An Algebraic Theory for Shared-State Concurrency*

Yotam Dvir[1][0000–0002–6507–3791], Ohad Kammar[2][0000–0002–2071–0929], and Ori Lahav[1][0000–0003–4305–6998]

[1] Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel
yotamdvir@mail.tau.ac.il, orilahav@tau.ac.il

[2] School of Informatics, University of Edinburgh, Edinburgh, Scotland
ohad.kammar@ed.ac.uk

Keywords: shared state · concurrency · denotational semantics · monads · equational theory · program refinement · program equivalence · compiler transformations · compiler optimisations

Abstract. We present a monadic denotational semantics for a higher-order programming language with shared-state concurrency, i.e. global-state in the presence of interleaving concurrency. Central to our approach is the use of Plotkin and Power's algebraic effect methodology: designing an equational theory that captures the intended semantics, and proving a monadic representation theorem for it. We use Hyland et al.'s equational theory of resumptions that extends non-deterministic global-state with an operator for yielding to the environment. The representation is based on Brookes-style traces. Based on this representation we define a denotational semantics that is directionally adequate with respect to a standard operational semantics. We use this semantics to justify compiler transformations of interest: redundant access eliminations, each following from a mundane algebraic calculation; while structural transformations follow from reasoning over the monad's interface.

1 Introduction

Denotational semantics $[\![-]\!]$ associates every program M with its meaning, i.e. its denotation, $[\![M]\!]$. A key feature of a denotational semantics is compositionality: the denotation of a program depends only on the denotations of its constituents.

As a concrete example, consider an imperative language that manipulates a memory store $\sigma \in \mathbb{S}$, and denotational semantics for it that associates with each

* Supported by the Israel Science Foundation (grant numbers 1566/18 and 814/22) and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 851811); and by a Royal Society University Research Fellowship and Enhancement Award. First published in the APLAS 2022: Programming Languages and Systems pp 324 by Springer Nature, https://doi.org/10.1007/978-3-031-21037-2_1.

program M a denotation $\llbracket M \rrbracket : \mathbb{S} \to \mathbb{S}$ modelling how M transforms the store. For example – denoting by $\sigma[\mathsf{a} \mapsto v]$ the store that is the same as σ but with its value at a changed to v – we have $\llbracket \mathsf{a} \mathrel{\mathbf{:=}} v \rrbracket \sigma = \sigma[\mathsf{a} \mapsto v]$. Compositionality manifests in the semantics of program sequencing: $\llbracket M \mathbin{;} N \rrbracket \sigma = \llbracket N \rrbracket (\llbracket M \rrbracket \sigma)$. Thus $\llbracket \mathsf{a} \mathrel{\mathbf{:=}} 0 \mathbin{;} \mathsf{a} \mathrel{\mathbf{:=}} 1 \rrbracket \sigma = \llbracket \mathsf{a} \mathrel{\mathbf{:=}} 1 \rrbracket (\llbracket \mathsf{a} \mathrel{\mathbf{:=}} 0 \rrbracket \sigma) = (\sigma[\mathsf{a} \mapsto 0])[\mathsf{a} \mapsto 1] = \sigma[\mathsf{a} \mapsto 1]$. Incidentally, we also have $\llbracket \mathsf{a} \mathrel{\mathbf{:=}} 1 \rrbracket \sigma = \sigma[\mathsf{a} \mapsto 1]$, and so $\llbracket \mathsf{a} \mathrel{\mathbf{:=}} 0 \mathbin{;} \mathsf{a} \mathrel{\mathbf{:=}} 1 \rrbracket = \llbracket \mathsf{a} \mathrel{\mathbf{:=}} 1 \rrbracket$.

A desirable property of a denotational semantics $\llbracket - \rrbracket$ is *adequacy*, meaning that $\llbracket M \rrbracket = \llbracket N \rrbracket$ implies that M and N are contextually equivalent: replacing N with M within some larger program does not affect the possible results of executing that program. Contextual equivalence is useful for optimizations: for example, M could have better runtime performance than N. Adequate denotational semantics can justify optimizations without quantifying over all program contexts, serving in this way as a basis for validating compiler optimizations.

Returning to the example above, although $\llbracket \mathsf{a} \mathrel{\mathbf{:=}} 0 \mathbin{;} \mathsf{a} \mathrel{\mathbf{:=}} 1 \rrbracket = \llbracket \mathsf{a} \mathrel{\mathbf{:=}} 1 \rrbracket$, in the presence of concurrency $\mathsf{a}\mathbf{:=}0\mathbf{;}\mathsf{a}\mathbf{:=}1$ and $\mathsf{a}\mathbf{:=}1$ are not contextually equivalent. For example, if $\mathsf{b}\mathbf{:=}\mathsf{a}\mathbf{?}$ (read from a and write the result to b) is executed concurrently, it could write 0 to b only with the first program. Therefore, the semantics we defined is inadequate for a concurrent programming language; differentiating between these programs requires a more sophisticated denotational semantics.

Moreover, the *transformation* $\mathsf{a} \mathrel{\mathbf{:=}} 0 \mathbin{;} \mathsf{a} \mathrel{\mathbf{:=}} 1 \twoheadrightarrow \mathsf{a} \mathrel{\mathbf{:=}} 1$ eliminating the first, redundant memory access is valid in the presence of concurrency, even though the programs are not equivalent. Indeed, a compiler applying this simplification within a program would not introduce any additional possible results (though it may eliminate some), and in particular it would conserve the correctness of the program. We would like our semantics to be able to justify such transformations.

This leads us to the concept of *directional adequacy*, a useful refinement of adequacy. Given a partial order $\leqslant$ on the set of denotations, the denotational semantics is directionally adequate (w.r.t. $\leqslant$) if $\llbracket M \rrbracket \leqslant \llbracket N \rrbracket$ implies that M contextually refines N: replacing N with M within some larger program does not introduce new possible results of executing that program. Thus, directional adequacy can justify the transformation $N \twoheadrightarrow M$ even if it is not an equivalence.

In this paper we define directionally-adequate denotational semantics for a higher-order language, subsuming the imperative language above, that justifies the above transformation along with other standard memory access eliminations:

$$\begin{array}{rcll}
\ell \mathrel{\mathbf{:=}} w \mathbin{;} \ell \mathrel{\mathbf{:=}} v & \twoheadrightarrow & \ell \mathrel{\mathbf{:=}} v & \text{(write ; write)} \\
\ell \mathrel{\mathbf{:=}} v \mathbin{;} \ell\mathbf{?} & \twoheadrightarrow & \ell \mathrel{\mathbf{:=}} v \mathbin{;} v & \text{(write ; read)} \\
\mathbf{let}\,\mathtt{x} = \ell\mathbf{?}\,\mathbf{in}\,\ell \mathrel{\mathbf{:=}} \mathtt{x} \mathbin{;} \mathtt{x} & \twoheadrightarrow & \ell\mathbf{?} & \text{(read ; write)} \\
\mathbf{let}\,\mathtt{x} = \ell\mathbf{?}\,\mathbf{in}\,\mathbf{let}\,\mathtt{y} = \ell\mathbf{?}\,\mathbf{in}\,\langle \mathtt{x}, \mathtt{y} \rangle & \twoheadrightarrow & \mathbf{let}\,\mathtt{x} = \ell\mathbf{?}\,\mathbf{in}\,\langle \mathtt{x}, \mathtt{x} \rangle & \text{(read ; read)} \\
\ell\mathbf{?} \mathbin{;} M & \twoheadrightarrow & M & \text{(irrelevant read)}
\end{array}$$

Other transformations and equivalences this semantics validates are structural ones, such as $\mathbf{if}\,M\,\mathbf{then}\,N\,\mathbf{else}\,N \cong M \mathbin{;} N$; and concurrency-related ones, such as $(M \parallel N) \mathbin{;} K \twoheadrightarrow M \mathbin{;} N \mathbin{;} K$.

None of these transformations are novel. Rather, the contribution of this paper is in the methodology that is used to justify them, fitting shared-state concurrency semantics into a general, uniform model structure. In particular, each memory access eliminations is proven correct via a mundane algebraic calculation; and the structural transformations can be justified using simple structural arguments that abstract away the details of our particular semantics.

Methodology The use of monads to study the denotational semantics of effects [30] has proven fruitful, especially with its recent refinement with algebraic operators and equational theories [9, 10, 23, 35, 36]. We follow the algebraic approach to define denotational semantics for a simple higher-order concurrent programming language, using an equational theory extending non-deterministic global-state with a single unary algebraic operator for yielding computation to a concurrently running program [1]. We find a concrete representation of the monad this theory induces based on sets of traces [4, 6], and use it to define a directionally adequate denotational semantics. From this adequacy result we deduce various program transformations via routine calculations.

The advantages of denotational semantics defined with this approach include:

Uniformity. Theories are stated using the same general framework. This uniformity means that many theoretical results can be stated in general terms, applying to all theories. Even if a theorem, like adequacy, must be proven separately for each theory, it is likely that a similar proof technique can be used, and experience can guide the construction of the proof.

Comparability. Comparing and contrasting theories is convenient due to uniformity [23]. While our language and semantics is very different from Abadi and Plotkin's [1], the equational theory we obtain easily compares to theirs.

Modularity. Since the theories are stated algebraically, using operators and equations, they are amenable to be combined to form larger theories. Some combinations are the result of general theory-combinators, such as the theory of non-deterministic global-state resulting from combining the theory of global-state [34] with the theory of non-determinism [14]. In this combined theory, equations that are provable in each separate theory remain provable. Even if the combination is bespoke, using an algebraic theory breaks down the problem into smaller components [8].

Abstraction. The semantics we define for the fragment of our language without shared-state is identical in form to the standard semantics, by using the monad operations. Therefore, any structural transformation proven using these abstractions remains valid in the language extended with shared-state.

Implementability. Monads are ubiquitous as a computational device in functional programming languages, such as Haskell. Thus a theory based on a monad may in the future form a bridge to implementation.

Outline The remaining sections are as follows. The next section provides background to the problem and overviews our results in a simplified manner. Then we dive into the weeds, starting with a succinct presentation of the equational theory and related notions (§3). We then work our way gradually to define the induced monad's concrete representation (§4). Next we define the denotations using this representation (§5). With everything in place, we present our metatheoretical results and use them to justify program transformations and equivalences (§6). We conclude with a discussion of related work and future prospects (§7).

2 Overview

Our setting is a simple programming language with state. We fix a finite set of *locations* $\mathbb{L} \coloneqq \{\mathtt{l}_1, \ldots, \mathtt{l}_{\bar{\mathtt{n}}}\}$ and a finite set of *(storable) values* $\mathbb{V} \coloneqq \{\mathtt{v}_1, \ldots, \mathtt{v}_{\bar{\mathtt{m}}}\}$. A *store* σ is an element of $\mathbb{S} \coloneqq \mathbb{L} \to \mathbb{V}$, where $\sigma\left[\ell \mapsto v\right]$ is the store that is equal to σ except (perhaps) at ℓ, where it takes the value v. We use subscripts to apply stores to locations, i.e. we write σ_ℓ instead of $\sigma\ell$. In examples we often assume $\mathbb{L} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and $\mathbb{V} = \{0, 1\}$, and write stores in matrix notation, e.g. $\left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & 1 \end{smallmatrix}\right)$.

The language is based on a standard extension of Moggi's [30] computational lambda calculus with products and variants (labelled sums), further extended with three shared-state constructs. Many other constructs are defined using syntactic sugar, such as if-statements and booleans, let-bindings, and program sequencing. The core syntax is presented below (where $n \geq 0$):

$$\begin{aligned}
G &::= \mathbf{(}G_1 \ast \cdots \ast G_n\mathbf{)} \mid \{\iota_1 \ \mathbf{of}\ G_1 \mid \cdots \mid \iota_n \ \mathbf{of}\ G_n\} && \text{(Ground types)} \\
A, B &::= \mathbf{(}A_1 \ast \cdots \ast A_n\mathbf{)} \mid \{\iota_1 \ \mathbf{of}\ A_1 \mid \cdots \mid \iota_n \ \mathbf{of}\ A_n\} \mid A \texttt{ -> } B && \text{(Types)} \\
V, W &::= \langle V_1, \ldots, V_n\rangle \mid \iota\, V \mid \lambda\mathtt{x}.\, M && \text{(Values)} \\
M, N &::= \mathtt{x} \mid \langle M_1, \ldots, M_n\rangle \mid \iota\, M \mid \lambda\mathtt{x}.\, M \mid M\, N && \text{(Terms)} \\
&\quad \mid \mathbf{match}\, M\, \mathbf{with}\, \langle \mathtt{x}_1, \ldots, \mathtt{x}_n\rangle \texttt{ -> } N \\
&\quad \mid \mathbf{case}\, M\, \mathbf{of}\, \{\iota_1\, \mathtt{x}_1 \texttt{ -> } N_1 \mid \cdots \mid \iota_n\, \mathtt{x}_n \texttt{ -> } N_n\} \\
&\quad \mid M\mathbf{?} \mid M \mathrel{\mathbf{:=}} N \mid M \parallel N
\end{aligned}$$

The typing rules for the shared-state constructs appear at the top of Figure 1, where we define $\mathbf{Loc} \coloneqq \{\mathtt{l}_1\, \mathbf{of}\, \mathbf{()} \mid \cdots \mid \mathtt{l}_{\bar{\mathtt{n}}}\, \mathbf{of}\, \mathbf{()}\}$ and $\mathbf{Val} \coloneqq \{\mathtt{v}_1\, \mathbf{of}\, \mathbf{()} \mid \cdots \mid \mathtt{v}_{\bar{\mathtt{m}}}\, \mathbf{of}\, \mathbf{()}\}$.

The language is equipped with a call-by-value, small-step operational semantics $\sigma, M \rightsquigarrow \rho, N$, meaning that the program M executed from store σ progresses to N with store ρ. The operational semantics for the shared-state constructs appears at the bottom of Figure 1. Parallel execution is interpreted via a standard interleaving semantics, ultimately returning the pair of the results of each side, and synchronizing on their completion. The reflexive-transitive closure of $\rightsquigarrow$ is denoted by $\rightsquigarrow^{*}$. The operational semantics can be seen in action in Example 2.

2.1 Global-State (for Sequential Computation)

To make this exposition more accessible, we focus on sequential computation before advancing to denotational semantics of concurrent computation. Sequential computations with global state cause two kinds of side-effects: looking a value up

$$\boxed{\Gamma \vdash M : A}$$

$$\frac{\Gamma \vdash M : \mathbf{Loc}}{\Gamma \vdash M\mathbf{?} : \mathbf{Val}} \qquad \frac{\Gamma \vdash M : \mathbf{Loc} \qquad \Gamma \vdash N : \mathbf{Val}}{\Gamma \vdash M \mathrel{\mathbf{:=}} N : \mathbf{()}} \qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash M \parallel N : \mathbf{(}A * B\mathbf{)}}$$

$$\boxed{\sigma, M \rightsquigarrow \sigma', M'}$$

$$\frac{\sigma, M \rightsquigarrow \sigma', M'}{\sigma, M\mathbf{?} \rightsquigarrow \sigma', M'\mathbf{?}} \qquad \frac{}{\sigma, \ell\mathbf{?} \rightsquigarrow \sigma, \sigma_\ell} \qquad \frac{\sigma, M \rightsquigarrow \sigma', M'}{\sigma, M \mathrel{\mathbf{:=}} N \rightsquigarrow \sigma', M' \mathrel{\mathbf{:=}} N}$$

$$\frac{\sigma, N \rightsquigarrow \sigma', N'}{\sigma, V \mathrel{\mathbf{:=}} N \rightsquigarrow \sigma', V \mathrel{\mathbf{:=}} N'} \qquad \frac{}{\sigma, \ell \mathrel{\mathbf{:=}} v \rightsquigarrow \sigma\left[\ell \mapsto v\right], \langle\rangle} \qquad \frac{\sigma, M \rightsquigarrow \sigma', M'}{\sigma, M \parallel N \rightsquigarrow \sigma', M' \parallel N}$$

$$\frac{\sigma, N \rightsquigarrow \sigma', N'}{\sigma, M \parallel N \rightsquigarrow \sigma', M \parallel N'} \qquad \frac{}{\sigma, V \parallel W \rightsquigarrow \sigma, \langle V, W\rangle}$$

Fig. 1. Typing and operational semantics of the shared-state constructs.

in the store, and updating a value in the store. Plotkin and Power [34] propose a corresponding equational theory with two operators:

Lookup. Suppose $\ell \in \mathbb{L}$, and $\left(t_v\right)_{v\in\mathbb{V}}$ is a $\mathbb{V}$-indexed sequence of terms. Then $\mathrm{L}_\ell \left(t_v\right)_{v\in\mathbb{V}}$ is a term representing looking the value in ℓ up, calling it v, and continuing the computation with t_v. We write $\mathrm{L}_\ell \left(v.\ t_v\right)$ instead of $\mathrm{L}_\ell \left(t_v\right)_{v\in\mathbb{V}}$.

Update. Suppose $\ell \in \mathbb{L}$, $v \in \mathbb{V}$, and t is a term. Then $\mathrm{U}_{\ell,v} t$ is a term representing updating the value in ℓ to v and continuing the computation with t.

The equations of the theory of global-state are generated by taking the closure of the axioms – listed at the top of Figure 2 – under reflexivity, symmetry, transitivity, and substitution. The grayed-out axioms happen to be derivable, and are included for presentation's sake. The theory of global-state can be used to define adequate denotational semantics for the sequential fragment of our language, obtained by removing concurrent execution ($\parallel$).

Example 1. Global-state includes the following equation (1) which, when considering sequential programs, represents the program equivalence (2):

$$\mathrm{L}_{\mathsf{b}} \left(v.\ \mathrm{U}_{\mathsf{a},v} \mathrm{L}_{\mathsf{c}} \left(w.\ \mathrm{U}_{\mathsf{a},w} \left\langle\right\rangle\right)\right) = \mathrm{L}_{\mathsf{c}} \left(w.\ \mathrm{U}_{\mathsf{a},w} \left\langle\right\rangle\right) \tag{1}$$

$$\mathsf{a} \mathrel{\mathbf{:=}} \mathsf{b}\mathbf{?} \mathbin{\mathbf{;}} \mathsf{a} \mathrel{\mathbf{:=}} \mathsf{c}\mathbf{?} \cong \mathsf{a} \mathrel{\mathbf{:=}} \mathsf{c}\mathbf{?} \tag{2}$$

2.2 Shared-State

The equivalence (2) from Example 1 fails in the concurrent setting, since the two program can be differentiated by program contexts with concurrency:

Global-State

$\mathtt{UL}$-det	$\mathrm{U}_{\ell,w}\mathrm{L}_{\ell}\left(v.\ x_v\right) = \mathrm{U}_{\ell,w}x_w$	
$\mathtt{UU}$-last	$\mathrm{U}_{\ell,v}\mathrm{U}_{\ell,w}x = \mathrm{U}_{\ell,w}x$	
$\mathtt{LU}$-noop	$\mathrm{L}_{\ell}\left(v.\ \mathrm{U}_{\ell,v}x\right) = x$	
$\mathtt{LL}$-diag	$\mathrm{L}_{\ell}\left(v.\ \mathrm{L}_{\ell}\left(w.\ x_{v,w}\right)\right) = \mathrm{L}_{\ell}\left(v.\ x_{v,v}\right)$	
$\mathtt{UU}$-comm	$\mathrm{U}_{\ell,v}\mathrm{U}_{\ell',w}x = \mathrm{U}_{\ell',w}\mathrm{U}_{\ell,v}x$	$\ell \neq \ell'$
$\mathtt{LU}$-comm	$\mathrm{L}_{\ell}\left(v.\ \mathrm{U}_{\ell',w}x_v\right) = \mathrm{U}_{\ell',w}\mathrm{L}_{\ell}\left(v.\ x_v\right)$	$\ell \neq \ell'$
$\mathtt{LL}$-comm	$\mathrm{L}_{\ell}\left(v.\ \mathrm{L}_{\ell'}\left(w.\ x_{v,w}\right)\right) = \mathrm{L}_{\ell'}\left(w.\ \mathrm{L}_{\ell}\left(v.\ x_{v,w}\right)\right)$	

Non-Determinism

ND-return	$\bigvee_{\imath<1} x = x$	
ND-epi	$\bigvee_{\jmath<\beta} x_{\jmath} = \bigvee_{\imath<\alpha} x_{\varphi\imath}$	surjective $\varphi : \alpha \to \beta$
ND-join	$\bigvee_{\imath<\alpha} \bigvee_{\jmath<\beta_{\imath}} x_{\imath,\jmath} = \bigvee_{\jmath<\sum_{\imath<\alpha}\beta_{\imath}} x_{\varphi\jmath}$	bijective $\varphi : \sum_{\imath<\alpha}\beta_{\imath} \to \coprod_{\imath<\alpha}\beta_{\imath}$

Interaction with Non-Determinism

ND-$\mathtt{L}$	$\bigvee_{\imath<\alpha} \mathrm{L}_{\ell}\left(v.\ x_{v,\imath}\right) = \mathrm{L}_{\ell}\left(v.\ \bigvee_{\imath<\alpha} x_{v,\imath}\right)$	
ND-$\mathtt{U}$	$\bigvee_{\imath<\alpha} \mathrm{U}_{\ell,v}x_{\imath} = \mathrm{U}_{\ell,v} \bigvee_{\imath<\alpha} x_{\imath}$	
ND-$\mathtt{Y}$	$\bigvee_{\imath<\alpha} \mathrm{Y}x_{\imath} = \mathrm{Y} \bigvee_{\imath<\alpha} x_{\imath}$	

Fig. 2. The axiomatization of the algebraic theory

Example 2. Consider the program context $\Xi\left[-\right] = \left[-\right] \parallel \mathsf{a?}$, i.e. executing each program in parallel to a thread dereferencing the location a. Then there is no execution of $\Xi\left[\mathsf{a} := \mathsf{c?}\right]$ that starts with the store $\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right)$ and returns $\langle\langle\rangle, 0\rangle$, but there is such a execution of $\Xi\left[\mathsf{a} := \mathsf{b?}\,;\mathsf{a} := \mathsf{c?}\right]$:

$$\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right), \mathsf{a} := \mathsf{b?}\,; \mathsf{a} := \mathsf{c?} \parallel \mathsf{a?} \ \rightsquigarrow^{*}\ \left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\0&0&1\end{smallmatrix}\right), \mathsf{a} := \mathsf{c?} \parallel \mathsf{a?} \ \rightsquigarrow$$
$$\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\0&0&1\end{smallmatrix}\right), \mathsf{a} := \mathsf{c?} \parallel 0 \ \rightsquigarrow^{*}\ \left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right), \langle\rangle \parallel 0 \ \rightsquigarrow\ \left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right), \langle\langle\rangle, 0\rangle$$

Therefore, the aforementioned denotational semantics defined using the theory of global-state cannot be extended with a denotation for $(\parallel)$ while preserving adequacy. More sophistication is needed in the concurrent setting: the denotations, even of sequential programs, must account for environment effects.

We thus extend the global-state theory with a single unary operator:

Yield. Suppose t is a term. Then $\mathrm{Y}t$ is a term. Its intended meaning is to let the environment read and write and then continue with the computation t.

We also need to account for the non-determinism inherent in parallel executions. We do so by extending the resulting theory with operators for finite non-determinism with their equational theory, and further standard equations axiomatizing commutative interaction with the other operators [14]:

Choice. For every $\alpha \in \mathbb{N}$ there is a respective choice operator. Suppose $\left(t_{\imath}\right)_{\imath<\alpha}$ is a sequence of terms $t_0, \ldots, t_{\alpha-1}$. Then $\bigvee_{\alpha}\left(t_{\imath}\right)_{\imath<\alpha}$ is a term. Its intended meaning is to choose $\imath < \alpha$ non-deterministically and continue with the computation $t_\imath$. We write $\bigvee_{\imath<\alpha} t_\imath$ instead of $\bigvee_\alpha (t_\imath)_{\imath<\alpha}$; and when $\alpha = 2$ we use infix notation, i.e. instead of $\bigvee_{\imath<2} t_\imath$ we may write $t_0 \vee t_1$.

The axioms of the resulting theory of resumptions $\textsc{Res}$ [1, 14, 30] are listed in Figure 2. The novelty is not in the equational theory, but rather in the way we use it to define denotations. We compare to related work in §7.

2.3 Denotations

In §5 we define denotations of programs, but for the sake of this discussion we simplify, defining the denotation $[\![M]\!]$ of a program M to be an equivalence class $|t|$ of a particular term t in $\textsc{Res}$. Our actual denotations defined in §5 will use the concrete representation of the monad (developed in §4.3), similar to how state transformers represent equivalence classes of terms of global-state.

Dereference & Assignment We use the monadic bind $\gg\!=$ (defined in §4.3), which we can think of as "chaining operators"; and *possible yields* $\mathrm{Y}^?t := t \vee \mathrm{Y}t$:

$$[\![M\mathbf{?}]\!] := [\![M]\!] \gg\!= \lambda\ell.\ \left|\mathrm{L}_\ell\left(v.\ \mathrm{Y}^?v\right)\right|$$

$$[\![M \mathbin{\mathbf{:=}} N]\!] := [\![M]\!] \gg\!= \lambda\ell.\ [\![N]\!] \gg\!= \lambda v.\ \left|\mathrm{U}_{\ell,v}\mathrm{Y}^?\langle\rangle\right|$$

The main idea is to intersperse possible yields to block the use of global-state equations such as in (1) and to allow computations to interleave. Although equation (1) still holds in $\textsc{Res}$, it does not imply the program equivalence (2) because the programs in (2) do not map to the algebraic terms in (1). Rather:

Example 3. The denotations of the programs in Example 1 are:

$$[\![\mathsf{a} \mathbin{\mathbf{:=}} \mathsf{c}\mathbf{?}]\!] = \left|\mathrm{L}_\mathsf{c}\left(v.\ \mathrm{Y}^?v\right)\right| \gg\!= \lambda v.\ \left|\mathrm{U}_{\mathsf{a},v}\mathrm{Y}^?\ \langle\rangle\right| = \left|\mathrm{L}_\mathsf{c}\left(v.\ \mathrm{Y}^?\mathrm{U}_{\mathsf{a},v}\mathrm{Y}^?\ \langle\rangle\right)\right| \tag{3}$$

$$[\![\mathsf{a} \mathbin{\mathbf{:=}} \mathsf{b}\mathbf{?}\ \mathbf{;}\ \mathsf{a} \mathbin{\mathbf{:=}} \mathsf{c}\mathbf{?}]\!] = \left|\mathrm{L}_\mathsf{b}\left(w.\ \mathrm{Y}^?\mathrm{U}_{\mathsf{a},w}\mathrm{Y}^?\mathrm{L}_\mathsf{c}\left(v.\ \mathrm{Y}^?\mathrm{U}_{\mathsf{a},v}\mathrm{Y}^?\ \langle\rangle\right)\right)\right| \tag{4}$$

So the denotation (3) looks c up finding a value v, then possibly yields, then updates a to v, then possibly yields, and finally returns the empty tuple. The concrete representation (Theorem 1) immediately proves that (3) and (4) are not equal, in contrast to the situation in Example 1.

Parallel Execution Computations interleave using the yield operator. Interleaving execution of programs suggests, for example, the following calculation:

$$\begin{aligned}
[\![\ell\mathbf{?} \parallel \ell \mathbin{\mathbf{:=}} 0]\!] &= \left|\mathrm{L}_\ell\left(v.\ \mathrm{Y}^?\ [\![v \parallel \ell \mathbin{\mathbf{:=}} 0]\!]\right) \vee \mathrm{U}_{\ell,0}\mathrm{Y}^?\ [\![\ell\mathbf{?} \parallel \langle\rangle]\!]\right| \\
&= \left|\mathrm{L}_\ell\left(v.\ \mathrm{Y}^?\mathrm{U}_{\ell,0}\mathrm{Y}^?\ [\![v \parallel \langle\rangle]\!]\right) \vee \mathrm{U}_{\ell,0}\mathrm{Y}^?\mathrm{L}_\ell\left(v.\ \mathrm{Y}^?\ [\![v \parallel \langle\rangle]\!]\right)\right| \\
&= \left|\mathrm{L}_\ell\left(v.\ \mathrm{Y}^?\mathrm{U}_{\ell,0}\mathrm{Y}^?\ \langle v, \langle\rangle\rangle\right) \vee \mathrm{U}_{\ell,0}\mathrm{Y}^?\mathrm{L}_\ell\left(v.\ \mathrm{Y}^?\ \langle v, \langle\rangle\rangle\right)\right|
\end{aligned}$$

The problem with the above is that is lacks compositionality: $[\![\ell\mathbf{?} \parallel \ell \mathbin{\mathbf{:=}} 0]\!]$ should be defined in terms of $[\![\ell\mathbf{?}]\!]$ and $[\![\ell \mathbin{\mathbf{:=}} 0]\!]$, without referring to the underlying programs. In §4.6 we define a function $(|||)$ such that $[\![M \parallel N]\!] = [\![M]\!] \mathbin{|||} [\![N]\!]$. This definition relies on the concrete representation from §4.3.

2.4 Program Transformations

Our main result is directional adequacy (Theorem 5). Under this simplified view it can be stated, in terms of the partial-order on our denotations generated by $|t| \leqslant |t \vee s|$, as follows: if $[\![M]\!] \leqslant [\![N]\!]$ then the transformation $N \twoheadrightarrow M$ is valid in the concurrent setting. The following example illustrates how directional-adequacy can be used to validate program transformations of interest, of the relatively few that are valid in the strong memory-model we consider here.

Example 4. We validate (write ; read) (see also Example 9):

$$[\![\ell \mathbin{\pmb{:=}} v \mathbin{\pmb{;}} \ell\pmb{?}]\!] = \left|\mathrm{U}_{\ell,v}\mathrm{Y}^{?}\mathrm{L}_{\ell}\left(w.\ \mathrm{Y}^{?}w\right)\right| \geqslant \left|\mathrm{U}_{\ell,v}\mathrm{L}_{\ell}\left(w.\ \mathrm{Y}^{?}w\right)\right| = \left|\mathrm{U}_{\ell,v}\mathrm{Y}^{?}v\right| = [\![\ell \mathbin{\pmb{:=}} v \mathbin{\pmb{;}} v]\!]$$

We can similarly validate the other memory access eliminations from §1. By using $\mathrm{Y}^{?}$ rather than Y in the denotations, the cases where the environment is known to not interleave are taken into accounted explicitly. The relevant global-state equation can then be exploited to eliminate the redundant memory access.

2.5 Caveats and Limitations

Our goal in this work is to fit concurrency semantics on equal footing with other semantic models of computational effects. As a consequence, the proposed model has its fair share of fine print, which we bring to the front:

Memory Model. We study a very strong memory model: sequential consistency. Modern architectures adhere to much weaker memory-models, where further program transformations are valid.

Concurrency Model. Our semantics involves a simple form of concurrency in which threads interleave their computation without restriction, acting on a shared memory store. This is in contrast to a well-established line of work in which models include a causal partial-order in which incomparable events denote "truly" parallel execution [31]. These causal models are showing promise in modelling sophisticated (i.e. weak) shared-state models [7, 16, 17, 25]. We hope further work would fit these causal models into a relatable semantic footing that easily accommodates higher-order structure.

Features. Our analysis lacks many valuable features that appear in related work, such as recursion [1], higher-order state [4], probabilities [41], and infinitely many locations/values. This simplification is intended: we took to reductionism, finding a minimal model still accounting for core features of shared state. The benefit of the algebraic approach is that this model can be modularly combined with other features, hopefully using standard technology such as sum-and-tensor [14], domain-enrichment [27], and functor categories [21,32,33,38]. For example, to support recursion our model may be integrated with one of the known powerdomain theory-combinators [42]. This requires making a semantic-design choice that is orthogonal to shared-state concurrency, each with different trade-offs. We avoid making such choices.

Semantic Precision. The equational theory and denotational semantics based on it leave much room for improvement in terms of precision and abstraction. For example, our denotational model does not support the *introduction* of irrelevant reads, i.e. it does not justify the valid transformation $M \twoheadrightarrow \ell\mathbf{?}\,\mathbf{;}\,M$. Indeed, taking $M = \langle\rangle$, we have $[\![\ell\mathbf{?}\,\mathbf{;}\,\langle\rangle]\!] = \left|\mathrm{L}_\ell\left(v.\,\mathrm{Y}^?\,\langle\rangle\right)\right| = \left|\mathrm{Y}^?\,\langle\rangle\right| \nleq |\langle\rangle| = [\![\langle\rangle]\!]$. The problem stems from a "counting" issue: even though the value being looked-up in ℓ is discarded, the additional possible-yield remains. We hope further work could address this semantic inaccuracy.

Full Abstraction. Brookes's seminal work [1,6] defined denotational semantics for concurrency that is fully-abstract, meaning that the converse of adequacy holds: programs that are replaceable in every context have equal denotations. Our semantics is far from being fully-abstract: there is a first-order valid equivalence, $M \cong \ell\mathbf{?}\,\mathbf{;}\,M$, that our semantics does not support. Moreover, we do not include atomic block executions in our language as Brookes did, which was crucial for the proof of full-abstraction. However, even if our model was precise enough to capture the first-order equivalences, and even if we were to include atomic block executions, we still would not expect to obtain full-abstraction, since this result is infamously elusive for higher-order languages (see Abramsky's recent overview on the full-abstraction problem of PCF [2]).

3 Equational Theory

At the foundation of our approach is the equational theory of resumptions Res [1, 14,30] presented in §2, consisting of operators and equational axioms over them. We succinctly fill-in the formal details below, followed by related definitions.

The signature of Res consists of the following parameterized operators. The notation $O : A\,\langle P\rangle$ means that the arity of the operator O is the set A and it is parameterized over the set P:

$\mathrm{L} : \mathbb{V}\,\langle\mathbb{L}\rangle$	lookup	$\mathrm{Y} : \mathbb{1}\,\langle\mathbb{1}\rangle$	yield
$\mathrm{U} : \mathbb{1}\,\langle\mathbb{L}\times\mathbb{V}\rangle$	update	$\bigvee_\alpha : \alpha\,\langle\mathbb{1}\rangle$	non-deterministic choice for every $\alpha \in \mathbb{N}$

From now on, whenever we refer to an *operator*, we mean an operator of Res. We denote the set of terms freely generated by the signature over X by $\mathrm{Term}X$.

Figure 2 lists the axioms of Res, classified as follows: an axiomatization of the equational theory of global-state [34]; the standard axiomatization of non-determinism; and an axiomatization of the commutative interaction of non-determinism with the other operators [13] via the tensor [14].

A *Res-algebra* $\mathcal{A}$ consists of a carrier set $\underline{\mathcal{A}}$ together with interpretations $\tilde{O}^{\mathcal{A}} : \underline{\mathcal{A}}^A \times P \to \underline{\mathcal{A}}$ for each operator $O : A\,\langle P\rangle$. We elide the superscript if it is clear from context. For a set X, a *Res-algebra on* X consists of a Res-algebra $\mathcal{A}$ and a function $\mathrm{env} : X \to \underline{\mathcal{A}}$; which extends to $\mathrm{eval} : \mathrm{Term}X \to \underline{\mathcal{A}}$ homomorphically along the inclusion $X \hookrightarrow \mathrm{Term}X$. A *Res-model on* X is a Res-algebra on X that satisfies each axiom of Res, i.e. the same element of $\underline{\mathcal{A}}$ is obtained by applying eval to either side of the axiom.

In the following, we abbreviate using $\vec{\mathrm{L}}\left(\sigma.\, t_{\sigma}\right) \coloneqq \mathrm{L}_{\mathtt{l}_1}\left(v_1.\ \ldots \mathrm{L}_{\mathtt{l}_{\bar{\mathtt{n}}}}\left(v_{\bar{\mathtt{n}}}.\, t_{\lambda \mathtt{l}_i.\, v_i}\right)\right)$ and $\vec{\mathrm{U}}_{\sigma} t \coloneqq \mathrm{U}_{\mathtt{l}_1, \sigma_{\mathtt{l}_1}} \ldots \mathrm{U}_{\mathtt{l}_{\bar{\mathtt{n}}}, \sigma_{\mathtt{l}_{\bar{\mathtt{n}}}}} t$, in addition to $\mathrm{Y}^{?} t \coloneqq t \vee \mathrm{Y} t$ that we saw in §2.3. For example, $\vec{\mathrm{L}}\left(\sigma.\, \vec{\mathrm{U}}_{\left(\begin{smallmatrix}\mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & \sigma_{\mathsf{b}}\end{smallmatrix}\right)} \mathrm{Y}^{?} t_{\sigma_{\mathsf{c}}}\right) = \mathrm{L}_{\mathsf{a}}\left(v.\, \mathrm{L}_{\mathsf{b}}\left(w.\, \mathrm{L}_{\mathsf{c}}\left(u.\, \mathrm{U}_{\mathsf{a},1} \mathrm{U}_{\mathsf{b},0} \mathrm{U}_{\mathsf{c},w}\left(t_u \vee \mathrm{Y} t_u\right)\right)\right)\right)$. We use similar shorthands with interpretations of operators as well.

4 A Monad for Shared-State

The next part in our denotational semantics is a monad whose elements represent equivalence classes of Res. The monad can be obtained via a universal construction [28] (by quotienting the terms by the equational theory), but a concrete representation is crucial to reason about it; for example, to show that two denotations are different.

4.1 Difficulty of Term Normalization

To motivate the definitions building up to this concrete representation, we first find a representative for each equivalence class in $\mathrm{Term} X / \textsc{Res}$, by taking an arbitrary $t \in \mathrm{Term} X$ and transforming it via equations in Res to a particular form – a normal form – such that there is only one term of this form equal to t.

Consider an algebraic term $t \in \mathrm{Term} X$. Using `LU`-noop once for each location, a sequence of `LU`-comm, and ND-return, we find that $t = \vec{\mathrm{L}}\left(\sigma.\, \bigvee_{\imath < 1} \vec{\mathrm{U}}_{\sigma} t\right)$. Note:

$$\vec{\mathrm{U}}_{\sigma} \mathrm{L}_{\ell}\left(v.\, s_v\right) \overset{\textsc{Res}}{=} \vec{\mathrm{U}}_{\sigma} s_{\sigma_{\ell}} \qquad \vec{\mathrm{U}}_{\sigma} \mathrm{U}_{\ell, v} s \overset{\textsc{Res}}{=} \vec{\mathrm{U}}_{\sigma[\ell \mapsto v]} s \qquad \vec{\mathrm{U}}_{\sigma} \bigvee_{\imath < \alpha} s_{\imath} \overset{\textsc{Res}}{=} \bigvee_{\imath < \alpha} \vec{\mathrm{U}}_{\sigma} s_{\imath}$$

By applying these equalities left-to-right as long as possible, and applying ND-join and ND-epi to rearrange the sums, we find that t is equal to a term of the form $\vec{\mathrm{L}}\left(\sigma.\, \bigvee_{\imath < \alpha_{\sigma}} \vec{\mathrm{U}}_{\rho_{\imath, \sigma}} s_{\imath, \sigma}\right)$, where $s_{\imath, \sigma}$ is either in X or is of the form $\mathrm{Y} s'_{\imath, \sigma}$.

For every σ, we can rearrange the sum according to common prefixes, thus we find that t is equal to a term of the form: $\vec{\mathrm{L}}\left(\sigma.\, \bigvee_{\rho \in \mathbb{S}} \vec{\mathrm{U}}_{\rho} \bigvee_{\jmath < \alpha_{\rho, \sigma}} s_{\jmath, \rho, \sigma}\right)$ where $s_{\imath, \rho, \sigma}$ is either in X or is of the form $\mathrm{Y} s'_{\imath, \rho, \sigma}$ (we can take $\alpha_{\rho, \sigma} = 0$ when the prefix $\vec{\mathrm{U}}_{\rho}$ did not appear in t). For every ρ, we can rearrange to obtain the form:

$$\vec{\mathrm{L}}\left(\sigma.\, \bigvee_{\rho \in \mathbb{S}} \vec{\mathrm{U}}_{\rho}\left(\mathrm{Y} r_{\rho, \sigma} \vee \bigvee_{\jmath < \beta_{\rho, \sigma}} x_{\jmath, \rho, \sigma}\right)\right) \tag{5}$$

This is not yet a normal form, which to obtain would require recursively applying this procedure to $r_{\rho, \sigma}$ and propagating empty choice operators outward. Were we to continue in this way to find a normal form, we would still need to prove uniqueness and completeness. One standard way to achieve this is to show that this procedure equates the sides of every axiom and respects the deduction rules of equational logic. This requires a careful proof-theoretic analysis of this normalization procedure. Instead, we take a model-theoretic approach, akin to normalization-by-evaluation, constructing for every set a concrete representation of the free Res-model over it. This representation is based on finite sets of traces.

4.2 Traces

Brookes [6] defined a trace to be a non-empty sequence of transitions, where a *transition* is a pairs of stores; e.g. $\langle\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right),\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\0&0&1\end{smallmatrix}\right)\rangle\;\langle\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\0&1&1\end{smallmatrix}\right),\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right)\rangle$. Brookes used traces to define a denotational semantics for an imperative concurrent programming language. In Brookes's semantics, traces denote interrupted executions, where each transition corresponds to an uninterrupted sequence of computation steps that starts with the first store and end with the second store. The breaks between transitions are where the computation yields to the environment.

The concept was adapted by many, including Benton et al. [4], to define denotational semantics for a functional language, where they have added an additional value at the end of the sequence to refer to the value the computation returns. A *trace* in this paper will refer to this concept: a non-empty sequence of transitions followed by an additional return value. If we wish to specify X as the set of the return values, we will call it an X*-trace*. For example, if $x \in X$ then $\langle\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right),\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\0&0&1\end{smallmatrix}\right)\rangle\;\langle\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\0&1&1\end{smallmatrix}\right),\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right)\rangle\; x$ is an X-trace.

For sets Y, Z, we denote by Y^* the set of sequences over Y, by Y^+ the set of non-empty sequences over Y, and $Y \cdot Z \coloneqq \{yz \mid y \in Y, z \in Z\}$. That is, $(\cdot)$ is just notation for $(\times)$ which suggests that the elements of the set are written in sequence, eliding the tuple notation. Thus $(\mathbb{S} \times \mathbb{S})^+ \cdot X$ is the set of X-traces.

Following our discussion in §4.1, our representation will use finite sets of traces instead of the algebraic syntax. In particular, the form we have found in (5) suggests the recursive definition:

$$\operatorname{rep} t \coloneqq \{\langle\sigma,\rho\rangle\,\tau \mid \sigma,\rho \in \mathbb{S}, \tau \in \operatorname{rep} r_{\rho,\sigma}\} \cup \{\langle\sigma,\rho\rangle\, x_{\jmath,\rho,\sigma} \mid \sigma,\rho \in \mathbb{S}, \jmath < \beta_{\rho,\sigma}\} \tag{6}$$

The model-theoretic approach we use below obviates the need for the syntactic manipulation that leads to the form in (5) as part of finding the representation. In the model definition, eval will play the role of rep.

4.3 Model Definition

We represent elements of $\mathrm{Term}X/\textsc{Res}$ by $\underline{\mathcal{T}}X \coloneqq \mathcal{P}_{\mathrm{fin}}\big((\mathbb{S}\times\mathbb{S})^+ \cdot X\big)$, i.e. finite sets of X-traces. We equip $\underline{\mathcal{T}}X$ with a $\textsc{Res}$-algebra structure $\mathcal{F}X$:

$$\begin{aligned}\tilde{\mathrm{L}}_\ell\,(v.\,P_v) &\coloneqq \{\langle\sigma,\rho\rangle\,\tau \mid \langle\sigma,\rho\rangle\,\tau \in P_{\sigma_\ell}\} & \tilde{\bigvee_{\imath<\alpha}} P_\imath &\coloneqq \bigcup_{\imath<\alpha} P_\imath \\ \tilde{\mathrm{U}}_{\ell,v} P &\coloneqq \{\langle\sigma,\rho\rangle\,\tau \mid \langle\sigma\,[\ell \mapsto v],\rho\rangle\,\tau \in P\} & \tilde{\mathrm{Y}} P &\coloneqq \{\langle\sigma,\sigma\rangle\,\tau \mid \sigma \in \mathbb{S}, \tau \in P\}\end{aligned}$$

We further equip it with $\operatorname{env} x \coloneqq \{\langle\sigma,\sigma\rangle\, x \mid \sigma \in \mathbb{S}\}$ to make it a $\textsc{Res}$-algebra over X. We denote $\operatorname{env} x$ by $\operatorname{return} x$, or $\tilde{x}$ for shorthand. This $\textsc{Res}$-algebra is in fact a $\textsc{Res}$-model over X by virtue of satisfying the axioms of $\textsc{Res}$:

Example 5. We verify that $\langle\mathcal{F}X, \mathrm{return}\rangle$ satisfies the axiom `LU`-noop:

$$\begin{aligned}\operatorname{eval}\left(\mathrm{L}_\ell\,(v.\,\mathrm{U}_{\ell,v} x)\right) &= \tilde{\mathrm{L}}_\ell\left(v.\,\tilde{\mathrm{U}}_{\ell,v}\tilde{x}\right) = \tilde{\mathrm{L}}_\ell\,(v.\;\{\langle\sigma,\sigma\,[\ell \mapsto v]\rangle\, x \mid \sigma \in \mathbb{S}\}) \\ &= \{\langle\sigma,\sigma\,[\ell \mapsto \sigma_\ell]\rangle\, x \mid \sigma \in \mathbb{S}\} = \{\langle\sigma,\sigma\rangle\, x \mid \sigma \in \mathbb{S}\} = \tilde{x} = \operatorname{eval} x\end{aligned}$$

4.4 Correspondence to Non-Deterministic Global-State

The theory of non-deterministic global-state (the fragment of Res excluding Y) admits a concrete representation using non-deterministic state transformers $\mathbb{S} \to \mathcal{P}_{\mathrm{fin}}(\mathbb{S}X)$ [14]. This representation corresponds to the one we have defined in an interesting way. Namely, there is a bijection between $\underline{\mathcal{T}}X$ and the set of functions mapping stores to finite sets of X-traces-with-the-first-store-removed:

$$\lambda P \in \underline{\mathcal{T}}X.\ \lambda\sigma \in \mathbb{S}.\ \left\{\rho\tau \in \mathbb{S}\cdot(\mathbb{S}\times\mathbb{S})^{*}\cdot X \;\middle|\; \langle\sigma,\rho\rangle\,\tau \in P\right\}$$
$$\lambda\psi \in \mathbb{S} \to \mathcal{P}_{\mathrm{fin}}\left(\mathbb{S}\cdot(\mathbb{S}\times\mathbb{S})^{*}\cdot X\right).\ \bigcup_{\sigma\in\mathbb{S}}\left\{\langle\sigma,\rho\rangle\,\tau \in (\mathbb{S}\times\mathbb{S})^{+}\cdot X \;\middle|\; \rho\tau \in \psi\sigma\right\}$$

Implicitly identifying the two, the model from §4.3 can be defined using formulas that look exactly like the non-deterministic global-state ones:

$$\tilde{\mathrm{L}}_{\ell}\,(v.\ P_v) \coloneqq \lambda\sigma.\ P_{\sigma_\ell}\sigma \qquad \tilde{\bigvee_{\imath<\alpha}} P_\imath \coloneqq \lambda\sigma.\ \bigcup_{\imath<\alpha} P_\imath\sigma$$
$$\tilde{\mathrm{U}}_{\ell,v}P \coloneqq \lambda\sigma.\ P\,(\sigma\,[\ell \mapsto v]) \qquad \tilde{x} \coloneqq \lambda\sigma.\ \{\sigma x\}$$

However, these are not the same formulas – they are defined for different elements (sets of traces as opposed to non-deterministic state transformers).

Using this identification for the yield operator, we obtained the definition $\tilde{\mathrm{Y}}P \coloneqq \lambda\sigma.\ \{\sigma\tau \mid \tau \in P\}$, which we understand as "the thread does not modify the state, then allows the environment to intervene, and then continues as before."

4.5 Representation Theorem

The model $\langle \mathcal{F}X, \mathrm{return}\rangle$ defined in §4.3 *represents* $\mathrm{Term}X/\mathrm{Res}$ because – according to the representation theorem – this is a *free* Res-model on X, and therefore equivalent to the model of equivalence classes we used in §2 or the model of syntactic normal forms to which we have alluded in §4.1.

To prove that the model is free we first equip the family of sets $\underline{\mathcal{T}}$ with a monad structure. For every Res-model $\mathcal{A}$ and function $f : X \to \underline{\mathcal{A}}$, define $- \gg= f : \underline{\mathcal{T}}X \to \underline{\mathcal{A}}$, the homomorphic extension of f along return, recursively; where $R_P^{\langle\sigma,\rho\rangle} \coloneqq \left\{\tau \in (\mathbb{S}\times\mathbb{S})^{+}\cdot X \;\middle|\; \langle\sigma,\rho\rangle\,\tau \in P\right\}$ and $X_{P,f}^{\langle\sigma,\rho\rangle} \coloneqq \tilde{\bigvee}^{\mathcal{A}}_{\langle\sigma,\rho\rangle x \in P} fx$:

$$\varnothing \gg= f \coloneqq \tilde{\bigvee_{0}}^{\mathcal{A}} \varnothing \qquad P \gg= f \coloneqq \vec{\tilde{L}}^{\mathcal{A}}\left(\sigma.\ \tilde{\bigvee_{\rho\in\mathbb{S}}}^{\mathcal{A}} \vec{\tilde{\mathrm{U}}}^{\mathcal{A}}_{\rho}\left(\tilde{\mathrm{Y}}^{\mathcal{A}}\left(R_P^{\langle\sigma,\rho\rangle} \gg= f\right)\tilde{\vee}^{\mathcal{A}}\, X_{P,f}^{\langle\sigma,\rho\rangle}\right)\right)$$

A simpler definition is available when there exists a set Y such that $\mathcal{A} = \mathcal{F}Y$:

$$\varnothing \gg= f \coloneqq \varnothing \qquad P \gg= f \coloneqq \{\alpha\,\langle\sigma,\varsigma\rangle\,\tau \mid \exists\rho.\ \alpha\,\langle\sigma,\rho\rangle\,x \in P \wedge \langle\rho,\varsigma\rangle\,\tau \in fx\}$$

The recursion is well-founded since $R_P^{\langle\sigma,\rho\rangle}$ is smaller than P when measured by the length of the longest trace in the set.

Thus we have our monad structure $\mathcal{T} \coloneqq \langle\underline{\mathcal{T}}, \mathrm{return}, \gg=\rangle$. We show it is induced by the aforementioned family of free Res-models:

Theorem 1 (Representation for shared-state). *The pair* $\langle \mathcal{F}X, \mathrm{return}\rangle$ *is a free* Res*-model on* X*: for every* Res*-model* $\mathcal{A}$ *and* $f : X \to \underline{\mathcal{A}}$*, the function* $- \gg\!= f : \underline{\mathcal{T}}X \to \underline{\mathcal{A}}$ *is the unique homomorphism* g *satisfying* $f = g \circ \mathrm{return}$.

As a direct consequence:

Corollary 1 (Model is sound and complete). *Terms over* X *are equal in* Res *iff they have the same representation in* $\langle \mathcal{F}X, \mathrm{return}\rangle$.

4.6 Synchronization

To define the denotational semantics of $(\parallel)$ in §5, we will define a corresponding function $(|||)$ on elements of the monad. To this end we first define the *trace synchronization*, an inductively defined relation $\tau_1 \parallel \tau_2 \Longrightarrow \tau$ presented below, that relates $\tau_i \in (\mathbb{S} \times \mathbb{S})^+ \cdot X_i$ and $\tau \in (\mathbb{S} \times \mathbb{S})^+ \cdot (X_1 \times X_2)$, representing the fact that τ_1 and τ_2 can synchronize to form τ:

$$\boxed{\tau \parallel \pi \Longrightarrow \omega} \qquad \frac{}{\langle \sigma, \rho\rangle\, x \parallel \langle \rho, \varsigma\rangle\, \beta y \Longrightarrow \langle \sigma, \varsigma\rangle\, \beta\, \langle x, y\rangle}\ (\textsc{Var-Left})$$

$$\frac{\tau \parallel \pi \Longrightarrow \omega}{\langle \sigma, \rho\rangle\, \tau \parallel \pi \Longrightarrow \langle \sigma, \rho\rangle\, \omega}\ (\textsc{Brk-Left}) \qquad \frac{\tau \parallel \pi \Longrightarrow \langle \rho, \varsigma\rangle\, \omega}{\langle \sigma, \rho\rangle\, \tau \parallel \pi \Longrightarrow \langle \sigma, \varsigma\rangle\, \omega}\ (\textsc{Seq-Left})$$

Symmetrically: (Var-Right) (Brk-Right) (Seq-Right)

One way to understand these rules is to concentrate on the first transition on the left trace $\tau_1 = \langle \sigma, \rho\rangle\, \tau_1'$; the right-sided rules are treated symmetrically. If the first transition is also the last, i.e. $\tau_1' \in X$, then ρ must be the initial store when the execution continues (recall that only a break *between transitions* reflects a yield to the environment). This is why Var-Left combines the transitions as it does. The value in τ_3 is the pair of the values in τ_1 and τ_2, reflecting the operational semantics of $(\parallel)$ returning the pair of the results. If, on the other hand, the first transition is not the last, then we may combine the transition with the continuation of the computation (Seq-Left), or we may not (Brk-Left). The first option means the yield was used-up in this synchronization; while in the second option yield remains available to ambient synchronizations.

From this relation we derive the *semantic synchronization* function:

$$(|||) : \underline{\mathcal{T}}X \times \underline{\mathcal{T}}Y \to \underline{\mathcal{T}}(X \times Y) \qquad P \mathbin{|||} Q \coloneqq \left\{ \omega \;\middle|\; \exists \tau \in P, \pi \in Q.\ \tau \parallel \pi \Longrightarrow \omega \right\}$$

Example 6. For $\sigma, \rho \in \mathbb{S}$, we may synchronize $\langle \sigma, \rho\rangle \langle \rho, \sigma\rangle \langle\rangle$ and $\langle \rho, \rho\rangle\, 0$ so:

$$\frac{\dfrac{}{\langle \rho, \sigma\rangle \langle\rangle \parallel \langle \rho, \rho\rangle\, 0 \Longrightarrow \langle \rho, \sigma\rangle \langle \langle\rangle, 0\rangle}\ \textsc{Var-Right}}{\langle \sigma, \rho\rangle \langle \rho, \sigma\rangle \langle\rangle \parallel \langle \rho, \rho\rangle\, 0 \Longrightarrow \langle \sigma, \sigma\rangle \langle \langle\rangle, 0\rangle}\ \textsc{Seq-Left}$$

Therefore, if $\langle \sigma, \rho\rangle \langle \rho, \sigma\rangle \langle\rangle \in P$ and $\langle \rho, \rho\rangle\, 0 \in Q$, then $\langle \sigma, \sigma\rangle \langle\langle\rangle, 0\rangle \in P \mathbin{|||} Q$.

The use of Seq-Left was possible since the stores happen to match, resulting in a trace that does not allow the environment to interfere. By using Brk-Left we could find a different synchronization, one that does yield to the environment.

5 Denotational Semantics

With the monad in place, denotations of types and contexts are standard [30]:

$$\begin{aligned} \llbracket (A_1 \mathbin{*} \cdots \mathbin{*} A_n) \rrbracket &\coloneqq \llbracket A_1 \rrbracket \times \cdots \times \llbracket A_n \rrbracket & \llbracket A \mathbin{\texttt{->}} B \rrbracket &\coloneqq \llbracket A \rrbracket \to \mathcal{T}\llbracket B \rrbracket \\ \llbracket \{\iota_1 \textbf{ of } A_1 \mid \cdots \mid \iota_n \textbf{ of } A_n\} \rrbracket &\coloneqq \textstyle\bigcup_{i=1}^{n} \{\iota_i\} \times \llbracket A_i \rrbracket & \llbracket \Gamma \rrbracket &\coloneqq \textstyle\prod_{(\mathtt{x}:A)\in\Gamma} \llbracket A \rrbracket \end{aligned}$$

Define the extension of $\gamma \in \llbracket \Gamma \rrbracket$ to $\gamma[\mathtt{x} \mapsto y] \in \llbracket \Gamma, \mathtt{x} : A \rrbracket$ by $\gamma[\mathtt{x} \mapsto y]\,\mathtt{x} \coloneqq y$.

On the above we base two kinds of denotations for programs $\Gamma \vdash M : A$:

Computational. $\llbracket M \rrbracket^{\mathsf{c}} : \llbracket \Gamma \rrbracket \to \mathcal{T}\llbracket A \rrbracket$. When Γ is empty we may write $\llbracket M \rrbracket^{\mathsf{c}}$ instead of $\llbracket M \rrbracket^{\mathsf{c}} \langle\rangle$. We write $\llbracket M \rrbracket^{\mathsf{c}} \subseteq \llbracket N \rrbracket^{\mathsf{c}}$ for $\forall \gamma \in \llbracket \Gamma \rrbracket .\ \llbracket M \rrbracket^{\mathsf{c}} \gamma \subseteq \llbracket N \rrbracket^{\mathsf{c}} \gamma$.

Valuational. $\llbracket V \rrbracket^{\mathsf{v}} : \llbracket \Gamma \rrbracket \to \llbracket A \rrbracket$ defined solely for values, and satisfying $\llbracket V \rrbracket^{\mathsf{c}} \gamma = \mathrm{return}\left(\llbracket V \rrbracket^{\mathsf{v}} \gamma\right)$. When Γ is empty we may write $\llbracket V \rrbracket^{\mathsf{v}}$ instead of $\llbracket V \rrbracket^{\mathsf{v}} \langle\rangle$; and if furthermore A is a ground type, we may write V instead of $\llbracket V \rrbracket^{\mathsf{v}}$, noting that the restriction of $\llbracket - \rrbracket^{\mathsf{v}}$ to closed programs of ground type is a bijection.

Most denotations of programs are standard as well, such as:

$$\begin{aligned} \llbracket \langle\rangle \rrbracket^{\mathsf{v}} \gamma &\coloneqq \langle\rangle & \llbracket \lambda\mathtt{x}.\, M \rrbracket^{\mathsf{v}} \gamma &\coloneqq \lambda y.\ \llbracket M \rrbracket^{\mathsf{c}} \gamma[\mathtt{x} \mapsto y] \\ \llbracket \mathtt{x} \rrbracket^{\mathsf{v}} \gamma &\coloneqq \gamma\mathtt{x} & \llbracket N M \rrbracket^{\mathsf{c}} \gamma &\coloneqq \llbracket N \rrbracket^{\mathsf{c}} \gamma \gg= \lambda f.\ \llbracket M \rrbracket^{\mathsf{c}} \gamma \gg= f \end{aligned}$$

The denotations of the state effects allow the environment to intervene:

$$\begin{aligned} \llbracket M\mathbf{?} \rrbracket^{\mathsf{c}} \gamma &\coloneqq \llbracket M \rrbracket^{\mathsf{c}} \gamma \gg= \lambda \ell.\ \tilde{\mathrm{L}}_{\ell}\left(v.\ \tilde{\mathrm{Y}}^{?} \tilde{v}\right) \\ \llbracket M \mathbin{\mathbf{:=}} N \rrbracket^{\mathsf{c}} \gamma &\coloneqq \llbracket M \rrbracket^{\mathsf{c}} \gamma \gg= \lambda \ell.\ \llbracket N \rrbracket^{\mathsf{c}} \gamma \gg= \lambda v.\ \tilde{\mathrm{U}}_{\ell, v} \tilde{\mathrm{Y}}^{?} \tilde{\langle\rangle} \\ \llbracket M \parallel N \rrbracket^{\mathsf{c}} \gamma &\coloneqq \llbracket M \rrbracket^{\mathsf{c}} \gamma \mathbin{|||} \llbracket N \rrbracket^{\mathsf{c}} \gamma \end{aligned}$$

Example 7. With the definitions above, we can state the denotations from Example 3 precisely. For instance, (4) becomes:

$$\llbracket \mathsf{a} \mathbin{\mathbf{:=}} \mathsf{b}\mathbf{?}\,\mathbf{;}\, \mathsf{a} \mathbin{\mathbf{:=}} \mathsf{c}\mathbf{?} \rrbracket^{\mathsf{c}} = \tilde{\mathrm{L}}_{\mathsf{b}}\left(w.\ \tilde{\mathrm{Y}}^{?} \tilde{\mathrm{U}}_{\mathsf{a}, w} \tilde{\mathrm{Y}}^{?} \tilde{\mathrm{L}}_{\mathsf{c}}\left(v.\ \tilde{\mathrm{Y}}^{?} \tilde{\mathrm{U}}_{\mathsf{a}, v} \tilde{\mathrm{Y}}^{?} \tilde{\langle\rangle}\right)\right)$$

Example 8. We can explain the execution of $\mathsf{a} \mathbin{\mathbf{:=}} \mathsf{b}\mathbf{?}\,\mathbf{;}\, \mathsf{a} \mathbin{\mathbf{:=}} \mathsf{c}\mathbf{?} \parallel \mathsf{a}\mathbf{?}$ from Example 2 in denotational terms. First we find traces to synchronize:

$$\begin{aligned} \left\langle \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 0 & 0 & 1 \end{smallmatrix}\right) \right\rangle \left\langle \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 0 & 0 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & 1 \end{smallmatrix}\right) \right\rangle\ \langle\rangle &\in \llbracket \mathsf{a} \mathbin{\mathbf{:=}} \mathsf{b}\mathbf{?}\,\mathbf{;}\, \mathsf{a} \mathbin{\mathbf{:=}} \mathsf{c}\mathbf{?} \rrbracket^{\mathsf{c}} \\ \left\langle \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 0 & 0 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 0 & 0 & 1 \end{smallmatrix}\right) \right\rangle\ 0 &\in \llbracket \mathsf{a}\mathbf{?} \rrbracket^{\mathsf{c}} \end{aligned}$$

Following from the derivation in Example 6 with $\sigma = \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & 1 \end{smallmatrix}\right)$ and $\rho = \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 0 & 0 & 1 \end{smallmatrix}\right)$:

$$\left\langle \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & 1 \end{smallmatrix}\right) \right\rangle\ \langle 0, \langle\rangle\rangle \in \llbracket \mathsf{a} \mathbin{\mathbf{:=}} \mathsf{b}\mathbf{?}\,\mathbf{;}\, \mathsf{a} \mathbin{\mathbf{:=}} \mathsf{c}\mathbf{?} \parallel \mathsf{a}\mathbf{?} \rrbracket^{\mathsf{c}}$$

This trace corresponds to the (uninterrupted) execution presented in Example 2.

6 Metatheoretical Results

First we find that the single-transition traces in the denotation of a program account for the possible executions of that program:

Theorem 2 (Soundness). *If* $\sigma, M \rightsquigarrow\!\ast\, \rho, V$ *then* $\langle \sigma, \rho \rangle [\![V]\!]^{\mathsf{v}} \in [\![M]\!]^{\mathsf{c}}$*.*

For the proof, omitted for brevity, we instrument the operational-semantics with *actions*, elements of $\{\mathrm{U}_{\ell,v}, \mathrm{L}_{\ell}, \varepsilon\}$ signifying the effect caused by the step, to analyse the change to the denotation of the program as it runs.

Working our way up to the fundamental lemma, we define a unary logical relation: functions $\mathcal{V}\llparenthesis - \rrparenthesis$ and $\mathcal{E}\llparenthesis - \rrparenthesis$ from types to sets of closed programs by mutual recursion. Specifically, $\mathcal{V}\llparenthesis A \rrparenthesis$ is a set of closed values of type A, and $\mathcal{E}\llparenthesis A \rrparenthesis$ is a set of closed programs of type A. The definition of $\mathcal{V}\llparenthesis - \rrparenthesis$ is standard:

$$\begin{aligned}
\mathcal{V}\llparenthesis A \texttt{ -> } B \rrparenthesis &\coloneqq \{\lambda \mathtt{x}.\, M \mid \forall V \in \mathcal{V}\llparenthesis A \rrparenthesis.\ M[V/\mathtt{x}] \in \mathcal{E}\llparenthesis B \rrparenthesis\} \\
\mathcal{V}\llparenthesis \texttt{(}A_1 \,\texttt{*}\, \cdots \,\texttt{*}\, A_n\texttt{)} \rrparenthesis &\coloneqq \{\langle V_1, \ldots, V_n \rangle \mid \forall i.\ V_i \in \mathcal{V}\llparenthesis A_i \rrparenthesis\} \\
\mathcal{V}\llparenthesis \{\iota_1 \textbf{ of } A_1 \mid \cdots \mid \iota_n \textbf{ of } A_n\} \rrparenthesis &\coloneqq \textstyle\bigcup_i \{\iota_i V \mid V \in \mathcal{V}\llparenthesis A_i \rrparenthesis\}
\end{aligned}$$

The definition of $\mathcal{E}\llparenthesis - \rrparenthesis$ is also standard in that it ensures programs in $\mathcal{E}\llparenthesis A \rrparenthesis$ compute to values in $\mathcal{V}\llparenthesis A \rrparenthesis$, but bespoke in its requirement about *how* they compute. This requirement is based on the way traces specify interrupted executions, a notion we have discussed in §4.2 and now make precise. For a non-empty sequence of transitions $\alpha = \langle \sigma_1, \rho_1 \rangle \ldots \langle \sigma_m, \rho_m \rangle$ we write $M \xrightarrow{\alpha} N$ when there exist $M = M_1, M_2 \ldots M_m, M_{m+1} = N$ such that $\sigma_i, M_i \rightsquigarrow\!\ast\, \rho_i, M_{i+1}$ for all $i \in \{1, \ldots, m\}$. We write $M \xrightarrow{\alpha x} V$ when $M \xrightarrow{\alpha} V$ and $[\![V]\!]^{\mathsf{v}} = x$. We now define:

$$\mathcal{E}\llparenthesis A \rrparenthesis \coloneqq \left\{ M \in \cdot \vdash A \;\middle|\; \forall \tau \in [\![M]\!]^{\mathsf{c}}\, \exists V \in \mathcal{V}\llparenthesis A \rrparenthesis.\ M \xrightarrow{\tau} V \right\}$$

The last component needed is the function $\mathcal{G}\llparenthesis - \rrparenthesis$ from typing contexts to sets of program substitutions: $\mathcal{G}\llparenthesis \Gamma \rrparenthesis \coloneqq \{\Theta \mid \forall (\mathtt{x} : A) \in \Gamma.\ \Theta\mathtt{x} \in \mathcal{V}\llparenthesis A \rrparenthesis\}$. The *semantic typing judgment* $\Gamma \vDash M : A$ is then defined as: $\forall \Theta \in \mathcal{G}\llparenthesis \Gamma \rrparenthesis.\ \Theta M \in \mathcal{E}\llparenthesis A \rrparenthesis$.

Theorem 3 (Fundamental Lemma). *If* $\Gamma \vdash M : A$ *then* $\Gamma \vDash M : A$*.*

This brings us one step closer to proving the theorem of directional adequacy. One piece is still missing: since the theorem assumes set inclusion of denotations rather than equality, we will need a different form of compositionality of the denotations than the one that holds by definition.

To state this form of compositionality we first define the standard notion of a program with holes. A function $\Xi[-] : \Gamma \vdash A \to \Delta \vdash B$ is a *program context* (or *context* for short) if, in the language extended with a program $\bullet$ and additional axioms $\Gamma' \vdash \bullet : A$ for all $\Gamma' \geq \Gamma$, we have $\Delta \vdash \Xi[\bullet] : B$; and if $\Gamma \vdash M : A$, then $\Xi[M]$ is obtained from $\Xi[\bullet]$ by replacing every occurrence of $\bullet$ with M.

Theorem 4 (Compositionality). *Let* $\Xi[-] : \Gamma \vdash A \to \cdot \vdash G$ *be a context for ground* G*, and* $M, N \in \Gamma \vdash A$*. If* $[\![M]\!]^{\mathsf{c}} \subseteq [\![N]\!]^{\mathsf{c}}$ *then* $[\![\Xi[M]]\!]^{\mathsf{c}} \subseteq [\![\Xi[N]]\!]^{\mathsf{c}}$*.*

The condition that the context be closed and ground is necessary, so an attempt to prove directly by induction on the structure of the context fails. The proof, omitted for brevity, instead uses a binary logical relation approximating containment that identifies with it on ground types; the main ingredient being:

$$\mathcal{E}^{\circ}\llparenthesis A \rrparenthesis := \{\langle P, Q\rangle \in \mathcal{T}\llbracket A\rrbracket \times \mathcal{T}\llbracket A\rrbracket \mid \forall \alpha x \in P \exists \beta y \in Q.\, \alpha = \beta \wedge \langle x, y\rangle \in \mathcal{V}^{\circ}\llparenthesis A \rrparenthesis\}$$

With this compositionality in hand we are finally ready to prove the main result of this paper, that we will then use to justify program transformation. To state it we first spell out the standard definition of contextual refinement.

Suppose that $M, N \in \Gamma \vdash A$. We say that M *refines* N, and write $M \sqsubseteq N$, if $\sigma, \Xi[M] \rightsquigarrow{*}\, \rho, V$ implies $\sigma, \Xi[N] \rightsquigarrow{*}\, \rho, V$ whenever $\Xi[-] : \Gamma \vdash A \to \cdot \vdash G$ is a context for ground G. This justifies the transformation $N \twoheadrightarrow M$, since replacing N with M within a larger program introduces no additional behaviours.

Theorem 5 (Directional Adequacy). *If* $\llbracket M\rrbracket^{\mathsf{c}} \subseteq \llbracket N\rrbracket^{\mathsf{c}}$ *then* $M \sqsubseteq N$.

Proof. Let $\Xi[-] : \Gamma \vdash A \to \cdot \vdash G$ be a program context for some ground G, and assume $\sigma, \Xi[M] \rightsquigarrow{*}\, \rho, V$ for some V. By soundness, $\langle \sigma, \rho\rangle \llbracket V\rrbracket^{\mathsf{v}} \in \llbracket \Xi[M]\rrbracket^{\mathsf{c}}$. Using compositionality, by assumption $\langle \sigma, \rho\rangle \llbracket V\rrbracket^{\mathsf{v}} \in \llbracket \Xi[N]\rrbracket^{\mathsf{c}}$. By the fundamental lemma, $\Xi[N] \xrightarrow{\langle \sigma, \rho\rangle} W$ for some W such that $\llbracket W\rrbracket^{\mathsf{v}} = \llbracket V\rrbracket^{\mathsf{v}}$. They are of ground type, so $W = V$. Therefore, $\sigma, \Xi[N] \rightsquigarrow{*}\, \rho, V$.

6.1 Example Transformations

Thanks to directional adequacy, we can now justify various transformations and equivalences using rather mundane calculations, requiring no reasoning about the context in which these transformations are to take place.

Example 9. We make the reasoning from Example 4 precise.

Denote $\Gamma := \mathrm{x} : \mathbf{Loc}, \mathrm{y} : \mathbf{Val}$. We have $\Gamma \vdash \mathtt{x{:=}y;x?} : \mathbf{Val}$ and $\Gamma \vdash \mathtt{x{:=}y;y} : \mathbf{Val}$. Let $\gamma \in \llbracket \Gamma \rrbracket$, and denote $\ell := \gamma\mathrm{x}$ and $v := \gamma\mathrm{y}$. Calculating, we have:

$$\llbracket \mathtt{x := y \,;\, x?}\rrbracket^{\mathsf{c}} \gamma = \tilde{\mathrm{U}}_{\ell, v} \tilde{\mathrm{Y}}^{?} \tilde{\mathrm{L}}_{\ell}\left(w.\, \tilde{\mathrm{Y}}^{?} \tilde{w}\right) \supseteq \tilde{\mathrm{U}}_{\ell, v} \tilde{\mathrm{Y}}^{?} \tilde{v} = \llbracket \mathtt{x := y \,;\, y}\rrbracket^{\mathsf{c}} \gamma$$

By directional adequacy, $\mathtt{x := y \,;\, y} \sqsubseteq \mathtt{x := y \,;\, x?}$.

Example 10. We validate elimination of irrelevant reads, i.e. $M \sqsubseteq \mathtt{x? \,;}\, M$:

$$\llbracket \mathtt{x? \,;}\, M\rrbracket^{\mathsf{c}} \gamma = \llbracket (\lambda\_.\, M)\, \mathtt{x?}\rrbracket^{\mathsf{c}} \gamma = \llbracket \mathtt{x?}\rrbracket^{\mathsf{c}} \gamma \gg\!= \lambda v.\, \llbracket M\rrbracket^{\mathsf{c}} \gamma = \tilde{\mathrm{Y}}^{?}\left(\llbracket M\rrbracket^{\mathsf{c}} \gamma\right) \supseteq \llbracket M\rrbracket^{\mathsf{c}} \gamma$$

As mentioned in §2.5, the semantics does not validate *introduction* of irrelevant reads, i.e. we have $\llbracket \mathtt{x? \,;}\, M\rrbracket^{\mathsf{c}} \nsubseteq \llbracket M\rrbracket^{\mathsf{c}}$ even though $\mathtt{x? \,;}\, M \sqsubseteq M$.

Example 11. Thanks to our use of standard monad-based semantics, structural transformations and equivalences follow from structural reasoning, avoiding considerations relating to shared-state. For instance:

$$\begin{aligned}\llbracket \mathbf{if}\, \mathrm{y}\, \mathbf{then}\, \lambda \mathrm{x}.\, K_{\mathbf{true}}\, \mathbf{else}\, \lambda \mathrm{x}.\, K_{\mathbf{false}}\rrbracket^{\mathsf{c}} \gamma &= \left\llbracket \lambda \mathrm{x}.\, K_{\gamma \mathrm{y}}\right\rrbracket^{\mathsf{c}} \gamma \\ &= \mathrm{return}\, \lambda z.\, \left\llbracket K_{\gamma \mathrm{y}}\right\rrbracket^{\mathsf{c}} (\gamma[\mathrm{x} \mapsto z]) = \llbracket \lambda \mathrm{x}.\, \mathbf{if}\, \mathrm{y}\, \mathbf{then}\, K_{\mathbf{true}}\, \mathbf{else}\, K_{\mathbf{false}}\rrbracket^{\mathsf{c}} \gamma\end{aligned}$$

Therefore, $\mathbf{if}\, \mathrm{y}\, \mathbf{then}\, \lambda \mathrm{x}.\, K_{\mathbf{true}}\, \mathbf{else}\, \lambda \mathrm{x}.\, K_{\mathbf{false}} \cong \lambda \mathrm{x}.\, \mathbf{if}\, \mathrm{y}\, \mathbf{then}\, K_{\mathbf{true}}\, \mathbf{else}\, K_{\mathbf{false}}$.

Finally, adequacy can help validate expected transformations involving ($\parallel$):

Example 12. Defining $\operatorname{map}\psi P \coloneqq \{\alpha\,(\psi x) \mid \alpha x \in P\}$ we have:

$$\begin{aligned}
[\![\langle M, N\rangle]\!]^{\mathsf{c}}\,\gamma &\subseteq [\![M \parallel N]\!]^{\mathsf{c}}\,\gamma && \text{(Sequencing)}\\
[\![M \parallel V]\!]^{\mathsf{c}}\,\gamma &= \operatorname{map}\big(\lambda x.\ \big\langle x, [\![V]\!]^{\mathsf{v}}\big\rangle\big)\big([\![M]\!]^{\mathsf{c}}\,\gamma\big) && \text{(Neutrality)}\\
[\![M \parallel N]\!]^{\mathsf{c}}\,\gamma &= \operatorname{map}\left(\lambda\langle y, x\rangle.\ \langle x, y\rangle\right)\big([\![N \parallel M]\!]^{\mathsf{c}}\,\gamma\big) && \text{(Symm.)}\\
[\![(M \parallel N) \parallel K]\!]^{\mathsf{c}}\,\gamma &= \operatorname{map}\left(\lambda\langle x, \langle y, z\rangle\rangle.\ \langle\langle x, y\rangle, z\rangle\right)\big([\![M \parallel (N \parallel K)]\!]^{\mathsf{c}}\,\gamma\big) && \text{(Assoc.)}
\end{aligned}$$

Unlike the previous examples, proving the above involves careful reasoning at the level of the traces. We still gain the benefit of justifying equivalences and transformations of programs – even open ones – without resorting to analysis under arbitrary program contexts and substitutions:

$$\begin{aligned}
\langle M, N\rangle &\sqsubseteq M \parallel N && \text{(Sequencing)}\\
\langle M, V\rangle &\cong M \parallel V && \text{(Neutrality)}\\
M \parallel N &\cong \mathbf{match}\, N \parallel M\, \mathbf{with}\, \langle \mathtt{y}, \mathtt{x}\rangle \; \texttt{->} \; \langle \mathtt{x}, \mathtt{y}\rangle && \text{(Symm.)}\\
(M \parallel N) \parallel K &\cong \mathbf{match}\, M \parallel (N \parallel K)\, \mathbf{with}\, \langle\langle \mathtt{x}, \langle \mathtt{y}, \mathtt{z}\rangle\rangle\rangle \; \texttt{->} \; \langle \mathtt{x}, \mathtt{y}\rangle\,, \mathtt{z} && \text{(Assoc.)}
\end{aligned}$$

Coordinating the returned values make these somewhat awkward. More convenient but less informative forms are derivable, such as $M\,\mathbf{;}\,N\,\mathbf{;}\,K \sqsubseteq (M \parallel N)\,\mathbf{;}\,K$ (mentioned as a transformation in §1) which is a consequence of (Sequencing).

7 Conclusion, Related Work, and Future Work

We have defined a monad-based denotational semantics for a language for shared-state providing standard higher-order semantics supporting standard meta-theoretic development. This monad is a representation of the one induced by the equational theory of resumptions, which extends non-deterministic global-state with a delaying/yielding operator [14].

Abadi and Plotkin [1] design a modification for the theory of resumptions to define a denotational semantics for a concurrent imperative programming language with cooperative asynchronous threads. We have shown that the theory of resumptions can be used as-is to define denotational semantics for concurrency, albeit of a different kind. It is interesting to note that they interpret the unary operator analogously to our interpretation of $\mathrm{Y}^{?}$, rather than Y. By decomposing into a sum we were able to validate transformations that are not equivalences.

Benton et al. [4] also define a monad for higher-order shared-state, with additional features such as recursion and abstract locations, using Brookes's style of semantics. Contrasting, the monad we defined is presented algebraically, and has finite sets of traces, whereas Benton et al.'s denotations are infinite even for recursion-free programs. Although this finiteness makes our definition simpler, we saw in Example 10 that it leads to a resumption-counting issue, thus less abstract semantics. It would be interesting to analyse their semantic model from the algebraic perspective as it may lead to more abstract semantics.

Like in previous work, including those mentioned above, our semantics is based on the sets of traces, originally used by Brookes [6] to define denotational semantics for an imperative concurrent language. Brookes proved that this semantics is not only directionally adequate, but also fully abstract. The proof makes crucial use of atomic execution blocks which we have not included.

Birkedal et al. [5] provide an interesting related model, given by logical relations (step-indexed, Kripke, etc.) over syntactic terms as semantics. Their language is substantially more expressive including higher-order local store, and accounts for a type-and-effect system semantics. A more precise model could lead to a monadic account that reproduces these results less syntactically.

Also of note are process calculi and algebraic laws concerning the structure of programs. Hoare and van Staden [12] give such an account for concurrent programs, unifying previous work. Their laws are much more general, parameterizing over the notions of sequencing programs and running programs in parallel. It would be interesting to discover if and how our semantics is an instance of theirs. There is also a lot of work on semantics of "while" languages where all information flows through the state, which support more advanced features such as probabilistic choice [3,11,41]. Others approach the study of concurrency through game semantics, such as Jaber and Murawski's [15] study of the semantics of a higher-order call-by-value concurrent language. Trace semantics features in their study too, though their traces are quite different, being sequences of player/opponent actions that incrementally transform configurations.

In the future we plan to refine the type system into a type-and-effect system [18,20,22,29,39,40], by annotating the typing judgments with the allowed effects. The denotations then depend on the effect annotations, with each annotation having its own associated equational theory. This may allow additional transformations that are currently beyond this model's reach. For example, the converse of (Sequencing) under certain syntactic and static guarantees would enable compiler parallelism.

Atomic constructs that disallow interference from the environment are a common feature of concurrent languages. Adding such constructs may be a simple matter, since we have a dedicated operator, yield, for allowing interference. Nevertheless, in the spirit of reductionism, we leave this investigation to future work.

We would also like to see how well our approach extends to weak-memory models. In particular, we believe that the timestamp-based operational semantics of the release-acquire memory model [19, 24, 26, 37] is amenable to a similar treatment by using more sophisticated traces.

Acknowledgments We thank Andrés Goens for providing his perspective on a previous version of this paper, and the anonymous APLAS reviewers for their helpful feedback.

References

1. Abadi, M., Plotkin, G.D.: A model of cooperative threads. Log. Methods Comput. Sci. **6**(4) (2010). https://doi.org/10.2168/LMCS-6(4:2)2010, https://doi.org/10.2168/LMCS-6(4:2)2010
2. Abramsky, S.: Intensionality, Definability and Computation, pp. 121–142. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-06025-5_5, https://doi.org/10.1007/978-3-319-06025-5_5
3. Anderson, C.J., Foster, N., Guha, A., Jeannin, J., Kozen, D., Schlesinger, C., Walker, D.: Netkat: semantic foundations for networks. In: Jagannathan, S., Sewell, P. (eds.) The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014. pp. 113–126. ACM (2014). https://doi.org/10.1145/2535838.2535862, https://doi.org/10.1145/2535838.2535862
4. Benton, N., Hofmann, M., Nigam, V.: Effect-dependent transformations for concurrent programs. In: Cheney, J., Vidal, G. (eds.) Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016. pp. 188–201. ACM (2016). https://doi.org/10.1145/2967973.2968602, https://doi.org/10.1145/2967973.2968602
5. Birkedal, L., Sieczkowski, F., Thamsborg, J.: A Concurrent Logical Relation. In: Cégielski, P., Durand, A. (eds.) Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL. Leibniz International Proceedings in Informatics (LIPIcs), vol. 16, pp. 107–121. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2012). https://doi.org/10.4230/LIPIcs.CSL.2012.107, http://drops.dagstuhl.de/opus/volltexte/2012/3667
6. Brookes, S.D.: Full abstraction for a shared-variable parallel language. Inf. Comput. **127**(2), 145–163 (1996). https://doi.org/10.1006/inco.1996.0056, https://doi.org/10.1006/inco.1996.0056
7. Castellan, S.: Weak memory models using event structures. In: Signoles, J. (ed.) Vingt-septièmes Journées Francophones des Langages Applicatifs (JFLA 2016). Saint-Malo, France (Jan 2016), https://hal.inria.fr/hal-01333582
8. Fiore, M., Saville, P.: List objects with algebraic structure. In: Miller, D. (ed.) 2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK. LIPIcs, vol. 84, pp. 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017). https://doi.org/10.4230/LIPIcs.FSCD.2017.16, https://doi.org/10.4230/LIPIcs.FSCD.2017.16
9. Forster, Y., Kammar, O., Lindley, S., Pretnar, M.: On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control. J. Funct. Program. **29**, e15 (2019). https://doi.org/10.1017/S0956796819000121, https://doi.org/10.1017/S0956796819000121
10. Gibbons, J., Hinze, R.: Just do it: simple monadic equational reasoning. In: Chakravarty, M.M.T., Hu, Z., Danvy, O. (eds.) Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011. pp. 2–14. ACM (2011). https://doi.org/10.1145/2034773.2034777, https://doi.org/10.1145/2034773.2034777
11. Grathwohl, N.B.B., Kozen, D., Mamouras, K.: KAT + b! In: Henzinger, T.A., Miller, D. (eds.) Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014. pp. 44:1–44:10. ACM (2014). https://doi.org/10.1145/2603088.2603095, https://doi.org/10.1145/2603088.2603095

12. Hoare, T., van Staden, S.: The laws of programming unify process calculi. Sci. Comput. Program. **85**, 102–114 (2014). https://doi.org/10.1016/j.scico.2013.08.012, https://doi.org/10.1016/j.scico.2013.08.012
13. Hyland, M., Levy, P.B., Plotkin, G.D., Power, J.: Combining algebraic effects with continuations. Theor. Comput. Sci. **375**(1-3), 20–40 (2007). https://doi.org/10.1016/j.tcs.2006.12.026, https://doi.org/10.1016/j.tcs.2006.12.026
14. Hyland, M., Plotkin, G.D., Power, J.: Combining effects: Sum and tensor. Theor. Comput. Sci. **357**(1-3), 70–99 (2006). https://doi.org/10.1016/j.tcs.2006.03.013, https://doi.org/10.1016/j.tcs.2006.03.013
15. Jaber, G., Murawski, A.S.: Complete trace models of state and control. In: Yoshida, N. (ed.) Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12648, pp. 348–374. Springer (2021). https://doi.org/10.1007/978-3-030-72019-3_13, https://doi.org/10.1007/978-3-030-72019-3_13
16. Jagadeesan, R., Jeffrey, A., Riely, J.: Pomsets with preconditions: A simple model of relaxed memory. Proc. ACM Program. Lang. **4**(OOPSLA) (nov 2020). https://doi.org/10.1145/3428262, https://doi.org/10.1145/3428262
17. Jeffrey, A., Riely, J., Batty, M., Cooksey, S., Kaysin, I., Podkopaev, A.: The leaky semicolon: Compositional semantic dependencies for relaxed-memory concurrency. Proc. ACM Program. Lang. **6**(POPL) (jan 2022). https://doi.org/10.1145/3498716, https://doi.org/10.1145/3498716
18. Jouvelot, P., Gifford, D.K.: Algebraic reconstruction of types and effects. In: Wise, D.S. (ed.) Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages, Orlando, Florida, USA, January 21-23, 1991. pp. 303–310. ACM Press (1991). https://doi.org/10.1145/99583.99623, https://doi.org/10.1145/99583.99623
19. Kaiser, J., Dang, H., Dreyer, D., Lahav, O., Vafeiadis, V.: Strong logic for weak memory: Reasoning about release-acquire consistency in iris. In: Müller, P. (ed.) 31st European Conference on Object-Oriented Programming, ECOOP 2017, June 19-23, 2017, Barcelona, Spain. LIPIcs, vol. 74, pp. 17:1–17:29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017). https://doi.org/10.4230/LIPIcs.ECOOP.2017.17, https://doi.org/10.4230/LIPIcs.ECOOP.2017.17
20. Kammar, O.: Algebraic theory of type-and-effect systems. Ph.D. thesis, University of Edinburgh, UK (2014), http://hdl.handle.net/1842/8910
21. Kammar, O., Levy, P.B., Moss, S.K., Staton, S.: A monad for full ground reference cells. In: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017. pp. 1–12. IEEE Computer Society (2017). https://doi.org/10.1109/LICS.2017.8005109, https://doi.org/10.1109/LICS.2017.8005109
22. Kammar, O., McDermott, D.: Factorisation systems for logical relations and monadic lifting in type-and-effect system semantics. In: Staton, S. (ed.) Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2018, Dalhousie University, Halifax, Canada, June 6-9, 2018. Electronic Notes in Theoretical Computer Science, vol. 341, pp. 239–260. Elsevier (2018). https://doi.org/10.1016/j.entcs.2018.11.012, https://doi.org/10.1016/j.entcs.2018.11.012
23. Kammar, O., Plotkin, G.D.: Algebraic foundations for effect-dependent optimisations. In: Field, J., Hicks, M. (eds.) Proceedings of the 39th ACM

SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012. pp. 349–360. ACM (2012). https://doi.org/10.1145/2103656.2103698, https://doi.org/10.1145/2103656.2103698

24. Kang, J., Hur, C., Lahav, O., Vafeiadis, V., Dreyer, D.: A promising semantics for relaxed-memory concurrency. In: Castagna, G., Gordon, A.D. (eds.) Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. pp. 175–189. ACM (2017). https://doi.org/10.1145/3009837.3009850, https://doi.org/10.1145/3009837.3009850
25. Kavanagh, R., Brookes, S.: A denotational semantics for sparc tso. Electronic Notes in Theoretical Computer Science **336**, 223–239 (2018). https://doi.org/https://doi.org/10.1016/j.entcs.2018.03.025, https://www.sciencedirect.com/science/article/pii/S1571066118300288, the Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII)
26. Lahav, O., Giannarakis, N., Vafeiadis, V.: Taming release-acquire consistency. In: Bodík, R., Majumdar, R. (eds.) Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016. pp. 649–662. ACM (2016). https://doi.org/10.1145/2837614.2837643, https://doi.org/10.1145/2837614.2837643
27. Levy, P.B.: Call-By-Push-Value: A Functional/Imperative Synthesis, Semantics Structures in Computation, vol. 2. Springer (2004)
28. Linton, F.E.J.: An outline of functorial semantics. In: Eckmann, B. (ed.) Seminar on Triples and Categorical Homology Theory. pp. 7–52. Springer Berlin Heidelberg, Berlin, Heidelberg (1969)
29. Lucassen, J.M., Gifford, D.K.: Polymorphic effect systems. In: Ferrante, J., Mager, P. (eds.) Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988. pp. 47–57. ACM Press (1988). https://doi.org/10.1145/73560.73564, https://doi.org/10.1145/73560.73564
30. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**(1), 55–92 (1991). https://doi.org/10.1016/0890-5401(91)90052-4, https://doi.org/10.1016/0890-5401(91)90052-4
31. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theor. Comput. Sci. **13**, 85–108 (1981). https://doi.org/10.1016/0304-3975(81)90112-2, https://doi.org/10.1016/0304-3975(81)90112-2
32. Oles, F.J.: A Category-Theoretic Approach to the Semantics of Programming Languages. Ph.D. thesis (1983)
33. Oles, F.J.: Type algebras, functor categories, and block structure. DAIMI Report Series (156) (1983)
34. Plotkin, G.D., Power, J.: Notions of computation determine monads. In: Nielsen, M., Engberg, U. (eds.) Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2303, pp. 342–356. Springer (2002). https://doi.org/10.1007/3-540-45931-6_24, https://doi.org/10.1007/3-540-45931-6_24
35. Plotkin, G.D., Power, J.: Algebraic operations and generic effects. Appl. Categorical Struct. **11**(1), 69–94 (2003). https://doi.org/10.1023/A:1023064908962, https://doi.org/10.1023/A:1023064908962

36. Plotkin, G.D., Pretnar, M.: Handlers of algebraic effects. In: Castagna, G. (ed.) Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5502, pp. 80–94. Springer (2009). https://doi.org/10.1007/978-3-642-00590-9_7, https://doi.org/10.1007/978-3-642-00590-9_7
37. Podkopaev, A., Sergey, I., Nanevski, A.: Operational aspects of C/C++ concurrency. CoRR **abs/1606.01400** (2016), http://arxiv.org/abs/1606.01400
38. Reynolds, J.C.: The essence of algol. In: de Bakker, J.W., van Vliet, J.C. (eds.) Algorithmic Languages. pp. 345–372. International Symposium on Algorithmic Languages, Amsterdam; New York: North-Holland Pub. Co. (1981)
39. Talpin, J., Jouvelot, P.: Polymorphic type, region and effect inference. J. Funct. Program. **2**(3), 245–271 (1992). https://doi.org/10.1017/S0956796800000393, https://doi.org/10.1017/S0956796800000393
40. Talpin, J., Jouvelot, P.: The type and effect discipline. Inf. Comput. **111**(2), 245–296 (1994). https://doi.org/10.1006/inco.1994.1046, https://doi.org/10.1006/inco.1994.1046
41. Wagemaker, J., Foster, N., Kappé, T., Kozen, D., Rot, J., Silva, A.: Concurrent netkat: Modeling and analyzing stateful, concurrent networks. CoRR **abs/2201.10485** (2022), https://arxiv.org/abs/2201.10485
42. Winskel, G.: On powerdomains and modality. Theor. Comput. Sci. **36**, 127–137 (1985). https://doi.org/10.1016/0304-3975(85)90037-4, https://doi.org/10.1016/0304-3975(85)90037-4

A Programming Language

We define a simple idealized call-by-value language for shared-state. The language is parameterized by the number of locations $\bar{\mathtt{n}}$ and the number of values $\bar{\mathtt{m}}$. The technical development is mostly standard, included here for the sake of completeness, while highlighting key points along the way.

A.1 Syntax

The language is higher order, extending Moggi's [30] computational lambda calculus with products and sums; and three shared-state constructs:

$$M, N ::= \quad \dots \quad | \quad M\mathbf{?} \quad | \quad M \mathrel{\mathbf{:=}} N \quad | \quad M \parallel N$$

Figure 3 presents the full syntax; Figure 4 presents syntactic sugar that makes the programs written in this language more familiar and readable.

A.2 Typing

We define a *typing context* as a finitely supported partial map Γ from variable names $\mathtt{x}$ to types A. We write $\cdot$ for the typing context with empty domain. We will often omit the empty typing context $\cdot$ from typing judgments. A typing context Δ *extends* a typing context Γ, written $\Gamma \leq \Delta$, when Δ extends Γ as a

G	$::=$		Ground type
		$(G_1 * \cdots * G_n)$	product
	$\mid$	$\{\iota_1\ \textbf{of}\ G_1 \mid \cdots \mid \iota_n\ \textbf{of}\ G_n\}$	variant
A, B	$::=$		Types
		$(A_1 * \cdots * A_n)$	product
	$\mid$	$\{\iota_1\ \textbf{of}\ A_1 \mid \cdots \mid \iota_n\ \textbf{of}\ A_n\}$	variant
	$\mid$	$A \texttt{->} B$	function
V, W	$::=$		Values
		$\langle V_1, \ldots, V_n\rangle$	tuple
	$\mid$	$\iota\, V$	constructor
	$\mid$	$\lambda\mathtt{x}.\, M$	abstraction

M, N	$::=$		Terms
		$\mathtt{x}$	variable
	$\mid$	$\langle M_1, \ldots, M_n\rangle$	tuple
	$\mid$	$\iota\, M$	constructor
	$\mid$	$\lambda\mathtt{x}.\, M$	abstraction
	$\mid$	$M\, N$	application
	$\mid$	$\textbf{match}\ M\ \textbf{with}\ \langle \mathtt{x}_1, \ldots, \mathtt{x}_n\rangle \texttt{->} N$	matching
	$\mid$	$\textbf{case}\ M\ \textbf{of}\ \{\ \iota_1\, \mathtt{x}_1 \texttt{->} N_1 \mid \cdots \mid \iota_n\, \mathtt{x}_n \texttt{->} N_n\}$	case split
	$\mid$	$M\textbf{?}$	dereference
	$\mid$	$M \textbf{:=} N$	assignment
	$\mid$	$M \parallel N$	par. exec.

Fig. 3. The language's terms and types

Sugar	Elaboration	
• $\mathbf{1}$	$\mathbf{()}$	Unit type
• $\{\cdots\ \iota \cdots\}$	$\{\cdots\ \iota\ \textbf{of}\ \mathbf{1} \cdots\}$	Enumeration variant
• ι	$\iota\ \langle\rangle$	Enumeration constructor
• **Loc**	$\{\mathtt{l}_1 \mid \cdots \mid \mathtt{l}_{\bar{\mathtt{n}}}\}$	Locations type
• **Val**	$\{\mathtt{v}_1 \mid \cdots \mid \mathtt{v}_{\bar{\mathtt{m}}}\}$	Values type
• **Bool**	$\{\textbf{true} \mid \textbf{false}\}$	Booleans
• **if** M **then** N_1 **else** N_2	**case** M **of** $\{$ **true** $\texttt{->} N_1$ **false** $\texttt{->} N_2\}$	Conditionals
• **match** M **with** $\langle\langle\mathtt{x}, \mathtt{y}\rangle, \mathtt{z}\rangle \texttt{->} N$	**match** M **with** $\langle\mathtt{w}, \mathtt{z}\rangle \texttt{->}$ **match** $\mathtt{w}$ **with** $\langle\mathtt{x}, \mathtt{y}\rangle \texttt{->} N$	Nested matching (example), choose a w fresh for N
• **let** $\mathtt{x} = M$ **in** N	$(\lambda\mathtt{x}.\, N)\, M$	Sequencing
• _	$\mathtt{x}$	In a binding occurrence, choose a fresh x for its scope
• $M\ \mathbf{;}\ N$	**let** _ $= M$ **in** N	Non-binding sequencing

Fig. 4. The language's syntactic sugar

partial map: $\mathrm{dom}\,\Gamma \subseteq \mathrm{dom}\,\Delta$, and $\forall \mathtt{x} \in \mathrm{dom}\,\Gamma.\ \Gamma\mathtt{x} = \Delta\mathtt{x}$. We write $\Gamma, \mathtt{x} : A$ for the *shadowing extension* of Γ by the *type assignment* $\mathtt{x} : A$:

$$(\Gamma, \mathtt{x} : A)\mathtt{y} \coloneqq \begin{cases} A & \mathtt{y} = \mathtt{x} \\ \Gamma\mathtt{y} & \text{otherwise} \end{cases}$$

When shadowing the empty typing context we will simply omit it, i.e. we write $\mathtt{x} : A$ instead of $\cdot, \mathtt{x} : A$. We also write $(x : A) \in \Gamma$ for $\Gamma x = A$.

Figure 5 presents the *typing judgment* $\Gamma \vdash M : A$, an inductively defined relation asserting that M is a well-formed term of type A, under typing context Γ. When $\vdash M : A$ we say that M is *closed*. We define $\Gamma \vdash A$ be the set of all programs M such that $\Gamma \vdash M : A$.

$$\boxed{\Gamma \vdash M : A}$$

$$\frac{}{\Gamma \vdash \mathtt{x} : A}(\Gamma\mathtt{x} = A) \qquad \frac{\Gamma, \mathtt{x} : A \vdash M : B}{\Gamma \vdash \lambda\mathtt{x}.\, M : A \mathrel{\texttt{->}} B} \qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A \mathrel{\texttt{->}} B}{\Gamma \vdash N M : B}$$

$$\frac{\forall i.\ \Gamma \vdash M_i : A_i}{\Gamma \vdash \langle M_1, \ldots, M_n \rangle : \mathbf{(}A_1 \mathbin{\mathbf{*}} \cdots \mathbin{\mathbf{*}} A_n\mathbf{)}} \qquad \frac{\Gamma \vdash M : A_i}{\Gamma \vdash \iota_i\, M : \{\iota_1 \ \mathbf{of}\ A_1 \mid \cdots \mid \iota_n \ \mathbf{of}\ A_n\}}$$

$$\frac{\Gamma \vdash M : \mathbf{(}A_1 \mathbin{\mathbf{*}} \cdots \mathbin{\mathbf{*}} A_n\mathbf{)} \qquad \Gamma, \mathtt{x}_1 : A_1, \ldots, \mathtt{x}_n : A_n \vdash N : A}{\Gamma \vdash \mathbf{match}\, M\, \mathbf{with}\, \langle \mathtt{x}_1, \ldots, \mathtt{x}_n \rangle \mathrel{\texttt{->}} N : A}$$

$$\frac{\Gamma \vdash M : \{\iota_1 \ \mathbf{of}\ A_1 \mid \cdots \mid \iota_n \ \mathbf{of}\ A_n\} \qquad \forall i.\ \Gamma, \mathtt{x}_i : A_i \vdash N_i : A}{\Gamma \vdash \mathbf{case}\, M\, \mathbf{of}\, \{\iota_1\, \mathtt{x}_1 \mathrel{\texttt{->}} N_1 \mid \cdots \mid \iota_n\, \mathtt{x}_n \mathrel{\texttt{->}} N_n\} : A} \qquad \frac{\Gamma \vdash M : \mathbf{Loc}}{\Gamma \vdash M\mathbf{?} : \mathbf{Val}}$$

$$\frac{\Gamma \vdash M : \mathbf{Loc} \qquad \Gamma \vdash N : \mathbf{Val}}{\Gamma \vdash M \mathbin{\mathbf{:=}} N : \mathbf{1}} \qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash M \parallel N : \mathbf{(}A \mathbin{\mathbf{*}} B\mathbf{)}}$$

Fig. 5. The language's type system

A.3 Substitutions

A *program substitution* (or just *substitution*) is a partial function from program variables to closed values. We write $M[V_1/\mathtt{x}_1 \ldots V_m/\mathtt{x}_m]$ for the application of the substitution that maps $\mathtt{x}_i \mapsto V_i$ on M. For a substitution Θ and a set of program variables $\mathtt{X}$, we obtain $\Theta|_{\notin \mathtt{X}}$ by removing $\mathtt{X}$ from Θ's domain. A substitution Θ extends to a function defined on all programs: first by acting as the identity on all other variables; and then by recursively applying to subprograms, removing variables from the domain of the substitution when going under binders, e.g.:

$$\begin{aligned}&\Theta\,(\mathbf{match}\,M\;\mathbf{with}\,\langle \mathtt{x}_1,\ldots,\mathtt{x}_n\rangle \;\texttt{->}\; N)\\ &\qquad\qquad\qquad := \mathbf{match}\,\Theta M\;\mathbf{with}\,\langle \mathtt{x}_1,\ldots,\mathtt{x}_n\rangle \;\texttt{->}\; \Theta|_{\notin\{\mathtt{x}_1,\ldots\mathtt{x}_n\}}N\end{aligned}$$

We relate substitutions to denotations in the following lemma.

Lemma 1 (Substitution). *Let Γ and Δ be typing contexts such that $\Gamma \le \Delta$. Let Θ be a substitution such that $\operatorname{dom}\Theta = \operatorname{dom}\Delta \setminus \operatorname{dom}\Gamma$ and $\forall \mathtt{x} \in \operatorname{dom}\Theta.\ \cdot \vdash \Theta\mathtt{x} : \Delta\mathtt{x}$. Let $\gamma \in [\![\Gamma]\!]$, and define $\gamma^{\Theta} \in [\![\Delta]\!]$ by $\forall \mathtt{x} \in \operatorname{dom}\Theta.\ \gamma^{\Theta}\mathtt{x} := [\![\Theta\mathtt{x}]\!]^{\mathrm{v}}$. Assume $\Delta \vdash M : A$ for some A. Then $[\![M]\!]^{\mathrm{c}}\,\gamma^{\Theta} = [\![\Theta M]\!]^{\mathrm{c}}\,\gamma$.*

A.4 Operational Semantics

The *reduction step* $\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'$ is an inductively defined relation asserting that, with initial store σ, the closed term M reduces in a single computation step — via *action* $a \in \{\mathrm{U}_{\ell,v}, \mathrm{L}_{\ell}, \varepsilon\}$ — to the closed term M', and the store changes to σ'. Figure 6 presents the inference rules for the shared-state constructs. The other rules reflect standard small-step rules, where the action is ε in axioms, and the action carries over in unary rules; e.g. $\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right), \mathsf{b} := \mathsf{a?} \overset{\mathrm{L}_{\mathsf{a}}}{\rightsquigarrow} \left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right), \mathsf{b} := 1$. The standard operational semantics is derived: we write $\sigma, M \rightsquigarrow \sigma', M'$ when there exists an action a such that $\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'$; e.g. $\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right), \mathsf{b} := \mathsf{a?} \rightsquigarrow \left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right), \mathsf{b} := 1$. We write $\rightsquigarrow\!*$ for its reflexive-transitive closure; e.g. $\left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&0&1\end{smallmatrix}\right), \mathsf{b}{:=}\mathsf{a?} \rightsquigarrow\!* \left(\begin{smallmatrix}\mathsf{a}&\mathsf{b}&\mathsf{c}\\1&1&1\end{smallmatrix}\right), \langle\rangle$.

B Proofs

The proofs below were omitted from the body of the paper for the sake of brevity.

B.1 Representation Theorem

Without further ado:

Proof (Theorem 1). We will use the following notation for conditionals:

$$\mathcal{C}\;?\;\mathtt{t} : \mathtt{f} := \begin{cases}\mathtt{t} & \mathcal{C}\\ \mathtt{f} & \text{else}\end{cases}$$

and a notation that helps with decomposing sets of traces by their first transition:

$$^{-}/_{-} : (\mathbb{S}\times\mathbb{S}) \times \underline{\mathcal{T}}X \to \mathcal{P}_{\mathrm{fin}}\left((\mathbb{S}\times\mathbb{S})^{*}\cdot X\right) \quad {}^{\langle\sigma,\rho\rangle}/P := \left\{\tau \in (\mathbb{S}\times\mathbb{S})^{*}\cdot X \;\middle|\; \langle\sigma,\rho\rangle\,\tau \in P\right\}$$

In the notations of §4.5, $R_P^{\langle\sigma,\rho\rangle} = {}^{\langle\sigma,\rho\rangle}/P \setminus X$ and $X_{P,f}^{\langle\sigma,\rho\rangle} = \tilde{\bigvee}^{\mathcal{A}}_{x\in\left({}^{\langle\sigma,\rho\rangle}/P\right)\cap X} fx$.

First show that $f = (\gg\!= f) \circ \mathrm{return}$ by using the equations of Res:

$$\mathrm{return}\,x \gg\!= f = \vec{\tilde{L}}^{\mathcal{A}}\left(\sigma.\ \tilde{\bigvee_{\rho\in\mathbb{S}}}^{\mathcal{A}}\ \vec{\tilde{\mathrm{U}}}^{\mathcal{A}}_{\rho}\left(\tilde{\mathrm{Y}}^{\mathcal{A}}\,(\varnothing \gg\!= f)\ \tilde{\vee}^{\mathcal{A}} \tilde{\bigvee_{y\in\{z\in X \mid \sigma=\rho\wedge z=x\}}}^{\mathcal{A}} fy\right)\right)$$

$$\boxed{\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'}$$

$$\frac{\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'}{\sigma, M\mathbf{?} \overset{a}{\rightsquigarrow} \sigma', M'\mathbf{?}} \qquad \frac{}{\sigma, \ell\mathbf{?} \overset{\mathrm{L}_\ell}{\rightsquigarrow} \sigma, \sigma_\ell} \qquad \frac{\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'}{\sigma, M \mathbin{\mathbf{:=}} N \overset{a}{\rightsquigarrow} \sigma', M' \mathbin{\mathbf{:=}} N}$$

$$\frac{\sigma, N \overset{a}{\rightsquigarrow} \sigma', N'}{\sigma, V \mathbin{\mathbf{:=}} N \overset{a}{\rightsquigarrow} \sigma', V \mathbin{\mathbf{:=}} N'} \qquad \frac{}{\sigma, \ell \mathbin{\mathbf{:=}} v \overset{\mathrm{U}_{\ell,v}}{\rightsquigarrow} \sigma\,[\ell \mapsto v]\,, \langle\rangle} \qquad \frac{\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'}{\sigma, M \parallel N \overset{a}{\rightsquigarrow} \sigma', M' \parallel N}$$

$$\frac{\sigma, N \overset{a}{\rightsquigarrow} \sigma', N'}{\sigma, M \parallel N \overset{a}{\rightsquigarrow} \sigma', M \parallel N'} \qquad \frac{}{\sigma, V \parallel W \overset{\varepsilon}{\rightsquigarrow} \sigma, \langle V, W\rangle} \qquad \frac{\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'}{\sigma, MN \overset{a}{\rightsquigarrow} \sigma', M'N}$$

$$\frac{\sigma, N \overset{a}{\rightsquigarrow} \sigma', N'}{\sigma, VN \overset{a}{\rightsquigarrow} \sigma', VN'} \qquad \frac{}{\sigma, (\lambda \mathtt{x}.\, M)V \overset{\varepsilon}{\rightsquigarrow} \sigma, M[V/\mathtt{x}]} \qquad \frac{\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'}{\sigma, \iota\, M \overset{a}{\rightsquigarrow} \sigma', \iota\, M'}$$

$$\frac{\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'}{\sigma, \mathbf{case}\, M\, \mathbf{of}\, \{\ldots \mid \iota_i\, \mathtt{x}_i \mathrel{\mathtt{->}} N_i \mid \ldots\} \overset{a}{\rightsquigarrow} \sigma', \mathbf{case}\, M'\, \mathbf{of}\, \{\ldots \mid \iota_i\, \mathtt{x}_i \mathrel{\mathtt{->}} N_i \mid \ldots\}}$$

$$\frac{}{\sigma, \mathbf{case}\, \iota_i\, V\, \mathbf{of}\, \{\iota_1\, \mathtt{x}_1 \mathrel{\mathtt{->}} M_1 \mid \ldots \mid \iota_n\, \mathtt{x}_n \mathrel{\mathtt{->}} M_n\} \overset{\varepsilon}{\rightsquigarrow} \sigma, M_i[V/\mathtt{x}_i]}$$

$$\frac{\sigma, M_{i+1} \overset{a}{\rightsquigarrow} \sigma', M'_{i+1}}{\sigma, \langle V_1, \ldots, V_i, M_{i+1}, \ldots, M_n\rangle \overset{a}{\rightsquigarrow} \sigma', \big\langle V_1, \ldots, V_i, M'_{i+1}, \ldots, M_n\big\rangle}$$

$$\frac{\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'}{\sigma, \mathbf{match}\, M\, \mathbf{with}\, \langle \mathtt{x}_1, \ldots, \mathtt{x}_n\rangle \mathrel{\mathtt{->}} N \overset{a}{\rightsquigarrow} \sigma', \mathbf{match}\, M'\, \mathbf{with}\, \langle \mathtt{x}_1, \ldots, \mathtt{x}_n\rangle \mathrel{\mathtt{->}} N}$$

$$\frac{}{\sigma, \mathbf{match}\, \langle V_1, \ldots, V_n\rangle\, \mathbf{with}\, \langle \mathtt{x}_1, \ldots, \mathtt{x}_n\rangle \mathrel{\mathtt{->}} N \overset{\varepsilon}{\rightsquigarrow} \sigma, N[V_1/\mathtt{x}_1 \ldots V_n/\mathtt{x}_n]}$$

Fig. 6. Operational semantics for shared-state constructs

$$= \vec{\tilde{L}}^{\mathcal{A}}\left(\sigma.\ \mathop{\tilde{\bigvee}^{\mathcal{A}}}_{\rho\in\mathbb{S}} \vec{\tilde{\mathrm{U}}}^{\mathcal{A}}_{\rho}\left(\varnothing\ \tilde{\vee}^{\mathcal{A}} \mathop{\tilde{\bigvee}^{\mathcal{A}}}_{\imath<(\sigma=\rho?1:0)} fx\right)\right) = \vec{\tilde{L}}^{\mathcal{A}}\left(\sigma.\ \vec{\tilde{\mathrm{U}}}^{\mathcal{A}}_{\sigma}\left(fx\right)\right) = fx$$

Next, we show that $(\mathrel{\rangle\!\rangle\!=} f)$ is a homomorphism case-by-case:

Lookup. Since $\mathrm{L}_{\ell}\left(v.\ \vec{\mathrm{L}}\left(\sigma.\ x_{\sigma,v}\right)\right) \overset{\textsc{Res}}{=} \vec{\mathrm{L}}\left(\sigma.\ x_{\sigma,\sigma_{\ell}}\right)$, the equation holds in any Res-model $\mathcal{A}$. Note also that ${}^{\langle\sigma,\rho\rangle}/\tilde{\mathrm{L}}_{\ell}\left(v.\ P_v\right) = {}^{\langle\sigma,\rho\rangle}/P_{\sigma_{\ell}}$. So:

$$\begin{aligned}
&\tilde{\mathrm{L}}^{\mathcal{A}}_{\ell}\left(v.\ P_v \mathrel{\rangle\!\rangle\!=} f\right)\\
&= \tilde{\mathrm{L}}^{\mathcal{A}}_{\ell}\left(v.\ \vec{\tilde{L}}^{\mathcal{A}}\left(\sigma.\ \mathop{\tilde{\bigvee}^{\mathcal{A}}}_{\rho\in\mathbb{S}} \vec{\tilde{\mathrm{U}}}^{\mathcal{A}}_{\rho}\left(\begin{array}{c}\tilde{\mathrm{Y}}^{\mathcal{A}}\left(\left({}^{\langle\sigma,\rho\rangle}/P_v\right)\setminus X \mathrel{\rangle\!\rangle\!=} f\right)\\ \tilde{\vee}^{\mathcal{A}}\\ \tilde{\bigvee}^{\mathcal{A}}_{x\in\left({}^{\langle\sigma,\rho\rangle}/P_v\right)\cap X} fx\end{array}\right)\right)\right)\\
&= \vec{\tilde{L}}^{\mathcal{A}}\left(\sigma.\ \mathop{\tilde{\bigvee}^{\mathcal{A}}}_{\rho\in\mathbb{S}} \vec{\tilde{\mathrm{U}}}^{\mathcal{A}}_{\rho}\left(\begin{array}{c}\tilde{\mathrm{Y}}^{\mathcal{A}}\left(\left({}^{\langle\sigma,\rho\rangle}/P_{\sigma_{\ell}}\right)\setminus X \mathrel{\rangle\!\rangle\!=} f\right)\\ \tilde{\vee}^{\mathcal{A}}\\ \tilde{\bigvee}^{\mathcal{A}}_{x\in\left({}^{\langle\sigma,\rho\rangle}/P_{\sigma_{\ell}}\right)\cap X} fx\end{array}\right)\right)\\
&= \vec{\tilde{L}}^{\mathcal{A}}\left(\sigma.\ \mathop{\tilde{\bigvee}^{\mathcal{A}}}_{\rho\in\mathbb{S}} \vec{\tilde{\mathrm{U}}}^{\mathcal{A}}_{\rho}\left(\begin{array}{c}\tilde{\mathrm{Y}}^{\mathcal{A}}\left(\left({}^{\langle\sigma,\rho\rangle}/\tilde{\mathrm{L}}_{\ell}\left(v.\ P_v\right)\right)\setminus X \mathrel{\rangle\!\rangle\!=} f\right)\\ \tilde{\vee}^{\mathcal{A}}\\ \tilde{\bigvee}^{\mathcal{A}}_{x\in\left({}^{\langle\sigma,\rho\rangle}/\tilde{\mathrm{L}}_{\ell}(v.\ P_v)\right)\cap X} fx\end{array}\right)\right)\\
&= \tilde{\mathrm{L}}_{\ell}\left(v.\ P_v\right) \mathrel{\rangle\!\rangle\!=} f
\end{aligned}$$

Update. $\tilde{\mathrm{U}}^{\mathcal{A}}_{\ell,v}\left(P \mathrel{\rangle\!\rangle\!=} f\right) = \tilde{\mathrm{U}}_{\ell,v}P \mathrel{\rangle\!\rangle\!=} f$ follows from $\mathrm{U}_{\ell,v}\vec{\mathrm{L}}\left(\sigma.\ \bigvee_{\rho\in\mathbb{S}}\vec{\mathrm{U}}_{\rho}x_{\sigma,\rho}\right) \overset{\textsc{Res}}{=} \vec{\mathrm{L}}\left(\sigma.\ x_{\sigma[\ell\mapsto v],\rho}\right)$ and ${}^{\langle\sigma,\rho\rangle}/\tilde{\mathrm{U}}_{\ell,v}P = {}^{\langle\sigma[\ell\mapsto v],\rho\rangle}/P$, similarly to the lookup case.

Yield. We have ${}^{\langle\sigma,\rho\rangle}/\tilde{\mathrm{Y}}P = \sigma = \rho\ ?\ P : \varnothing \subseteq \left(\mathbb{S}\times\mathbb{S}\right)^{+}\cdot X$. Thus the right-hand-side of the $\tilde{\vee}^{\mathcal{A}}$ vanishes, and more specifically we have:

$$\begin{aligned}
&\tilde{\mathrm{Y}}^{\mathcal{A}}\left(\left({}^{\langle\sigma,\rho\rangle}/\tilde{\mathrm{Y}}P\right)\setminus X \mathrel{\rangle\!\rangle\!=} f\right)\tilde{\vee}^{\mathcal{A}} \mathop{\tilde{\bigvee}^{\mathcal{A}}}_{x\in\left({}^{\langle\sigma,\rho\rangle}/\tilde{\mathrm{Y}}P\right)\cap X} fx\\
&\qquad = \tilde{\mathrm{Y}}^{\mathcal{A}}\left(\left(\sigma = \rho\ ?\ P : \varnothing\right) \mathrel{\rangle\!\rangle\!=} f\right) = \mathop{\tilde{\bigvee}^{\mathcal{A}}}_{\imath<(\sigma=\rho?1:0)} \tilde{\mathrm{Y}}^{\mathcal{A}}\left(P \mathrel{\rangle\!\rangle\!=} f\right)
\end{aligned}$$

The desired $\tilde{\mathrm{Y}}^{\mathcal{A}}\left(P \mathrel{\rangle\!\rangle\!=} f\right) = \tilde{\mathrm{Y}}P \mathrel{\rangle\!\rangle\!=} f$ now follows from:

$$\vec{\mathrm{L}}\left(\sigma.\ \bigvee_{\rho\in\mathbb{S}}\vec{\mathrm{U}}_{\rho}\left(\bigvee_{\imath<(\sigma=\rho?1:0)} x\right)\right) \overset{\textsc{Res}}{=} x$$

Choice. By induction on the length of the longest trace in $\tilde{\bigvee}_{\imath<\alpha}P_{\imath}$. We have

$$\left({}^{\langle\sigma,\rho\rangle}/\tilde{\bigvee_{\imath<\alpha}}P_{\imath}\right)\cap T = \left({}^{\langle\sigma,\rho\rangle}/\bigcup_{\imath<\alpha}P_{\imath}\right)\cap T = \left(\bigcup_{\imath<\alpha}{}^{\langle\sigma,\rho\rangle}/P_{\imath}\right)\cap T$$

$$= \bigcup_{\imath<\alpha} \left({}^{\langle\sigma,\rho\rangle}/P_\imath\right) \cap T = \tilde{\bigvee_{\imath<\alpha}} \left({}^{\langle\sigma,\rho\rangle}/P_\imath\right) \cap T \quad (\star)$$

In particular, $\left({}^{\langle\sigma,\rho\rangle}/\tilde{\bigvee}_{\imath<\alpha} P_\imath\right) \smallsetminus X = \tilde{\bigvee}_{\imath<\alpha} \left({}^{\langle\sigma,\rho\rangle}/P_\imath\right) \smallsetminus X$. The longest trace here is shorter than in $\tilde{\bigvee}_{\imath<\alpha} P_\imath$. By induction we have

$$\left(\tilde{\bigvee_{\imath<\alpha}} \left({}^{\langle\sigma,\rho\rangle}/P_\imath\right) \smallsetminus X\right) \gg= f = \tilde{\bigvee_{\imath<\alpha}}^{\mathcal{A}} \left(\left({}^{\langle\sigma,\rho\rangle}/P_\imath\right) \smallsetminus X \gg= f\right)$$

Also by $(\star)$, we have $\left({}^{\langle\sigma,\rho\rangle}/\tilde{\bigvee}_{\imath<\alpha} P_\imath\right) \cap X = \bigcup_{\imath<\alpha} \left({}^{\langle\sigma,\rho\rangle}/P_\imath\right) \cap X$. By ND-join, we have $\tilde{\bigvee}^{\mathcal{A}}_{x \in \left({}^{\langle\sigma,\rho\rangle}/\tilde{\bigvee}_{\imath<\alpha} P_\imath\right) \cap X} fx = \tilde{\bigvee}^{\mathcal{A}}_{\imath<\alpha} \tilde{\bigvee}^{\mathcal{A}}_{x \in \left({}^{\langle\sigma,\rho\rangle}/P_\imath\right) \cap X} fx$. By further rearranging $\tilde{\bigvee}^{\mathcal{A}}$, the desired $\tilde{\bigvee}^{\mathcal{A}}_{\imath<\alpha} (P \gg= f) = \tilde{\bigvee}^{\mathcal{A}}_{\imath<\alpha} P \gg= f$ follows.

Finally, to show that $\gg= f$ uniquely satisfies the above, suppose g is a homomorphism satisfying $f = g \circ \mathrm{return}$. For any $P \in \underline{\mathcal{T}}X$ and $h : X \to \mathcal{A}$, by induction on the length of the longest trace in P, we have $g\,(P \gg= h) = P \gg= g \circ h$; since $\gg=$ is defined in terms of the operators and g is a homomorphism. Furthermore, $(\gg= \mathrm{return})$ is the identity (easily verified using the simplified definition of $\gg=$), and therefore $gP = g\,(P \gg= \mathrm{return}) = P \gg= g \circ \mathrm{return} = P \gg= f$.

B.2 Metatheoretical Results

Soundness First, we note that the homomorphic extension is monotonic:

Lemma 2. *If $P \subseteq Q$ then $P \gg= f \subseteq Q \gg= f$.*

Proof. By induction on the length of the longest trace in Q.

To show soundness, we first show that a single step respects the denotations:

Lemma 3. *If $\sigma, M \overset{a}{\rightsquigarrow} \sigma', M'$ then $a^\sharp\, [\![M']\!]^{\mathsf{c}}\, \sigma \subseteq [\![M]\!]^{\mathsf{c}}\, \sigma$, where:*

$$\mathrm{U}^\sharp_{\ell,v} P \coloneqq \tilde{\mathrm{U}}_{\ell,v} \tilde{\mathrm{Y}}^{?} P \qquad \mathrm{L}^\sharp_{\ell} P \coloneqq \tilde{\mathrm{Y}}^{?} P \qquad \varepsilon^\sharp P \coloneqq P$$

For example, $\left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & 1 \end{smallmatrix}\right), \mathsf{a?} \parallel \mathsf{a} \coloneqq 0 \overset{\mathrm{U}_{\mathsf{a},0}}{\rightsquigarrow} \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 0 & 0 & 1 \end{smallmatrix}\right), \mathsf{a?} \parallel \langle\rangle$, and indeed one can calculate that:

$$\mathrm{U}^\sharp_{\mathsf{a},0} [\![\mathsf{a?} \parallel \langle\rangle]\!]^{\mathsf{c}} \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & 1 \end{smallmatrix}\right) \subseteq [\![\mathsf{a?} \parallel \mathsf{a} \coloneqq 0]\!]^{\mathsf{c}} \left(\begin{smallmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ 1 & 0 & 1 \end{smallmatrix}\right)$$

Proof. By induction on the reduction step's derivation:

$\sigma, (\lambda\mathrm{x}.\, M)V \overset{\varepsilon}{\rightsquigarrow} \sigma, M[V/\mathrm{x}]$**.** By Lemma 1.
$\sigma, V \coloneqq N \overset{a}{\rightsquigarrow} \sigma', V \coloneqq N'$**.** We calculate:

$$\begin{aligned} [\![V \coloneqq N]\!]^{\mathsf{c}}\, \sigma &= \left([\![V]\!]^{\mathsf{c}} \gg= \lambda\ell.\ [\![N]\!]^{\mathsf{c}} \gg= \lambda v.\ \tilde{\mathrm{U}}_{\ell,v} \tilde{\mathrm{Y}}^{?} \tilde{\langle\rangle}\right) \sigma \\ &= \left(\mathrm{return}\, [\![V]\!]^{\mathsf{v}} \gg= \lambda\ell.\ [\![N]\!]^{\mathsf{c}} \gg= \lambda v.\ \tilde{\mathrm{U}}_{\ell,v} \tilde{\mathrm{Y}}^{?} \tilde{\langle\rangle}\right) \sigma \end{aligned}$$

$$\begin{aligned}
&= \left(\llbracket N \rrbracket^{\mathsf{c}} \gg\!= \lambda v.\, \tilde{\mathrm{U}}_{\llbracket V \rrbracket^{\mathsf{v}},v} \tilde{\mathrm{Y}}^{?} \widetilde{\langle\rangle}\right) \sigma \\
&\supseteq \left(a^{\sharp} \llbracket N' \rrbracket^{\mathsf{c}} \gg\!= \lambda v.\, \tilde{\mathrm{U}}_{\llbracket V \rrbracket^{\mathsf{v}},v} \tilde{\mathrm{Y}}^{?} \widetilde{\langle\rangle}\right) \sigma \\
&= a^{\sharp} \left(\llbracket N' \rrbracket^{\mathsf{c}} \gg\!= \lambda v.\, \tilde{\mathrm{U}}_{\llbracket V \rrbracket^{\mathsf{v}},v} \tilde{\mathrm{Y}}^{?} \widetilde{\langle\rangle}\right) \sigma \\
&= a^{\sharp} \left(\llbracket V \rrbracket^{\mathsf{c}} \gg\!= \lambda \ell.\, \llbracket N' \rrbracket^{\mathsf{c}} \gg\!= \lambda v.\, \tilde{\mathrm{U}}_{\ell,v} \tilde{\mathrm{Y}}^{?} \widetilde{\langle\rangle}\right) \sigma \\
&= a^{\sharp} \llbracket V \mathrel{\boldsymbol{:=}} N' \rrbracket^{\mathsf{c}} \sigma
\end{aligned}$$

where in the $\supseteq$ step we used the induction hypothesis and Lemma 2, and in the following step we used the fact that $a^{\sharp}$ is defined in terms of the operators and $\gg\!=$ is homomorphic.

$\sigma, \ell\boldsymbol{?} \overset{\mathrm{L}_{\ell}}{\rightsquigarrow} \sigma, \sigma_{\ell}$**.** We calculate:

$$\begin{aligned}
\llbracket \ell\boldsymbol{?} \rrbracket^{\mathsf{c}} \sigma &= \left(\llbracket \ell \rrbracket^{\mathsf{c}} \gg\!= \lambda \ell.\, \tilde{\mathrm{L}}_{\ell} \left(v.\, \tilde{\mathrm{Y}}^{?} \tilde{v}\right)\right) \sigma \\
&= \left(\tilde{\mathrm{L}}_{\ell} \left(v.\, \tilde{\mathrm{Y}}^{?} \tilde{v}\right)\right) \sigma = \left(\tilde{\mathrm{Y}}^{?} \tilde{\sigma_{\ell}}\right) \sigma = \mathrm{L}_{\ell}^{\sharp} \llbracket \sigma_{\ell} \rrbracket^{\mathsf{c}} \sigma
\end{aligned}$$

$\sigma, M \parallel N \overset{\mathrm{U}_{\ell,v}}{\rightsquigarrow} \sigma', M' \parallel N$**.** Let $\langle \sigma, \rho \rangle \omega \in \tilde{\mathrm{U}}_{\ell,v} \tilde{\mathrm{Y}}^{?} \llbracket M' \parallel N \rrbracket^{\mathsf{c}}$. We split to cases, showing in both that $\langle \sigma, \rho \rangle \omega \in \llbracket M \parallel N \rrbracket^{\mathsf{c}}$:

$\langle \sigma, \rho \rangle \omega \in \tilde{\mathrm{U}}_{\ell,v} \tilde{\mathrm{Y}} \llbracket M' \parallel N \rrbracket^{\mathsf{c}}$**.** So $\rho = \sigma[\ell \mapsto v]$ and $\omega \in \llbracket M' \parallel N \rrbracket^{\mathsf{c}}$. Thus there exist $\tau \in \llbracket M' \rrbracket^{\mathsf{c}}$ and $\pi \in \llbracket N \rrbracket^{\mathsf{c}}$ such that $\tau \,\|\, \pi \implies \omega$. By Brk-Left, $\langle \sigma, \rho \rangle \tau \,\|\, \pi \implies \langle \sigma, \rho \rangle \omega$. Note that $\langle \sigma, \rho \rangle \tau \in \tilde{\mathrm{U}}_{\ell,v} \tilde{\mathrm{Y}} \llbracket M' \rrbracket^{\mathsf{c}} \subseteq \llbracket M \rrbracket^{\mathsf{c}}$ by the induction hypothesis. Therefore, $\langle \sigma, \rho \rangle \omega \in \llbracket M \parallel N \rrbracket^{\mathsf{c}}$.

$\langle \sigma, \rho \rangle \tau \in \tilde{\mathrm{U}}_{\ell,v} \llbracket M' \parallel N \rrbracket^{\mathsf{c}}$**.** $\langle \sigma[\ell \mapsto v], \rho \rangle \omega \in \llbracket M' \parallel N \rrbracket^{\mathsf{c}}$. Thus there exist $\tau \in \llbracket M' \rrbracket^{\mathsf{c}}$ and $\pi \in \llbracket N \rrbracket^{\mathsf{c}}$ such that $\tau \,\|\, \pi \implies \langle \sigma[\ell \mapsto v], \rho \rangle \omega$. By Seq-Left, $\langle \sigma, \sigma[\ell \mapsto v] \rangle \tau \,\|\, \pi \implies \langle \sigma, \rho \rangle \omega$. Note that $\langle \sigma, \sigma[\ell \mapsto v] \rangle \tau \in \tilde{\mathrm{U}}_{\ell,v} \tilde{\mathrm{Y}} \llbracket M' \rrbracket^{\mathsf{c}} \subseteq \llbracket M \rrbracket^{\mathsf{c}}$ by the induction hypothesis. Therefore, $\langle \sigma, \rho \rangle \omega \in \llbracket M \parallel N \rrbracket^{\mathsf{c}}$.

Other cases are similar to one of the above.

Soundness of the standard operational semantics follows:

Proof (Soundness). By induction on the number of steps, in each step using Lemma 3 and always "choosing" not to take $\tilde{\mathrm{Y}}$ from $\tilde{\mathrm{Y}}^{?}$. For example, consider the case $\varsigma, N \overset{\mathrm{U}_{\ell,v}}{\rightsquigarrow} \varsigma[\ell \mapsto v], M \rightsquigarrow\!\!* \rho, V$. By the induction hypothesis:

$$\langle \varsigma[\ell \mapsto v], \rho \rangle \llbracket V \rrbracket^{\mathsf{v}} \in \llbracket M \rrbracket^{\mathsf{c}}$$

Thus $\langle \varsigma, \rho \rangle \llbracket V \rrbracket^{\mathsf{v}} \in \tilde{\mathrm{U}}_{\ell,v} \llbracket M \rrbracket^{\mathsf{c}} \varsigma \subseteq \llbracket N \rrbracket^{\mathsf{c}} \varsigma$ by Lemma 3.

Fundamental Lemma The logical relation defined in §6 is defined so that the theorem can be proven by induction on the typing derivation. The following lemma will be helpful in the parallel execution case:

Lemma 4. *Assume $\tau \,\|\, \pi \implies \omega$. If $M \xrightarrow{\tau} V$ and $N \xrightarrow{\pi} W$, then $M \parallel N \xrightarrow{\omega} \langle V, W \rangle$.*

Proof. By induction on the assumed synchronization.

Var-Left. There exist $\sigma, \rho, x, \varsigma, \alpha, y$ such that:

1. $\tau = \langle \sigma, \rho \rangle \, x$
2. $\pi = \langle \rho, \varsigma \rangle \, \alpha y$
3. $\omega = \langle \sigma, \varsigma \rangle \, \alpha \, \langle x, y \rangle$

By #1 & #2:

$$M \parallel N \xrightarrow{\langle \sigma, \rho \rangle} V \parallel N \xrightarrow{\langle \rho, \varsigma \rangle \alpha} V \parallel W \overset{\varepsilon}{\rightsquigarrow} \langle V, W \rangle$$

Combining the executions and using transitivity, by #3 we are done.

Brk-Left. There exist $\sigma, \rho, \alpha, \beta, \eta, x, y$ such that:

1. $\tau = \langle \sigma, \rho \rangle \, \alpha x$
2. $\pi = \beta y$
3. $\omega = \langle \sigma, \rho \rangle \, \eta \, \langle x, y \rangle$
4. $\alpha x \parallel \beta y \Longrightarrow \eta \, \langle x, y \rangle$

By #1 & #2, there exists K such that:

$$\begin{aligned} M \xrightarrow{\langle \sigma, \rho \rangle} K &\xrightarrow{\alpha x} V \\ N &\xrightarrow{\beta y} W \end{aligned}$$

Thus by #4, using the induction hypothesis:

$$M \parallel N \xrightarrow{\langle \sigma, \rho \rangle} K \parallel N \xrightarrow{\eta \langle x, y \rangle} \langle V, W \rangle$$

By #3 we are done.

Seq-Left. There exist $\sigma, \rho, \varsigma, \alpha, \beta, \eta, x, y$ such that:

1. $\tau = \langle \sigma, \rho \rangle \, \alpha x$
2. $\pi = \beta y$
3. $\omega = \langle \sigma, \varsigma \rangle \, \eta \, \langle x, y \rangle$
4. $\alpha x \parallel \beta y \Longrightarrow \langle \rho, \varsigma \rangle \, \eta \, \langle x, y \rangle$

By #1 & #2, there exists K such that:

$$\begin{aligned} M \xrightarrow{\langle \sigma, \rho \rangle} K &\xrightarrow{\alpha x} V \\ N &\xrightarrow{\beta y} W \end{aligned}$$

Thus by #4, using the induction hypothesis:

$$M \parallel N \xrightarrow{\langle \sigma, \rho \rangle} K \parallel N \xrightarrow{\langle \rho, \varsigma \rangle \eta \langle x, y \rangle} \langle V, W \rangle$$

By #3 we are done.

Other cases are symmetric.

We find the following concept relevant for the proof of the Fundamental Lemma: we say a trace is *preserving* if it consists solely of transitions in which the stores are equal; e.g. $[\![\ell\mathbf{?}]\!]^{\mathsf{c}}$ consists solely of preserving traces.

Proof (Fundamental Lemma). By induction on the typing judgment.

$\mathit{\Gamma} \vdash NM : B$**.** Let $\mathit{\Theta} \in \mathcal{G}\llparenthesis \mathit{\Gamma} \rrparenthesis$, and let $\tau \in \llbracket \mathit{\Theta}\,(NM) \rrbracket^{\mathsf{c}} = \llbracket \mathit{\Theta} N \rrbracket^{\mathsf{c}} \gg\!= \lambda f.\ \llbracket \mathit{\Theta} M \rrbracket^{\mathsf{c}} \gg\!= f$. Therefore, there exist $\alpha, \sigma, \rho, f, \varsigma, \pi$ such that:

1. $\alpha \langle \sigma, \rho \rangle f \in \llbracket \mathit{\Theta} N \rrbracket^{\mathsf{c}}$
2. $\langle \rho, \varsigma \rangle \pi \in \llbracket \mathit{\Theta} M \rrbracket^{\mathsf{c}} \gg\!= f$
3. $\tau = \alpha \langle \sigma, \varsigma \rangle \pi$

By #2, there exist $\eta, \varrho, \theta, x, \mu, \omega$ such that:

4. $\eta \langle \varrho, \theta \rangle x \in \llbracket \mathit{\Theta} M \rrbracket^{\mathsf{c}}$
5. $\langle \theta, \mu \rangle \omega \in f x$
6. $\langle \rho, \varsigma \rangle \pi = \eta \langle \varrho, \mu \rangle \omega$

By #6, there exists β such that:

7. $\eta = \langle \rho, \varsigma \rangle \beta$
8. $\pi = \beta \langle \varrho, \mu \rangle \omega$

By #1, #4, & #7, using the induction hypothesis, we have:

$$\exists W \in \mathcal{V}\llparenthesis A \text{ -> } B \rrparenthesis.\ \mathit{\Theta} N \xrightarrow{\alpha\langle\sigma,\rho\rangle f} W$$

$$\exists V \in \mathcal{V}\llparenthesis A \rrparenthesis.\ \mathit{\Theta} M \xrightarrow{\langle\rho,\varsigma\rangle\beta\langle\varrho,\theta\rangle x} V$$

In particular, there exist $\mathtt{x}, K$ such that $W = \lambda\mathtt{x}.\, K$ and $K[V/\mathtt{x}] \in \mathcal{E}\llparenthesis B \rrparenthesis$. So:

$$\mathit{\Theta}\,(NM) \xrightarrow{\alpha\langle\sigma,\rho\rangle} (\lambda\mathtt{x}.\, K)\,(\mathit{\Theta} M) \xrightarrow{\langle\rho,\varsigma\rangle\beta\langle\varrho,\theta\rangle} (\lambda\mathtt{x}.\, K)\, V \xrightarrow{\langle\theta,\theta\rangle} K[V/\mathtt{x}]$$

Furthermore, we have $\llbracket \lambda\mathtt{x}.\, K \rrbracket^{\mathsf{v}} = f$ and $\llbracket V \rrbracket^{\mathsf{v}} = x$. By Lemma 1,

$$\llbracket K[V/\mathtt{x}] \rrbracket^{\mathsf{c}} = \llbracket K \rrbracket^{\mathsf{c}} \left[\mathtt{x} \mapsto \llbracket V \rrbracket^{\mathsf{v}}\right] = \llbracket \lambda\mathtt{x}.\, K \rrbracket^{\mathsf{v}} \llbracket V \rrbracket^{\mathsf{v}} = f x$$

By #5:

$$\exists U \in \mathcal{V}\llparenthesis B \rrparenthesis.\ K[V/\mathtt{x}] \xrightarrow{\langle\theta,\mu\rangle\omega} U$$

Combining the executions and using transitivity:

$$\exists U \in \mathcal{V}\llparenthesis B \rrparenthesis.\ \mathit{\Theta}\,(NM) \xrightarrow{\alpha\langle\sigma,\varsigma\rangle\beta\langle\varrho,\mu\rangle\omega} U$$

By #3 & #8, $\alpha \langle \sigma, \varsigma \rangle \beta \langle \varrho, \mu \rangle \omega = \tau$, so we are done.

$\mathit{\Gamma} \vdash M\mathbf{?} : \mathbf{Val}$**.** Let $\mathit{\Theta} \in \mathcal{G}\llparenthesis \mathit{\Gamma} \rrparenthesis$, and let $\tau \in \llbracket \mathit{\Theta}\,(M\mathbf{?}) \rrbracket^{\mathsf{c}} = \llbracket \mathit{\Theta} M \rrbracket^{\mathsf{c}} \gg\!= \lambda\ell.\ \tilde{\mathrm{L}}_{\ell}\left(v.\ \tilde{\mathrm{Y}}^{?}\tilde{v}\right)$. Therefore, there exist $\alpha, \sigma, \rho, \ell, \varsigma, \pi$ such that:

1. $\alpha \langle \sigma, \rho \rangle \ell \in \llbracket \mathit{\Theta} M \rrbracket^{\mathsf{c}}$
2. $\langle \rho, \varsigma \rangle \pi \in \tilde{\mathrm{L}}_{\ell}\left(v.\ \tilde{\mathrm{Y}}^{?}\tilde{v}\right)$
3. $\tau = \alpha \langle \sigma, \varsigma \rangle \pi$

By #2, $\langle \rho, \varsigma \rangle \pi \in \tilde{\mathrm{Y}}^{?}\tilde{\rho}_{\ell}$. In particular, $\langle \rho, \varsigma \rangle \pi$ is preserving. By #1, using the induction hypothesis, we have:

$$\exists V \in \mathcal{V}\llparenthesis \mathbf{Loc} \rrparenthesis.\ \mathit{\Theta} M \xrightarrow{\alpha\langle\sigma,\rho\rangle\ell} V$$

Since **Loc** is a ground type, $V = \ell$; and since $\langle \rho, \varsigma \rangle\, \pi$ is preserving:

$$\varTheta\,(M\mathbf{?}) \xrightarrow{\alpha\langle\sigma,\rho\rangle} \ell\mathbf{?} \xrightarrow{\langle\rho,\rho\rangle} \rho_\ell \xrightarrow{\langle\rho,\varsigma\rangle\pi} \rho_\ell$$

Combining the executions and using transitivity:

$$\varTheta\,(M\mathbf{?}) \xrightarrow{\alpha\langle\sigma,\varsigma\rangle\pi} \rho_\ell$$

Since $\rho_\ell \in \mathcal{V}\llparenthesis \mathbf{Val} \rrparenthesis$, by #3 we are done.

$\varGamma \vdash M \parallel N : \boldsymbol{(}A * B\boldsymbol{)}$. Let $\varTheta \in \mathcal{G}\llparenthesis \varGamma \rrparenthesis$, and let $\omega \in [\![\varTheta\,(M \parallel N)]\!]^{\mathsf{c}} = [\![\varTheta M]\!]^{\mathsf{c}} \mathrel{|||} [\![\varTheta N]\!]^{\mathsf{c}}$. So there exist $\tau \in [\![\varTheta M]\!]^{\mathsf{c}}$ and $\pi \in [\![\varTheta N]\!]^{\mathsf{c}}$ such that $\tau \parallel \pi \implies \omega$.

By the induction hypothesis it follows that:

$$\exists V \in \mathcal{V}\llparenthesis A \rrparenthesis.\ \varTheta M \xrightarrow{\tau} V$$
$$\exists W \in \mathcal{V}\llparenthesis B \rrparenthesis.\ \varTheta N \xrightarrow{\pi} W$$

By Lemma 4, $\varTheta\,(M \parallel N) \xrightarrow{\omega} \langle V, W \rangle \in \mathcal{V}\llparenthesis \boldsymbol{(}A * B\boldsymbol{)} \rrparenthesis$.

Other cases are similar.

Compositionality Proving the theorem directly by induction on the structure of the program context fails, in particular when attempting to handle the abstraction (λ) case. The condition in the theorem that the context be closed and ground is necessary, so this failure is expected. To surpass this hurdle we define a "contains up-to extensionality" logical relation.

We define functions $\mathcal{V}^{\circ}\llparenthesis - \rrparenthesis$ and $\mathcal{E}^{\circ}\llparenthesis - \rrparenthesis$ from types to binary relations by mutual recursion. Specifically, $\mathcal{V}^{\circ}\llparenthesis A \rrparenthesis$ is a relation on $[\![A]\!]$, and $\mathcal{E}^{\circ}\llparenthesis A \rrparenthesis$ on $\mathcal{T}\,[\![A]\!]$.

$$\begin{aligned}
\mathcal{V}^{\circ}\llparenthesis A \mathrel{\texttt{->}} B \rrparenthesis &\coloneqq \{\langle f, g\rangle \mid \forall\, \langle x, y\rangle \in \mathcal{V}^{\circ}\llparenthesis A \rrparenthesis.\ \langle f x, g y\rangle \in \mathcal{E}^{\circ}\llparenthesis B \rrparenthesis\} \\
\mathcal{V}^{\circ}\llparenthesis \boldsymbol{(}A_1 * \cdots * A_n\boldsymbol{)} \rrparenthesis &\coloneqq \{\langle\langle x_1, \ldots, x_n\rangle, \langle y_1, \ldots, y_n\rangle\rangle \mid \forall i.\ \langle x_i, y_i\rangle \in \mathcal{V}^{\circ}\llparenthesis A_i \rrparenthesis\} \\
\mathcal{V}^{\circ}\llparenthesis \{\iota_1 \ \mathbf{of}\ A_1 \mid \cdots \mid \iota_n \ \mathbf{of}\ A_n\} \rrparenthesis &\coloneqq \textstyle\bigcup_i \{\langle \iota_i\, x, \iota_i\, y\rangle \mid \langle x, y\rangle \in \mathcal{V}^{\circ}\llparenthesis A_i \rrparenthesis\}
\end{aligned}$$

$$\mathcal{E}^{\circ}\llparenthesis A \rrparenthesis \coloneqq \{\langle P, Q\rangle \mid \forall \alpha x \in P \exists \beta y \in Q.\ \alpha = \beta \wedge \langle x, y\rangle \in \mathcal{V}^{\circ}\llparenthesis A \rrparenthesis\}$$

For every typing context $\varGamma$ we define a binary relation $\mathcal{G}^{\circ}\llparenthesis \varGamma \rrparenthesis$ over $[\![\varGamma]\!]$:

$$\mathcal{G}^{\circ}\llparenthesis \varGamma \rrparenthesis \coloneqq \{\langle \gamma, \delta\rangle \mid \forall\, (\mathtt{x} : A) \in \varGamma.\ \langle \gamma \mathtt{x}, \delta \mathtt{x}\rangle \in \mathcal{V}^{\circ}\llparenthesis A \rrparenthesis\}$$

Define the judgment $\varGamma \vDash M \lesssim N : A$ as follows:

$$\forall\, \langle \gamma, \delta\rangle \in \mathcal{G}^{\circ}\llparenthesis \varGamma \rrparenthesis.\ \left\langle [\![M]\!]^{\mathsf{c}}\, \gamma, [\![N]\!]^{\mathsf{c}}\, \delta \right\rangle \in \mathcal{E}^{\circ}\llparenthesis A \rrparenthesis$$

Lemma 5. *$\vDash$ is closed under typing rules. That is, for $\varXi\,[-] : \varGamma \vdash A \to \varDelta \vdash B$, if $\varGamma \vDash M \lesssim N : A$ then $\varDelta \vDash \varXi\,[M] \lesssim \varXi\,[N] : B$*

Proof. By induction on the typing of $\varXi\,[\bullet]$.

Proposition 1. *$\mathit{\Gamma} \vDash M \lesssim M : A$ for every $M \in \mathit{\Gamma} \vdash A$.*

Proof. Use Lemma 5 with M itself as $\mathit{\Xi}\,[\bullet]$.

Proposition 2. *If $\mathcal{T}\,[\![A]\!] \ni P' \subseteq P$ and $\langle P, Q\rangle \in \mathcal{E}^{\circ}\llparenthesis A \rrparenthesis$ then $\langle P', Q\rangle \in \mathcal{E}^{\circ}\llparenthesis A \rrparenthesis$.*

Proof. Trivial, by quantification structure.

Lemma 6. *For $M, N \in \mathit{\Gamma} \vdash A$, if $[\![M]\!]^{\mathsf{c}} \subseteq [\![N]\!]^{\mathsf{c}}$ then $\mathit{\Gamma} \vDash M \lesssim N : A$.*

Proof. Let $\langle \gamma, \delta \rangle \in \mathcal{G}^{\circ}\llparenthesis \mathit{\Gamma} \rrparenthesis$. By assumption, $[\![M]\!]^{\mathsf{c}}\,\gamma \subseteq [\![N]\!]^{\mathsf{c}}\,\gamma$. By Proposition 1, $\left\langle [\![N]\!]^{\mathsf{c}}\,\gamma, [\![N]\!]^{\mathsf{c}}\,\delta \right\rangle \in \mathcal{E}^{\circ}\llparenthesis A \rrparenthesis$. Done by Proposition 2.

Remark 1. We could have gone through M first and used the containment with δ second, by first proving Proposition 2 symmetrically with $Q \subseteq Q'$.

Lemma 7. *$\mathcal{V}^{\circ}\llparenthesis G \rrparenthesis$ is equality for every ground G.*

Proof. By induction on G.

Proposition 3. *$\mathcal{E}^{\circ}\llparenthesis G \rrparenthesis$ is containment for ground G.*

Proof. By Lemma 7, $\alpha = \beta \wedge \langle x, y\rangle \in \mathcal{V}^{\circ}\llparenthesis G \rrparenthesis$ is equivalent to $\alpha x = \beta y$.

Proof (Theorem 4). Assume $[\![M]\!]^{\mathsf{c}} \subseteq [\![N]\!]^{\mathsf{c}}$. By Lemma 6, $\mathit{\Gamma} \vDash M \lesssim N : A$. By Lemma 5, $\vDash \mathit{\Xi}\,[M] \lesssim \mathit{\Xi}\,[N] : G$; i.e. $\left\langle [\![\mathit{\Xi}\,[M]]\!]^{\mathsf{c}}, [\![\mathit{\Xi}\,[N]]\!]^{\mathsf{c}} \right\rangle \in \mathcal{E}^{\circ}\llparenthesis G \rrparenthesis$. By Lemma 7, $[\![\mathit{\Xi}\,[M]]\!]^{\mathsf{c}} \subseteq [\![\mathit{\Xi}\,[N]]\!]^{\mathsf{c}}$.