know that $\varphi_\alpha(x) = \varphi_\beta(x) \not\Rightarrow \text{stuck}_\alpha$, therefore $x \in \text{ext}_\alpha$. The latter case is a contradiction, since $x \in \text{Bot}_\alpha \uparrow_\beta$ means that there is some $y \in \text{Bot}_\alpha$ such that $y <_\beta x$, which entails that $\varphi_\beta(x) \Rightarrow \varphi_\beta(y)$; however, $\varphi_\beta(y) = \varphi_\alpha(y) \Rightarrow \text{stuck}_\alpha$ and, by combining these facts, we get that $\varphi_\beta(x) \Rightarrow \text{stuck}_\alpha$. \blacktriangleleft

► **Lemma F.3** (Monotonicity of br). *If $\alpha \sqsubseteq_{\text{lpo}} \beta$, then $\text{br}_\alpha = \{\psi \in \text{br}_\beta \mid \psi \Rightarrow \neg \text{stuck}_\alpha\}$.*

Proof. We show the inclusion in both directions.

For the forward inclusion, take $\psi \in \text{br}_\alpha$, i.e. $\psi = \varphi_\alpha(S)$, for some $S \subseteq_{\text{fin}} \text{ext}_\alpha$ maximally satisfiable and that does not imply stuck_α ; we want to show that $\psi = \varphi_\beta(S) \in \text{br}_\beta$. By Lemma F.2, we know that $\text{ext}_\alpha \subseteq \text{ext}_\beta$, therefore $S \subseteq_{\text{fin}} \text{ext}_\beta$. Further, since $\varphi_\alpha(x) = \varphi_\beta(x)$ for all $x \in N_\alpha$, then $\varphi_\beta(S) = \varphi_\alpha(S) \Rightarrow \neg \text{stuck}_\alpha$; by using the contrapositive of Lemma F.1, we get that $\varphi_\beta(S) \Rightarrow \neg \text{stuck}_\beta$. Last, take any T such that $S \subset T \subseteq \text{ext}_\beta$. If $T \subseteq \text{ext}_\alpha$, we already know that $\varphi_\alpha(T)$ cannot be satisfiable; so suppose that there is some node $x \in T$ such that $x \notin \text{ext}_\alpha$, meaning that either $x \notin N_\alpha$ or $\varphi_\alpha(x) \Rightarrow \text{stuck}_\alpha$. In the first case, by Lemma C.2 $x \in \text{Bot}_\alpha \uparrow_\beta$, so there is a $z \in \text{Bot}_\alpha$ such that $z <_\beta x$, which also means that $\varphi_\beta(x) \Rightarrow \varphi_\beta(z) \Rightarrow \text{stuck}_\alpha$. So, in either case, we have $\varphi_\beta(x) \Rightarrow \text{stuck}_\alpha$, while, since $\varphi_\alpha(S) \in \text{br}_\alpha$, we know that $\varphi_\alpha(S) \Rightarrow \neg \text{stuck}_\alpha$. These give us:

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

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

For the reverse inclusion, take any $\psi \in \text{br}_\beta$ such that $\psi \Rightarrow \neg \text{stuck}_\alpha$. We know that $\psi = \varphi_\beta(S)$ where $S \subseteq \text{ext}_\beta$. By Lemma F.2, for each $x \in S$, either $x \in \text{ext}_\alpha$ or $\varphi_\beta(x) \Rightarrow \text{stuck}_\alpha$. But the latter case is not possible, otherwise we would be able to contradict $\varphi_\beta(S) = \psi \Rightarrow \neg \text{stuck}_\alpha$. Therefore, $S \subseteq \text{ext}_\alpha$. Now take any T such that $S \subset T \subseteq_{\text{fin}} \text{ext}_\alpha$. Since $\psi \in \text{br}_\beta$, 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, $\psi \in \text{br}_\alpha$. \blacktriangleleft

► **Lemma F.4** (Monotonicity of $\mathbin{\text{\textcircled{;}}}$ for LPOFs). *For all $\alpha, \alpha', \beta, \beta' \in \text{lpo}_{\text{fin}}(L)$ and $f \in \text{copy}_{\alpha', \beta'}$ such that $\alpha \sqsubseteq_{\text{lpo}} \alpha'$ and $\beta \sqsubseteq_{\text{lpo}} \beta'$, there exists $g \in \text{copy}_{\alpha, \beta}$ such that $\alpha \mathbin{\text{\textcircled{;}}}_g \beta \sqsubseteq_{\text{lpo}} \alpha' \mathbin{\text{\textcircled{;}}}_f \beta'$.*

Proof. Let $\beta'_\psi = f(\psi)$ for each $\psi \in \text{br}_{\alpha'}$. Since $\beta' \equiv f(\psi)$ for all ψ , then for each $\psi \in \text{br}_{\alpha'}$ there exists a bijection $f_\psi: N_{\beta'} \rightarrow N_{\beta'_\psi}$. Since $\beta \sqsubseteq_{\text{lpo}} \beta'$, 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_ψ s have pairwise disjoint nodes. By Lemma F.3, $\text{br}_\alpha \subseteq \text{br}_{\alpha'}$; so, we have such a β_ψ for all $\psi \in \text{br}_\alpha$, and we can define $g \in \text{copy}_{\alpha,\beta}$ as $g(\psi) = \beta_\psi$ for all $\psi \in \text{br}_\alpha$. Also, $\beta_\psi = g_\psi(\beta) \sqsubseteq_{\text{lpo}} f_\psi(\beta') = \beta'_\psi$ for all $\psi \in \text{br}_\alpha$, since $\beta \sqsubseteq_{\text{lpo}} \beta'$ and $g_\psi \subseteq f_\psi$.

We start with proving that $N_{\alpha \circledast_g \beta} \subseteq_\downarrow N_{\alpha' \circledast_f \beta'}$. Take any $x \in N_{\alpha \circledast_g \beta}$; so, either $x \in N_\alpha (\subseteq N_{\alpha'})$ or $x \in N_{\beta_\psi} (\subseteq N_{\beta'_\psi})$, for some $\psi \in \text{br}_\alpha$. Clearly $x \in N_{\alpha' \circledast_f \beta'}$, so the subset inclusion holds; we now show that it is downward closed. Take any $y <_{\alpha' \circledast_f \beta'} x$. If $x \in N_\alpha$, then clearly $y <_{\alpha'} x$, since \circledast does not add any causality into elements of $N_{\alpha'}$, and we therefore conclude that $y \in N_\alpha \subseteq N_{\alpha \circledast_g \beta}$, since $N_\alpha \subseteq_\downarrow N_{\alpha'}$. If instead $x \in N_{\beta_\psi}$ for some $\psi \in \text{br}_\alpha$, then we know by Lemma F.3 that $\psi \in \text{br}_{\alpha'}$ and therefore $x \in N_{\beta'_\psi} \subseteq N_{\alpha' \circledast_f \beta'}$. Since $y <_{\alpha' \circledast_f \beta'} x$, then either $y <_{\beta'_\psi} x$, or $\psi \Rightarrow \varphi_{\alpha'}(y)$. In the former case, then clearly $y \in N_{\beta_\psi} \subseteq N_{\alpha \circledast_g \beta}$ 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 C.2 there is some $z \in \text{Bot}_\alpha$ such that $z <_{\alpha'} y$, which also means that $\varphi_{\alpha'}(y) \Rightarrow \varphi_{\alpha'}(z)$. Since $\alpha \sqsubseteq_{\text{lpo}} \alpha'$ and $z \in \text{Bot}_\alpha$, we have that $\psi \Rightarrow \varphi_{\alpha'}(y) \Rightarrow \varphi_{\alpha'}(z) \Leftrightarrow \varphi_\alpha(z) \Rightarrow \text{stuck}_\alpha$; this contradicts $\psi \Rightarrow \neg \text{stuck}_\alpha$, that holds since $\psi \in \text{br}_\alpha$.

Next, we show the condition on the order:

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

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

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

Then, by Lemma F.3 $\text{br}_\alpha \subseteq \text{br}_{\alpha'}$, and, for all $\psi \in \text{br}_{\alpha'} \setminus \text{br}_\alpha$, the nodes of β'_ψ do not appear in $N_{\alpha \circledast_g \beta}$; hence, we can reduce the bounds of the union in (\dagger) and obtain:

$$\begin{aligned}
& \bigcup_{\psi \in \text{br}_{\alpha'}} (<_{\beta'_\psi} \cap (N_{\alpha \circledast_g \beta} \times N_{\alpha \circledast_g \beta})) \cup ((\{x \mid \psi \Rightarrow \varphi_{\alpha'}(x)\} \times N_{\beta'_\psi}) \cap (N_{\alpha \circledast_g \beta} \times N_{\alpha \circledast_g \beta})) \\
&= \bigcup_{\psi \in \text{br}_\alpha} (<_{\beta'_\psi} \cap (N_{\alpha \circledast_g \beta} \times N_{\alpha \circledast_g \beta})) \cup ((\{x \mid \psi \Rightarrow \varphi_{\alpha'}(x)\} \times N_{\beta'_\psi}) \cap (N_{\alpha \circledast_g \beta} \times N_{\alpha \circledast_g \beta})) \\
&= \bigcup_{\psi \in \text{br}_\alpha} (<_{\beta_\psi} \cap (N_{\beta_\psi} \times N_{\beta_\psi})) \cup (\{x \mid \psi \Rightarrow \varphi_{\alpha'}(x)\} \times N_{\beta_\psi}) \\
&= \bigcup_{\psi \in \text{br}_\alpha} (<_{\beta_\psi} \cup (\{x \mid \psi \Rightarrow \varphi_\alpha(x)\} \times N_{\beta_\psi}))
\end{aligned}$$

where the last two equalities hold because $\beta_\psi \sqsubseteq_{\text{lpo}} \beta'_\psi$ and because the nodes of β'_ψ are disjoint from those of α and of any other isomorphic copy of β . Thus, (\dagger) is equal to $<_\alpha \cup \bigcup_{\psi \in \text{br}_\alpha} (<_{\beta_\psi} \cup (\{x \mid \psi \Rightarrow \varphi_\alpha(x)\} \times N_{\beta_\psi}))$ that, by definition, is $<_{\alpha \circledast_g \beta}$.

Now, take any $x \in N_{\alpha \circ \beta}$. Since $\alpha \sqsubseteq_{\text{lpof}} \alpha'$ and $\beta_\psi \sqsubseteq_{\text{lpof}} \beta'_\psi$, 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:

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

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

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

For successors, we know by Lemma C.4(3) that $\text{succ}_{\alpha \circ \beta}(x) \subseteq \text{succ}_{\alpha' \circ \beta'}(x) \setminus \text{Bot}_{\alpha \circ \beta} \uparrow_{\alpha' \circ \beta'}$, we now must show the reverse inclusion. Take any $y \in \text{succ}_{\alpha' \circ \beta'}(x)$ such that $y \notin \text{Bot}_{\alpha \circ \beta} \uparrow_{\alpha' \circ \beta'}$. Given the way that $<_{\alpha' \circ \beta'}$ is constructed, there are three cases to consider:

1. $y \in \text{succ}_{\alpha'}(x)$. Since $\alpha \sqsubseteq_{\text{lpof}} \alpha'$, we know that $\text{succ}_\alpha(x) = \text{succ}_{\alpha'}(x) \setminus \text{Bot}_\alpha \uparrow_{\alpha'}$. However, $y \notin \text{Bot}_\alpha \uparrow_{\alpha'}$ (since $\text{Bot}_\alpha \uparrow_{\alpha'} \subseteq \text{Bot}_{\alpha \circ \beta} \uparrow_{\alpha' \circ \beta'}$); therefore, it must be that $y \in \text{succ}_\alpha(x)$, and so $y \in \text{succ}_{\alpha \circ \beta}(x)$.
2. $y \in \text{succ}_{\beta'_\psi}(x)$ for some $\psi \in \text{br}_\alpha$. 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 $y' \in \text{succ}_{\beta'}(x')$. Since $\beta \sqsubseteq_{\text{lpof}} \beta'$, then $\text{succ}_\beta(x') = \text{succ}_{\beta'}(x') \setminus \text{Bot}_\beta \uparrow_{\beta'}$. However, since $y \notin \text{Bot}_{\alpha \circ \beta} \uparrow_{\alpha' \circ \beta'}$, then it must be that $y' \notin \text{Bot}_\beta \uparrow_{\beta'}$; so we must have that $y' \in \text{succ}_\beta(x')$. Since $x', y' \in N_\beta$, then $g_\psi(x') = f_\psi(x') = x$ and $g_\psi(y') = f_\psi(y') = y$; so, $y \in \text{succ}_{\beta_\psi}(x)$ and therefore $y \in \text{succ}_{\alpha \circ \beta}(x)$.
3. $x \in N_\alpha$ and $y \in N_{\beta'_\psi}$ for some $\psi \in \text{br}_{\alpha'}$. By Lemma F.3, either $\psi \in \text{br}_\alpha$ or $\psi \not\sqsubseteq \neg \text{stuck}_\alpha$. For the first case, $x <_{\alpha' \circ \beta'} y$, $x \in N_{\alpha'}$ 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 $\beta_\psi \sqsubseteq_{\text{lpof}} \beta'_\psi$, then y must also be the root of β_ψ ; so, $x <_{\alpha \circ \beta} y$ and, by Lemma C.4(1), $y \in \text{succ}_{\alpha \circ \beta}(x)$.

To conclude, we show that the second case is impossible. Suppose that $\psi \not\sqsubseteq \neg \text{stuck}_\alpha$. From $\psi \in \text{br}_{\alpha'}$ we also know that $\psi \Rightarrow \neg \text{stuck}_{\alpha'}$. Therefore, there is some v such that $v \models \psi$, $v \models \text{stuck}_\alpha$ and $v \not\models \text{stuck}_{\alpha'}$. From $v \models \text{stuck}_\alpha$, we get that $v \models \varphi_\alpha(z)$, for some $z \in \text{Bot}_\alpha$; so $v \models \varphi_{\alpha'}(z)$ too (since $\varphi_\alpha(z) = \varphi_{\alpha'}(z)$). Since $v \models \varphi_{\alpha'}(z)$ and $v \not\models \text{stuck}_{\alpha'}$, then $\varphi_{\alpha'}(z) \not\sqsubseteq \text{stuck}_{\alpha'}$; therefore $z \in \text{ext}_{\alpha'}$. Moreover, since $v \models \psi$ and $v \models \varphi_{\alpha'}(z)$, 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 $z <_{\alpha' \circ \beta'} y$ (by the definition of $<_{\alpha' \circ \beta'}$), in contradiction with $y \notin \text{Bot}_{\alpha \circ \beta} \uparrow_{\alpha' \circ \beta'}$. \blacktriangleleft

► **Lemma F.5** (Monotonicity of \circ). *If $\alpha \sqsubseteq_{\text{pom}} \alpha'$ and $\beta \sqsubseteq_{\text{pom}} \beta'$, then $\alpha \circ \beta \sqsubseteq_{\text{pom}} \alpha' \circ \beta'$*

Proof. Take any $\gamma \in \alpha' \circ \beta'$; by construction, $\gamma = \alpha' \circ_f \beta'$, for some $\alpha' \in \alpha'$, $\beta' \in \beta'$, and $f \in \text{copy}_{\alpha', \beta'}$. By Lemma D.1(3), there exist $\alpha \in \alpha$ and $\beta \in \beta$ such that $\alpha \sqsubseteq_{\text{lpof}} \alpha'$ and $\beta \sqsubseteq_{\text{lpof}} \beta'$. So, by Lemma F.4 there exists a $g \in \text{copy}_{\alpha, \beta}$ such that $\alpha \circ_g \beta \sqsubseteq_{\text{lpof}} \alpha' \circ_f \beta'$. So, by Lemma D.1(3) $\alpha \circ \beta \sqsubseteq_{\text{pom}} \alpha' \circ \beta'$. \blacktriangleleft

► **Corollary F.6.** \circ is Scott continuous.

Proof. Immediate from Lemmas 3.10 and F.5. \blacktriangleleft

F.2 Guarded Branching

► **Lemma F.7** (Monotonicity of guard for LPOFs). *For every $\ell \in L$, $x \in \text{nodes}$ and $\alpha, \alpha', \beta, \beta' \in \text{lpoF}_{\text{fin}}(L)$ such that $\alpha \sqsubseteq_{\text{lpoF}} \alpha'$, $\beta \sqsubseteq_{\text{lpoF}} \beta'$, $N_{\alpha'} \cap N_{\beta'} = \emptyset$ and $x \notin N_{\alpha'} \cup N_{\beta'}$, it holds that $\text{guard}(x, \ell, \alpha, \beta) \sqsubseteq_{\text{lpoF}} \text{guard}(x, \ell, \alpha', \beta')$.*

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

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 $\text{pred}_{\gamma'}(x) = \emptyset$. 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,

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

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

1. $y = x$: by construction, $\lambda_{\gamma}(y) = \lambda_{\gamma'}(y) = \ell$ and $\varphi_{\gamma}(y) = \varphi_{\gamma'} = \text{true}$.
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 $\text{succ}_{\gamma}(y) \supseteq \text{succ}_{\gamma'}(y) \setminus \text{Bot}_{\gamma} \uparrow_{\gamma'}$ (since the other inclusion holds for Lemma C.4). Thus, take $z \in \text{succ}_{\gamma'}(y)$; we have three cases to consider:

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

► **Lemma F.8** (Monotonicity of guard). *If $\alpha \sqsubseteq_{\text{pom}} \alpha'$ and $\beta \sqsubseteq_{\text{pom}} \beta'$, then $\text{guard}(\ell, \alpha, \beta) \sqsubseteq_{\text{pom}} \text{guard}(\ell, \alpha', \beta')$.*

Proof. Take any $\gamma \in \text{guard}(\ell, \alpha', \beta')$; by construction, $\gamma = \text{guard}(x, \ell, \alpha', \beta')$, for some $\alpha' \in \alpha'$ and $\beta' \in \beta'$ such that $N_{\alpha'} \cap N_{\beta'} = \emptyset$ and $x \notin N_{\alpha'} \cup N_{\beta'}$. By Lemma D.1(3), there exist $\alpha \in \alpha$ and $\beta \in \beta$ such that $\alpha \sqsubseteq_{\text{lpoF}} \alpha'$ and $\beta \sqsubseteq_{\text{lpoF}} \beta'$. So, by Lemma F.7 $\text{guard}(x, \ell, \alpha, \beta) \sqsubseteq_{\text{lpoF}} \text{guard}(x, \ell, \alpha', \beta')$. So, by Lemma D.1(3) $\text{guard}(\ell, \alpha, \beta) \sqsubseteq_{\text{pom}} \text{guard}(\ell, \alpha', \beta')$. ◀

► **Corollary F.9.** *guard is Scott continuous.*

Proof. Immediate from Lemmas 3.10 and F.8. ◀

G

 Denotational Semantics

G.1 Linearization

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

Proof. We show the set inclusion in both directions. Take any $y \in \text{next}(\alpha, \psi, S)$, 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 $y \in \text{next}(\beta, \psi, S)$.

For the reverse inclusion, take any $y \in \text{next}(\beta, \psi, S)$, 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 C.2, we get that $y \in \text{Bot}_\alpha \uparrow_\beta$. So there is a $z \in \text{Bot}_\alpha$ such that $z <_\beta y$. Since $y \downarrow_\beta \subseteq S$, then $z \in S$. However, this is a contradiction since we assumed that $S \cap \text{Bot}_\alpha = \emptyset$. ◀

For any set X , the pointwise extension $\langle X \rightarrow D(S), \sqsubseteq^\bullet \rangle$ 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.

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

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

Proof. Let $T_{S,\psi} = \{x \in N_\alpha \mid x \notin S, \text{sat}(\psi \wedge \varphi_\alpha(x))\}$ be the set of nodes still to schedule. The proof is by induction on the size of the set $T_{S,\psi}$.

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

Now suppose the claim holds for all sets smaller than $T_{S,\psi}$, and let f be defined as follows:

$$f(\gamma, \psi, S, x)(s) = \begin{cases} \mathcal{L}_{lpof}(\gamma, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_\gamma(x) \rrbracket_{act}(s)) & \text{if } \lambda_\gamma(x) \in act \\ \mathcal{L}_{lpof}(\gamma, \psi \wedge \langle x = \llbracket \lambda_\gamma(x) \rrbracket \rangle, S \cup \{x\})(s) & \text{if } \lambda_\gamma(x) \in test \\ \perp & \text{if } \lambda_\gamma(x) = \perp \\ \mathcal{L}_{lpof}(\gamma, \psi, S \cup \{x\})(s) & \text{if } \lambda_\gamma(x) = fork \end{cases}$$

So, $\mathcal{L}_{lpof}(\gamma, \psi, S)(s) = \bigwedge_{x \in \text{next}(\gamma, \psi, S)} f(\gamma, \psi, S, x)(s)$. We start by showing that $f(\alpha, \psi, S, x)(s) \sqsubseteq f(\beta, \psi, S, x)(s)$. First, we remark that $S \cap \text{Bot}_\alpha = \emptyset$ and so, if $x \notin \text{Bot}_\alpha$, then $(S \cup \{x\}) \cap \text{Bot}_\alpha = \emptyset$, 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 $\llbracket \lambda_\alpha(x) \rrbracket_{act}(s) \sqsubseteq \llbracket \lambda_\beta(x) \rrbracket_{act}(s)$. Note that $T_{S \cup \{x\}, \psi} \subset T_{S, \psi}$, 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:

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

2. If $\lambda_\alpha(x) \in \text{test}$, then again $\lambda_\beta(x) \in \text{test}$ as well and $\lambda_\alpha(x) = \lambda_\beta(x)$ since the order over tests is flat. Since $T_{S \cup \{x\}, \psi \wedge \llbracket x = \llbracket b \rrbracket_{\text{test}}(s) \rrbracket} \subset T_{S, \psi}$, then we can use the induction hypothesis to conclude that:

$$\begin{aligned} f(\alpha, \psi, S, x)(s) &= \mathcal{L}_{\text{lpof}}(\alpha, \psi \wedge \llbracket x = \llbracket \lambda_\alpha(x) \rrbracket_{\text{test}}(s) \rrbracket, S \cup \{x\})(s) \\ &\sqsubseteq \mathcal{L}_{\text{lpof}}(\beta, \psi \wedge \llbracket x = \llbracket \lambda_\beta(x) \rrbracket_{\text{test}}(s) \rrbracket, S \cup \{x\})(s) \\ &= f(\beta, \psi, S, x)(s) \end{aligned}$$

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

$$f(\alpha, \psi, S, x)(s) = \perp \sqsubseteq f(\beta, \psi, S, x)(s)$$

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

$$f(\alpha, \psi, S, x)(x) = \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})(x) \sqsubseteq \mathcal{L}_{\text{lpof}}(\beta, \psi, S \cup \{x\})(x) = f(\beta, \psi, S, x)(x)$$

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

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

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

$$= \bigotimes_{x \in \text{next}(\beta, \psi, S)} f(\alpha, \psi, S, x)(s)$$

Since $\&$ is monotone, and given the claim we just proved about f :

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

► **Lemma G.3** (Monotonicity of \mathcal{L}_{fin}). *If $\alpha \sqsubseteq_{\text{lpof}} \beta$, then $\mathcal{L}_{\text{fin}}(\alpha) \sqsubseteq^\bullet \mathcal{L}_{\text{fin}}(\beta)$.*

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

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

► **Corollary G.4.** $\mathcal{L}: \text{pom} \rightarrow (\mathcal{S} \rightarrow D(\mathcal{S}))$ is Scott continuous.

Proof. Immediate from Lemmas 3.10 and G.3.

G.2 Properties of Linearization: Skip and Actions

► **Lemma G.5.** $\mathcal{L}(\llbracket \text{skip} \rrbracket) = \eta$ and $\mathcal{L}(\llbracket a \rrbracket) = \llbracket a \rrbracket_{\text{act}}$.

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

$$\mathcal{L}(\llbracket \text{skip} \rrbracket)(s) = \mathcal{L}(\langle \text{fork} \rangle)(s)$$

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

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

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

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

◀

G.3 Properties of Linearization: Sequential Composition

We first introduce a few definitions:

$$\begin{aligned} \text{next}^*(\alpha, \psi, S) &= \{x \in N_\alpha \setminus S \mid \text{sat}(\psi \wedge \varphi_\alpha(x))\} \\ \text{tests}_\alpha(S) &= \{x \in S \mid \lambda_\alpha(x) \in \text{test}\} \\ \text{forms}(X) &= \left\{ \bigwedge_{x \in X} \langle x = b_x \rangle \mid \forall x \in X. b_x \in \mathbb{B} \right\} \\ \text{valid}_\alpha(S) &= \{\psi \mid \psi \in \text{forms}(\text{tests}_\alpha(S)), \forall x \in S. \psi \Rightarrow \varphi_\alpha(x)\} \end{aligned}$$

The set $\text{next}^*(\alpha, \psi, S)$ 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 $\text{tests}_\alpha(S)$ contains all the test nodes from S . The set $\text{forms}(X)$ 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, $\text{valid}_\alpha(S)$ contains all the formulae over $\text{tests}_\alpha(S)$ 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 G.6.** *If α is binary branching, $S \subseteq \text{nBot}_\alpha$, $\text{next}^*(\alpha, \psi, S) = \emptyset$ and $\psi \in \text{valid}_\alpha(S)$, then $\psi \in \text{br}_\alpha$ and $\text{next}(\alpha \circ_f \beta) = \min_{f(\psi)}$.*

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

1. $\psi \Rightarrow \neg \text{stuck}_\alpha \Leftrightarrow \neg \bigvee_{x \in \text{Bot}_\alpha} \varphi_\alpha(x) \Leftrightarrow \bigwedge_{x \in \text{Bot}_\alpha} \neg \varphi_\alpha(x)$. It will suffice to show that $\psi \Rightarrow \neg \varphi_\alpha(x)$ for all $x \in \text{Bot}_\alpha$. Take any $v \models \psi$. We know that $\psi \wedge \varphi_\alpha(x)$ is unsatisfiable since $x \in \text{Bot}_\alpha$, therefore $v \models \neg \varphi_\alpha(x)$, so we are done.
2. $\psi \Leftrightarrow \bigwedge_{x \in S} \varphi_\alpha(x)$. From $\psi \in \text{valid}_\alpha(S)$, we immediately get that $\psi \Rightarrow \bigwedge_{x \in S} \varphi_\alpha(x)$. We now also show the reverse. Since $\psi \in \text{valid}_\alpha(S)$, then $\psi \in \text{forms}(\text{tests}_\alpha(S))$. We now show that for every $y \in \text{tests}_\alpha(S)$, either $\bigwedge_{x \in S} \varphi_\alpha(x) \Rightarrow y$ or $\bigwedge_{x \in S} \varphi_\alpha(x) \Rightarrow \neg y$. Take any $y \in \text{tests}_\alpha(S)$. By the binary branching property, $\text{succ}_\alpha(y) = \{z_1, z_2\}$ 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 $y \in \text{tests}_\alpha(S)$ and $\psi \in \text{valid}_\alpha(S)$. 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 $y \in \text{vars}(\psi)$, and so $\bigwedge_{x \in S} \varphi_\alpha(x) \Rightarrow \psi$.

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

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

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

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

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

Take any $x \in \text{next}(\alpha \circ_f \beta, \psi, S)$, so $x \in N_{\alpha \circ_f \beta}$, $x \notin S$, $x \downarrow_{\alpha \circ_f \beta} \subseteq S$, and $\psi \Rightarrow \varphi_{\alpha \circ_f \beta}(x)$. Clearly $x \notin N_\alpha$, otherwise one of $x \notin S$ or $\psi \Rightarrow \varphi_{\alpha \circ_f \beta}(x) = \varphi_\alpha(x)$ would be violated. Now, take any $y <_{\alpha \circ_f \beta} x$, we know from $x \downarrow_{\alpha \circ_f \beta} \subseteq S$ that $y \in S \subseteq N_\alpha$. So, since $\psi \Rightarrow \varphi_{\alpha \circ_f \beta}(x)$, 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 $x \in N_{\alpha \circ_f \beta}$ and $x \notin S$. By construction, $\varphi_{\alpha \circ_f \beta}(x) = \varphi_{f(\psi)}(x) \wedge \psi \Leftrightarrow \text{true} \wedge \psi$ ($\varphi_{f(\psi)}(x) \Leftrightarrow \text{true}$ since x is minimal and therefore no branches have occurred), so clearly $\psi \Rightarrow \varphi_{\alpha \circ_f \beta}(x)$. Now take any $y <_{\alpha \circ_f \beta} x$, 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 $x \downarrow_{\alpha \circ_f \beta} \subseteq S$, and we have shown all the conditions to guarantee that $x \in \text{next}(\alpha \circ_f \beta, \psi, S)$. \blacktriangleleft

► **Lemma G.7.** *If α is binary branching, $S \subseteq \text{nBot}_\alpha$, $\text{next}^*(\alpha, \psi, S) \neq \emptyset$, and $\psi \in \text{valid}_\alpha(S)$, then $\text{next}(\alpha \circ_f \beta, \psi, S) = \text{next}(\alpha, \psi, S) \neq \emptyset$*

Proof. Since $\text{next}^*(\alpha, \psi, S) \neq \emptyset$, then there exists $x \in \text{next}^*(\alpha, \psi, S)$, 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 $y \in \text{next}^*(\alpha, \psi, S)$ such that $y <_\alpha x$.

We now show that $x \downarrow_\alpha \subseteq S$. Take any $y <_\alpha x$. We know that $y \notin \text{next}^*(\alpha, \psi, S)$, 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 $x \in \text{succ}_\alpha(y)$ 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 $\psi \in \text{valid}_\alpha(S)$ 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 $\varphi_\alpha(x) \Leftrightarrow \bigwedge_{y \in \text{pred}_\alpha(x)} \varphi_\alpha(y)$, 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 $x \in \text{next}(\alpha, \psi, S)$.

Finally, we show that $\text{next}(\alpha, \psi, S) = \text{next}(\alpha \circ_f \beta, \psi, S)$ by showing the set inclusion in both directions. Take any $y \in \text{next}(\alpha, \psi, S)$, so $y \in N_\alpha \setminus S$ such that $y \downarrow_\alpha \subseteq S$ and $\psi \Rightarrow \varphi_\alpha(y)$. Since $N_\alpha \subseteq_\downarrow N_{\alpha \circ_f \beta}$, and $\varphi_\alpha(y) = \varphi_{\alpha \circ_f \beta}(y)$, this immediately gives us that $y \in N_{\alpha \circ_f \beta} \setminus S$, $y \downarrow_{\alpha \circ_f \beta} \subseteq S$, and $\psi \Rightarrow \varphi_{\alpha \circ_f \beta}(x)$ therefore $y \in \text{next}(\alpha \circ_f \beta, \psi, S)$.

Now, for the reverse inclusion, take any $y \in \text{next}(\alpha \circ_f \beta, \psi, S)$, so $y \in N_{\alpha \circ_f \beta} \setminus S$ such that $y \downarrow_{\alpha \circ_f \beta} \subseteq S$ and $\psi \Rightarrow \varphi_{\alpha \circ_f \beta}(x)$. 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 $\psi' \in \text{br}_\alpha$. This implies that $x <_{\alpha \circ_f \beta} y$, as we will now show. First, $\psi \Rightarrow \varphi_{\alpha \circ_f \beta}(y) = \varphi_{f(\psi')}(y) \wedge \psi' \Rightarrow \psi'$, so $\psi' \wedge \varphi_\alpha(x)$ is satisfiable. If $x \notin \text{ext}_\alpha$, then $\varphi_\alpha(x) \Rightarrow \text{stuck}_\alpha$, but we know that $\psi' \Rightarrow \neg \text{stuck}_\alpha$, which contradicts that $\psi' \wedge \varphi_\alpha(x)$ is

satisfiable. Therefore, $x \in \text{ext}_\alpha$, and since ψ' is maximal, then it must be the case that $\psi' \Rightarrow \varphi_\alpha(x)$ already, which implies that $x <_{\alpha \circ_f \beta} y$. However, since $x \notin S$, then $y \downarrow_{\alpha \circ_f \beta} \not\subseteq S$, which is a contradiction. Therefore, it must be the case that $y \in N_\alpha$. We therefore have that $y \in N_\alpha \setminus S$, $y \downarrow_\alpha = y \downarrow_{\alpha \circ_f \beta} \subseteq S$, and $\psi \Rightarrow \varphi_{\alpha \circ_f \beta}(y) = \varphi_\alpha(y)$; therefore, $y \in \text{next}(\alpha, \psi, S)$. ◀

► **Lemma G.8.** *For any binary branching $\alpha, \beta \in \text{lpo}_f(L)$, $S \subseteq \text{nBot}_\alpha$, and $\psi \in \text{valid}_\alpha(S)$:*

$$\mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S) = \mathcal{L}_{\text{lpo}_f}(\beta, \text{true}, \emptyset)^\dagger \circ \mathcal{L}_{\text{lpo}_f}(\alpha, \psi, S)$$

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

In the base case, $\text{next}^*(\alpha, \psi, S)$ is empty, so by Lemma G.6, we know that $\psi \in \text{br}_\alpha$ and $\text{next}(\alpha \circ_f \beta, \psi, S) = \min_{f(\psi)}$. Since each lpo is single-rooted, then $\min_{f(\psi)} = \{x\}$ for some $x \in N_{f(\psi)}$. This gives us:

$$\begin{aligned} & \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S)(s) \\ &= \bigwedge_{x \in \text{next}(\alpha \circ_f \beta, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_{\alpha \circ_f \beta}(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_{\alpha \circ_f \beta}(x) \in \text{act} \\ \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi \wedge (x = \llbracket \lambda_{\alpha \circ_f \beta}(x) \rrbracket(s)), S \cup \{x\})(s) & \text{if } \lambda_{\alpha \circ_f \beta}(x) \in \text{test} \\ \perp & \text{if } \lambda_{\alpha \circ_f \beta}(x) = \perp \\ \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S \cup \{x\})(s) & \text{if } \lambda_{\alpha \circ_f \beta}(x) = \text{fork} \end{cases} \\ &= \begin{cases} \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_{\alpha \circ_f \beta}(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_{\alpha \circ_f \beta}(x) \in \text{act} \\ \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi \wedge (x = \llbracket \lambda_{\alpha \circ_f \beta}(x) \rrbracket(s)), S \cup \{x\})(s) & \text{if } \lambda_{\alpha \circ_f \beta}(x) \in \text{test} \\ \perp & \text{if } \lambda_{\alpha \circ_f \beta}(x) = \perp \\ \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S \cup \{x\})(s) & \text{if } \lambda_{\alpha \circ_f \beta}(x) = \text{fork} \end{cases} \\ &= \begin{cases} \mathcal{L}_{\text{lpo}_f}(\beta_\psi, \text{true}, \{x\})^\dagger(\llbracket \lambda_{\beta_\psi}(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_{\beta_\psi}(x) \in \text{act} \\ \mathcal{L}_{\text{lpo}_f}(\beta_\psi, (x = \llbracket \lambda_{\beta_\psi}(x) \rrbracket(s)), \{x\})(s) & \text{if } \lambda_{\beta_\psi}(x) \in \text{test} \\ \perp & \text{if } \lambda_{\beta_\psi}(x) = \perp \\ \mathcal{L}_{\text{lpo}_f}(\beta_\psi, \text{true}, \{x\})(s) & \text{if } \lambda_{\beta_\psi}(x) = \text{fork} \end{cases} \\ &= \bigwedge_{x \in \text{next}(\beta, \text{true}, \emptyset)} \begin{cases} \mathcal{L}_{\text{lpo}_f}(\beta, \text{true}, \{x\})^\dagger(\llbracket \lambda_\beta(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\beta(x) \in \text{act} \\ \mathcal{L}_{\text{lpo}_f}(\beta, (x = \llbracket \lambda_\beta(x) \rrbracket(s)), \{x\})(s) & \text{if } \lambda_\beta(x) \in \text{test} \\ \perp & \text{if } \lambda_\beta(x) = \perp \\ \mathcal{L}_{\text{lpo}_f}(\beta, \text{true}, \{x\})(s) & \text{if } \lambda_\beta(x) = \text{fork} \end{cases} \\ &= \mathcal{L}_{\text{lpo}_f}(\beta, \text{true}, \emptyset)(s) \\ &= \mathcal{L}_{\text{lpo}_f}(\beta, \text{true}, \emptyset)^\dagger(\eta(s)) \\ &= \mathcal{L}_{\text{lpo}_f}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpo}_f}(\alpha, \psi, S)(s)) \end{aligned}$$

Now suppose $\text{next}^*(\alpha, \psi, S) \neq \emptyset$. By Lemma G.7, we know that $\text{next}(\alpha \circ_f \beta, \psi, S) = \text{next}(\alpha, \psi, S) \neq \emptyset$. Given this, we have:

$$\begin{aligned} & \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S)(s) \\ &= \bigwedge_{x \in \text{next}(\alpha \circ_f \beta, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_{\alpha \circ_f \beta}(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_{\alpha \circ_f \beta}(x) \in \text{act} \\ \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi \wedge (x = \llbracket \lambda_{\alpha \circ_f \beta}(x) \rrbracket(s)), S \cup \{x\})(s) & \text{if } \lambda_{\alpha \circ_f \beta}(x) \in \text{test} \\ \perp & \text{if } \lambda_{\alpha \circ_f \beta}(x) = \perp \\ \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S \cup \{x\})(s) & \text{if } \lambda_{\alpha \circ_f \beta}(x) = \text{fork} \end{cases} \\ &= \bigwedge_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi \wedge (x = \llbracket \lambda_\alpha(x) \rrbracket(s)), S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \perp & \text{if } \lambda_\alpha(x) = \perp \\ \mathcal{L}_{\text{lpo}_f}(\alpha \circ_f \beta, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases} \end{aligned}$$

Clearly $\text{next}^*(\alpha, \psi, S \cup \{x\}) \subset \text{next}^*(\alpha, \psi, S)$ and $\psi \in \text{valid}_\alpha(S \cup \{x\})$, if x is not a test node and therefore $\text{valid}_\alpha(S \cup \{x\}) = \text{valid}_\alpha(S)$. So, by the induction hypothesis and monad laws, we get:

$$= \bigotimes_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s))) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\alpha \circ \beta, \psi \wedge \llbracket x = \llbracket \lambda_\alpha(x) \rrbracket(s) \rrbracket, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \perp & \text{if } \lambda_\alpha(x) = \perp \\ \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})(s)) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases}$$

We also have that $\text{next}^*(\alpha, S \cup \{x\}, \psi \wedge \llbracket x = \llbracket \lambda_\alpha(x) \rrbracket_{\text{test}}(s) \rrbracket) \subset \text{next}^*(\alpha, S, \psi)$. In addition, $\psi \wedge \llbracket x = \llbracket \lambda_\alpha(x) \rrbracket_{\text{test}}(s) \rrbracket \in \text{valid}_\alpha(S \cup \{x\})$. So, by the induction hypothesis:

$$= \bigotimes_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s))) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpof}}(\alpha, \psi \wedge \llbracket x = \llbracket \lambda_\alpha(x) \rrbracket(s) \rrbracket, S \cup \{x\})(s)) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \perp & \text{if } \lambda_\alpha(x) = \perp \\ \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})(s)) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases}$$

Finally since $f^\dagger(\perp) = \perp$ for any f , we get:

$$= \bigotimes_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s))) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpof}}(\alpha, \psi \wedge \llbracket x = \llbracket \lambda_\alpha(x) \rrbracket(s) \rrbracket, S \cup \{x\})(s)) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\perp) & \text{if } \lambda_\alpha(x) = \perp \\ \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})(s)) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases}$$

Since $f^\dagger(X \& Y) = f^\dagger(X) \& f^\dagger(Y)$:

$$\begin{aligned} &= \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger \left(\bigotimes_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi \wedge \llbracket x = \llbracket \lambda_\alpha(x) \rrbracket(s) \rrbracket, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \perp & \text{if } \lambda_\alpha(x) = \perp \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases} \right) \\ &= \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger(\mathcal{L}_{\text{lpof}}(\alpha, \psi, S)(s)) \quad \blacktriangleleft \end{aligned}$$

► **Lemma G.9.** $\mathcal{L}(\llbracket C_1; C_2 \rrbracket) = \mathcal{L}(\llbracket C_2 \rrbracket)^\dagger \circ \mathcal{L}(\llbracket C_1 \rrbracket)$.

Proof.

$$\mathcal{L}(\llbracket C_1; C_2 \rrbracket) = \mathcal{L}(\llbracket C_1 \rrbracket \circ \llbracket C_2 \rrbracket)$$

By the definition of \circ for infinite pomsets:

$$= \mathcal{L} \left(\sup_{\alpha \ll \llbracket C_1 \rrbracket} \sup_{\beta \ll \llbracket C_2 \rrbracket} \alpha \circ \beta \right)$$

Since \mathcal{L} is Scott continuous:

$$\begin{aligned} &= \sup_{\alpha \ll \llbracket C_1 \rrbracket} \sup_{\beta \ll \llbracket C_2 \rrbracket} \mathcal{L}(\alpha \circ \beta) \\ &= \sup_{\alpha \ll \llbracket C_1 \rrbracket} \sup_{\beta \ll \llbracket C_2 \rrbracket} \sup_{\gamma \ll \alpha \circ \beta} \mathcal{L}_{\text{fin}}(\gamma) \end{aligned}$$

Since $\alpha \circ \beta$ is finite, then $\alpha \circ \beta \ll \alpha \circ \beta$ and therefore since \mathcal{L}_{fin} is monotone, then $\sup_{\gamma \ll \alpha \circ \beta} \mathcal{L}_{\text{fin}}(\gamma) = \mathcal{L}_{\text{fin}}(\alpha \circ \beta)$.

$$= \sup_{\alpha \ll \llbracket C_1 \rrbracket} \sup_{\beta \ll \llbracket C_2 \rrbracket} \mathcal{L}_{\text{fin}}(\alpha \circ \beta)$$

$$= \sup_{\alpha \ll [C_1]} \sup_{\beta \ll [C_2]} \mathcal{L}_{\text{fin}}(\{\alpha \circ_f \beta \mid \alpha \in \alpha, \beta \in \beta, f \in \text{copy}_{\alpha, \beta}\})$$

The set above is an equivalence class, so fixing an arbitrary $\alpha \in \alpha$, $\beta \in \beta$, and $f \in \text{copy}_{\alpha, \beta}$, we can rewrite the expression as follows:

$$\begin{aligned} &= \sup_{\alpha \ll [C_1]} \sup_{\beta \ll [C_2]} \mathcal{L}_{\text{fin}}([\alpha \circ_f \beta]) \\ &= \sup_{\alpha \ll [C_1]} \sup_{\beta \ll [C_2]} \mathcal{L}_{\text{lpof}}(\alpha \circ_f \beta, \text{true}, \emptyset) \end{aligned}$$

By Lemma G.8:

$$\begin{aligned} &= \sup_{\alpha \ll [C_1]} \sup_{\beta \ll [C_2]} \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)^\dagger \circ \mathcal{L}_{\text{lpof}}(\alpha, \text{true}, \emptyset) \\ &= \sup_{\alpha \ll [C_1]} \sup_{\beta \ll [C_2]} \mathcal{L}_{\text{fin}}([\beta])^\dagger \circ \mathcal{L}_{\text{fin}}([\alpha]) \end{aligned}$$

By continuity of Kleisli extension, and since $\alpha = [\alpha]$ and $\beta = [\beta]$:

$$\begin{aligned} &= \left(\sup_{\beta \ll [C_2]} \mathcal{L}_{\text{fin}}(\beta) \right)^\dagger \circ \sup_{\alpha \ll [C_1]} \mathcal{L}_{\text{fin}}(\alpha) \\ &= \mathcal{L}([C_2])^\dagger \circ \mathcal{L}([C_1]) \end{aligned}$$

◀

G.4 Properties of Linearization: Guarded Branching

► **Lemma G.10.** *For any $\alpha, \beta \in \text{lpof}_{\text{fin}}(\text{act})$, $b \in \text{test}$, and $x \in \text{nodes} \setminus (N_\alpha \cup N_\beta)$:*

$$\mathcal{L}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \text{true}, \emptyset)(s) = \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha, \text{true}, \emptyset)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases}$$

Proof.

$$\begin{aligned} &\mathcal{L}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \text{true}, \emptyset)(s) \\ &= \bigwedge_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \text{true}, \{x\})^\dagger(\llbracket \lambda_{\text{guard}(x, b, \alpha, \beta)}(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(x) \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \llbracket x = \llbracket \lambda_{\text{guard}(x, b, \alpha, \beta)}(x) \rrbracket(s) \rrbracket, \{x\})(s) & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(x) \in \text{test} \\ \perp & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(x) = \perp \\ \mathcal{L}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \text{true}, \{x\})(s) & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(x) = \text{fork} \end{cases} \end{aligned}$$

Since x is the root of $\text{guard}(x, b, \alpha, \beta)$, then $\text{next}(\alpha, \psi, S) = \{x\}$ and $\lambda_{\text{guard}(x, b, \alpha, \beta)}(x) \in \text{test}$.

$$\begin{aligned} &= \mathcal{L}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \llbracket x = \llbracket \lambda_{\text{guard}(x, b, \alpha, \beta)}(x) \rrbracket(s) \rrbracket, \{x\})(s) \\ &= \begin{cases} \mathcal{L}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), x, \{x\})(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \mathcal{L}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \neg x, \{x\})(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases} \end{aligned}$$

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:

$$= \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha, \text{true}, \emptyset)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = \text{true} \\ \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = \text{false} \end{cases}$$

◀

► **Lemma G.11.**

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

Proof.

$$\begin{aligned}
& \mathcal{L}[\text{if } b \text{ then } C_1 \text{ else } C_2](s) \\
&= \mathcal{L}(\text{guard}(b, \llbracket C_1 \rrbracket, \llbracket C_2 \rrbracket))(s) \\
&= \sup_{\alpha \ll \llbracket C_1 \rrbracket} \sup_{\beta \ll \llbracket C_2 \rrbracket} \mathcal{L}_{\text{fin}}(\text{guard}(b, \alpha, \beta))(s) \\
&= \sup_{\alpha \ll \llbracket C_1 \rrbracket} \sup_{\beta \ll \llbracket C_2 \rrbracket} \mathcal{L}_{\text{fin}}(\{\text{guard}(x, b, \alpha, \beta) \mid x \in \text{nodes}, \alpha \in \alpha, \beta \in \beta\})(s) \\
&= \sup_{\alpha \ll \llbracket C_1 \rrbracket} \sup_{\beta \ll \llbracket C_2 \rrbracket} \mathcal{L}_{\text{fin}}(\llbracket \text{guard}(x, b, \alpha, \beta) \rrbracket)(s) \\
&= \sup_{\alpha \ll \llbracket C_1 \rrbracket} \sup_{\beta \ll \llbracket C_2 \rrbracket} \mathcal{L}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \text{true}, \emptyset)(s) \\
&= \sup_{\alpha \ll \llbracket C_1 \rrbracket} \sup_{\beta \ll \llbracket C_2 \rrbracket} \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha, \text{true}, \emptyset)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \mathcal{L}_{\text{lpof}}(\beta, \text{true}, \emptyset)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases} \quad (\text{By Lemma G.10}) \\
&= \begin{cases} \sup_{\alpha \ll \llbracket C_1 \rrbracket} \mathcal{L}_{\text{fin}}(\alpha)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \sup_{\beta \ll \llbracket C_2 \rrbracket} \mathcal{L}_{\text{fin}}(\beta)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases} \\
&= \begin{cases} \mathcal{L}(\llbracket C_1 \rrbracket)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \mathcal{L}(\llbracket C_2 \rrbracket)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases} \quad \blacktriangleleft
\end{aligned}$$

G.5 Properties of Linearization: While Loops

► **Lemma G.12.** Let $D(S)$ be the dcpo such that $\mathcal{L}: \text{pom} \rightarrow S \rightarrow D(S)$, and let $\perp_{\text{pom}} \in \text{pom}$ be the bottom of the pomset order, $\perp_D \in D(S)$ be the bottom of $D(S)$, and $\perp_D^\bullet: S \rightarrow D(S)$ be the bottom of the pointwise order on $D(S)$. Then

$$\mathcal{L}(\Phi_{\langle C, b \rangle}^n(\perp_{\text{pom}})) = \Psi_{\langle \mathcal{L}(\llbracket C \rrbracket), b \rangle}^n(\perp_D^\bullet)$$

Proof. The proof is by induction on n . If $n = 0$, then we have:

$$\mathcal{L}(\Phi_{\langle C, b \rangle}^0(\perp_{\text{pom}}))(s) = \mathcal{L}(\perp_{\text{pom}})(s) = \perp_D = \perp_D^\bullet(s) = \Psi_{\langle \mathcal{L}(\llbracket C \rrbracket), b \rangle}^0(\perp_D^\bullet)(s)$$

Now, suppose the claim holds for n , then we have:

$$\begin{aligned}
\mathcal{L}(\Phi_{\langle C, b \rangle}^{n+1}(\perp_{\text{pom}}))(s) &= \mathcal{L}(\Phi_{\langle C, b \rangle}(\Phi_{\langle C, b \rangle}^n(\perp_{\text{pom}})))(s) \\
&= \mathcal{L}(\text{guard}(b, \llbracket C \rrbracket \circ \Phi_{\langle C, b \rangle}^n(\perp_{\text{pom}}), \llbracket \text{skip} \rrbracket))(s)
\end{aligned}$$

By Lemma G.10.

$$= \begin{cases} \mathcal{L}(\llbracket C \rrbracket \circ \Phi_{\langle C, b \rangle}^n(\perp_{\text{pom}}))(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \mathcal{L}(\llbracket \text{skip} \rrbracket)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases}$$

By Lemmas G.5 and G.9.

$$= \begin{cases} \mathcal{L}(\Phi_{\langle C, b \rangle}^n(\perp_{\text{pom}}))^\dagger(\mathcal{L}(\llbracket C \rrbracket)(s)) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \eta(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases}$$

By the induction hypothesis.

$$= \begin{cases} (\Psi_{\langle \mathcal{L}(\llbracket C \rrbracket), b \rangle}^n(\perp_D^\bullet))^\dagger(\mathcal{L}(\llbracket C \rrbracket)(s)) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \eta(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases}$$

$$\begin{aligned}
&= \Psi_{\langle \mathcal{L}[\![C]\!], b \rangle} \left(\Psi_{\langle \mathcal{L}[\![C]\!], b \rangle}^n (\perp_D^\bullet) \right) (s) \\
&= \Psi_{\langle \mathcal{L}[\![C]\!], b \rangle}^{n+1} (\perp_D^\bullet) (s)
\end{aligned}$$

► **Lemma G.13.** $\mathcal{L}(\llbracket \text{while } b \text{ do } C \rrbracket) = \text{lfp} (\Psi_{\langle \mathcal{L}[\![C]\!], b \rangle})$ where

$$\Psi_{\langle f, b \rangle}(g)(s) \triangleq \begin{cases} g^\dagger(f(s)) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \eta(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases}$$

Proof.

$$\begin{aligned}
\mathcal{L}(\llbracket \text{while } b \text{ do } C \rrbracket) &= \mathcal{L}(\text{lfp} (\Phi_{\langle C, b \rangle})) \\
&= \mathcal{L} \left(\sup_{n \in \mathbb{N}} \Phi_{\langle C, b \rangle}^n (\perp_{\text{pom}}) \right) && \text{(By the Kleene fixed point theorem)} \\
&= \sup_{n \in \mathbb{N}} \mathcal{L} \left(\Phi_{\langle C, b \rangle}^n (\perp_{\text{pom}}) \right) && \text{(By Scott continuity of } \mathcal{L} \text{)} \\
&= \sup_{n \in \mathbb{N}} \Psi_{\langle \mathcal{L}[\![C]\!], b \rangle}^n (\perp_D^\bullet) && \text{(By Lemma G.12)} \\
&= \text{lfp} (\Psi_{\langle \mathcal{L}[\![C]\!], b \rangle}) && \text{(By the Kleene fixed point theorem)} \quad \blacktriangleleft
\end{aligned}$$

G.6 Main Lemma

► **Lemma 5.2** (Linearization). *The following properties hold:*

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

$$\mathcal{L}(\llbracket \text{if } b \text{ then } C_1 \text{ else } C_2 \rrbracket)(s) = \begin{cases} \mathcal{L}(\llbracket C_1 \rrbracket)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \mathcal{L}(\llbracket C_2 \rrbracket)(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases}$$

$$\mathcal{L}(\llbracket \text{while } b \text{ do } C \rrbracket) = \text{lfp} (\Psi_{\langle \mathcal{L}[\![C]\!], b \rangle}) \text{ where } \Psi_{\langle f, b \rangle}(g)(s) \triangleq \begin{cases} g^\dagger(f(s)) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 1 \\ \eta(s) & \text{if } \llbracket b \rrbracket_{\text{test}}(s) = 0 \end{cases}$$

Proof. Follows immediately from Lemmas G.5, G.9, G.11, and G.13 ◀

H 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: $\sup_{i \in I} S_i \triangleq \bigcup_{i \in I} S_i$.

We begin by defining standard pomset language semantics in Figure 5. Although the semantics is defined using pomsets with tests, $\text{test}_{\text{PL}} = \emptyset$, so the formula of every node in α is true. In addition, all pomsets are finite and \perp nodes are never introduced, meaning that α consists only of actions and fork nodes. We therefore get that the definition sequential composition is the standard one, e.g., [9, 17, 38, 49].

► **Lemma H.1.** *If $\text{Bot}_\alpha = \emptyset$, $\varphi_\alpha(x) = \text{true}$ for all $x \in N_\alpha$, and $f = [\text{true} \mapsto \beta]$, then:*

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

Proof. First note that since $\text{Bot}_\alpha = \emptyset$, then $\text{stuck}_\alpha = \text{false}$ and so $\text{ext}_\alpha = N_\alpha$. Therefore, $\text{br}_\alpha = \{\bigwedge_{x \in N_\alpha} \varphi_\alpha(x)\} = \{\text{true}\}$. We now show the condition on nodes.

$$N_{\alpha \circ f \beta} = N_\alpha \cup \bigcup_{\psi \in \text{br}_\alpha} N_{f(\psi)} = N_\alpha \cup N_{f(\text{true})} = N_\alpha \cup N_\beta$$

We now also show the condition on orders:

$$\begin{aligned}
<_{\alpha \circ_f \beta} &= <_{\alpha} \cup \bigcup_{\psi \in \text{br}_{\alpha}} (<_{f(\psi)} \cup \{x \in N_{\alpha} \mid \psi \Rightarrow \varphi_{\alpha}(x)\} \times N_{f(\psi)}) \\
&= <_{\alpha} \cup <_{f(\text{true})} \cup \{x \in N_{\alpha} \mid \text{true} \Rightarrow \text{true}\} \times N_{f(\text{true})} \\
&= <_{\alpha} \cup <_{\beta} \cup (N_{\alpha} \times N_{\beta})
\end{aligned}$$

◀

Loops are interpreted as the least fixed point of $\Xi_{\langle C, b \rangle}$, which is Scott continuous in the subset \subseteq order, since suprema (given by \cup) are associative and commutative. The **skip** appears in the false branch to correspond to $\Phi_{\langle C, b \rangle}$.

H.1 Definition of Translation

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.

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

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

$$\begin{aligned}
N_{\psi} &\triangleq \{x \in N_{\alpha} \mid \psi \Rightarrow \varphi_{\alpha}(x)\} \\
<_{\psi} &\triangleq <_{\alpha} \cap (N_{\psi} \times N_{\psi}) \\
\lambda_{\psi}(x) &\triangleq \begin{cases} \lambda_{\alpha}(x) & \text{if } \lambda_{\alpha}(x) \notin \text{test} \\ \mathbf{assume} \ \lambda_{\alpha}(x) & \text{if } \lambda_{\alpha}(x) \in \text{test} \text{ and } \psi \Rightarrow x \\ \mathbf{assume} \ \neg \lambda_{\alpha}(x) & \text{if } \lambda_{\alpha}(x) \in \text{test} \text{ and } \psi \Rightarrow \neg x \end{cases}
\end{aligned}$$

The nodes of $\text{tr}_{\text{lpoF}}(\alpha, \psi)$ are those nodes $x \in N_{\alpha}$ such that $\psi \Rightarrow \varphi_{\alpha}(x)$, indicating that x agrees with the outcomes of the tests described by ψ . 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 $\psi \in \text{br}_{\alpha}$.

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

Finite translation is monotonic (Lemma H.3)— $\text{tr}_{\text{fin}}(\alpha) \subseteq \text{tr}_{\text{fin}}(\beta)$ if $\alpha \sqsubseteq_{\text{pom}} \beta$ —therefore we can extend to infinite pomsets with the extension lemma.

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

H.2 Properties of the Translation Function

► **Lemma H.2** (Monotonicity of tr_{lpoF}). *If $\alpha \sqsubseteq_{\text{lpoF}} \alpha'$ and $\psi \in \text{br}_{\alpha}$, then $\text{tr}_{\text{lpoF}}(\alpha, \psi) = \text{tr}_{\text{lpoF}}(\alpha', \psi)$*

Proof. We first show that $N_{\text{tr}_{\text{lpoF}}(\alpha, \psi)} = N_{\text{tr}_{\text{lpoF}}(\alpha', \psi)}$, or in other words, that $\{x \in N_{\alpha} \mid \psi \Rightarrow \varphi_{\alpha}(x)\} = \{x \in N_{\alpha'} \mid \psi \Rightarrow \varphi_{\alpha'}(x)\}$. The forward inclusion is immediate since $N_{\alpha} \subseteq N_{\alpha'}$ and $\varphi_{\alpha}(x) = \varphi_{\alpha'}(x)$ for all $x \in N_{\alpha}$. We now show the reverse inclusion. Take any $x \in N_{\alpha'}$ such that $\psi \Rightarrow \varphi_{\alpha'}(x)$. If $x \notin N_{\alpha}$, then by Lemma C.2, $x \in \text{Bot}_{\alpha} \uparrow_{\alpha'}$. This means that there is some $y \in \text{Bot}_{\alpha}$ such that $y <_{\alpha'} x$, and so $\varphi_{\alpha'}(x) \Rightarrow \varphi_{\alpha'}(y) \Rightarrow \text{stuck}_{\alpha}$. But this contradicts the fact that $\psi \Rightarrow \varphi_{\alpha'}(x)$ and $\psi \in \text{br}_{\alpha}$. Therefore, it must be that $x \in N_{\alpha}$, and so the claim holds.

Now, for the order, we have:

$$\begin{aligned} <_{\text{tr}_{\text{lpof}}(\alpha, \psi)} &= <_{\alpha} \cap (N_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \times N_{\text{tr}_{\text{lpof}}(\alpha, \psi)}) \\ &= (<_{\alpha'} \cap (N_{\alpha} \times N_{\alpha})) \cap (N_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \times N_{\text{tr}_{\text{lpof}}(\alpha, \psi)}) \end{aligned}$$

Since $N_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \subseteq N_{\alpha}$:

$$\begin{aligned} &= <_{\alpha'} \cap (N_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \times N_{\text{tr}_{\text{lpof}}(\alpha, \psi)}) \\ &= <_{\alpha'} \cap (N_{\text{tr}_{\text{lpof}}(\alpha', \psi)} \times N_{\text{tr}_{\text{lpof}}(\alpha', \psi)}) \\ &= <_{\text{tr}_{\text{lpof}}(\alpha', \psi)} \end{aligned}$$

Finally, for any $x \in N_{\text{tr}_{\text{lpof}}(\alpha, \psi)}$, we have:

$$\lambda_{\text{tr}_{\text{lpof}}(\alpha, \psi)}(x) = \begin{cases} \lambda_{\alpha}(x) & \text{if } \lambda_{\alpha}(x) \notin \text{test} \\ \text{assume } \lambda_{\alpha}(x) & \text{if } \lambda_{\alpha}(x) \in \text{test} \text{ and } \psi \Rightarrow x \\ \text{assume } \neg \lambda_{\alpha}(x) & \text{if } \lambda_{\alpha}(x) \in \text{test} \text{ and } \psi \Rightarrow \neg x \end{cases}$$

By virtue of the fact that $x \in N_{\text{tr}_{\text{lpof}}(\alpha, \psi)}$, we know that $\psi \Rightarrow \varphi_{\alpha}(x)$, and since $\psi \in \text{br}_{\alpha}$, then $\psi \Rightarrow \neg \text{stuck}_{\alpha}$, therefore $\lambda_{\alpha}(x)$ cannot be \perp , and since *label* is a flat cpo, then it must be that $\lambda_{\alpha}(x) = \lambda_{\alpha'}(x)$.

$$\begin{aligned} &= \begin{cases} \lambda_{\alpha'}(x) & \text{if } \lambda_{\alpha'}(x) \notin \text{test} \\ \text{assume } \lambda_{\alpha'}(x) & \text{if } \lambda_{\alpha'}(x) \in \text{test} \text{ and } \psi \Rightarrow x \\ \text{assume } \neg \lambda_{\alpha'}(x) & \text{if } \lambda_{\alpha'}(x) \in \text{test} \text{ and } \psi \Rightarrow \neg x \end{cases} \\ &= \lambda_{\text{tr}_{\text{lpof}}(\alpha', \psi)}(x) \end{aligned}$$

And finally, clearly we have $\varphi_{\text{tr}_{\text{lpof}}(\alpha, \psi)}(x) = \varphi_{\text{tr}_{\text{lpof}}(\alpha', \psi)}(x) = \text{true}$. ◀

► **Lemma H.3** (Monotonicity of tr_{fin}). *If $\alpha \sqsubseteq_{\text{pom}} \alpha'$, then $\text{tr}_{\text{fin}}(\alpha) \subseteq \text{tr}_{\text{fin}}(\alpha')$.*

Proof. Since $\alpha \sqsubseteq_{\text{pom}} \alpha'$, then by Lemma D.1 there exists an $\alpha \in \alpha$ and $\alpha' \in \alpha'$ such that $\alpha \sqsubseteq_{\text{lpof}} \alpha'$. We now prove the claim as follows:

$$\begin{aligned} \text{tr}_{\text{fin}}(\alpha) &= \text{tr}_{\text{fin}}([\alpha]) \\ &= \{[\beta] \mid \psi \in \text{br}_{\alpha}, \beta \in \text{tr}_{\text{lpof}}(\alpha)\} \\ &\subseteq \{[\beta] \mid \psi \in \text{br}_{\alpha'}, \beta \in \text{tr}_{\text{lpof}}(\alpha)\} && \text{By Lemma F.3.} \\ &= \{[\beta] \mid \psi \in \text{br}_{\alpha'}, \beta \in \text{tr}_{\text{lpof}}(\alpha')\} && \text{By Lemma H.2.} \\ &= \text{tr}_{\text{fin}}([\alpha']) \\ &= \text{tr}_{\text{fin}}(\alpha') \end{aligned}$$

H.3 Translation and Pomset Operations

► **Lemma H.4.** *For any $\ell \in \text{act} \cup \{\text{fork}\}$:*

$$\text{tr}(\langle \ell \rangle) = \{\langle \ell \rangle\}$$

Proof. Since $\langle \ell \rangle$ is a finite pomset:

$$\text{tr}(\langle \ell \rangle) = \text{tr}_{\text{fin}}(\langle \ell \rangle)$$

Fixing any arbitrary $x \in \text{nodes}$, we have that $\langle \ell \rangle = [\langle \ell \rangle_x]$.

$$\begin{aligned} &= \text{tr}_{\text{fin}}([\langle \ell \rangle_x]) \\ &= \{[\text{tr}_{\text{lpof}}(\langle \ell \rangle_x, \psi)] \mid \psi \in \text{br}_{\langle \ell \rangle_x}\} \end{aligned}$$

Since $\text{br}_{\langle \ell \rangle_x} = \{\text{true}\}$ and $\text{tr}_{\text{lpof}}(\langle \ell \rangle_x, \text{true}) = \langle \ell \rangle_x$:

$$\begin{aligned} &= \{[\langle \ell \rangle_x]\} \\ &= \{\langle \ell \rangle\} \end{aligned} \quad \blacktriangleleft$$

For sequential composition of LPOFs, where $\text{br}_\alpha = \{\text{true}\}$, we will write $\alpha \circ \beta$ (omitting the copy function f), to mean $\alpha \circ_f \beta$ where $f \triangleq [\text{true} \mapsto \beta]$, i.e., f does not perform any renaming, since only a single isomorphic copy of β is needed.

► **Lemma H.5.** For any $\alpha, \beta \in \text{lpof}_{\text{fin}}(\text{label})$, $f \in \text{copy}_{\alpha, \beta}$, $\psi \in \text{br}_\alpha$, and $\psi' \in \text{br}_{f(\psi)}$,

$$\text{tr}_{\text{lpof}}(\alpha \circ_f \beta, \psi \wedge \psi') = \text{tr}_{\text{lpof}}(\alpha, \psi) \circ \text{tr}_{\text{lpof}}(f(\psi), \psi')$$

Proof. We show the equality component-wise. Some of the steps use Lemma H.1. First, for the nodes, we have:

$$N_{\text{tr}_{\text{lpof}}(\alpha \circ_f \beta, \psi \wedge \psi')} = \{x \in N_{\alpha \circ_f \beta} \mid \psi \wedge \psi' \Rightarrow \varphi_{\alpha \circ_f \beta}(x)\}$$

Since $\psi \in \text{br}_\alpha$, then we will only take nodes from $f(\psi)$ and no other $\psi'' \in \text{br}_\alpha$

$$\begin{aligned} &= \{x \in N_\alpha \mid \psi \wedge \psi' \Rightarrow \varphi_\alpha(x)\} \cup \{x \in N_{f(\psi)} \mid \psi \wedge \psi' \Rightarrow \varphi_{f(\psi)}(x) \wedge \psi\} \\ &= \{x \in N_\alpha \mid \psi \Rightarrow \varphi_\alpha(x)\} \cup \{x \in N_{f(\psi)} \mid \psi' \Rightarrow \varphi_{f(\psi)}(x)\} \\ &= N_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \cup N_{\text{tr}_{\text{lpof}}(f(\psi), \psi')} \\ &= N_{\text{tr}_{\text{lpof}}(\alpha, \psi) \circ \text{tr}_{\text{lpof}}(f(\psi), \psi')} \end{aligned}$$

Now, we show that the orders are the same:

$$\begin{aligned} &<_{\text{tr}_{\text{lpof}}(\alpha \circ_f \beta, \psi \wedge \psi')} \\ &= <_{\alpha \circ_f \beta} \cap (N_{\text{tr}_{\text{lpof}}(\alpha \circ_f \beta, \psi \wedge \psi')} \times N_{\text{tr}_{\text{lpof}}(\alpha \circ_f \beta, \psi \wedge \psi')}) \\ &= (<_\alpha \cap (N_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \times N_{\text{tr}_{\text{lpof}}(\alpha, \psi)})) \cup (<_{f(\psi)} \cap (N_{\text{tr}_{\text{lpof}}(f(\psi), \psi')} \times N_{\text{tr}_{\text{lpof}}(f(\psi), \psi')})) \cup (N_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \times N_{\text{tr}_{\text{lpof}}(f(\psi), \psi')}) \\ &= <_{\text{tr}_{\text{lpof}}(\alpha, \psi) \circ \text{tr}_{\text{lpof}}(f(\psi), \psi')} \end{aligned}$$

Now, for any x , clearly we have:

$$\varphi_{\text{tr}_{\text{lpof}}(\alpha \circ_f \beta, \psi \wedge \psi')}(x) = \varphi_{\text{tr}_{\text{lpof}}(\alpha, \psi) \circ \text{tr}_{\text{lpof}}(f(\psi), \psi')}(x) = \text{true}$$

If $x \in N_\alpha$, then:

$$\begin{aligned} \lambda_{\text{tr}_{\text{lpof}}(\alpha \circ_f \beta, \psi \wedge \psi')}(x) &= \begin{cases} \lambda_{\alpha \circ_f \beta}(x) & \text{if } \lambda_{\alpha \circ_f \beta}(x) \notin \text{test} \\ \text{assume } \lambda_{\alpha \circ_f \beta}(x) & \text{if } \lambda_{\alpha \circ_f \beta}(x) \in \text{test} \text{ and } \psi \wedge \psi' \Rightarrow x \\ \text{assume } \neg \lambda_{\alpha \circ_f \beta}(x) & \text{if } \lambda_{\alpha \circ_f \beta}(x) \in \text{test} \text{ and } \psi \wedge \psi' \Rightarrow \neg x \end{cases} \\ &= \begin{cases} \lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \notin \text{test} \\ \text{assume } \lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \in \text{test} \text{ and } \psi \Rightarrow x \\ \text{assume } \neg \lambda_\alpha(x) & \text{if } \lambda_\alpha(x) \in \text{test} \text{ and } \psi \Rightarrow \neg x \end{cases} \\ &= \lambda_{\text{tr}_{\text{lpof}}(\alpha, \psi)}(x) \\ &= \lambda_{\text{tr}_{\text{lpof}}(\alpha, \psi) \circ \text{tr}_{\text{lpof}}(f(\psi), \psi')}(x) \end{aligned}$$

The case for $x \in f(\psi)$ is nearly identical. ◀

► **Lemma H.6.** $\text{tr}(\alpha \circ \beta) = \{\alpha' \circ \beta' \mid \alpha' \in \text{tr}(\alpha), \beta' \in \text{tr}(\beta)\}.$

Proof.

$$\begin{aligned} \text{tr}(\alpha \circ \beta) &= \text{tr}\left(\sup_{\alpha' \ll \alpha} \sup_{\beta' \ll \beta} \alpha' \circ \beta'\right) \\ &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \text{tr}_{\text{fin}}(\alpha' \circ \beta') \\ &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \text{tr}_{\text{fin}}(\{\alpha \circ_f \beta \mid \alpha \in \alpha', \beta \in \beta', f \in \text{copy}_{\alpha, \beta}\}) \end{aligned}$$

Since the above set is an equivalence class, we can fix any $\alpha \in \alpha'$, $\beta \in \beta'$, and $f \in \text{copy}_{\alpha, \beta}$.

$$\begin{aligned} &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \text{tr}_{\text{fin}}([\alpha \circ_f \beta]) \\ &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \{\text{tr}_{\text{lpof}}(\alpha \circ_f \beta, \psi) \mid \psi \in \text{br}_{\alpha \circ_f \beta}\} \\ &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \{\text{tr}_{\text{lpof}}(\alpha \circ_f \beta, \psi \wedge \psi') \mid \psi \in \text{br}_{\alpha}, \psi' \in \text{br}_{f(\psi)}\} \\ &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \{\text{tr}_{\text{lpof}}(\alpha, \psi) \circ \text{tr}_{\text{lpof}}(f(\psi), \psi') \mid \psi \in \text{br}_{\alpha}, \psi' \in \text{br}_{f(\psi)}\} \quad (\text{By Lemma H.5}) \\ &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \text{tr}_{\text{fin}}(\alpha') \circ \text{tr}_{\text{fin}}(\beta') \\ &= \left(\bigcup_{\alpha' \ll \alpha} \text{tr}_{\text{fin}}(\alpha') \right) \circ \left(\bigcup_{\beta' \ll \beta} \text{tr}_{\text{fin}}(\beta') \right) \\ &= \text{tr}(\alpha) \circ \text{tr}(\beta) \end{aligned} \quad \blacktriangleleft$$

► **Lemma H.7.** $\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x) = \langle \text{assume } b \rangle_x \circ \text{tr}_{\text{lpof}}(\alpha, \psi)$ and $\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge \neg x) = \langle \text{assume } \neg b \rangle_x \circ \text{tr}_{\text{lpof}}(\beta, \psi).$

Proof. We prove the equality for the first case. The second is entirely symmetrical. We first show equality of the node sets:

$$N_{\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)} = \{y \in N_{\text{guard}(x, b, \alpha, \beta)} \mid \psi \wedge x \Rightarrow \varphi_{\text{guard}(x, b, \alpha, \beta)}(y)\}$$

By the construction of guard, x is true in all nodes in the α branch, and the root x has formula true.

$$\begin{aligned} &= \{x\} \cup \{y \in N_{\alpha} \mid \psi \Rightarrow \varphi_{\alpha}(y)\} \\ &= N_{\langle \text{assume } b \rangle_x} \cup N_{\text{tr}(\alpha, \psi)} \\ &= N_{\langle \text{assume } b \rangle_x \circ \text{tr}(\alpha, \psi)} \end{aligned}$$

For the order, we have:

$$\begin{aligned} &<_{\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)} \\ &= <_{\text{guard}(x, b, \alpha, \beta)} \cap (N_{\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)} \times N_{\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)}) \\ &= (<_{\alpha} \cup <_{\beta} \cup (\{x\} \times (N_{\alpha} \cup N_{\beta}))) \cap (N_{\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)} \times N_{\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)}) \end{aligned}$$

Using what we just showed about the node sets:

$$\begin{aligned}
 &= (<_{\alpha} \cap (N_{\text{tr}(\alpha, \psi)} \times N_{\text{tr}(\alpha, \psi)})) \cup (\{x\} \times N_{\text{tr}(\alpha, \psi)}) \\
 &= <_{\text{tr}(\alpha, \psi)} \cup (\{x\} \times N_{\text{tr}(\alpha, \psi)}) \\
 &= <_{\langle \text{assume } b \rangle_x \text{tr}(\alpha, \psi)}
 \end{aligned}$$

For every y , clearly:

$$\varphi_{\text{tr}_{\text{tpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)}(y) = \varphi_{\langle \text{assume } b \rangle_x \text{tr}(\alpha, \psi)}(y) = \text{true}$$

If $y = x$, then we have:

$$\begin{aligned}
 &\lambda_{\text{tr}_{\text{tpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)}(y) \\
 &= \begin{cases} \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) \notin \text{test} \\ \text{assume } \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) \in \text{test} \text{ and } \psi \wedge x \Rightarrow y \\ \text{assume } \neg \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) \in \text{test} \text{ and } \psi \wedge x \Rightarrow \neg y \end{cases} \\
 &= \text{assume } \lambda_{\text{guard}(x, b, \alpha, \beta)}(x) \\
 &= \text{assume } b \\
 &= \lambda_{\langle \text{assume } b \rangle_x}(x) \\
 &= \lambda_{\langle \text{assume } b \rangle_x \text{tr}(\alpha, \psi)}(x)
 \end{aligned}$$

If not, then $y \in \text{tr}(\alpha, \psi) \subseteq N_{\alpha}$:

$$\begin{aligned}
 &\lambda_{\text{tr}_{\text{tpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)}(y) \\
 &= \begin{cases} \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) \notin \text{test} \\ \text{assume } \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) \in \text{test} \text{ and } \psi \wedge x \Rightarrow y \\ \text{assume } \neg \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) & \text{if } \lambda_{\text{guard}(x, b, \alpha, \beta)}(y) \in \text{test} \text{ and } \psi \wedge x \Rightarrow \neg y \end{cases} \\
 &= \begin{cases} \lambda_{\alpha}(y) & \text{if } \lambda_{\alpha}(y) \notin \text{test} \\ \text{assume } \lambda_{\alpha}(y) & \text{if } \lambda_{\alpha}(y) \in \text{test} \text{ and } \psi \Rightarrow y \\ \text{assume } \neg \lambda_{\alpha}(y) & \text{if } \lambda_{\alpha}(y) \in \text{test} \text{ and } \psi \Rightarrow \neg y \end{cases} \\
 &= \lambda_{\text{tr}(\alpha, \psi)}(y) \\
 &= \lambda_{\langle \text{assume } b \rangle_x \text{tr}(\alpha, \psi)}(y)
 \end{aligned}$$

► **Lemma H.8.**

$$\text{tr}(\text{guard}(b, \alpha, \beta)) = \{\langle \text{assume } b \rangle \text{ ; } \alpha' \mid \alpha' \in \text{tr}(\alpha)\} \cup \{\langle \text{assume } \neg b \rangle \text{ ; } \beta' \mid \beta' \in \text{tr}(\beta)\}$$

Proof.

$$\begin{aligned}
 &\text{tr}(\text{guard}(b, \alpha, \beta)) \\
 &= \text{tr} \left(\sup_{\alpha' \ll \alpha} \sup_{\beta' \ll \beta} \text{guard}(b, \alpha', \beta') \right) \\
 &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \text{tr}_{\text{fin}}(\text{guard}(b, \alpha', \beta')) \\
 &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \text{tr}_{\text{fin}}(\{\text{guard}(x, b, \alpha, \beta) \mid \alpha \in \alpha', \beta \in \beta', x \in \text{nodes}\})
 \end{aligned}$$

Since the set above is an equivalence class, we can fix any x , α , and β and write:

$$\begin{aligned}
&= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \text{tr}_{\text{fin}}([\text{guard}(x, b, \alpha, \beta)]) \\
&= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \{[\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi)] \mid \psi \in \text{br}_{\text{guard}(x, b, \alpha, \beta)}\} \\
&= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \{[\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge x)] \mid \psi \in \text{br}_{\alpha}\} \cup \{[\text{tr}_{\text{lpof}}(\text{guard}(x, b, \alpha, \beta), \psi \wedge \neg x)] \mid \psi \in \text{br}_{\beta}\}
\end{aligned}$$

By Lemma H.7:

$$\begin{aligned}
&= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \{[\langle \text{assume } b \rangle_x \circ \text{tr}_{\text{lpof}}(\alpha, \psi)] \mid \psi \in \text{br}_{\alpha}\} \cup \{[\langle \text{assume } \neg b \rangle_x \circ \text{tr}_{\text{lpof}}(\beta, \psi)] \mid \psi \in \text{br}_{\beta}\} \\
&= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta' \ll \beta} \{\langle \text{assume } b \rangle \circ \alpha'' \mid \alpha'' \in \text{tr}_{\text{fin}}(\alpha')\} \cup \{\langle \text{assume } \neg b \rangle \circ \beta'' \mid \beta'' \in \text{tr}_{\text{fin}}(\beta')\} \\
&= \{\langle \text{assume } b \rangle \circ \alpha'' \mid \alpha'' \in \text{tr}(\alpha)\} \cup \{\langle \text{assume } \neg b \rangle \circ \beta'' \mid \beta'' \in \text{tr}(\beta)\} \quad \blacktriangleleft
\end{aligned}$$

► **Lemma H.9.** For any $\alpha, \beta \in \text{lpof}_{\text{fin}}$, $\psi \in \text{br}_{\alpha}$, and $\psi' \in \text{br}_{\beta}$:

$$\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi') = \text{tr}_{\text{lpof}}(\alpha, \psi) \parallel_x \text{tr}_{\text{lpof}}(\beta, \psi')$$

Proof. We show the equality for each component. First, for the nodes, we have:

$$\begin{aligned}
N_{\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')} &= \{y \in N_{\alpha \parallel_x \beta} \mid \psi \wedge \psi' \Rightarrow \varphi_{\alpha \parallel_x \beta}(y)\} \\
&= \{x\} \cup \{y \in \text{nFork}_{\alpha} \mid \psi \Rightarrow \varphi_{\alpha}(y)\} \cup \{z \in \text{nFork}_{\beta} \mid \psi' \Rightarrow \varphi_{\beta}(z)\} \\
&= \{x\} \cup \text{nFork}_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \cup \text{nFork}_{\text{tr}_{\text{lpof}}(\beta, \psi')} \\
&= N_{\text{tr}_{\text{lpof}}(\alpha, \psi) \parallel_x \text{tr}_{\text{lpof}}(\beta, \psi')}
\end{aligned}$$

Now, for the orders:

$$\begin{aligned}
&<_{\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')} \\
&= <_{\alpha \parallel_x \beta} \cap (N_{\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')} \times N_{\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')}) \\
&= ((\{x\} \times (\text{nFork}_{\alpha} \cup \text{nFork}_{\beta})) \cup (<_{\alpha} \cap (\text{nFork}_{\alpha} \times \text{nFork}_{\alpha})) \cup (<_{\beta} \cap (\text{nFork}_{\beta} \times \text{nFork}_{\beta}))) \\
&\quad \cap (N_{\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')} \times N_{\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')}) \\
&= (\{x\} \times (\text{nFork}_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \cup \text{nFork}_{\text{tr}_{\text{lpof}}(\beta, \psi')})) \\
&\quad \cup (<_{\alpha} \cap (\text{nFork}_{\text{tr}_{\text{lpof}}(\alpha, \psi)} \times \text{nFork}_{\text{tr}_{\text{lpof}}(\alpha, \psi)})) \\
&\quad \cup (<_{\beta} \cap (\text{nFork}_{\text{tr}_{\text{lpof}}(\beta, \psi')} \times \text{nFork}_{\text{tr}_{\text{lpof}}(\beta, \psi')})) \\
&= <_{\text{tr}_{\text{lpof}}(\alpha, \psi) \parallel_x \text{tr}_{\text{lpof}}(\beta, \psi')}
\end{aligned}$$

Now take any y , clearly we have:

$$\varphi_{\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')}(y) = \varphi_{\text{tr}_{\text{lpof}}(\alpha, \psi) \parallel_x \text{tr}_{\text{lpof}}(\beta, \psi')}(y) = \text{true}$$

Now, if $y = x$, then $\lambda_{\alpha \parallel_x \beta}(y) = \text{fork}$, so:

$$\lambda_{\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')}(y) = \begin{cases} \lambda_{\alpha \parallel_x \beta}(y) & \text{if } \lambda_{\alpha \parallel_x \beta}(y) \notin \text{test} \\ \text{assume } \lambda_{\alpha \parallel_x \beta}(y) & \text{if } \lambda_{\alpha \parallel_x \beta}(y) \in \text{test} \text{ and } \psi \wedge \psi' \Rightarrow x \\ \text{assume } \neg \lambda_{\alpha \parallel_x \beta}(y) & \text{if } \lambda_{\alpha \parallel_x \beta}(y) \in \text{test} \text{ and } \psi \wedge \psi' \Rightarrow \neg x \end{cases}$$

$$\begin{aligned}
&= \lambda_{\alpha \parallel_x \beta}(y) \\
&= \text{fork} \\
&= \lambda_{\text{tr}_{\text{lpof}}(\alpha, \psi) \parallel_x \text{tr}_{\text{lpof}}(\beta, \psi')}(y)
\end{aligned}$$

Now, if $y \in \text{nFork}_\alpha$, then:

$$\begin{aligned}
\lambda_{\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')}(y) &= \begin{cases} \lambda_{\alpha \parallel_x \beta}(y) & \text{if } \lambda_{\alpha \parallel_x \beta}(y) \notin \text{test} \\ \mathbf{assume} \lambda_{\alpha \parallel_x \beta}(y) & \text{if } \lambda_{\alpha \parallel_x \beta}(y) \in \text{test} \text{ and } \psi \wedge \psi' \Rightarrow x \\ \mathbf{assume} \neg \lambda_{\alpha \parallel_x \beta}(y) & \text{if } \lambda_{\alpha \parallel_x \beta}(y) \in \text{test} \text{ and } \psi \wedge \psi' \Rightarrow \neg x \end{cases} \\
&= \begin{cases} \lambda_\alpha(y) & \text{if } \lambda_\alpha(y) \notin \text{test} \\ \mathbf{assume} \lambda_\alpha(y) & \text{if } \lambda_\alpha(y) \in \text{test} \text{ and } \psi \Rightarrow x \\ \mathbf{assume} \neg \lambda_\alpha(y) & \text{if } \lambda_\alpha(y) \in \text{test} \text{ and } \psi \Rightarrow \neg x \end{cases} \\
&= \lambda_{\text{tr}(\alpha, \psi)}(y) \\
&= \lambda_{\text{tr}_{\text{lpof}}(\alpha, \psi) \parallel_x \text{tr}_{\text{lpof}}(\beta, \psi')}(y)
\end{aligned}$$

The case where $y \in \text{nBot}(\beta)$ is nearly identical. \blacktriangleleft

Since \parallel is not monotone (see Remark 4.2), we need a different way to approximate infinite pomsets composed in parallel, which we now describe. Let $\text{root}(\gamma)$ be the label of the root node of the pomset γ and

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

i.e., $\gamma \ll_1 \gamma'$ is similar to the standard approximation order \ll on pomsets, but it prevents a fork node at the root of γ' from being converted to \perp , which would cause the root of a parallel composition involving γ' vs γ to have a different number of successors. Coming back to the example of Remark 4.2, we have that $[\alpha] \sqsubseteq_{\text{pom}} [\beta]$ and also $[\alpha] \ll [\beta]$, whereas $[\alpha] \not\ll_1 [\beta]$, since the root of β is **fork** but not so is the root of α .

We need two properties of \ll_1 . First:

$$\sup_{\gamma \ll_1 \gamma'} \gamma = \sup_{\gamma \ll \gamma'} \gamma \tag{7}$$

By Lemma 3.9 and Corollary E.9, we have that $\sup_{\gamma \ll \gamma'} \gamma = \gamma'$. If $\text{root}(\gamma') = \text{fork}$, then $\{\gamma \mid \gamma \ll_1 \gamma'\} = \{\gamma \mid \gamma \ll \gamma'\} \setminus \{\perp_{\text{pom}}\}$, which clearly has the same supremum, as we have only removed the bottom element from the set. If $\text{root}(\gamma') \neq \text{fork}$, then the orders are equivalent. The second property that we need is

$$[\alpha \parallel \beta]_{\text{fin}} \setminus \{\perp_{\text{pom}}\} = \{\alpha' \parallel \beta' \mid \alpha' \ll_1 \alpha, \beta' \ll_1 \beta\} \tag{8}$$

This can be easily seen by observing that \perp_{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 α and β .

► **Lemma H.10.** $\text{tr}(\alpha \parallel \beta) = \{\alpha' \parallel \beta' \mid \alpha' \in \text{tr}(\alpha), \beta' \in \text{tr}(\beta)\}$.

Proof.

$$\text{tr}(\alpha \parallel \beta) = \text{tr} \left(\sup_{\gamma \in [\alpha \parallel \beta]_{\text{fin}}} \gamma \right)$$

$$= \text{tr}(\langle \perp \rangle) \cup \text{tr} \left(\sup_{\gamma \in \lfloor \alpha \parallel \beta \rfloor_{\text{fin}} \setminus \{\perp_{\text{pom}}\}} \gamma \right)$$

By Equation (8).

$$\begin{aligned} &= \emptyset \cup \text{tr} \left(\sup_{\alpha' \ll_1 \alpha} \sup_{\beta' \ll_1 \beta} \alpha' \parallel \beta' \right) \\ &= \bigcup_{\alpha' \ll_1 \alpha} \bigcup_{\beta' \ll_1 \beta} \text{tr}_{\text{fin}}(\alpha' \parallel \beta') \\ &= \bigcup_{\alpha' \ll_1 \alpha} \bigcup_{\beta' \ll_1 \beta} \text{tr}_{\text{fin}}(\{\alpha \parallel_x \beta \mid \alpha \in \alpha', \beta \in \beta', x \in \text{nodes}\}) \\ &= \bigcup_{\alpha' \ll_1 \alpha} \bigcup_{\beta' \ll_1 \beta} \text{tr}_{\text{fin}}([\alpha \parallel_x \beta]) \\ &= \bigcup_{\alpha' \ll_1 \alpha} \bigcup_{\beta' \ll_1 \beta} \{[\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi)] \mid \psi \in \text{br}_{\alpha \parallel_x \beta}\} \end{aligned}$$

By Lemma H.9.

$$\begin{aligned} &= \bigcup_{\alpha' \ll_1 \alpha} \bigcup_{\beta' \ll_1 \beta} \{[\text{tr}_{\text{lpof}}(\alpha \parallel_x \beta, \psi \wedge \psi')]\mid \psi \in \text{br}_{\alpha}, \psi' \in \text{br}_{\beta}\} \\ &= \bigcup_{\alpha' \ll_1 \alpha} \bigcup_{\beta' \ll_1 \beta} \{[\{\alpha' \parallel_x \beta' \mid \alpha' \in \text{tr}_{\text{lpof}}(\alpha, \psi), \beta' \in \text{tr}_{\text{lpof}}(\beta, \psi')\}]\mid \psi \in \text{br}_{\alpha}, \psi' \in \text{br}_{\beta}\} \\ &= \bigcup_{\alpha' \ll_1 \alpha} \bigcup_{\beta' \ll_1 \beta} \{\alpha'' \parallel \beta'' \mid \alpha'' \in \{[\text{tr}_{\text{lpof}}(\alpha, \psi)] \mid \psi \in \text{br}_{\alpha}\}, \beta'' \in \{[\text{tr}_{\text{lpof}}(\beta, \psi')]\mid \psi' \in \text{br}_{\beta}\}\} \\ &= \bigcup_{\alpha' \ll_1 \alpha} \bigcup_{\beta' \ll_1 \beta} \{\alpha'' \parallel \beta'' \mid \alpha'' \in \text{tr}_{\text{fin}}(\alpha'), \beta'' \in \text{tr}_{\text{fin}}(\beta')\} \end{aligned}$$

By Equation (7).

$$= \{\alpha'' \parallel \beta'' \mid \alpha'' \in \text{tr}(\alpha), \beta'' \in \text{tr}(\beta)\} \quad \blacktriangleleft$$

► **Lemma H.11.** *If $\text{tr}(\llbracket C \rrbracket) = \llbracket C \rrbracket_{\text{PL}}$, for all $n \in \mathbb{N}$:*

$$\text{tr}(\Phi_{\langle C, b \rangle}^n(\perp_{\text{pom}})) = \Xi_{\langle C, b \rangle}^n(\emptyset)$$

where above we use $\perp_{\text{pom}} = \langle \perp \rangle$ to disambiguate.

Proof. The proof is by induction on n . If $n = 0$, then we have:

$$\text{tr}(\Phi_{\langle C, b \rangle}^0(\perp_{\text{pom}})) = \text{tr}(\perp_{\text{pom}}) = \{[\text{tr}_{\text{lpof}}(\langle \perp \rangle_x, \psi)] \mid \psi \in \text{br}_{\langle \perp \rangle_x}\} = \emptyset = \Xi_{\langle C, b \rangle}^0(\emptyset)$$

Where the second to last step follows from the fact that $\text{br}_{\langle \perp \rangle_x} = \emptyset$. Now, suppose the claim holds for n , we have:

$$\begin{aligned} &\text{tr}(\Phi_{\langle C, b \rangle}^{n+1}(\perp_{\text{pom}})) \\ &= \text{tr}(\Phi_{\langle C, b \rangle}(\Phi_{\langle C, b \rangle}^n(\perp_{\text{pom}}))) \\ &= \text{tr}(\text{guard}(b, \llbracket C \rrbracket \circ \Phi_{\langle C, b \rangle}^n(\emptyset), \llbracket \text{skip} \rrbracket)) \end{aligned}$$

By Lemma H.8.

$$= \left\{ \langle \text{assume } b \rangle \circ \alpha \mid \alpha \in \text{tr}(\llbracket C \rrbracket \circ \Phi_{\langle C, b \rangle}^n(\emptyset)) \right\} \cup \left\{ \langle \text{assume } b \rangle \circ \beta \mid \beta \in \text{tr}(\llbracket \text{skip} \rrbracket) \right\}$$

By Lemmas H.4 and H.6.

$$= \left\{ \langle \text{assume } b \rangle ; \alpha ; \beta \mid \alpha \in \text{tr}(\llbracket C \rrbracket), \beta \in \text{tr}(\Phi_{\langle C, b \rangle}^n(\emptyset)) \right\} \cup \{ \langle \text{assume } b \rangle ; \llbracket \text{skip} \rrbracket \}$$

By assumption, and the induction hypothesis:

$$\begin{aligned} &= \left\{ \langle \text{assume } b \rangle ; \alpha ; \beta \mid \alpha \in \llbracket C \rrbracket_{\text{PL}}, \beta \in \Xi_{\langle C, b \rangle}^n(\emptyset) \right\} \cup \{ \langle \text{assume } b \rangle ; \llbracket \text{skip} \rrbracket \} \\ &= \Xi_{\langle C, b \rangle} \left(\Xi_{\langle C, b \rangle}^n(\emptyset) \right) \\ &= \Xi_{\langle C, b \rangle}^{n+1}(\emptyset) \end{aligned}$$

H.4 Translations and Program Semantics

► **Lemma H.12.** *For any program $C \in \text{cmd}$, $\llbracket C \rrbracket_{\text{PL}} = \text{tr}(\llbracket C \rrbracket)$.*

Proof. The proof is by induction on the structure of the program.

- $C = \text{skip}$ or $C \in \text{act}$. Follows immediately from Lemma H.4.
- $C = C_1; C_2$.

$$\begin{aligned} \llbracket C_1; C_2 \rrbracket_{\text{PL}} &= \{ \alpha ; \beta \mid \alpha \in \llbracket C_1 \rrbracket_{\text{PL}}, \beta \in \llbracket C_2 \rrbracket_{\text{PL}} \} \\ &= \{ \alpha ; \beta \mid \alpha \in \text{tr}(\llbracket C_1 \rrbracket), \beta \in \text{tr}(\llbracket C_2 \rrbracket) \} && \text{(By the induction hypothesis.)} \\ &= \text{tr}(\llbracket C_1 \rrbracket ; \llbracket C_2 \rrbracket) && \text{(By Lemma H.6.)} \\ &= \text{tr}(\llbracket C_1; C_2 \rrbracket) \end{aligned}$$

- $C = C_1 \mid C_2$.

$$\begin{aligned} \llbracket C_1 \mid C_2 \rrbracket_{\text{PL}} &= \{ \alpha \parallel \beta \mid \alpha \in \llbracket C_1 \rrbracket_{\text{PL}}, \beta \in \llbracket C_2 \rrbracket_{\text{PL}} \} \\ &= \{ \alpha \parallel \beta \mid \alpha \in \text{tr}(\llbracket C_1 \rrbracket), \beta \in \text{tr}(\llbracket C_2 \rrbracket) \} && \text{(By the induction hypothesis.)} \\ &= \text{tr}(\llbracket C_1 \rrbracket \parallel \llbracket C_2 \rrbracket) && \text{(By Lemma H.10.)} \\ &= \text{tr}(\llbracket C_1 \mid C_2 \rrbracket) \end{aligned}$$

- $C = \text{if } b \text{ then } C_1 \text{ else } C_2$.

$$\begin{aligned} &\llbracket \text{if } b \text{ then } C_1 \text{ else } C_2 \rrbracket_{\text{PL}} \\ &= \{ \langle \text{assume } b \rangle ; \alpha \mid \alpha \in \llbracket C_1 \rrbracket_{\text{PL}} \} \cup \{ \langle \text{assume } \neg b \rangle ; \beta \mid \beta \in \llbracket C_2 \rrbracket_{\text{PL}} \} \end{aligned}$$

By the induction hypothesis:

$$= \{ \langle \text{assume } b \rangle ; \alpha \mid \alpha \in \text{tr}(\llbracket C_1 \rrbracket) \} \cup \{ \langle \text{assume } \neg b \rangle ; \beta \mid \beta \in \text{tr}(\llbracket C_2 \rrbracket) \}$$

By Lemma H.8.

$$\begin{aligned} &= \text{tr}(\text{guard}(b, \llbracket C_1 \rrbracket, \llbracket C_2 \rrbracket)) \\ &= \text{tr}(\llbracket \text{if } b \text{ then } C_1 \text{ else } C_2 \rrbracket) \end{aligned}$$

- $C = \text{while } b \text{ do } C'$.

$$\begin{aligned} &\llbracket \text{while } b \text{ do } C' \rrbracket_{\text{PL}} \\ &= \text{lfp}(\Xi_{\langle C', b \rangle}) \end{aligned}$$

$$\begin{aligned}
&= \bigcup_{n \in \mathbb{N}} \Xi_{\langle C', b \rangle}^n(\emptyset) && \text{(By the Kleene fixed point theorem.)} \\
&= \bigcup_{n \in \mathbb{N}} \text{tr} \left(\Phi_{\langle C', b \rangle}^n(\perp_{\text{pom}}) \right) && \text{(By Lemma H.11, and the induction hypothesis.)} \\
&= \text{tr} \left(\sup_{n \in \mathbb{N}} \Phi_{\langle C', b \rangle}^n(\perp_{\text{pom}}) \right) && \text{(By Scott continuity of tr.)} \\
&= \text{tr} \left(\text{lfp} \left(\Phi_{\langle C', b \rangle}^n \right) \right) && \text{(By the Kleene fixed point theorem.)} \\
&= \text{tr} (\llbracket \text{while } b \text{ do } C' \rrbracket) && \blacktriangleleft
\end{aligned}$$

H.5 Translations and Linearization

► **Lemma H.13.** *For any binary branching $\alpha \in \text{lpof}_{\text{fin}}(\text{label})$, $S \subseteq \text{nBot}_\alpha$, and $\psi \in \text{valid}_\alpha(S)$, then:*

$$\bigcup_{\psi' \in \text{br}_\alpha : \psi' \Rightarrow \psi} \text{next}(\text{tr}_{\text{lpof}}(\alpha, \psi'), \text{true}, S) = \text{next}(\alpha, \psi, S) \setminus \{x \in N_\alpha \mid \varphi_\alpha(x) \Rightarrow \text{stuck}_\alpha\}$$

Proof. We show the inclusion in both directions.

Take any $x \in \bigcup_{\psi' \in \text{br}_\alpha : \psi' \Rightarrow \psi} \text{next}(\text{tr}_{\text{lpof}}(\alpha, \psi'), \text{true}, S)$, so there is a $\psi' \in \text{br}_\alpha$ such that $\psi' \Rightarrow \psi$ and $x \in N_{\text{tr}_{\text{lpof}}(\alpha, \psi')} \setminus S$ such that $x \downarrow_{\text{tr}_{\text{lpof}}(\alpha, \psi')} \subseteq S$. From $x \in N_{\text{tr}_{\text{lpof}}(\alpha, \psi')} \setminus S$, we get that $x \in N_\alpha \setminus S$ and $\psi' \Rightarrow \varphi_\alpha(x)$. 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 $y \in N_{\text{tr}_{\text{lpof}}(\alpha, \psi')}$, and therefore $y <_{\text{tr}_{\text{lpof}}(\alpha, \psi')} x$, 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 $\psi \in \text{valid}_\alpha(S)$, then ψ must assign a truth value to every test in S , and since we just showed that $x \downarrow_\alpha \subseteq S$, then $\text{vars}(\varphi_\alpha(x)) \subseteq S$, and so we must have that $\psi' \Rightarrow \varphi_\alpha(x)$. Therefore, we have established all the conditions of $x \in \text{next}(\alpha, \psi, S)$. Clearly also $\varphi_\alpha(x) \not\Rightarrow \text{stuck}_\alpha$ since $\psi' \Rightarrow \varphi_\alpha(x)$ and $\psi' \Rightarrow \neg \text{stuck}_\alpha$.

Now take any $x \in \text{next}(\alpha, \psi, S) \setminus \{x \in N_\alpha \mid \varphi_\alpha(x) \Rightarrow \text{stuck}_\alpha\}$, so $x \in N_\alpha \setminus S$, $\psi \Rightarrow \varphi_\alpha(x)$, $x \downarrow_\alpha \subseteq S$, and $\varphi_\alpha(x) \not\Rightarrow \text{stuck}_\alpha$. Since $\varphi_\alpha(x) \not\Rightarrow \text{stuck}_\alpha$, then $x \in \text{ext}_\alpha$, and so there is clearly a $\psi' \in \text{br}_\alpha$ such that $\psi' \Rightarrow \varphi_\alpha(x)$. This means that $x \in N_{\text{tr}_{\text{lpof}}(\alpha, \psi')} \setminus S$. Since by definition $\varphi_{\text{tr}_{\text{lpof}}(\alpha, \psi')}(x) = \text{true}$, then clearly $\text{true} \Rightarrow \varphi_{\text{tr}_{\text{lpof}}(\alpha, \psi')}(x)$. Finally, we will show that $x \downarrow_{\text{tr}_{\text{lpof}}(\alpha, \psi')} \subseteq S$. Take any $y <_{\text{tr}_{\text{lpof}}(\alpha, \psi')} x$, this means that $y <_\alpha x$, so clearly $y \in S$. Therefore, we have that $x \in \text{next}(\text{tr}_{\text{lpof}}(\alpha, \psi'), \text{true}, S) \subseteq \bigcup_{\psi' \in \text{br}_\alpha : \psi' \Rightarrow \psi} \text{next}(\text{tr}_{\text{lpof}}(\alpha, \psi'), \text{true}, S)$. \blacktriangleleft

► **Lemma H.14.** *If $S \subseteq \text{nBot}_\alpha$, $\psi \in \text{valid}_\alpha(S)$ and $\psi \Rightarrow \text{stuck}_\alpha$, then $\mathcal{L}_{\text{lpof}}(\alpha, \psi, S) = \emptyset$.*

Proof. The proof is by induction on the size of the set $\text{next}^*(\alpha, \psi, S)$. If $\text{next}^*(\alpha, \psi, S) = \emptyset$, then by Lemma G.6 we know that $\psi \in \text{br}_\alpha$, but this is impossible since $\psi \Rightarrow \text{stuck}_\alpha$. Therefore, it cannot be the case that $\text{next}^*(\alpha, \psi, S) = \emptyset$.

Now suppose that $\text{next}^*(\alpha, \psi, S) \neq \emptyset$. By Lemma G.7, we know that $\text{next}(\alpha, \psi, S) \neq \emptyset$. So, we have:

$$\mathcal{L}_{\text{lpof}}(\alpha, \psi, S) = \bigcup_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi \wedge \langle x = \llbracket \lambda_\alpha(x) \rrbracket_{\text{test}}(s) \rangle, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \emptyset & \text{if } \lambda_\alpha(x) = \perp \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases}$$

By the induction hypothesis in the first, second, and fourth cases, we get:

$$\begin{aligned}
&= \bigcup_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \emptyset & \text{if } \lambda_\alpha(x) \in \text{act} \\ \emptyset & \text{if } \lambda_\alpha(x) \in \text{test} \\ \emptyset & \text{if } \lambda_\alpha(x) = \perp \\ \emptyset & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases} \\
&= \bigcup_{x \in \text{next}(\alpha, \psi, S)} \emptyset = \emptyset
\end{aligned}$$

► **Lemma H.15.** For any binary branching $\alpha \in \text{lpof}_{\text{fin}}(\text{label})$, $S \subseteq \text{nBot}_\alpha$, and $\psi \in \text{valid}_\alpha(S)$:

$$\bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi} \mathcal{L}_{\text{lpof}}(\text{tr}_{\text{lpof}}(\alpha, \psi'), \text{true}, S)(s) = \mathcal{L}_{\text{lpof}}(\alpha, \psi, S)(s)$$

Proof. The proof is by induction on the size of the set $\text{next}^*(\alpha, \psi, S)$. If $\text{next}^*(\alpha, \psi, S) = \emptyset$, then clearly $\text{next}(\alpha, \psi, S) = \emptyset$ and by Lemma G.6, we know that $\psi \in \text{br}_\alpha$. That means that the union on the left side is only over a single term, generated from ψ . Now, $\text{tr}_{\text{lpof}}(\alpha, \psi)$ contains all those nodes x such that $\psi \Rightarrow \varphi_\alpha(x)$, but we know from $\text{next}^*(\alpha, \psi, S) = \emptyset$ that all such nodes are already in S , therefore by Lemma H.13, $\text{next}(\text{tr}_{\text{lpof}}(\alpha, \psi), \text{true}, S) = \emptyset$ too. This gives us

$$\bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi} \mathcal{L}_{\text{lpof}}(\text{tr}_{\text{lpof}}(\alpha, \psi'), \text{true}, S)(s) = \mathcal{L}_{\text{lpof}}(\text{tr}_{\text{lpof}}(\alpha, \psi), \text{true}, S)(s) = \{s\} = \mathcal{L}_{\text{lpof}}(\alpha, \psi, S)(s)$$

Now, assume that $\text{next}^*(\alpha, \psi, S) \neq \emptyset$, so by Lemma G.7, then $\text{next}(\alpha, \psi, S) \neq \emptyset$.

Suppose that $\varphi_\alpha(x) \Rightarrow \text{stuck}_\alpha$ for some $x \in \text{next}(\alpha, \psi, S)$. This means that $\psi \Rightarrow \varphi_\alpha(x) \Rightarrow \text{stuck}_\alpha$, so clearly there can be no $\psi \in \text{br}_\alpha$ such that $\psi' \Rightarrow \psi$, so $\bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi} \mathcal{L}_{\text{lpof}}(\text{tr}_{\text{lpof}}(\alpha, \psi'), \text{true}, S)(s) = \emptyset$. In addition, by Lemma H.14, $\mathcal{L}_{\text{lpof}}(\alpha, \psi, S)(s) = \emptyset$ too.

So, we are left with the case where $\varphi_\alpha(x) \not\Rightarrow \text{stuck}_\alpha$ for all $x \in \text{next}(\alpha, \psi, S)$. Let $\alpha_{\psi'} = \text{tr}_{\text{lpof}}(\alpha, \psi')$. We now have:

$$\begin{aligned}
&\bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi} \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S) \\
&= \bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi} \bigcup_{x \in \text{next}(\alpha_{\psi'}, \text{true}, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})^\dagger(\llbracket \lambda_{\alpha_{\psi'}}(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_{\alpha_{\psi'}}(x) \in \text{act}_{\text{PL}} \\ \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true} \wedge \llbracket x = \llbracket \lambda_{\alpha_{\psi'}}(x) \rrbracket_{\text{test}}(s) \rrbracket, S \cup \{x\})(s) & \text{if } \lambda_{\alpha_{\psi'}}(x) \in \text{test}_{\text{PL}} \\ \emptyset & \text{if } \lambda_{\alpha_{\psi'}}(x) = \perp \\ \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})(s) & \text{if } \lambda_{\alpha_{\psi'}}(x) = \text{fork} \end{cases}
\end{aligned}$$

Since $\alpha_{\psi'}$ has no test or \perp 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.

$$\begin{aligned}
&= \bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi} \bigcup_{x \in \text{next}(\alpha_{\psi'}, \text{true}, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})^\dagger(\llbracket \text{assume } \lambda_\alpha(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) \in \text{test} \text{ and } \psi' \Rightarrow x \\ \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})^\dagger(\llbracket \text{assume } \neg \lambda_\alpha(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) \in \text{test} \text{ and } \psi' \Rightarrow \neg x \\ \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases}
\end{aligned}$$

By Lemma H.13, and the fact that $\psi \Rightarrow \varphi_\alpha(x)$ for all $x \in \text{next}(\alpha, \psi, S)$, we can rearrange the unions and move the inner one inside of the conditional.

$$\begin{aligned}
&= \bigcup_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi} \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi \wedge x} \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{test}}(s)) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi \wedge \neg x} \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})^\dagger(\llbracket \neg \lambda_\alpha(x) \rrbracket_{\text{test}}(s)) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \bigcup_{\psi' \in \text{br}_\alpha: \psi' \Rightarrow \psi} \mathcal{L}_{\text{lpof}}(\alpha_{\psi'}, \text{true}, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases}
\end{aligned}$$

By the induction hypothesis.

$$= \bigcup_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi \wedge x, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{test}}(s)) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \cup \mathcal{L}_{\text{lpof}}(\alpha, \psi \wedge \neg x, S \cup \{x\})^\dagger(\llbracket \neg \lambda_\alpha(x) \rrbracket_{\text{test}}(s)) & \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases}$$

Exactly one of $\llbracket \lambda_\alpha(x) \rrbracket_{\text{test}}(s)$ and $\llbracket \neg \lambda_\alpha(x) \rrbracket_{\text{test}}(s)$ will evaluate to $\{s\}$ and the other will evaluate to \emptyset , so we can consolidate the terms.

$$= \bigcup_{x \in \text{next}(\alpha, \psi, S)} \begin{cases} \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})^\dagger(\llbracket \lambda_\alpha(x) \rrbracket_{\text{act}}(s)) & \text{if } \lambda_\alpha(x) \in \text{act} \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi \wedge (x = \llbracket \lambda_\alpha(x) \rrbracket_{\text{test}}(s)), S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) \in \text{test} \\ \mathcal{L}_{\text{lpof}}(\alpha, \psi, S \cup \{x\})(s) & \text{if } \lambda_\alpha(x) = \text{fork} \end{cases}$$

$$= \mathcal{L}_{\text{lpof}}(\alpha, \psi, S)$$

► **Lemma H.16.** *For any $\alpha \in \text{pom}$ and $s \in S$, $\mathcal{L}(\alpha)(s) = \mathcal{L}_{\text{PL}}(\text{tr}(\alpha))(s)$.*

Proof.

$$\begin{aligned} \mathcal{L}(\alpha)(s) &= \bigcup_{\alpha' \ll \alpha} \mathcal{L}_{\text{fin}}(\alpha')(s) \\ &= \bigcup_{[\alpha] \ll \alpha} \mathcal{L}_{\text{lpof}}(\alpha, \text{true}, \emptyset)(s) \\ &= \bigcup_{[\alpha] \ll \alpha} \bigcup_{\psi \in \text{br}_\alpha} \mathcal{L}_{\text{lpof}}(\text{tr}_{\text{lpof}}(\alpha, \psi), \text{true}, \emptyset)(s) && \text{By Lemma H.15} \\ &= \bigcup_{\alpha' \ll \alpha} \bigcup_{\beta \in \text{tr}_{\text{fin}}(\alpha')} \mathcal{L}_{\text{fin}}(\beta)(s) \\ &= \bigcup_{\beta \in \bigcup_{\alpha' \ll \alpha} \text{tr}_{\text{fin}}(\alpha')} \mathcal{L}_{\text{fin}}(\beta)(s) \\ &= \bigcup_{\beta \in \text{tr}(\alpha)} \mathcal{L}_{\text{fin}}(\beta)(s) \\ &= \mathcal{L}_{\text{PL}}(\text{tr}(\alpha)) \end{aligned}$$

H.6 Main Theorem

► **Theorem 6.1** (Equivalence of Semantics). *The following diagram commutes:*

$$\begin{array}{ccc} \text{cmd} & \xrightarrow{\llbracket - \rrbracket} & \text{pom} \\ \llbracket - \rrbracket_{\text{PL}} \downarrow & \text{tr} \swarrow & \downarrow \mathcal{L} \\ \text{pomLang} & \xrightarrow{\mathcal{L}_{\text{PL}}} & (S \rightarrow \mathcal{P}(S)) \end{array}$$

Proof. It suffices to show that $\llbracket - \rrbracket_{\text{PL}} = \text{tr} \circ \llbracket - \rrbracket$ and $\mathcal{L}_{\text{PL}} \circ \text{tr} = \mathcal{L}$, which follows immediately from Lemmas H.12 and H.16. We then have:

$$\mathcal{L}_{\text{PL}} \circ \llbracket - \rrbracket_{\text{PL}} = \mathcal{L}_{\text{PL}} \circ (\text{tr} \circ \llbracket - \rrbracket) = (\mathcal{L}_{\text{PL}} \circ \text{tr}) \circ \llbracket - \rrbracket = \mathcal{L} \circ \llbracket - \rrbracket$$