A reactive operational semantics for a lambda-calculus with time warps

Adrien Guatto, Christine Tasson, Ada Vienot

A reactive operational semantics for a lambda-calculus with time warps

Adrien Guatto[1], Christine Tasson[2], and Ada Vienot[1]

[1] Université de Paris, CNRS, IRIF, Paris, 75006, France
guatto@irif.fr vienot@irif.fr
[2] Sorbonne Université, LIP6, Paris, 75005, France
christine.tasson@lip6.fr

This work is supported by the Emergence project "ReaLiSe" funded by the city of Paris.

Abstract

Many computer systems are *reactive*: they execute in continuous interaction with their environment. From the perspective of functional programming, reactive systems can be modeled and programmed as stream functions. Several lines of research have exploited this point of view. Works based on Nakano's guarded recursion use type-theoretical modalities to guarantee productivity. Works based on synchronous programming use more specialized methods but guarantee incremental processing of stream fragments.

In this paper, we contribute to a recent synthesis between the two approaches by describing how a language with a family of type-theoretical modalities can be given an incremental semantics in the synchronous style. We prove that this semantics coincides with a more usual big-step semantics.

1 Introduction

Reactive systems can be found in a variety of application domains, ranging from hard real-time and life-critical systems such as airplane control to more mundane areas like video games and graphical user interfaces. Their specificity as computer systems lies in their continuous interaction with an external environment.

At a low level of abstraction, reactive systems can be modeled as state machines. Yet, reasoning about state machines and, more generally, state transition systems requires beautiful but complex notions of bisimilarity. Furthermore, in practice, the dominant approach to the implementation of state machines is via a mix of callbacks and mutable state, which makes compositional reasoning delicate. These difficulties have motivated a lot of works in the imperative world, such as the well-known model-view-controller pattern.

An alternative approach to the programming of reactive systems is to use functional programming with infinite data structures. In this view, a reactive program is seen as a pure function from some infinite data structure describing incoming events to some other infinite data structure describing outgoing events. This idea can be traced back to the seminal work of Kahn [10], in which he observed how interacting parallel processes could be described as well-behaved functions between infinite sequences — or *streams*, for short. Thanks to this approach, all the classical tools from functional programming (e.g., higher-order functions) can be applied to reactive systems.

The idea of reactive programming with pure functions has been fertile, and has developed into several distinct lines of research. To introduce these works, and explain how the present paper fits into them, let us consider the example below, given in functional pseudocode.

```
sum : Stream Nat -> Stream Nat
sum xs = ys where rec ys = map2 (+) xs (0 :: ys)
```

This function takes a stream of integers and returns its running sum, that is, the ith element of the output stream `ys` is the sum of the first i elements of the input stream `xs`. Here `(::)` denotes the stream constructor and `map2 f xs ys` applies the function `f` to `xs` and `ys` pointwise.

Handling stream functions like `sum` in a programming language for reactive programming raises two interconnected questions.

1. What constructions and data types does the language offer, and under what conditions? For example, should one allow arbitrary higher-order functions? Arbitrary recursive definitions? Other infinite data types such as infinite trees or streams of streams?
2. How is the language implemented? A reactive program, being in continuous interaction with its environment, ought to be able to process streams in an incremental fashion. Moreover, this processing should use as little space and time as possible.

Different research lines have answered these questions differently, leading to distinct tradeoffs between expressiveness, safety, and efficiency.

A first answer is that of *synchronous functional programming* languages in the vein of Lustre [5], which favor efficiency and safety over expressiveness. In synchronous functional programming, base types are that of streams of scalars, computed along an infinite sequence of execution *ticks*. In exchange for this restrictive focus, synchronous languages benefit from specialized yet powerful type systems:

- *causality types* classify *productive* recursive definitions, e.g., rejecting a variant of `sum` without the initial 0 on the last argument to `map2`;
- *clock types* classify stream types according to its *clock*: the set of ticks at which a new stream element must appear.

These analyses serve as a foundation to the backends of synchronous compilers, which turn clocked stream functions into state machines realized as memory-frugal imperative code.

While well-suited to hard real-time and life-critical systems, synchronous programming can feel restrictive from the perspective of mainstream functional programming. Indeed, even an expressive higher-order synchronous language such as Lucid Synchrone [6, 16] does not support streams of streams, nor streams of arbitrary functions. This is to be contrasted with the situation in a second approach to reactive programming, *functional reactive programming*.

Functional reactive programming, as originally proposed by Elliott and Hudak [8], consists in realizing infinite data structures by way of laziness in a language such as Haskell. The expressiveness of Haskell makes it simple to write concise and elegant reactive code, making it especially relevant for prototyping. Unfortunately, using the full power of a lazy language and its facilities for general recursion comes at a cost. This cost is the ease with which one can inadvertently write a non productive definition or introduce unbounded memory usage [13].

To be used in applications with performance or safety needs, functional reactive programming has to be tamed. This can be achieved in two ways. One possibility is to implement a restrictive domain-specific language for reactive programming as a Haskell library. From this perspective, Haskell serves as a powerful macro language for the embedded language. This is the approach used in, e.g., *causal commutative arrows* [13].

$$\textsc{GuardedRec}\quad \frac{\Gamma, x : \triangleright A \vdash M : A}{\Gamma \vdash \mathsf{rec}\, x.M : A}$$

Figure 1 Guarded recursion

Another choice is to drop Haskell and develop a standalone domain-specific language. This approach has become increasingly popular since the seminal work of Krishnaswami and Benton [12]. They remarked that a certain unary connective (or *modality*) studied in foundational works on modal logic [15] and denotational

semantics [1, 4] can be used to give a type-theoretical criterion for productive recursion. This criterion consists in Figure 1. The type $\triangleright A$ (pronounced "later A") is a slightly weaker version of A that does not give any information at the beginning of execution. This *guarded* recursion rule, by requiring the body of the definition to have type $\triangleright A \to A$, makes sure that the definition actually extends `x` before using it. In particular, computing the fixed point of the identity function is no longer permitted. This simple idea has led to many developments, including memory-conscious implementations [11, 2].

Interestingly, the introduction of the later modality in functional reactive programming brings it closer to synchronous concepts. Indeed, the type $\triangleright A$ should be thought of as "A but without the data available at the first execution tick." For example, the type $\triangleright \mathsf{Stream}\ \mathbb{N}$ is that of streams that start growing at the second tick rather than the first one. Thus, the later modality introduces an idea of global discrete time akin to that found in synchronous programming. Moreover, its effect on types is tantalizingly similar to that of the clocks used in synchronous programming. In light of these similarities, the first author built a unifying calculus, termed λ^*, that bridges synchronous programming and functional reactive programming [9].

In λ^*, the later modality arises as a special case of a more general *warping modality*. The warping modality $p \Rrightarrow -$ acts upon a type as prescribed by the *time warp* denoted p. A time warp is, briefly speaking, a well-behaved map from the discrete time scale to itself. The type $p \Rrightarrow A$ should be thought of as a variant of A suitably stretched or compressed so that, at tick n, one can observe from it what can be observed from A at tick $p(n)$. The calculus can be thought of as connecting two lines of research in the sense that all synchronous clocks are time warps and all modalities used in guarded recursion can be described as time warps. Moreover, many time warps where hitherto unavailable as type connectives.

In addition to a type system, λ^* comes equipped with an execution mechanism expressed as an operational semantics. It can be seen as an evaluator parameterized with a quantity of fuel. It is a theorem of λ^* that a closed term of type $\mathsf{Stream}\ \mathbb{B}$, when evaluated with n units of fuel, converges to a list of n booleans. This list is a prefix of the ideal conceptually-infinite stream computed by the program. While this is satisfactory from a metatheoretical perspective, this process leaves much to be desired in the reactive setting. The evaluation process is not incremental: the evaluation process with $n+1$ units of fuel cannot reuse any of the computations that arose while computing the shorter prefixes.

Contributions. In this paper, we continue to bridge the gap between synchronous functional programming and functional reactive programming by introducing an incremental semantics for λ^*. Taking inspiration from compilation techniques introduced for Lucid Synchrone [7, 16], we introduce a notion of typed program *states* that store results to be reused, and replace prefix values with collections of instantaneous deltas that we call *increments*. We prove by a logical-relation argument that our incremental semantics is sound: glueing together the consecutive increments recovers the prefix computed by the reference semantics.

In Section 2, we introduce the first author's λ^* calculus, and our modifications of its type system needed to type-check program states. We then describe our incremental semantics in depth in Section 3, including its metatheory and the adequacy theorem for our logical relation. Finally, in Section 4 we discuss our results and their relation to previous works.

Notations. We write ω for the first infinite ordinal and $\omega+1$ for its successor ordinal $\omega \cup \{\omega\}$. Addition and subtraction extend to $\omega+1$ by setting $n+\omega=\omega$ and $n-\omega=0$ for all $n \in \omega+1$. The *domain* of a partial map $f : X \rightharpoonup Y$, denoted $\mathrm{dom}(f)$, is defined as $f^{-1}(Y)$. A *finite map* is a partial map of finite domain. Given a finite map f, the notation $f(x)$ implies $x \in \mathrm{dom}(f)$. We write $f, x : y$ or $f[y/x]$ for the finite map sending x to y and $x' \in \mathrm{dom}(f) \setminus \{x\}$ to $f(x')$.

2 The λ^* calculus

In order to save space, we leave aside the treatment of product and sum types in the short version of this paper. A complete treatment of λ^* can be found in the full version of the paper, available at http://hal.archives-ouvertes.fr/hal-03465519.

2.1 Type system

Time warps and terms. In reactive languages, types are implicitly indexed by logical time steps. In λ^*, this indexing can be changed through the use of the warping modality $p \Rrightarrow -$. To explain how this modality works, we must first define what a time warp p exactly is.

Definition 2.1. A *time warp* is a suprema-preserving map $p : \omega + 1 \to \omega + 1$.

Equivalently, a time warp is a monotonic map from $\omega + 1$ to itself such that $p(0) = 0$ and $p(\omega) = \max_{i<\omega} p(i)$. Time warps are denoted p, q, r. The time warps **id** and **lat** respectively correspond to the maps $n \mapsto n$ and $n \mapsto n - 1$.

Time warps appear in the terms of λ^* as well as in its types: they decorate the term constructors $\mathsf{delay}^{q \leqslant p}(M)$ and $M \,\mathsf{by}\, p$. The other terms of the language are those of a simply-typed λ-calculus with streams, constants c belonging to some fixed set C, and recursive definitions.

$$M, N ::= x \mid c \mid \lambda x^A.\, M \mid MN \mid \mathsf{rec}\, x^A.\, M \mid M :: N \mid \mathsf{hd}\, M \mid \mathsf{tl}\, M \mid M \,\mathsf{by}\, p \mid \mathsf{delay}^{q \leqslant p}(M)$$

Types. The types of λ^* are those of the simply-typed λ-calculus with a fixed set of ground types G (whose elements are denoted ν) and streams, as well as the warping modality $p \Rrightarrow (-)$.[1]

$$A, B ::= \nu \mid \mathsf{Str}\, A \mid A \overset{S}{\Rightarrow} B \mid p \Rrightarrow A$$

The label S annotating the arrow type $A \overset{S}{\Rightarrow} B$ is new to the present paper. It is linked to the proposed incremental semantics. We explain the purpose this *state type* S serves in Section 3.

Informally, the values of type $p \Rrightarrow A$ at tick n are those of A at tick $p(n)$. Another possible way to think of $p \Rrightarrow A$ is as a type where time passes "p-times" faster than in type A. For example, a value of type $p \Rrightarrow \mathsf{Str}\, \mathsf{Nat}$ at tick n is a list of $p(n)$ numbers.

We assume that for every ground type ν there exists a set $C_\nu \subseteq C$ such that every constant $c \in C$ belongs to exactly one such C_ν.

Typing contexts. Typing contexts are finite maps from identifiers to types, and are denoted Γ, Δ. We write $p \Rrightarrow \Gamma$ for the typing context sending $x \in \mathrm{dom}(\Gamma)$ to $p \Rightarrow \Gamma(x)$.

Typing judgements. The typing judgment $\Gamma \vdash M : A \mid S$ states that if the free variables of M respect the types prescribed by Γ then M computes a result of type A using a state of type S. Its rules are given in Figure 2. Most of them are those of the simply-typed λ-calculus, thus we focus our explanation on the rules specific to temporal constructs. The state-type aspect is new and we discuss it in the next section.

In λ^*, the type $\mathsf{Str}\, A$ classifies streams that unfold at the rate of one element per step. This is reflected by the rules Head, Tail, and Cons, which state that the head of a stream is available now while its tail is only available at type $\mathbf{lat} \Rrightarrow \mathsf{Str}\, A$. This type describes streams which start to grow strictly after the first tick. Thus, the ith element of a stream of type $\mathsf{Str}\, A$ is only available at tick i.

[1] The warping modality was denoted $p * (-)$ in the original λ^* paper [9].

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

$$\textsc{Var}\ \frac{}{\Gamma, x{:}A \vdash x : A \mid \mathbb{1}} \qquad \textsc{Scalar}\ \frac{c \in C_\nu}{\Gamma \vdash c : \nu \mid \mathbb{1}} \qquad \textsc{Lam}\ \frac{\Gamma, x{:}A \vdash M : B \mid S}{\Gamma \vdash \lambda x^A.\, M : A \overset{S}{\Rightarrow} B \mid \mathbb{1}}$$

$$\textsc{App}\ \frac{\Gamma \vdash M : A \overset{S''}{\Longrightarrow} B \mid S \qquad \Gamma \vdash N : A \mid S'}{\Gamma \vdash MN : B \mid S \times S' \times S''} \qquad \textsc{Head}\ \frac{\Gamma \vdash M : \mathsf{Str}\, A \mid S}{\Gamma \vdash \mathsf{hd}\, M : A \mid S} \qquad \textsc{Tail}\ \frac{\Gamma \vdash M : \mathsf{Str}\, A \mid S}{\Gamma \vdash \mathsf{tl}\, M : \mathbf{lat} \Rightarrow \mathsf{Str}\, A \mid S}$$

$$\textsc{Cons}\ \frac{\Gamma \vdash M : A \mid S \qquad \Gamma \vdash N : \mathbf{lat} \Rightarrow \mathsf{Str}\, A \mid S'}{\Gamma \vdash M :: N : \mathsf{Str}\, A \mid S \times S'} \qquad \textsc{Rec}\ \frac{\Gamma, x{:}\mathbf{lat} \Rightarrow A \vdash M : A \mid S}{\Gamma \vdash \mathsf{rec}\, x^A.\, M : A \mid S \times \int A} \qquad \textsc{By}\ \frac{\Gamma \vdash M : A \mid S}{p \Rightarrow \Gamma \vdash M\ \mathsf{by}\ p : p \Rightarrow A \mid p \Rightarrow S}$$

$$\textsc{Delay}\ \frac{q \leqslant p \qquad \Gamma \vdash M : p \Rightarrow A \mid S}{\Gamma \vdash \mathsf{delay}^{q \leqslant p}(M) : q \Rightarrow A \mid S \times \int (p \Rightarrow A)}$$

Figure 2 Typing judgement for λ^*

Rule Rec is that of guarded recursive definitions. It mirrors the rule GuardedRec described in the introduction, but expresses the later modality $\triangleright$ as a special application of the warping modality with **lat** as time warp. This rule enforces productivity by forbidding instantaneous dependencies in recursive definitions.

The by construction (Rule By) is used to introduce and eliminate the warp modality. It states that if a program M produces an output at rate A by consuming inputs at rate Γ, then one can always speed M up to produce an output at rate $p \Rightarrow A$, at the cost of consuming inputs at rate $p \Rightarrow \Gamma$, for any p.

If $q(n) \leqslant p(n)$ then the values of type $p \Rightarrow A$ at the nth tick can be *truncated* into values of type $q \Rightarrow A$ at the nth tick. Rule Delay generalizes this reasoning to all ticks: the delay construction turns results of type $p \Rightarrow A$ into results of type $q \Rightarrow A$ as long as $q \leqslant p$, where $\leqslant$ denotes the pointwise ordering. We describe the operational content of truncation in the next section.

2.2 Prefix semantics

The terms of λ^* manipulate ideal, infinite data structures, but they have to execute on a concrete machine that only deals with finite syntax. For this reason, the reference semantics of the language computes finite yet arbitrarily precise approximants, or *prefixes*, of streams and other infinite data structures. The length of these prefixes is controlled by a parameter to the evaluation judgment, which can be thought of alternatively as the current tick or as a quantity of fuel (as in step-indexing [1]). Using this parameter, one may evaluate, e.g., a program computing the stream of natural numbers at tick 1 and obtain the prefix $0 :: \mathbf{stop}$, or evaluate at tick 2 and obtain the prefix $0 :: 1 :: \mathbf{stop}$, and so on. For this reason, we call this evaluation semantics the *prefix semantics*.

Prefix values and environments. Prefix values, denoted V, should be thought of as prefixes of a certain length of an infinite object of a certain type. A prefix of length n is the result of a computation at tick n. Figure 3 describes the typing rules of the judgment $V : A \mathfrak{C} n$, which states that V is a prefix of length n of an object of type A.

The special value **stop** is a prefix of every value at 0. Prefixes at ω should be infinite but

$$\boxed{V : A \mathbin{@} n} \qquad \textsc{VStop}\ \frac{}{\mathbf{stop} : A \mathbin{@} 0} \qquad \textsc{VBox}\ \frac{\Gamma \vdash M : A \mid S \qquad E : \Gamma \mathbin{@} \omega}{\mathbf{box}(M)\{E\} : A \mathbin{@} \omega} \qquad \textsc{VScalar}\ \frac{c \in C_\nu}{c : \nu \mathbin{@} n+1}$$

$$\textsc{VWarp}\ \frac{V : A \mathbin{@} p(n+1)}{\mathbf{w}\,(p, V) : p \Rightarrow A \mathbin{@} n+1} \qquad \textsc{VClo}\ \frac{\Gamma, x{:}A \vdash M : B \mid S \qquad E : \Gamma \mathbin{@} n+1}{(x.M)\{E\} : A \overset{S}{\Rightarrow} B \mathbin{@} n+1} \qquad \textsc{VStream}\ \frac{V : A \mathbin{@} n+1 \qquad V' : \mathsf{Str}\,A \mathbin{@} n}{V :: V' : \mathsf{Str}\,A \mathbin{@} n+1}$$

$$\boxed{E : \Gamma \mathbin{@} n} \qquad \textsc{VEnv}\ \frac{\mathrm{dom}(\Gamma) = \mathrm{dom}(E) \qquad \forall x \in \mathrm{dom}(\Gamma).\ E(x) : \Gamma(x) \mathbin{@} n}{E : \Gamma \mathbin{@} n}$$

Figure 3 Typing judgement for prefixes

are modeled as suspended computations (*thunks*). All other rules apply for finite positive n. Warped values of type $p \Rightarrow A$ at n are prefixes of length $p(n)$ of A. Functional values are closures. Values $V :: V'$ of type $\mathsf{Str}\,A$ are formed of a head value V at n and of a tail value V' at $n-1$.

Environments, denoted E, are partial maps from identifiers to values. We write $\mathbf{w}\,(p, E)$ for the environment E' sending $x \in \mathrm{dom}(E)$ to $\mathbf{w}\,(p, E(x))$.

$$\boxed{\lfloor V \rfloor_n} \qquad \lfloor \mathbf{w}\,(p, V) \rfloor_{n+1} = \mathbf{w}\left(p, \lfloor V \rfloor_{p(n+1)}\right)$$

$$\lfloor V \rfloor_0 = \mathbf{stop} \qquad \lfloor (x.M)\{E\} \rfloor_{n+1} = (x.M)\{\lfloor E \rfloor_{n+1}\}$$

$$\lfloor V \rfloor_\omega = V \qquad \lfloor V_1 :: V_2 \rfloor_{n+1} = \lfloor V_1 \rfloor_{n+1} :: \lfloor V_2 \rfloor_n$$

$$\lfloor c \rfloor_{n+1} = c \qquad \lfloor \mathbf{box}(M)\{E\} \rfloor_{n+1} = V \text{ if } M\,;\lfloor E \rfloor_{n+1} \Downarrow_{n+1} V$$

Figure 4 Truncation of prefixes

As mentioned before, prefixes can be truncated. Given a prefix V of length m and $n \leq m$, one may obtain a shorter prefix $\lfloor V \rfloor_n$ as defined in Figure 4. This extends to environments pointwise. Since the evaluation judgment is deterministic [9], $\lfloor - \rfloor_n$ defines a partial (because of the **box** case) function rather than a relation.

The special value **stop** is a 0-length prefix of every other value. Truncating at ω does nothing. Truncating a closure means truncating its environment. Truncating a stream prefix of length $n > 0$ involves truncating its tail at length $n-1$; for example $\lfloor 1 :: 2 :: \mathbf{stop} \rfloor_1 = 1 :: \mathbf{stop}$. To truncate a thunk one has to evaluate it.

In addition to being total on well-typed terms, truncation preserves typing [9].

Theorem 2.1. *If $V : A \mathbin{@} n$, then for all $m \leqslant n$, we have $\lfloor V \rfloor_m : A \mathbin{@} m$.*

Evaluation judgement. The judgment given in Figure 5 is mostly that of a big-step semantics for call-by-value λ-calculus with environments and closures, extended to depend on a time step n.

We explain the rules that have a time-specific aspect. Rule PZero states that a program evaluated at time 0 always returns the same initial value named **stop**. Rule POmega freezes both the code and the environment in a thunk. When an actual value is required, the program can then be evaluated as needed to any finite length. Rule PBy formalizes the idea that type $p \Rightarrow A$ is composed of p-long prefixes of type A. Rule PRec is reminiscent of the Kleene fixed point theorem

as, in essence, it computes the value of $\mathsf{rec}\, x.\, M$ at n as the iterated application $(\lambda x.\, M)^n(\mathbf{stop})$. This process is formalized by the *iteration* judgment $M\,;E\,;x\,;V \Uparrow_m^n V'$, whose iterative behavior is captured by the lemma below.

Lemma 2.1. *If $M\,;E\,;x\,;V \Uparrow_i^k V'$ then for all j such that $i \leqslant j \leqslant k$, there is V'' such that $M\,;\lfloor E\rfloor_j\,;x\,;V \Uparrow_i^j V''$ and $M\,;E\,;x\,;V'' \Uparrow_j^k V'$.*

$$\boxed{M\,;E \Downarrow_n V}$$

$$\textsc{PZero}\ \frac{}{M\,;E \Downarrow_0 \mathbf{stop}} \qquad \textsc{POmega}\ \frac{}{M\,;E \Downarrow_\omega \mathbf{box}(M)\{E\}} \qquad \textsc{PScalar}\ \frac{}{c\,;E \Downarrow_{n+1} c} \qquad \textsc{PVar}\ \frac{x \in \mathrm{dom}(E)}{x\,;E \Downarrow_{n+1} E(x)}$$

$$\textsc{PLam}\ \frac{}{\lambda x^A.\, M\,;E \Downarrow_{n+1} (x.M)\{E\}} \qquad \textsc{PApp}\ \frac{M\,;E \Downarrow_{n+1} (x.P)\{E'\} \qquad N\,;E \Downarrow_{n+1} V \qquad P\,;E'[V/x] \Downarrow_{n+1} V'}{MN\,;E \Downarrow_{n+1} V'}$$

$$\textsc{PHead}\ \frac{M\,;E \Downarrow_{n+1} V :: V'}{\mathsf{hd}\, M\,;E \Downarrow_{n+1} V} \qquad \textsc{PTail}\ \frac{M\,;E \Downarrow_{n+1} V :: V'}{\mathsf{tl}\, M\,;E \Downarrow_{n+1} \mathbf{w}\,(\mathbf{lat}, V')} \qquad \textsc{PCons}\ \frac{M\,;E \Downarrow_{n+1} V \qquad N\,;E \Downarrow_{n+1} \mathbf{w}\,(\mathbf{lat}, V')}{M :: N\,;E \Downarrow_{n+1} V :: V'}$$

$$\textsc{PBy}\ \frac{M\,;E \Downarrow_{p(n+1)} V}{M\ \mathsf{by}\, p\,;\mathbf{w}\,(p,E) \Downarrow_{n+1} \mathbf{w}\,(p,V)} \qquad \textsc{PRec}\ \frac{M\,;E\,;x\,;\mathbf{stop} \Uparrow_0^{n+1} V}{\mathsf{rec}\, x^A.\, M\,;E \Downarrow_{n+1} V} \qquad \textsc{PDelay}\ \frac{M\,;E \Downarrow_{n+1} \mathbf{w}\,(p,V)}{\mathsf{delay}^{q\leqslant p}(M)\,;E \Downarrow_{n+1} \mathbf{w}\left(q, \lfloor V\rfloor_{q(n+1)}\right)}$$

$$\boxed{M\,;E\,;x\,;V \Uparrow_m^n V'}$$

$$\textsc{PIFinish}\ \frac{}{M\,;E\,;x\,;V \Uparrow_n^n V} \qquad \textsc{PIIter}\ \frac{m < n \qquad M\,;E\,;x\,;V' \Uparrow_{m+1}^n V'' \qquad M\,;E[\mathbf{w}\,(\mathbf{lat}, V)/x] \Downarrow_{m+1} V'}{M\,;E\,;x\,;V \Uparrow_m^n V''}$$

Figure 5 Prefix evaluation judgement

As an illustration on how the prefix evaluation works, consider the following program defining the stream of natural numbers. The function $\mathsf{incr} : \mathsf{Str}\,\mathsf{Int} \overset{\mathbb{1}}{\Rightarrow} \mathsf{Str}\,\mathsf{Int}$ adds one to all of the constituents of a stream of integers. For the example to be well-typed, we need to use the function $\mathrm{shift}_A : A \overset{\mathbb{1}}{\Rightarrow} (\mathbf{lat} \Rightarrow A)$, which acts as a causality-preserving type coercion.[2]

$$\mathsf{let}\ \mathrm{incr}' = \mathrm{shift}\ \mathrm{incr}\ \mathsf{in}\ \mathsf{let}\ \mathsf{rec}\ \mathrm{nat} = 0 :: (\mathrm{incr}'\,\mathrm{nat}\ \mathsf{by}\ \mathbf{lat})$$

The table below gives the first prefixes produced by evaluating the program nat.

n	1	2	3
nat	$0 :: \mathbf{stop}$	$0 :: 1 :: \mathbf{stop}$	$0 :: 1 :: 2 :: \mathbf{stop}$

The program nat is recursive. The fixed point rule evaluates the body of the rec block by recomputing the value obtained at the previous tick, which is incremented by calling incr'. At the first step, this yields **stop**. For the recursion to be guarded, the by **lat** wraps it under a special constructor to indicate its provenance: this yields $\mathbf{w}\,(\mathbf{lat}, \mathbf{stop})$. Finally, 0 is concatenated to it, yielding $0 :: \mathbf{stop}$. Next step, the process is repeated: we obtain $0 :: \mathbf{stop}$ again at the first tick. We increment it using incr', which yields $1 :: \mathbf{stop}$, and wrap it under a constructor, turning it into $\mathbf{w}\,(\mathbf{lat}, 1 :: \mathbf{stop})$. Concatenating 0 to it yields $0 :: 1 :: \mathbf{stop}$, and so on.

[2] For brevity's sake, we do not make explicit how this function works. An explanation can be found in Guatto's original paper[9]. We discuss this choice in more detail in Section 4.1.

Metatheory. We state the correctness theorems of [9] that will be important to the rest of the paper. Combined, they express that the evaluation of well-typed terms defines a total function, and that this function is monotonic with respect to truncation.

Theorem 2.2 (Subject reduction)**.** *If* $\Gamma \vdash M : A \mid S$ *and* $M\,;E \Downarrow_n V$ *with* $E : \Gamma \mathbin{@} n$*, then* $V : A \mathbin{@} n$*.*

Theorem 2.3 (Totality)**.** *If* $\Gamma \vdash M : A \mid S$ *and* $E : \Gamma \mathbin{@} n$*, then there exists* V *such that* $M\,;E \Downarrow_n V$*.*

Theorem 2.4 (Determinism)**.** *If* $M\,;E \Downarrow_n V_1$ *and* $M\,;E \Downarrow_n V_2$*, then* $V_1 = V_2$*.*

Theorem 2.5 (Monotonicity)**.** *If* $m \leqslant n$ *and* $M\,;E \Downarrow_n V$ *then* $M\,;\lfloor E \rfloor_m \Downarrow_m \lfloor V \rfloor_m$*.*

3 Reactive semantics for λ^*

In this section, we present a new semantics for the λ^* calculus, quite different from the prefix one. In this semantics, program execution consists in a sequence of reaction steps, each of which consumes a slice of the inputs, produces a slice of the outputs, and updates some internal state. To define this semantics, we first describe states and their types, as well as *increments*, a new sort of values describing these "slices." We show in Section 3.4 that this semantics computes the same results as the prefix semantics, in the appropriate sense.

3.1 Increments and states

Increments. Increments, denoted δ, are syntactic objects representing the information gained by running the program for one step. For example, each time a program of type $\mathsf{Str}\,\mathsf{Nat}$ does a step, it produces a number. The syntax of increments is defined by the following grammar.

$$\delta, \delta' ::= \mathbf{nil} \mid c \mid (x.M)\{\gamma\} \mid \delta :: \delta' \mid \triangleright(\delta_i)_i$$

The null increment is denoted **nil**, c denotes a constant, $(x.M)\{\gamma\}$ denotes an incremental closure, $\delta :: \delta'$ denotes a temporal sequence of increments and $\triangleright(\delta_i)_{i<n}$ denotes a warped family of increments.

Incremental environments, denoted γ, are finite maps from identifiers to increments. If the context permits no ambiguity, they will also be referred to simply as "environments" for the sake of brevity. By abuse of notation, if $(\gamma_i)_{i<n}$ is a family of environments with a common domain D, we write $\triangleright(\gamma_i)_{i<n}$ for the environment sending $x \in D$ to $\triangleright(\gamma_i(x))_{i<n}$.

Families of increments and typing. Another intuition behind increments is that they represent one-step-wide slices of prefixes. A collection of n increments then represents a prefix of length n. Such collections are represented by possibly infinite families of increments $(\delta_i)_{i<n}$. The empty family is written ε. The concatenation of two families of increments $(\delta_i)_{i<n}$ and $(\delta'_i)_{i<m}$, denoted $(\delta_i)_{i<n} \oplus (\delta'_i)_{i<m}$, is the family $(\delta''_i)_{i<n+m}$ such that δ''_i is either δ_i when $i < n$ or δ'_{i-m} when $m \leqslant i < m + n$.

As families of increments are intended to represent prefixes, they are typed according to the judgement rules given in Figure 6. We explain three interesting rules.

Rule ISCALAR states that scalars do not evolve as time passes. They yield all of their informational content at the very first tick, and they bring no additional information during subsequent ticks. Thus, families of increments of a scalar type comprise an initial scalar value followed by the null increment **nil** forever after.

$$\boxed{(\delta_i)_{i<n} : A \mathbin{@} n} \qquad \textsc{IStop}\ \frac{}{\varepsilon : A \mathbin{@} 0} \qquad \textsc{IScalar}\ \frac{c \in C_\nu}{(c) \oplus (\mathbf{nil})_{i<n} : \nu \mathbin{@} n+1}$$

$$\textsc{IWarp}\ \frac{(\delta_i)_{i<p(n+1)} : A \mathbin{@} p(n+1)}{\left(\triangleright\, (\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n+1} : p \Rrightarrow A \mathbin{@} n+1} \qquad \textsc{IClo}\ \frac{\Gamma,\, x{:}A \vdash M : B \mid S \qquad (\gamma_i)_{i<n+1} : \Gamma \mathbin{@} n+1}{\left((x.M)\{\gamma_i\}\right)_{i<n+1} : A \overset{S}{\Rightarrow} B \mathbin{@} n+1}$$

$$\textsc{IStream}\ \frac{(\delta_i)_{i<n+1} : A \mathbin{@} n+1 \qquad (\delta'_i)_{i<n+1} : \mathbf{lat} \Rrightarrow \mathsf{Str}\, A \mathbin{@} n+1}{(\delta_i :: \delta'_i)_{i<n+1} : \mathsf{Str}\, A \mathbin{@} n+1} \qquad \textsc{IBox}\ \frac{\forall m \in \omega.\ (\delta_i)_{i<m} : A \mathbin{@} m}{(\delta_i)_{i<\omega} : A \mathbin{@} \omega}$$

$$\boxed{(\gamma_i)_{i<n} : \Gamma \mathbin{@} n} \qquad \textsc{IEnv}\ \frac{\forall i < n.\ \ \mathrm{dom}(\Gamma) = \mathrm{dom}(\gamma_i) \qquad \forall x \in \mathrm{dom}(\Gamma).\ (\gamma_i(x))_{i<n} : \Gamma(x) \mathbin{@} n}{(\gamma_i)_{i<n} : \Gamma \mathbin{@} n}$$

Figure 6 Typing judgement for λ^* increments

Rule IWarp expresses that flattening the n lists under the $\triangleright$ constructor yields a family of size $p(n)$. Recall that a prefix of $p \Rrightarrow A$ at time n is a prefix of A at time $p(n)$. Thus, while one step from time $n-1$ to n is performed on the outside of the program, $\partial p(n) = p(n) - p(n-1)$ steps are performed under the by during this time. Since increments represent the informational content of the program during one step, an increment of $p \Rrightarrow A$ at time n is a family of increments of size $\partial p(n)$, encompassing steps from $p(n-1)$ to $p(n) - 1$.

Finally, rule IBox is the incremental counterpart to rule VBox. The main difference is that we no longer represent infinite prefixes as thunks (as in the prefix semantics) but as infinite families of increments. For this reason, truncation in the incremental semantics consists in restricting a (possibly infinite) family of increments to one of its initial segments. This is simpler than in the prefix semantics, where truncating a thunk actually consists in evaluating its body. However, this means that a program may return an infinite family of increments in finite time, e.g., as the result of M by p at the ith tick for p such that $p(i) = \omega$. We discuss the consequences of this choice in Section 4.

We now prove a counterpart to Theorem 2.1 by induction over typing derivations. A well-typed family of increments can be truncated and preserve its typing, albeit at a shorter length.

Theorem 3.1 (Type safety of increment truncation)**.** *If* $(\delta_i)_{i<n} : A \mathbin{@} n$ *and* $m \leqslant n$ *then* $(\delta_i)_{i<m} : A \mathbin{@} m$.

Notice that the typing judgement is only applicable on family of increments, and not increments alone. As an illustration, let $x{:}A \vdash M : B \mid S$ and $x{:}A \vdash N : B \mid S$ be two different well-typed terms. Now, consider the family of increments $\left((x.M)\{\}, (x.N)\{\}\right)$ with M and N unrelated terms of the same type. It maps to no prefix of type $A \overset{S}{\Rightarrow} B$, since the code inside the two closures differs. If we could somehow type increments on their own, there would be families of increments that are well-typed but do not correspond to a prefix. This is undesirable, as we want well-typed families of increments to map one-to-one to prefixes.

States and their types. States, denoted s, are syntactic objects. They are classified by a special category of types, *state types*, denoted S, according to the rules given in Figure 7.

$$s, s' ::= \mathbf{stop} \mid \mathbf{done} \mid \mathbf{unit} \mid \mathbf{w}\,(p, s) \mid (s, s') \mid \mathbf{inc}\,((\delta_i)_i)$$
$$S, S' ::= \mathbb{1} \mid p \Rrightarrow S \mid S \times S' \mid \textstyle\int A$$

These rules are similar to the ones used for typing prefixes (cf. Figure 3), except for SDone and SIncr. Rule SDone reflects the fact that a program that has performed ω reactions no longer needs to store any relevant state. We will develop this point shortly. Rule SIncr introduces the type constructor $\int$, which embeds families of increments into states. As such, $\int A$ denotes a "buffer" able to store values of type A — this allows us to store previously computed values into our states for further computations. Notice that it is the only interesting base state type, as the only other one is $\mathbb{1}$, and denotes a stateless computation.

$$\boxed{s : S \,\mathfrak{G}\, n}$$

$$\textsc{SStop}\ \frac{}{\mathbf{stop} : S \,\mathfrak{G}\, 0} \qquad \textsc{SDone}\ \frac{}{\mathbf{done} : S \,\mathfrak{G}\, \omega} \qquad \textsc{SUnit}\ \frac{}{\mathbf{unit} : \mathbb{1} \,\mathfrak{G}\, n+1}$$

$$\textsc{SWarp}\ \frac{s : S \,\mathfrak{G}\, p(n+1)}{\mathbf{w}\,(p, s) : p \Rrightarrow S \,\mathfrak{G}\, n+1} \qquad \textsc{SPair}\ \frac{s_1 : S_1 \,\mathfrak{G}\, n+1 \qquad s_2 : S_2 \,\mathfrak{G}\, n+1}{(s_1, s_2) : S_1 \times S_2 \,\mathfrak{G}\, n+1}$$

$$\textsc{SIncr}\ \frac{(\delta_i)_{i<n+1} : A \mathbin{@} n+1}{\mathbf{inc}\,((\delta_i)_{i<n+1}) : \textstyle\int A \,\mathfrak{G}\, n+1}$$

Figure 7 Typing judgement for states

We can now explain the state-type component S of the typing judgment $\Gamma \vdash M : A \mid S$ that we deliberately ignored in Section 2.1. This state type describes the state that our incremental semantics associates to M. It is handled in the same way as effect annotations in type-and-effect disciplines [14]. In particular, App expresses that an application requires three substates: one for the function, one for the argument, and one for the body of the function. Most other rules follow simpler patterns, with the states from subterms becoming substates of the parent term. Only the rules dealing with the warping modality require further component.

Rule By asserts that the state of $p\,\mathsf{by}\,M$ is just the state of M at rhythm p. This is because running a term under a by is akin to running it under an independant local clock p — thus, its state also grows at this rhythm.

We have seen how the prefix semantics mimicks Kleene's iteration sequence, approximating the fixed point of M at n by $(\lambda x.\, M)^n(\mathbf{stop})$, so to speak. In the incremental semantics, at the $(n+1)$th tick, rather than recompute $n+1$ applications of M we apply M once to the previously-computed result. In order to be able to do so, we include a copy of the output of M in the state of $\mathsf{rec}\,x.\, M$, as expressed by Rec. This behavior is a key ingredient of the incremental semantics.

Rule Delay specifies a nontrivial state type due to our wish to incrementalize the computations. To understand the specified state type, imagine that we receive incremental results from some term M at rythm p, and wish to produce incremental results at rythm q. If q is slow enough with respect to p, we may have to output *now* increments received *previously*. Hence, we choose to store the output of M in the state of $\mathsf{delay}^{q \leqslant p}(M)$. (We discuss how much buffering is needed in Section 4.)

3.2 Evaluation judgements

Discrete derivatives. We are now interested in how data changes from one tick to the next. Thus we define the *discrete derivative* of a time warp p, denoted ∂p, as $n \mapsto p(n) - p(n-1)$.

$$\boxed{M \parallel s\,;\gamma \Longrightarrow s'\,;\delta}$$

$$\textsc{EScalar}\ \frac{}{c \parallel \mathbf{unit}\,;\gamma \Longrightarrow \mathbf{unit}\,;\mathbf{nil}} \qquad \textsc{EScalarI}\ \frac{c \in C_\nu}{c \parallel \mathbf{stop}\,;\gamma \Longrightarrow \mathbf{unit}\,;c}$$

$$\textsc{EVar}\ \frac{x \in \mathrm{dom}(\gamma)}{x \parallel \mathbf{unit}\,;\gamma \Longrightarrow \mathbf{unit}\,;\gamma(x)} \qquad \textsc{ELam}\ \frac{}{\lambda x^A.\,M \parallel \mathbf{unit}\,;\gamma \Longrightarrow \mathbf{unit}\,;(x.M)\{\gamma\}}$$

$$\textsc{EApp}\ \frac{M \parallel s_M\,;\gamma \Longrightarrow s'_M\,;(x.P)\{\gamma'\} \qquad N \parallel s_N\,;\gamma \Longrightarrow s'_N\,;\delta \qquad P \parallel s_P\,;\gamma'[\delta/x] \Longrightarrow s'_P\,;\delta'}{MN \parallel (s_M, s_N, s_P)\,;\gamma \Longrightarrow (s'_M, s'_N, s'_P)\,;\delta'}$$

$$\textsc{EHead}\ \frac{M \parallel s\,;\gamma \Longrightarrow s'\,;\delta :: \delta'}{\mathsf{hd}\, M \parallel s\,;\gamma \Longrightarrow s'\,;\delta} \qquad \textsc{ETail}\ \frac{M \parallel s\,;\gamma \Longrightarrow s'\,;\delta :: \delta'}{\mathsf{tl}\, M \parallel s\,;\gamma \Longrightarrow s'\,;\delta'} \qquad \textsc{ECons}\ \frac{M \parallel s_M\,;\gamma \Longrightarrow s'_M\,;\delta \qquad N \parallel s_N\,;\gamma \Longrightarrow s'_N\,;\delta'}{M :: N \parallel (s_M, s_N)\,;\gamma \Longrightarrow (s'_M, s'_N)\,;\delta :: \delta'}$$

$$\textsc{EBy}\ \frac{M \parallel s\,;(\gamma_i)_{i<\partial p(n)} \overset{\partial p(n)}{\Longrightarrow} s'\,;(\delta_i)_{i<\partial p(n)}}{M\ \mathsf{by}\ p \parallel \mathbf{w}\,(p,s)\,;\triangleright(\gamma_i)_{i<\partial p(n)} \Longrightarrow \mathbf{w}\,(p,s')\,;\triangleright(\delta_i)_{i<\partial p(n)}}$$

$$\textsc{ERecI}\ \frac{M \parallel \mathbf{stop}\,;\gamma[\triangleright\varepsilon/x] \Longrightarrow s'\,;\delta}{\mathsf{rec}\, x^A.\,M \parallel \mathbf{stop}\,;\gamma \Longrightarrow (s', \mathbf{inc}\,(\delta))\,;\delta} \qquad \textsc{ERec}\ \frac{M \parallel s\,;\gamma[\triangleright(\delta_{n-1})/x] \Longrightarrow s'\,;\delta_n}{\mathsf{rec}\, x^A.\,M \parallel \left(s, \mathbf{inc}\,((\delta_i)_{i<n})\right)\,;\gamma \Longrightarrow \left(s', \mathbf{inc}\,((\delta_i)_{i<n+1})\right)\,;\delta_n}$$

$$\textsc{EDelay}\ \frac{M \parallel s\,;\gamma \Longrightarrow s'\,;\triangleright(\delta_i)_{p(n)\leqslant i<p(n+1)}}{\mathsf{delay}^{q\leqslant p}(M) \parallel \left(s, \mathbf{inc}\left(\left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}\right)\right)\,;\gamma \Longrightarrow \left(s', \mathbf{inc}\left(\left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n+1}\right)\right)\,;\triangleright(\delta_i)_{q(n)\leqslant i<q(n+1)}}$$

Figure 8 Single-step evaluation judgement

Single-step evaluation judgement. The main idea behind the incremental semantics is to have a relation that advances a computation by one tick. Such a relation can be pictured by viewing the well-typed term $\Gamma \vdash M : A \mid S$ as an automaton (see Figure 9). It is given a state s of type S, and an incremental environment γ representing a slice of a prefix environment of type Γ. The automaton produces an increment δ representing a slice of the output value of type A, and a new state s' that is ready to be used to advance the program another step. To this end, we introduce the *single-step evaluation judgement*, denoted $M \parallel s\,;\gamma \Longrightarrow s'\,;\delta$. The rules composing this judgement are given in Figure 8.

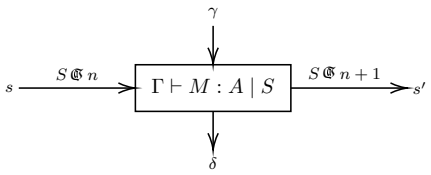

Figure 9 Terms as block diagrams

All rules — except two — that handle cases where the initial state is equal to **stop** have been omitted here for shortness sake.[3] The reason for singling out EScalarI and ERecI is that unlike the other rules, they are not just structural coercions. EScalarI expresses that once a scalar has yielded its full informational content at the first step, it becomes inert and returns the vacuous increment, **nil**. ERecI is in charge of initializing the recursion. It does so by binding $\triangleright\varepsilon$ in the environment, which corresponds to the first increment of a family of type $\mathbf{lat} \Rrightarrow A$.

[3]They can be found in the full version of the paper, available at http://hal.archives-ouvertes.fr/hal-03465519.

We detail three remarkable rules of the judgement.

Rule EBy makes use of the many-step evaluation judgement in its premise, which will formally be introduced in the next paragraph. It corresponds to repeated application of the single-step judgement. When the program M by p makes a step from time $n-1$ to time n, the subterm M makes a step from time $p(n-1)$ to $p(n)$, thus producing $\partial p(n)$ increments. The $\partial p(n)$-family of increments output by the subterm M is then turned into a single increment by the $\triangleright$ constructor. The input and output states are p-shifted in time, and protected under the constructor $\mathbf{w}\,(p,-)$.

Rule ERec is the most surprising one, due to the fact that it only needs to use the increment produced at the previous tick to produce the one for the current step. While it receives by way of its state a family of increments intended to represent the entierety of the previously-computed prefix, it only uses the very last one to produce a result. Apart from this particular observation, the rule behaves similarly to its prefix counterpart PRec, as the premise of ERec behaves similarly to the iteration judgement (cf. Figure 5): a value of type A and length $n-1$ is inserted under a constructor for a value of type $\mathbf{lat} \Rrightarrow A$, and bound to the environment. Once this increment has been produced, it is appended to the input state's increments.

Rule EDelay states that at time n, the subterm M of type $p \Rrightarrow A$ produces a warped increment family encompassing ticks from $p(n-1)$ to $p(n)-1$. To produce an increment of $q \Rrightarrow A$ at time n, we need a warped increment family encompassing ticks from $q(n-1)$ to $q(n)-1$. We buffer the increments produced by the subterm M by appending them to the input state. This yields a family of increments encompassing times 0 to times $p(n)-1$, which becomes a new output. From this family, we extract increments ranging from time $q(n-1)$ to time $q(n)-1$. This is always possible since $q \leqslant p$. By warping this subfamily extracted from the output state, we produce our final result (see Figure 10).

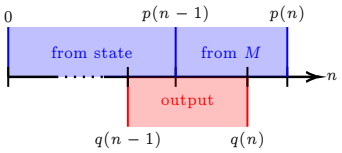

Figure 10 EDelay timings

Recall the program nat given in Section 2.2. The table below gives the first increments produced by evaluating it using the single-step reduction.

n	1	2	3
nat	$0 :: \triangleright[\,]$	$\mathbf{nil} :: \triangleright[1 :: \triangleright[\,]]$	$\mathbf{nil} :: \triangleright[\mathbf{nil} :: \triangleright[2 :: \triangleright[\,]]]$

The fixed point rule evaluates the body under rec. At the first tick, the warped empty increment $\triangleright[\,]$ is bound in the context. Applying incr′ on it leaves it unchanged. Notice that at the very first tick, the program '0' returns its value as an increment. Thus, when '$0 ::$' is called, it produces the increment $0 :: \triangleright[\,]$. The result is then stored in the state before being returned by the whole program. At the next tick, we take this state, and unwrap rec again. This time, the previous increment is $0 :: \triangleright[\,]$, and applying incr′ on it yields $1 :: \triangleright[\,]$. Then, we call '$0 ::$' on it. However, since we are not at the first tick, the program '0' is inert and produces the increment **nil**. Thus, when evaluated at tick 1, the program returns the increment $\mathbf{nil} :: \triangleright[1 :: \triangleright[\,]]$. The same reasoning applies for subsequent evaluations.

Many-step evaluation judgement. We define a relation $\overset{n}{\Rightarrow}$, the *many-step evaluation judgement*, whose rules are given in Figure 11. The statement $M \parallel s\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'\,;(\delta_i)_{i<n}$ expresses that repeated applications of single-step reduction starting from state s leads to state s', with step $i \in [0,n)$ consuming γ_i and producing δ_i.

Rule EOmega deserves special attention. Being able to run a program for ω steps means being able to run it for m steps, for every finite m. The infinite family of increments returned as output is defined elementwise by all the finite runs of the program. However, the output

state of all the finite runs is thrown away, and an infinite run always has output state **done**. Intuitively speaking, the final state of a program after ω steps is irrelevant since it can never do any further step. Since the rules in Figure 8 do not contain any rule that takes **done** as input state, ESTEP can never be applied, making **done** final — which is the intended behavior.

Any two many-step reductions originating from the same term, inputs, and initial state yield the same outputs for both state and produced increments. In other words, the automaton described by M is deterministic.

Theorem 3.2 (Determinism)**.** *If* $M \Vert s;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'_1\,;(\delta^1_i)_{i<n}$ *and* $M \parallel s\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'_2\,;(\delta^2_i)_{i<n}$*, then* $s'_1 = s'_2$ *and* $\delta^1_i = \delta^2_i$ *for all* i*.*

Since the many-step evaluation judgement depends on the single-step evaluation judgement and vice-versa due to rules ESTEP and EBY respectively, the proof has to proceed by mutual induction on the judgements using a lemma that proves determinism in the single-step case.

Lemma 3.1 (Determinism, single-step)**.** *If we have* $M \parallel s\,;\gamma \Longrightarrow s'_1\,;\delta_1$ *and* $M \parallel s\,;\gamma \Longrightarrow s'_2\,;\delta_2$ *then* $s'_1 = s'_2$ *and* $\delta_1 = \delta_2$*.*

$$\boxed{M \parallel s\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'\,;(\delta_i)_{i<n}}$$

EZERO

$$\frac{}{M \parallel s\,;\varepsilon \overset{0}{\Longrightarrow} s\,;\varepsilon}$$

ESTEP

$$\frac{M \parallel s\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'\,;(\delta_i)_{i<n} \qquad M \parallel s'\,;\gamma_n \Longrightarrow s''\,;\delta_n}{M \parallel s\,;(\gamma_i)_{i<n+1} \overset{n+1}{\Longrightarrow} s''\,;(\delta_i)_{i<n+1}}$$

EOMEGA

$$\frac{\forall m \leq \omega,\quad M \parallel s\,;(\gamma_i)_{i<m} \overset{m}{\Longrightarrow} s'_m\,;(\delta_i)_{i<m}}{M \parallel s\,;(\gamma_i)_{i<\omega} \overset{\omega}{\Longrightarrow} \mathbf{done}\,;(\delta_i)_{i<\omega}}$$

Figure 11 Many-step evaluation judgement

3.3 Metatheory

We now prove metatheoretical properties of the type system in relation with the many-step evaluation judgment. These theorems prove that our incremental semantics is as well-behaved as the prefix semantics: it is a type-respecting total function on well-typed terms. Since the type system of increments applies to families of increments indexed by a downward-closed subset of $\omega + 1$, the theorems only apply with **stop** as the initial state.

Many-step reduction is type-safe. Running a closed term of type A with initial state **stop** for n steps produces a family of increments $(\delta_i)_{i<n}$ of type A at n. This result, generalized to open terms, is proved by induction over typing derivations.

Theorem 3.3 (Type safety)**.** *Let* $\Gamma \vdash M : A \mid S$*, and* $n \leqslant \omega$*. For all* $(\gamma_i)_{i<n} : \Gamma \mathbin{@} n$ *and* $(\delta_i)_{i<n}$ *such that* $M \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s\,;(\delta_i)_{i<n}$*, we have* $(\delta_i)_{i<n} : A \mathbin{@} n$ *and* $s : S \mathfrak{C} n$*.*

Many-step reduction is a total function. For any well-typed closed term M, there exists a unique family of increments $(\delta_i)_{i<n}$ such that running M for n steps produces $(\delta_i)_{i<n}$. This is due to Theorem A.2 and to the theorem below, proved again by induction over typing derivations.

Theorem 3.4 (Totality)**.** *Let* $\Gamma \vdash M : A \mid S$*, and* $n \leqslant \omega$*. For all* $(\gamma_i)_{i<n} : \Gamma \mathbin{@} n$*, there exists* s *and* $(\delta_i)_{i\leq n}$ *such that* $M \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s\,;(\delta_i)_{i<n}$*.*

$$
\begin{array}{rll}
\varepsilon \models_0^A & \mathbf{stop} & \\
(\delta_i)_{i<\omega} \models_\omega^A & \mathbf{box}(M)\{E\} & \overset{\triangle}{\iff} \forall n < \omega.\ (\delta_i)_{i<n} \models_n^A \lfloor \mathbf{box}(M)\{E\} \rfloor_n \\
(c) \oplus (\mathbf{nil})_{i<n-1} \models_n^\nu & c & \\
\left(\triangleright (\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n} \models_n^{p \Rrightarrow A} & \mathbf{w}\,(p, V) & \overset{\triangle}{\iff} (\delta_i)_{i<p(n)} \models_{p(n)}^A V \\
\left(\delta_i :: \triangleright (\delta'_j)_{i-1\leqslant j<i}\right)_{i<n} \models_n^{\mathsf{Str}\,A} & V :: V' & \overset{\triangle}{\iff} (\delta_i)_{i<n} \models_n^A V \text{ and } (\delta'_i)_{i<n-1} \models_{n-1}^{\mathsf{Str}\,A} V' \\
\left((x.M)\{\gamma_i\}\right)_{i<n} \models_n^{A \overset{S}{\Rightarrow} B} & (x.N)\{E\} & \overset{\triangle}{\iff} \forall m \leqslant n, \forall (\delta_i)_{i<m} \models_m^A V, \exists s, (\delta'_i)_{i<m}, V', \\
 & & \begin{cases} M \,\|\, \mathbf{stop}\,;\left(\gamma_i[\delta_i/x]\right)_{i<m} \overset{m}{\Longrightarrow} s\,;(\delta'_i)_{i<m} \\ N\,;\lfloor E \rfloor_m [V/x] \Downarrow_m V' \\ (\delta'_i)_{i<m} \models_m^B V' \end{cases}
\end{array}
$$

Figure 12 Logical relation for values

3.4 Correctness

While the many-step evaluation judgments define a total function, it remains to be proved that it is correct, in the sense that it computes the same results as the prefix semantics. In particular, we wish to prove the following result.

Theorem 3.5. *If* $\vdash M : \mathsf{Nat} \mid S$ *and* $M\,;\varnothing \Downarrow_1 c$ *then there is* s *such that* $M \,\|\, \mathbf{stop}\,;\emptyset \Longrightarrow s\,;c$.

This theorem is more general than it looks. In the presence of by in the language, the term M may contain subterms which have to run for more than one step. In particular, the result c could be the ith element of some stream of numbers, for i fixed but arbitrary. For this reason, to prove this theorem, we must generalize it to many-step reduction, in addition to open terms of arbitrary types. We do so by way of the logical relation defined in Figure 12.

The logical relation specifies how families of increments assemble into prefix values. Given a family of increments $(\delta_i)_{i<n} : A \mathbin{@} n$ and a prefix value $V : A \mathbin{@} n$, we say that $(\delta_i)_i$ is a *coherent slicing* of V when $\Delta \models_n^A V$ holds. This relation is defined by well-founded induction on the lexicographical product $<_{\mathsf{ty}} \times_{\mathsf{lex}} <_{\omega+1}$ between the strict subtree ordering on types $<_{\mathsf{ty}}$ and the canonical strict ordering $<_{\omega+1}$ on $\omega + 1$.

The case of scalars, pairs, sums, and streams are unremarkable. Thunks representing infinite prefixes are coherently sliced into infinite families of prefixes in such a way that truncation at a finite length always produces coherent slicings. A warped increment family is a coherent slicing of a warped value if their unfoldings are coherent slicings of each other. Finally, functions follow the usual pattern of logical relations, adapted to our setting: a (prefix) closure is coherently sliced by a family of incremental closures when, applied to coherent arguments, they return coherent results.

The logical relation over values extends to incremental and prefix environments.

Definition 3.1 (Logical relation for environments)**.** A family $(\gamma_i)_i : \Gamma \mathbin{@} n$ is a *coherent slicing* of $E : \Gamma \mathbin{@} n$, denoted $(\gamma_i)_i \models_n^\Gamma E$, when, for all $x \in \mathrm{dom}(\Gamma)$, we have $(\gamma_i(x))_i \models_n^{\Gamma(x)} E(x)$.

We extend the logical relation to terms. Intuitively, two closed terms M and N are logically related when the increments produced by n reactions of M coherently slice the prefix of length n computed from N. The definition below generalizes this to open terms.

Definition 3.2 (Logical relation for terms)**.** The terms M and N are *logically related* at Γ and A, denoted $M \models^{\Gamma\vdash A} N$, when, for all n, E, $(\gamma_i)_{i<n}$, V, $(\delta_i)_{i<n}$ and s such that $(\gamma_i)_{i<n} \models_n^\Gamma E$, if $N\,;E \Downarrow_n V$ and $M \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s\,;(\delta_i)_{i<n}$, then $(\delta_i)_{i<n} \models_n^A V$.

Adequacy. The adequacy lemma for our logical relation immediately implies Theorem 3.5 by the definition of AScalar and the fact that the single-step reduction is a special case of the many-step one.

Lemma 3.2 (Adequacy)**.** *If* $\Gamma \vdash M : A \mid S$*, then* $M \models^{\Gamma\vdash A} M$.

The proof is done by induction over the typing hypothesis. Inversion lemmas are needed to invert whole chains of many-step reductions, as it is defined in term of single-step reductions. Such lemmas are not needed for inverting the prefix reduction. The induction hypothesis is then to be applied on subterms, which in turn proves that they produce coherent slicings. Application of the rules of Figure 12 yields the desired results.

To prove the adequacy lemma, we need a result connecting truncation in the original prefix semantics with our incremental semantics in the following sense.

Lemma 3.3 (Coherence of truncation)**.** *If* $(\delta_i)_{i<n} \models_n^A V$ *and* $m \leqslant n$*, then* $(\delta_i)_{i<m} \models_m^A \lfloor V \rfloor_m$.

The key case is the one of Rec. It is dealt with by an inner induction on the number of steps, using Lemma A.9 to connect the intermediate families of increments with the values computed by the prefix semantics.

4 Perspectives

4.1 Limitations

Increments storage in states. Because our increment typing judgement only classifies whole families of increments, states have to store entire families of increments as well to be well-typed. As a consequence, a naive implementation of our semantics of delays and fixed points would keep all past increments in memory, resulting in unbounded space usage. This is very unsatisfactory since the elimination of such spurious "space leaks" is a core concern in reactive programming. Fortunately, inspection of the single-step judgement shows that the rules that use state to store intermediate computations only rely on a subset of the increments stored in this state. For example, the semantics of fix actually uses only the last increment from its state, and thus only requires a buffer of size one.

Infinitary syntax. As mentioned in Section 3, in the original semantics, prefixes are always finite since prefixes of type $A \mathbin{@} \omega$ are thunks. In contrast, families of increments can be actually infinite. This choice makes for simple semantics but cannot be used in a non-lazy implementation. We believe that using thunks in the incremental semantics would be technically heavier but would not raise any conceptual issue.

Structural coercions of the warping modality. The original presentation of λ^* includes primitive constructs to mediate between isomorphic types such as $p \Rrightarrow (A \times B)$ and $(p \Rrightarrow A) \times (p \Rrightarrow B)$. We have omitted their unremarkable treatment from this paper for lack of space.

Recursive states. Our state types do not include recursive types such as streams. This restriction makes it impossible to capture most higher-order recursive functions, such as the general map function over streams. This function should have type $(A \overset{S}{\Rrightarrow} B) \overset{\mathbb{1}}{\Rrightarrow} \mathsf{Str}\,A \overset{\mathsf{Str}\,S}{\Longrightarrow} \mathsf{Str}\,B$, reflecting how map f xs creates a new copy of the state of f for each element of xs.

4.2 Related work

The implementation of reactive programs is a wide area that has been explored in distinct directions by distinct lines of research. We briefly discuss two of them.

Synchronous programming. Synchronous languages restrict stream elements to be of scalar types, eschewing types such as streams of streams. This restriction allows for efficient if specialized implementation techniques. Our incremental semantics is directly inspired from their "single-loop" state-passing transform [17, 6, 3], but handles streams of arbitrary element type, such as streams of streams, as well as higher-order functions.

We owe a particularly large debt to Lucid Synchrone [16], whose compiler introduced the idea of type-checking program states in a higher-order reactive language more than twenty years ago. In contrast with the present work, in Lucid Synchrone state types are almost[4] invisible at the source level and only appear in the generated OCaml code, relying on the OCaml compiler for this part of type-checking. Our choice to include state types in the source type system simplifies metatheoretical proofs and would facilitate an implementation of separate compilation. However, the absence of recursive state types forces us to reject many higher-order programs (see Section 4.2), some of which are accepted by Lucid Synchrone.

Functional reactive programming. Several works in functional reactive programming use techniques similar to that of synchronous programming. For example, causal commutative arrows [13] can be compiled to simple state-passing code but are purely first-order.

Several recent proposals such as that of Krishnaswami [11] or Bahr et al. [2] exploit a modal type discipline to reject certain recursion patterns and rule out implicit memory leaks in an expressive higher-order language. In their operational semantics, program state is untyped and dynamically allocated in a global store, while in ours it is typed and thus controlled by term structure. An in-depth comparison of the two styles remains for future work.

4.3 Conclusion

We have presented a new operational semantics for λ^*, a λ-calculus for productive reactive programming proposed in previous work. This semantics, in contrast with the existing one, is incremental. It can be seen as an adaptation of classic synchronous-language compilation techniques to a setting featuring rich temporal types such as streams of streams or streams of stream functions. Our incremental semantics enjoys the same desirable metatheoretical properties: type safety, totality, and determinism. Using a logical relation to specify how prefixes can be sliced into families of incremental values, we have proved that it is fully consistent with respect to the existing semantics.

In future work, we plan to refine the space usage of the new semantics, extend it beyond streams to arbitrary guarded recursive types, and capture its incremental character in a denotational model by adapting the existing interpretation of λ^* in the topos of trees. We would also like to adapt it into an implementation.

References

[1] Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. A Very Modal Model of a Modern, Major, General Type System. In *Principles of Programming Languages*

[4] Lucid Synchrone distinguishes between functions of unit state type and functions of non-unit state types at the source level, grouping all of the former functions into a single type.

(POPL'07). ACM, 2007.

[2] Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. The Clocks Are Ticking: No More Delays! Reduction Semantics for Type Theory with Guarded Recursion. In *Logic in Computer Science (LICS'17)*. Springer, 2017.

[3] Dariusz Biernacki, Jean-Louis Colaço, Gregoire Hamon, and Marc Pouzet. Clock-directed modular code generation for synchronous data-flow languages. In *Languages, Compilers and Tools for Embedded Systems (LCTES'08)*, 2008.

[4] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science*, 8(4), 2012.

[5] Paul Caspi, Daniel Pilaud, Nicolas Halbwachs, and John Plaice. LUSTRE: A declarative language for programming synchronous systems. In *Principles of Programming Languages (POPL'87)*, 1987.

[6] Paul Caspi and Marc Pouzet. Synchronous Kahn Networks. In *International Conference on Functional Programming (ICFP'96)*. ACM, 1996.

[7] Paul Caspi and Marc Pouzet. A co-iterative characterization of synchronous stream functions. *Electronic Notes in Theoretical Computer Science*, 11:1–21, 1998.

[8] Conal Elliott and Paul Hudak. Functional Reactive Animation. In *International Conference on Functional Programming (ICFP'97)*. ACM, 1997.

[9] Adrien Guatto. A Generalized Modality for Recursion. In *Logic in Computer Science (LICS'18)*, 2018.

[10] Gilles Kahn. The semantics of a simple language for parallel programming. In *Information Processing Congress (IFIP'74)*. IFIP, 1974.

[11] Neelakantan R Krishnaswami. Higher-Order Functional Reactive Programming without Spacetime Leaks. In *International Conference on Functional Programming (ICFP'13)*. ACM, 2013.

[12] Neelakantan R. Krishnaswami and Nick Benton. Ultrametric Semantics of Reactive Programs. In *Logic in Computer Science (LICS'11)*. IEEE, 2011.

[13] Hai Liu, Eric Cheng, and Paul Hudak. Causal commutative arrows. *Journal of Functional Programming*, 2011.

[14] John Lucassen and David Gifford. Polymorphic effect systems. In *Principles of Programming Languages (POPL'88)*. ACM, 1988.

[15] Hiroshi Nakano. A Modality for Recursion. In *Logic in Computer Science (LICS'00)*. IEEE, 2000.

[16] Marc Pouzet. *Lucid Synchrone, version 3. Tutorial and reference manual*. Université Paris-Sud, LRI, April 2006.

[17] Pascal Raymond. *Compilation efficace d'un langage déclaratif synchrone : Le générateur de code Lustre-V3*. PhD thesis, Institut National Polytechnique de Grenoble, 1991.

A Interesting proofs

A.1 Inversion and construction lemmas

A.1.1 Inversion lemmas

Lemma A.1 (Inversion lemma for by)**.** *Let $0 < n \leqslant \omega$. Suppose that:*

$$M \text{ by } p \parallel \mathbf{stop} \,;\, \left(\triangleright(\gamma_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n} \xRightarrow{n} s \,;\, (\delta_i)_{i<n}$$

Then, we have:

$$\exists s'.\ \exists(\delta'_i)_{i<p(n)}.\ \begin{cases} M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<p(n)} \xRightarrow{p(n)} s' \,;\, (\delta'_i)_{i<p(n)} \\ \delta_i = \triangleright(\delta'_j)_{p(i)<j\leqslant p(i+1)} \text{ for all } i < n \\ s = \begin{cases} \mathbf{w}\,(p, s') & \text{if } n < \omega \\ \mathbf{done} & \text{otherwise} \end{cases} \end{cases}$$

Proof. By induction on n.

- Case $n = 1$. By hypothesis and inversion of EStep we have:

  $$M \text{ by } p \parallel \mathbf{stop} \,;\triangleright(\gamma_i)_{i<p(1)} \Longrightarrow s \,;\delta_0$$

  Then, by inversion of EByI, we obtain what we wanted:

  $$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<p(1)} \xRightarrow{p(1)} s' \,;(\delta'_i)_{i<p(1)} \qquad \delta_0 = \triangleright(\delta'_i)_{i<p(1)} \qquad s = \mathbf{w}\,(p, s')$$

- Case $1 < n < \omega$. By hypothesis and inversion of EStep we have:

  $$M \text{ by } p \parallel \mathbf{stop} \,;\, \left(\triangleright(\gamma_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n-1} \xRightarrow{n-1} s_I \,;\, (\delta_i)_{i<n-1} \tag{a}$$

  $$M \text{ by } p \parallel s_I \,;\triangleright(\gamma_j)_{p(n-1)\leqslant j<p(n)} \Longrightarrow s \,;\delta_n \tag{b}$$

  Applying the induction hypothesis to (a) yields:

  $$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<p(n-1)} \xRightarrow{p(n-1)} s'_I \,;(\delta'_i)_{i<p(n-1)} \tag{c}$$

  $$\delta_i = \triangleright(\delta'_j)_{p(i)<j\leqslant p(i+1)} \quad \forall i < n-1 \tag{d}$$

  $$s_I = \mathbf{w}\,(p, s'_I) \tag{e}$$

  Now, we distinguish two cases.

  - $p(n-1) = \omega$. In this case, $s'_I = \mathbf{done}$, $p(n) = \omega$ and $\partial p(n) = 0$. We have by EZero that :

    $$M \parallel s'_I \,;\triangleright\varepsilon \xRightarrow{\partial p(n)} s'_I \,;\triangleright\varepsilon$$

    Thus, $\delta_{n-1} = \triangleright\varepsilon$. Substituting $p(n-1) = p(n) = \omega$ in the above equations yields what we wanted:

    $$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<p(n)} \xRightarrow{p(n)} s' \,;(\delta'_i)_{i<p(n)}$$

    $$\delta_i = \triangleright(\delta'_j)_{p(i)<j\leqslant p(i+1)} \quad \forall i < n$$

    $$s = \mathbf{w}\,(p, s') = \mathbf{w}\,(p, \mathbf{done})$$

– $p(n-1) < \omega$. In this case, $s_I = \mathbf{w}\,(p, s'_I)$. By substituting (e) in (b) and then inverting EBy, we obtain:

$$M \parallel s'_I \,;\, (\gamma_i)_{p(n-1)\leqslant i<p(n)} \xRightarrow{\partial p(n)} s' \,;\, (\delta'_i)_{p(n-1)\leqslant i<p(n)} \tag{f}$$

$$\delta_{n-1} = \triangleright(\delta'_j)_{p(n-1)<j\leqslant p(n)} \tag{g}$$

$$s = \begin{cases} \mathbf{w}\,(p, s') & \text{if } \partial p(n) < \omega \\ \mathbf{done} & \text{otherwise} \end{cases} \tag{h}$$

Notice that if $\partial p(n) = \omega$, then $p(n) = \omega$

Since $p(n) = p(n-1) + \partial p(n)$, combining (c) and (f) yields what we wanted:

$$M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<p(n)} \xRightarrow{p(n)} s' \,;\, (\delta'_i)_{i<p(n)}$$

$$\delta_i = \triangleright(\delta'_j)_{p(i)<j\leqslant p(i+1)} \quad \forall i < n$$

$$s = \begin{cases} \mathbf{w}\,(p, s') & \text{if } p(n) < \omega \\ \mathbf{done} & \text{otherwise} \end{cases}$$

- Case $n = \omega$. By hypothesis and inversion of EOmega we have:

$$\forall m < \omega. \quad M \text{ by } p \parallel \mathbf{stop} \,;\, \left(\triangleright(\gamma_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<m} \xRightarrow{m} s_m \,;\, (\delta_i)_{i<m}$$

The induction hypothesis applies, and gives :

$$\forall m < \omega. \quad \begin{cases} M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<p(m)} \xRightarrow{p(m)} s'_{p(m)} \,;\, (\delta'_i)_{i<p(m)} \\ \delta_i = \triangleright(\delta'_j)_{p(i)<j\leqslant p(i+1)} \quad \forall i < m \end{cases} \tag{i}$$

By inversion of EOmega on the hypothesis we have that $s = \mathbf{done}$. By (i), we obtain that $\forall i < \omega.\ \delta_i = \triangleright(\delta'_j)_{p(i)<j\leqslant p(i+1)}$.

We conclude in different ways depending on the finiteness of $p(\omega)$.

– $p(\omega) = \omega$. Let $k < \omega$. Since p is monotonic due to being a time warp, and $p(\omega) = \omega$, there exists a M such that $k \leqslant p(M)$. By $p(M) - k$ inversions of EStep (or EOmega if $p(M) = \omega$), we obtain that :

$$\exists s''_k. \quad M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<k} \xRightarrow{k} s''_k \,;\, (\delta'_i)_{i<k}$$

Thus, we have that :

$$\forall k < \omega. \quad M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<k} \xRightarrow{k} s''_k \,;\, (\delta'_i)_{i<k}$$

Immediately applying EOmega and substituting $p(\omega) = \omega$, we obtain :

$$M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<p(\omega)} \xRightarrow{p(\omega)} \mathbf{done} \,;\, (\delta'_i)_{i<p(\omega)}$$

Which concludes our proof.

– $p(\omega) < \omega$. In this case, p is ultimately constant: there is some finite M such that $p(M) = p(\omega)$. Thus, by instantiating (i) with $m := M$, we obtain :

$$M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<p(M)} \xRightarrow{p(M)} s'_M \,;\, (\delta'_i)_{i<p(M)}$$

By substituting $p(M)$ for $p(\omega)$ in the above, we prove that:

$$M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<p(\omega)} \overset{p(\omega)}{\Longrightarrow} s'_M \,;\, (\delta'_i)_{i<p(\omega)}$$

Which concludes our proof.

□

Lemma A.2 (Inversion lemma for rec)**.** *Let $0 < n \leqslant \omega$. Suppose that:*

$$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s \,;\, (\delta_i)_{i<n}$$

Then, we have:

$$\exists s'.\ \begin{cases} M \parallel \mathbf{stop} \,;\, \left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<n} \overset{n}{\Longrightarrow} s' \,;\, (\delta_i)_{i<n} \\ s = \begin{cases} \left(s', \mathbf{inc}\left((\delta_i)_{i<n}\right)\right) & \text{if } n < \omega \\ \mathbf{done} & \text{otherwise} \end{cases} \end{cases}$$

Proof. By induction on n.

- Case $n = 1$. By hypothesis and inversion of EStep we have:

  $$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, \gamma_0 \Longrightarrow s \,;\, \delta_0$$

  Then, by inversion of EFixI, we obtain:

  $$M \parallel \mathbf{stop} \,;\, \left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \Longrightarrow s' \,;\, \delta_0 \qquad\qquad s = \left(s', \mathbf{inc}\,(\delta)\right)$$

  Applying EStep yields what we wanted.

- Case $1 < n < \omega$. By hypothesis and inversion of EStep we have:

  $$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n-1} \overset{n-1}{\Longrightarrow} s_I \,;\, (\delta_i)_{i<n-1} \tag{a}$$

  $$\mathsf{rec}\, x^A.\, M \parallel s_I \,;\, \gamma_{n-1} \Longrightarrow s \,;\, \delta_{n-1} \tag{b}$$

  Applying the induction hypothesis to (a) yields :

  $$M \parallel \mathbf{stop} \,;\, \left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<n-1} \overset{n-1}{\Longrightarrow} s'_I \,;\, (\delta_i)_{i<n-1} \tag{c}$$

  $$s_I = \left(s'_I, \mathbf{inc}\left((\delta_i)_{i<n-1}\right)\right) \tag{d}$$

  By substituting (d) in (b) and then inverting ERec, we obtain:

  $$M \parallel s'_I \,;\, \left(\gamma_{n-1}\left[\triangleright(\delta_{n-2})/x\right]\right) \Longrightarrow s' \,;\, \delta_{n-1} \tag{e}$$

  $$s = \left(s', \mathbf{inc}\left((\delta_i)_{i<n}\right)\right) \tag{f}$$

  Together with (f), applying EStep to (c) and (e) yields what we wanted:

  $$M \parallel \mathbf{stop} \,;\, \left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<n} \overset{n}{\Longrightarrow} s' \,;\, (\delta_i)_{i<n}$$

  $$s = \left(s', \mathbf{inc}\left((\delta_i)_{i<n}\right)\right)$$

- Case $n = \omega$. By hypothesis and inversion of EOmega we have $s = \mathbf{done}$ and:

  $$\forall m < \omega.\ \ \mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<m} \overset{m}{\Longrightarrow} s'_m \,;\, (\delta_i)_{i<m}$$

The induction hypothesis applies, and gives :

$$\forall m < \omega.\ M \parallel \mathbf{stop}\,;\left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<m} \overset{m}{\Longrightarrow} s'_m\,;(\delta_i)_{i<m}$$

Applying EOmega to the above equation gives what we wanted.

□

Lemma A.3 (Inversion lemma for delay)**.** *Let $0 < n \leqslant \omega$. Suppose that:*

$$\mathsf{delay}^{q\leqslant p}(M) \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s\,;(\delta_i)_{i<n}$$

Then, we have:

$$\exists s'.\ \exists(\delta'_i)_{i<p(n)}.\ \begin{cases} M \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'\,;\left(\triangleright(\delta'_i)_{p(i)<j\leqslant p(i+1)}\right)_{i<n} \\ \delta_i = \triangleright(\delta'_j)_{q(i)<j\leqslant q(i+1)} \text{ for all } i<n \\ s = \begin{cases} \left(s', \mathbf{inc}\left(\left(\triangleright(\delta'_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}\right)\right) & \text{if } n<\omega \\ \mathbf{done} & \text{otherwise} \end{cases} \end{cases}$$

Proof. By induction on n.

- Case $n = 1$. By hypothesis and inversion of EStep we have:

  $$\mathsf{delay}^{q\leqslant p}(M) \parallel \mathbf{stop}\,;\gamma_0 \Longrightarrow s\,;\delta_0$$

  Then, by inversion of EDelayI and immediate application of EStep, we obtain what we wanted:

  $$M \parallel \mathbf{stop}\,;(\gamma_0) \overset{1}{\Longrightarrow} s'\,;\left(\triangleright(\delta'_i)_{i<q(1)}\right) \qquad \delta_0 = \triangleright(\delta'_i)_{i<q(1)} \qquad s = \left(s', \mathbf{inc}\left(\triangleright(\delta'_i)_{i<p(1)}\right)\right)$$

- Case $1 < n < \omega$. By hypothesis and inversion of EStep we have:

  $$\mathsf{delay}^{q\leqslant p}(M) \parallel \mathbf{stop}\,;(\gamma_i)_{i<n-1} \overset{n-1}{\Longrightarrow} s_I\,;(\delta_i)_{i<n-1} \tag{a}$$
  $$\mathsf{delay}^{q\leqslant p}(M) \parallel s_I\,;\gamma_n \Longrightarrow s\,;\delta_n \tag{b}$$

  Applying the induction hypothesis to (a) yields:

  $$M \parallel \mathbf{stop}\,;(\gamma_i)_{i<n-1} \overset{n-1}{\Longrightarrow} s'_I\,;\left(\triangleright(\delta'_i)_{p(i)<j\leqslant p(i+1)}\right)_{i<n-1} \tag{c}$$
  $$\delta_i = \triangleright(\delta'_j)_{q(i)<j\leqslant q(i+1)} \text{ for all } i<n-1 \tag{d}$$
  $$s_I = \left(s'_I, \mathbf{inc}\left(\left(\triangleright(\delta'_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n-1}\right)\right) \tag{e}$$

  Now, by substitution of (e) in (b) and immediate inversion of EDelay, we get:

  $$M \parallel s'_I\,;\gamma_n \Longrightarrow s'\,;\triangleright(\delta'_i)_{p(n-1)<j\leqslant p(n)} \tag{f}$$
  $$\delta_n = \triangleright(\delta'_j)_{q(n-1)<j\leqslant q(1)} \tag{g}$$
  $$s = \left(s', \mathbf{inc}\left(\left(\triangleright(\delta'_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}\right)\right) \tag{h}$$

Applying EStep on (c) and (f), merging (d) with (g), and (h) yield what we wanted:

$$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s' \,;\left(\triangleright(\delta'_i)_{p(i)<j\leqslant p(i+1)}\right)_{i<n}$$
$$\delta_i = \triangleright(\delta'_j)_{q(i)<j\leqslant q(i+1)} \text{ for all } i<n$$
$$s = \left(s', \mathbf{inc}\left(\left(\triangleright(\delta'_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}\right)\right)$$

- Case $n=\omega$. By hypothesis and inversion of EOmega we have:

$$\forall m<\omega.\quad \mathsf{delay}^{q\leqslant p}(M) \parallel \mathbf{stop} \,;(\gamma_i)_{i<m} \overset{m}{\Longrightarrow} s_m \,;(\delta_i)_{i<m}$$

The induction hypothesis applies, and gives :

$$\forall m<\omega.\quad \begin{cases} M \parallel \mathbf{stop} \,;(\gamma_i)_{i<m} \overset{m}{\Longrightarrow} s'_m \,;\left(\triangleright(\delta'_i)_{p(i)<j\leqslant p(i+1)}\right)_{i<m} \\ \delta_i = \triangleright(\delta'_j)_{q(i)<j\leqslant q(i+1)} \text{ for all } i<m \end{cases} \tag{i}$$

By inversion of EOmega on the hypothesis we have that $s = \mathbf{done}$. By (i), we obtain that $\forall i<\omega.\ \delta_i = \triangleright(\delta'_j)_{q(i)<j\leqslant q(i+1)}$.

Applying EOmega on (i) gives:

$$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<\omega} \overset{\omega}{\Longrightarrow} \mathbf{done} \,;\left(\triangleright(\delta'_i)_{p(i)<j\leqslant p(i+1)}\right)_{i<\omega}$$

Which concludes our proof.

□

A.1.2 Construction lemmas

Lemma A.4 (Construction lemma for variables)**.** *Let $0<n\leqslant\omega$. Let $(\gamma_i)_{i<n}$, such that:*

$$\forall i<n.\ x\in\mathrm{dom}(\gamma_i(x))$$

Then, we have:

$$x \parallel \mathbf{stop} \,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s \,;(\gamma_i(x))_{i<n} \text{ – with } s = \begin{cases} \mathbf{unit} \text{ if } n<\omega \\ \mathbf{done} \text{ elsewise} \end{cases}$$

Proof. By induction on n.

- Case $n=1$. By hypothesis, since $x\in\mathrm{dom}(\gamma_0)$, then EVarI applies and yields:

$$x \parallel \mathbf{stop} \,;\gamma_0 \Longrightarrow \mathbf{unit} \,;\gamma_0(x)$$

Immediate application of EStep yields what we wanted.

- Case $1<n<\omega$. The induction hypothesis gives:

$$x \parallel \mathbf{stop} \,;(\gamma_i)_{i<n-1} \overset{n-1}{\Longrightarrow} \mathbf{unit} \,;(\gamma_i(x))_{i<n-1}$$

By hypothesis, since $x\in\mathrm{dom}(\gamma_{n-1})$, then EVar applies and yields:

$$x \parallel \mathbf{unit} \,;\gamma_{n-1} \Longrightarrow \mathbf{unit} \,;\gamma_{n-1}(x)$$

Immediate application of EStep yields what we wanted.

- Case $n = \omega$. Since $x \in \mathrm{dom}(\gamma_m)$ for all $m < \omega$, the induction hypothesis applies and gives:
$$\forall m < \omega. \quad x \parallel \mathbf{stop} \,;(\gamma_i)_{i<m} \overset{m}{\Longrightarrow} \mathbf{unit} \,;(\gamma_i(x))_{i<m}$$
Immediate application of EOmega yields what we wanted.

□

Lemma A.5 (Construction lemma for by)**.** *Let $0 < n \leqslant \omega$. Suppose that:*
$$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<p(n)} \overset{p(n)}{\Longrightarrow} s' \,;(\delta_i)_{i<p(n)}$$
Then, we have:
$$M \text{ by } p \parallel \mathbf{stop} \,;\left(\triangleright(\gamma_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n} \overset{n}{\Longrightarrow} s \,;\left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}$$
With s such that:
$$s = \begin{cases} \mathbf{w}\,(p, s') & \text{if } n < \omega \\ \mathbf{done} & \text{elsewise} \end{cases}$$

Proof. By induction on n.

- Case $n = 1$. By hypothesis we have:
$$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<p(1)} \overset{p(1)}{\Longrightarrow} s' \,;(\delta_i)_{i<p(1)}$$
Notice that since p is a time warp, $p(0) = 0$. We apply EByI:
$$M \text{ by } p \parallel \mathbf{stop} \,;\triangleright(\delta_j)_{p(0)\leqslant j<p(1)} \Longrightarrow \mathbf{w}\,(p, s') \,;\triangleright(\gamma_j)_{p(0)\leqslant j<p(1)}$$
Applying EStep yields what we wanted.

- Case $1 < n < \omega$. By hypothesis and inversion of EStep we have:
$$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<p(n-1)} \overset{p(n-1)}{\Longrightarrow} s'_I \,;(\delta_i)_{i<p(n-1)} \tag{a}$$
$$M \parallel s'_I \,;(\gamma_i)_{p(n-1)\leqslant i<p(n)} \overset{\partial p(n)}{\Longrightarrow} s' \,;(\delta_i)_{p(n-1)\leqslant i<p(n)} \tag{b}$$
Applying the induction hypothesis to (a), and EBy to (b) yields :
$$M \text{ by } p \parallel \mathbf{stop} \,;\left(\triangleright(\gamma_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n-1} \overset{n-1}{\Longrightarrow} \mathbf{w}\,(p, s'_I) \,;\left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n-1}$$
$$M \text{ by } p \parallel \mathbf{w}\,(p, s'_I) \,;\triangleright(\gamma_j)_{p(n-1)\leqslant j<p(n)} \Longrightarrow \mathbf{w}\,(p, s') \,;\triangleright(\delta_j)_{p(n-1)\leqslant j<p(n)}$$
Immediate application of EStep yields what we wanted:
$$M \text{ by } p \parallel \mathbf{stop} \,;\left(\triangleright(\gamma_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n} \overset{n}{\Longrightarrow} \mathbf{w}\,(p, s') \,;\left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}$$

- Case $n = \omega$. By hypothesis we have:
$$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<p(\omega)} \overset{p(\omega)}{\Longrightarrow} s' \,;(\delta_i)_{i<p(\omega)} \tag{c}$$
We distinguish to cases, depending on the value of $p(\omega)$.

– Case $p(\omega) < \omega$. p is ultimately constant – this means that there is a finite N such that $p(N) = p(\omega)$. Thus, (c) rewrites to:

$$M \parallel \mathbf{stop} \,;(\gamma_i)_{i<p(N)} \xRightarrow{p(N)} s' \,;(\delta_i)_{i<p(N)}$$

By application of the induction hypothesis, we obtain:

$$M \text{ by } p \parallel \mathbf{stop} \,;\left(\triangleright(\gamma_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<N} \xRightarrow{N} \mathbf{w}\,(p,s') \,;\left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<N} \quad \text{(d)}$$

By EZero, we always have $M \parallel s' \,;\varepsilon \xRightarrow{0} s' \,;\varepsilon$. Thus, by EBy:

$$M \text{ by } p \parallel \mathbf{w}\,(p,s') \,;\triangleright\varepsilon \Longrightarrow \mathbf{w}\,(p,s') \,;\triangleright\varepsilon$$

Notice that since p is ultimately constant at N, then for all $m \geqslant N$, we have:

$$(\gamma_i)_{p(m)\leqslant i<p(m+1)} = (\delta_i)_{p(m)\leqslant i<p(m+1)} = \varepsilon$$

Thus, EStep applies any number of times to (d) and yields:

$$\forall m < \omega.\ M \text{ by } p \parallel \mathbf{stop} \,;\left(\triangleright(\gamma_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<m} \xRightarrow{m} \mathbf{w}\,(p,s') \,;\left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<m}$$

Applying EOmega proves what we wanted.

– Case $p(\omega) = \omega$. Immediate inversion of EOmega in (c) yields:

$$\forall m < \omega.\ M \parallel \mathbf{stop} \,;(\gamma_i)_{i<m} \xRightarrow{m} s'_m \,;(\delta_i)_{i<m}$$

This especially gives:

$$\forall m < \omega.\ M \parallel \mathbf{stop} \,;(\gamma_i)_{i<p(m)} \xRightarrow{p(m)} s'_{p(m)} \,;(\delta_i)_{i<p(m)}$$

Applying the induction hypothesis gives:

$$\forall m < \omega.\ M \text{ by } p \parallel \mathbf{stop} \,;\left(\triangleright(\gamma_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<m} \xRightarrow{m} \mathbf{w}\left(p, s'_{p(m)}\right) \,;\left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<m}$$

Applying EOmega proves what we wanted.

□

Lemma A.6 (Construction lemma for rec)**.** *Let $0 < n \leqslant \omega$. Suppose that:*

$$M \parallel \mathbf{stop} \,;\left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<n} \xRightarrow{n} s' \,;(\delta_i)_{i<n}$$

Then, we have:

$$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;(\gamma_i)_{i<n} \xRightarrow{n} s \,;(\delta_i)_{i<n} \text{ – with } s = \begin{cases} (s', \mathbf{inc}\,((\delta_i)_{i<n})) & \text{if } n < \omega \\ \mathbf{done} & \text{elsewise} \end{cases}$$

Proof. By induction on n.

- Case $n = 1$. By hypothesis and inversion of EStep we have:

$$M \parallel \mathbf{stop} \,;\gamma_0\left[\triangleright\varepsilon/x\right] \Longrightarrow s' \,;\delta_0$$

Then, by application of EFixI, we obtain:

$$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\gamma_0 \Longrightarrow (s', \mathbf{inc}\,(\delta_0)) \,;\delta_0$$

Applying EStep yields what we wanted.

- Case $1 < n < \omega$. By hypothesis and inversion of EStep we have:

$$M \parallel \mathbf{stop} \,;\, \left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<n-1} \stackrel{n-1}{\Longrightarrow} s'_I \,;\, (\delta_i)_{i<n-1} \tag{a}$$

$$M \parallel s_I \,;\, \left(\gamma_{n-1}\left[\triangleright(\delta_{n-2})/x\right]\right) \Longrightarrow s' \,;\, \delta_{n-1} \tag{b}$$

  Applying the induction hypothesis to (a), and ERec to (b) yields :

$$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n-1} \stackrel{n-1}{\Longrightarrow} (s'_I, \mathbf{inc}\left((\delta_i)_{i<n-1}\right)) \,;\, (\delta_i)_{i<n-1}$$

$$\mathsf{rec}\, x^A.\, M \parallel (s'_I, \mathbf{inc}\left((\delta_i)_{i<n-1}\right)) \,;\, \gamma_{n-1} \Longrightarrow (s', \mathbf{inc}\left((\delta_i)_{i<n}\right)) \,;\, \delta_{n-1}$$

  Immediate application of EStep yields what we wanted:

$$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n} \stackrel{n}{\Longrightarrow} (s', \mathbf{inc}\left((\delta_i)_{i<n}\right)) \,;\, (\delta_i)_{i<n}$$

- Case $n = \omega$. By hypothesis and inversion of EOmega we have:

$$\forall m < \omega. \quad M \parallel \mathbf{stop} \,;\, \left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<m} \stackrel{m}{\Longrightarrow} s'_m \,;\, (\delta_i)_{i<m}$$

  Applying the induction hypothesis gives:

$$\forall m < \omega. \quad \mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<m} \stackrel{m}{\Longrightarrow} \left(s'_m, \mathbf{inc}\left((\delta_i)_{i<m}\right)\right) \,;\, (\delta_i)_{i<m}$$

  Immediate application of EOmega yields what we wanted.

□

A.2 Metatheory

A.2.1 Preliminaries

We first prove some preliminary theorems.

Theorem A.1 (Type safety of increment truncation)**.** *If* $(\delta_i)_{i<n} : A \mathbin{@} n$ *and* $m \leqslant n$ *then* $(\delta_i)_{i<m} : A \mathbin{@} m$.

Proof. Let $m < n$, and $(\delta_i)_{i<n} : A \mathbin{@} n$. In the case where $m = 0$, then $(\delta_i)_{i<m} = \varepsilon$. Consequently, we have $(\delta_i)_{i<m} : A \mathbin{@} m$ by rule IStop, which proves what we wanted. In all the following, we thus suppose that $m > 0$. We proceed by induction over the typing judgement.

- *Rule* IStop. We have $n = 0$, and $(\delta_i)_{i<n} = \varepsilon$. Since $m > 0$ by the above and $m \leqslant n$, this case never happens (we have already demonstrated that the theorem always holds when $m = 0$).

- *Rule* IScalar. We have $n > 0$, $(\delta_i)_{i<n} = (c) \oplus (\mathbf{nil})_{i<n}$ and $A = \nu$. Since $m > 0$, rule IScalar gives that $(c) \oplus (\mathbf{nil})_{i<m}$. This proves what we wanted.

- *Rule* IProd. We have $n > 0$, $(\delta_i)_{i<n} = \left((\delta_i^1, \delta_i^2)\right)_{i<n}$ and $A = A_1 \times A_2$. Inversion of rule IProd gives that $(\delta_i^1)_{i<n} : A_1 \mathbin{@} n$ and $(\delta_i^2)_{i<n} : A_2 \mathbin{@} n$. The induction hypothesis gives us that $(\delta_i^1)_{i<m} : A_1 \mathbin{@} m$ and $(\delta_i^2)_{i<m} : A_2 \mathbin{@} m$. Immediate application of rule IProd allows us to conclude that $\left((\delta_i^1, \delta_i^2)\right)_{i<m} : A_1 \times A_2 \mathbin{@} m$. This proves what we wanted.

- *Rule* IInj. We have $n > 0$, $(\delta_i)_{i<n} = (\mathbf{inj}_j(\delta'_i))_{i<n}$ and $A = A_1 + A_2$. Inversion of rule IInj gives that $(\delta'_i)_{i<n} : A_j \ @ \ n$. The induction hypothesis gives us that $(\delta'_i)_{i<m} : A_j \ @ \ m$. Immediate application of rule IInj allows us to conclude that $(\mathbf{inj}_j(\delta'_i))_{i<m} : A_1 + A_2 \ @ \ m$. This proves what we wanted.

- *Rule* IWarp. We have $n > 0$, $(\delta_i)_{i<n} = \left(\triangleright(\delta'_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}$ and $A = p \Rrightarrow A'$. Inversion of rule IWarp gives that $(\delta'_i)_{i<p(n)} : A' \ @ \ p(n)$. Since $m \leqslant n \implies p(m) \leqslant p(n)$, the induction hypothesis gives us that $(\delta'_i)_{i<p(m)} : A' \ @ \ p(m)$. Immediate application of rule IWarp allows us to conclude that $\left(\triangleright(\delta'_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<m} : p \Rrightarrow A' \ @ \ m$. This proves what we wanted.

- *Rule* IClo. We have $n > 0$, $(\delta_i)_{i<n} = \left((x.M)\{\gamma_i\}\right)_{i<n}$ and $A = A \overset{S}{\Rightarrow} B$. Inversion of rule IClo gives that $\Gamma, x : A \vdash M : B \mid S$ and $(\gamma_i)_{i<n} : \Gamma \ @ \ n$. The induction hypothesis gives us that $(\gamma_i)_{i<m} : \Gamma \ @ \ m$. Immediate application of rule IClo allows us to conclude that $\left((x.M)\{\gamma_i\}\right)_{i<m} : A \overset{S}{\Rightarrow} B \ @ \ m$. This proves what we wanted.

- *Rule* IStream. We have $n > 0$, $(\delta_i)_{i<n} = (\delta_i^1 :: \delta_i^2)_{i<n}$ and $A = \mathsf{Str}\, A'$. Inversion of rule IStream gives that $(\delta_i^1)_{i<n} : A' \ @ \ n$ and $(\delta_i^2)_{i<n} : \mathbf{lat} \Rrightarrow \mathsf{Str}\, A' \ @ \ n$. The induction hypothesis gives us that $(\delta_i^1)_{i<m} : A' \ @ \ m$ and $(\delta_i^2)_{i<m} : \mathbf{lat} \Rrightarrow \mathsf{Str}\, A' \ @ \ m$. Immediate application of rule IStream allows us to conclude that $(\delta_i^1 :: \delta_i^2)_{i<m} : \mathsf{Str}\, A' \ @ \ m$. This proves what we wanted.

- *Rule* IBox. We have $n = \omega$. If $m = \omega$, we are done. Suppose $m < n$. Inversion of rule IBox gives that $\forall k < \omega.\ (\delta_i)_{i<k} : A \ @ \ k$. Since $m < \omega$, this especially gives us that $(\delta_i)_{i<m} : A \ @ \ m$. This proves what we wanted.

- *Rule* IEnv. We have $(\gamma_i)_{i<n} : \Gamma \ @ \ n$. Inversion IEnv gives that $\mathrm{dom}(\Gamma) = \mathrm{dom}\,(\gamma_i)$ for all $i < n$, and $(\gamma_i(x))_{i<n} : \Gamma(x) \ @ \ n$ for all $x \in \mathrm{dom}(\Gamma)$. Since $m \leqslant n$, we also have $\mathrm{dom}(\Gamma) = \mathrm{dom}\,(\gamma_i)$ for all $i < m$. Furtherore, the induction hypothesis applies, and gives $(\gamma_i(x))_{i<m} : \Gamma(x) \ @ \ m$ for all $x \in \mathrm{dom}(\Gamma)$. Immediate application of rule IEnv allows us to conclude that $(\gamma_i)_{i<m} : \Gamma \ @ \ m$. This proves what we wanted.

□

A.2.2 Determinism

In the following, Theorem A.2 and Lemma A.7 are proved by mutual induction.

Theorem A.2 (Determinism)**.** *If* $M \,\|\, s; (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'_1; (\delta_i^1)_{i<n}$ *and* $M \,\|\, s; (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'_2; (\delta_i^2)_{i<n}$*, then* $s'_1 = s'_2$ *and* $\delta_i^1 = \delta_i^2$ *for all* i*.*

Proof. Let us denote:

$$(\pi_1) \equiv M \parallel s\,; (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'_1\,; (\delta_i^1)_{i<n}$$
$$(\pi_2) \equiv M \parallel s\,; (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'_2\,; (\delta_i^2)_{i<n}$$

We proceed by structural induction over (π_1).

- Rule EZero. We have $n = 0$, $(\delta_i^1)_{i<0} = \varepsilon$ and $s'_1 = s$. Thus, substitution in (π_2) and immediate inversion yields that rule EZero applies and $(\delta_i^2)_{i<0} = \varepsilon$ and $s'_2 = s$. This gives

the following, which concludes the proof:

$$(\delta_i^1)_{i<0} = (\delta_i^2)_{i<0} = \varepsilon$$
$$s'1 = s_2' = s$$

- Rule ESTEP. We have $1 \leqslant n < \omega$, and:

$$M \parallel s\,;(\gamma_i)_{i<n-1} \stackrel{n-1}{\Longrightarrow} s_1''\,;(\delta_i^1)_{i<n-1} \tag{a}$$
$$M \parallel s_1''\,;\gamma_{n-1} \Longrightarrow s_1'\,;\delta_{n-1}^1 \tag{b}$$

Substitution of n in (π_2) and immediate inversion yields that rule ESTEP applies, and:

$$M \parallel s\,;(\gamma_i)_{i<n-1} \stackrel{n-1}{\Longrightarrow} s_2''\,;(\delta_i^2)_{i<n-1} \tag{c}$$
$$M \parallel s_2''\,;\gamma_{n-1} \Longrightarrow s_2'\,;\delta_{n-1}^2 \tag{d}$$

Application of the induction hypothesis over eq. (a) and eq. (c) gives:

$$\delta_i^1 = \delta_i^2 \text{ forall } i < n-1$$
$$s_1'' = s_2''$$

Application of the mutual induction hypothesis yields that Lemma A.7 applies on eq. (b) and eq. (d):

$$\delta_{n-1}^1 = \delta_{n-1}^2$$
$$s_1' = s_2'$$

This gives the following, which concludes the proof:

$$\delta_i^1 = \delta_i^2 \text{ forall } i < n$$
$$s_1' = s_2'$$

- Rule EOMEGA. We have $n = \omega$, and:

$$\forall m < \omega.\ M \parallel s\,;(\gamma_i)_{i<m} \stackrel{m}{\Longrightarrow} s_{1,m}''\,;(\delta_i^1)_{i<m} \tag{e}$$
$$s_1' = \mathbf{done} \tag{f}$$

Substitution of n in (π_2) and immediate inversion yields that rule ESTEP applies, and:

$$\forall m < \omega.\ M \parallel s\,;(\gamma_i)_{i<m} \stackrel{m}{\Longrightarrow} s_{2,m}''\,;(\delta_i^2)_{i<m} \tag{g}$$
$$s_2' = \mathbf{done} \tag{h}$$

Application of the induction hypothesis over eq. (e) and eq. (g) gives:

$$\delta_m^1 = \delta_m^2 \text{ forall } m < \omega$$
$$s_{1,m}'' = s_{2,m}''$$

The above especially gives $(\delta_i^1)_{i<\omega} = (\delta_i^2)_{i<\omega}$. Since $s_1' = s_2' = \mathbf{done}$, this concludes the proof.

□

Lemma A.7 (Determinism, single-step)**.** *If we have* $M \parallel s\,;\gamma \Longrightarrow s'_1\,;\delta_1$ *and* $M \parallel s\,;\gamma \Longrightarrow s'_2\,;\delta_2$ *then* $s'_1 = s'_2$ *and* $\delta_1 = \delta_2$*.*

Proof. Let us denote:

$$(\pi_1) \equiv M \parallel s\,;\gamma \Longrightarrow s'_1\,;\delta_1$$
$$(\pi_2) \equiv M \parallel s\,;\gamma \Longrightarrow s'_2\,;\delta_2$$

We proceed by structural induction over (π_1). For most cases, immediate inversion of the rule gives us values for M, s and γ. Then, by substitution, we notice that inversion of (pi_2) gives the exact same rule. Application of the induction hypothesis yields the desired result. The only cases which requires a slightly different argument are those of rules EBY and EBYI. Since their premises make use of the many-step judgement, we must use Theorem A.2 by using the mutual induction hypothesis. Due to the large number of rules and the many uninteresting cases, we will give a detailed proof only for those two cases, and rule EPAIR (as a more general example).

- Rule EPAIR. We have:

$$\frac{N \parallel s_N\,;\gamma \Longrightarrow s'_{1,N}\,;\delta_{1,N} \qquad P \parallel s_P\,;\gamma \Longrightarrow s'_{1,P}\,;\delta_{1,P}}{\underbrace{(N,P)}_{=M} \parallel \underbrace{(s_N,s_P)}_{=s}\,;\gamma \Longrightarrow \underbrace{(s'_{1,N},s'_{1,P})}_{=s'_1}\,;\underbrace{(\delta_{1,N},\delta_{1,P})}_{=\delta_1}} \tag{a}$$

  By immediate substitution of M, s and γ in (π_2), we obtain that its last deduction rule is also EPAIR. This gives:

$$\frac{N \parallel s_N\,;\gamma \Longrightarrow s'_{2,N}\,;\delta_{2,N} \qquad P \parallel s_P\,;\gamma \Longrightarrow s'_{2,P}\,;\delta_{2,P}}{\underbrace{(N,P)}_{=M} \parallel \underbrace{(s_N,s_P)}_{=s}\,;\gamma \Longrightarrow \underbrace{(s'_{2,N},s'_{2,P})}_{=s'_2}\,;\underbrace{(\delta_{2,N},\delta_{2,P})}_{=\delta_2}} \tag{b}$$

  Inversion of eq. (a) and eq. (b) yields:

$$N \parallel s_N\,;\gamma \Longrightarrow s'_{1,N}\,;\delta_{1,N} \quad P \parallel s_P\,;\gamma \Longrightarrow s'_{1,P}\,;\delta_{1,P}$$
$$N \parallel s_N\,;\gamma \Longrightarrow s'_{2,N}\,;\delta_{2,N} \quad P \parallel s_P\,;\gamma \Longrightarrow s'_{2,P}\,;\delta_{2,P}$$

  By the induction hypothesis, we have $s'_{1,N} = s'_{2,N}$, $s'_{1,P} = s'_{2,P}$, as well as $\delta_{1,N} = \delta_{2,N}$ and $\delta_{1,P} = \delta_{2,P}$. Thus, we have the following, which concludes the proof:

$$s'_1 = (s'_{1,N}, s'_{1,P}) = (s'_{2,N}, s'_{2,P}) = s'_2$$
$$\delta_1 = (\delta_{1,N}, \delta_{1,P}) = (\delta_{2,N}, \delta_{2,P}) = \delta_2$$

- Rule EBYI. We have:

$$\frac{N \parallel \mathbf{stop}\,;(\gamma_i)_{i<p(1)} \overset{p(1)}{\Longrightarrow} s''_1\,;(\delta^1_i)_{i<p(1)}}{\underbrace{N \text{ by } p}_{=M} \parallel \underbrace{\mathbf{stop}}_{=s}\,;\underbrace{\triangleright(\gamma_i)_{i<p(1)}}_{=\gamma} \Longrightarrow \underbrace{\mathbf{w}\,(p,s''_1)}_{=s'_1}\,;\underbrace{\triangleright(\delta^1_i)_{i<p(1)}}_{=\delta_1}} \tag{c}$$

  By immediate substitution of M, s and γ in (π_2), we obtain that its last deduction rule is

also EByI. This gives:

$$\frac{N \parallel \mathbf{stop}\,;(\gamma_i)_{i<p(1)} \xrightarrow{p(1)} s_2'';(\delta_i^2)_{i<p(1)}}{\underbrace{N \text{ by } p}_{=M} \parallel \underbrace{\mathbf{stop}}_{=s}\,;\underbrace{\triangleright(\gamma_i)_{i<p(1)}}_{=\gamma} \Longrightarrow \underbrace{\mathbf{w}\,(p,s_2'')}_{=s_2'}\,;\underbrace{\triangleright(\delta_i^2)_{i<p(1)}}_{=\delta_2}} \tag{d}$$

Inversion of eq. (c) and eq. (d) yields:

$$N \parallel \mathbf{stop}\,;(\gamma_i)_{i<p(1)} \xrightarrow{p(1)} s_1'';(\delta_i^1)_{i<p(1)}$$
$$N \parallel \mathbf{stop}\,;(\gamma_i)_{i<p(1)} \xrightarrow{p(1)} s_2'';(\delta_i^2)_{i<p(1)}$$

By the mutual induction hypothesis, Theorem A.2 applies and gives $s_1'' = s_2''$, and $\delta_i^1 = \delta_i^2$ for all $i < p(1)$. Thus, we have the following, which concludes the proof:

$$s_1' = \mathbf{w}\,(p,s_1'') = \mathbf{w}\,(p,s_2'') = s_2'$$
$$\delta_1 = \triangleright(\delta_i^1)_{i<p(1)} = \triangleright(\delta_i^2)_{i<p(1)} = \delta_2$$

- Rule EBy. We have:

$$\frac{N \parallel s_N\,;(\gamma_i)_{i<\partial p(n)} \xrightarrow{\partial p(n)} s_1'';(\delta_i^1)_{i<\partial p(n)}}{\underbrace{N \text{ by } p}_{=M} \parallel \underbrace{\mathbf{w}\,(p,s_N)}_{=s}\,;\underbrace{\triangleright(\gamma_i)_{i<\partial p(n)}}_{=\gamma} \Longrightarrow \underbrace{\mathbf{w}\,(p,s_1'')}_{=s_1'}\,;\underbrace{\triangleright(\delta_i^1)_{i<\partial p(n)}}_{=\delta_1}} \tag{e}$$

By immediate substitution of M, s and γ in (π_2), we obtain that its last deduction rule is also EBy. This gives:

$$\frac{N \parallel s_N\,;(\gamma_i)_{i<\partial p(n)} \xrightarrow{\partial p(n)} s_2'';(\delta_i^2)_{i<\partial p(n)}}{\underbrace{N \text{ by } p}_{=M} \parallel \underbrace{\mathbf{w}\,(p,s_N)}_{=s}\,;\underbrace{\triangleright(\gamma_i)_{i<\partial p(n)}}_{=\gamma} \Longrightarrow \underbrace{\mathbf{w}\,(p,s_2'')}_{=s_2'}\,;\underbrace{\triangleright(\delta_i^2)_{i<\partial p(n)}}_{=\delta_2}} \tag{f}$$

Inversion of eq. (e) and eq. (f) yields:

$$N \parallel s_N\,;(\gamma_i)_{i<\partial p(n)} \xrightarrow{\partial p(n)} s_1'';(\delta_i^1)_{i<\partial p(n)}$$
$$N \parallel s_N\,;(\gamma_i)_{i<\partial p(n)} \xrightarrow{\partial p(n)} s_2'';(\delta_i^2)_{i<\partial p(n)}$$

By the mutual induction hypothesis, Theorem A.2 applies and gives $s_1'' = s_2''$, and $\delta_i^1 = \delta_i^2$ for all $i < p(1)$. Thus, we have the following, which concludes the proof:

$$s_1' = \mathbf{w}\,(p,s_1'') = \mathbf{w}\,(p,s_2'') = s_2'$$
$$\delta_1 = \triangleright(\delta_i^1)_{i<p(1)} = \triangleright(\delta_i^2)_{i<p(1)} = \delta_2$$

□

A.2.3 Type safety

In order to simplify our proof, we deal with the base case $n = 0$ in a separate lemma.

Lemma A.8 (Type safety at $n = 0$)**.** *Let* $\Gamma \vdash M : A \mid S$.

$$\textit{If we have } \begin{cases} (\gamma_i)_{i<0} : \Gamma \ @ \ 0 \\ M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<0} \overset{0}{\Longrightarrow} s \,;\, (\delta_i)_{i<0} \end{cases} \textit{— then } \begin{cases} (\delta_i)_{i<0} : A \ @ \ 0 \\ s : S \,\mathfrak{G}\, 0 \end{cases}$$

Proof. Since $n = 0$, $(\gamma_i)_i = \varepsilon$. By EZero, we have $M \parallel \mathbf{stop} \,;\, \varepsilon \overset{0}{\Longrightarrow} \mathbf{stop} \,;\, \varepsilon$. Thus, $(\delta_i)_i = \varepsilon$ and $s = \mathbf{stop}$. Rules IStop and SStop yield $(\delta_i)_i = \varepsilon : A \ @ \ 0$ and $s = \mathbf{stop} : S \,\mathfrak{G}\, 0$ – which is what we wanted. □

Let us restate Theorem A.3:

Theorem A.3 (Type safety)**.** *Let* $\Gamma \vdash M : A \mid S$*, and* $n \leqslant \omega$*. For all* $(\gamma_i)_{i<n} : \Gamma \ @ \ n$ *and* $(\delta_i)_{i<n}$ *such that* $M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s \,;\, (\delta_i)_{i<n}$*, we have* $(\delta_i)_{i<n} : A \ @ \ n$ *and* $s : S \,\mathfrak{G}\, n$.

Proof. Let M be a well-typed term such that $(\pi) : \Gamma \vdash M : A \mid S$. By induction over (π).

Almost all cases can be dealt with by inverting the typing derivation, obtaining the shape of the term, observing that subject reduction holds for subterms due to the induction hypothesis, and using these hypotheses to conclude that the property holds for the term. Thus, we will focus on three nontrivial examples – By, Rec, and Delay.

- Case Rec. We have $\Gamma \vdash \mathsf{rec}\, x^A.\, M : A \mid S \times \int A$. By inversion of Rec, we obtain:

$$\Gamma,\, x : \mathbf{lat} \Rrightarrow A \vdash M : A \mid S \tag{a}$$

  Now, we proceed by induction over $n \in \omega + 1$.

  - $\underline{n = 0}$. True by Lemma A.8.
  - $\underline{1 \leqslant n < \omega}$. Suppose that we have:

$$(\gamma_i)_{i<n} : \Gamma \ @ \ n \tag{b}$$

$$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s \,;\, (\delta_i)_{i<n} \tag{c}$$

    By inversion of EStep in (c), we also have:

$$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n-1} \overset{n-1}{\Longrightarrow} s_I \,;\, (\delta_i)_{i<n-1} \tag{d}$$

    By Theorem A.1, we have $(\gamma_i)_{i<n-1} : \Gamma \ @ \ n-1$. Thus, the induction hypothesis over n applies to (d), and immediate application of IWarp yields:

$$\begin{gathered} (\delta_i)_{i<n-1} : A \ @ \ n-1 \\ (\triangleright\varepsilon) \oplus \big(\triangleright(\delta_{i-1})\big)_{1\leqslant i<n} : \mathbf{lat} \Rrightarrow A \ @ \ n \end{gathered} \tag{e}$$

    Applying IEnv to (b) and (e) gives:

$$\big(\gamma_0\big[\triangleright\varepsilon/x\big]\big) \oplus \big(\gamma_i\big[\triangleright(\delta_{i-1})/x\big]\big)_{1\leqslant i<n} : \mathbf{lat} \Rrightarrow A \ @ \ n \tag{f}$$

    Finally, we apply Lemma A.2 to (d) – this yields:

$$\exists s'.\ \begin{cases} M \parallel \mathbf{stop} \,;\, \big(\gamma_0\big[\triangleright\varepsilon/x\big]\big) \oplus \big(\gamma_i\big[\triangleright(\delta_{i-1})/x\big]\big)_{1\leqslant i<n} \overset{n}{\Longrightarrow} s' \,;\, (\delta_i)_{i<n} \\ s = \Big(s', \mathbf{inc}\,\big((\delta_i)_{i<n}\big)\Big) \end{cases} \tag{g}$$

Since, by (a), and (f):

$$\Gamma,\, x{:}\mathbf{lat} \Rrightarrow A \vdash M : A \mid S$$
$$\left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<n} : \mathbf{lat} \Rrightarrow A \mathbin{@} n$$

Then, the induction hypothesis applies to (g) and yields:

$$(\delta_i)_{i<n} : A \mathbin{@} n$$
$$s' : S \mathfrak{G} n$$

Immediate application of SPair and SIncr gives us:

$$(\delta_i)_{i<n} : A \mathbin{@} n$$
$$s = (s', \mathbf{inc}\,((\delta_i)_{i<n})) : S \times \textstyle\int A \mathfrak{G} n$$

Which is what we wanted.

– $\underline{n = \omega}$. Suppose that we have:

$$(\gamma_i)_{i<\omega} : \Gamma \mathbin{@} \omega \tag{h}$$
$$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop}\,;(\gamma_i)_{i<\omega} \overset{\omega}{\Longrightarrow} s\,;(\delta_i)_{i<\omega} \tag{i}$$

By inversion of EOmega in (i), we have:

$$s = \mathbf{done} \tag{j}$$
$$\forall m. \quad \mathsf{rec}\, x^A.\, M \parallel \mathbf{stop}\,;(\gamma_i)_{i<m} \overset{m}{\Longrightarrow} s_m\,;(\delta_i)_{i<m} \tag{k}$$

By SDone, we obtain :

$$s = \mathbf{done} : S \times \textstyle\int A \mathfrak{G} \omega \tag{l}$$

By applying Theorem A.1 on (h), then $(\gamma_i)_{i<m} : \Gamma \mathbin{@} m$ for all m. Consequently, the induction hypothesis applies on (k) and yields:

$$\forall m.\ (\delta_i)_{i<m} : A \mathbin{@} m$$

Which by immediate application of IBox yields:

$$(\delta_i)_{i<\omega} : A \mathbin{@} \omega \tag{m}$$

In the end, (l) and (m) prove what we wanted.

- Case By. We have $p \Rrightarrow \Gamma \vdash M \,\mathsf{by}\, p : p \Rrightarrow A \mid p \Rrightarrow S$. By inversion of By, we obtain:

$$\Gamma \vdash M : A \mid S \tag{a}$$

If $n = 0$, Lemma A.8 proves what we want. Thus, suppose that $n > 0$.

Let:

$$(\gamma_i)_{i<n} : p \Rrightarrow \Gamma \mathbin{@} n \tag{b}$$
$$M \,\mathsf{by}\, p \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s\,;(\delta_i)_{i<n} \tag{c}$$

Inversion of IEnv and IWarp in (b) gives:

$$\exists (\gamma'_i)_{i<p(n)}.\ \begin{cases} (\gamma'_i)_{i<p(n)} : \Gamma\ @\ p(n) \\ \gamma_i = \triangleright(\gamma'_j)_{p(i)\leqslant j<p(i+1)} \text{ for all } i<n \end{cases} \tag{d}$$

Substitution of (d) in (c) and immediate application of Lemma A.1 yields:

$$\exists s'.\ \exists (\delta'_i)_{i<p(n)}.\ \begin{cases} M \parallel \mathbf{stop}\,;(\gamma'_i)_{i<p(n)} \overset{p(n)}{\Longrightarrow} s'\,;(\delta'_i)_{i<p(n)} \\ \delta_i = \triangleright(\delta'_j)_{p(i)<j\leqslant p(i+1)} \text{ for all } i<n \\ s = \begin{cases} \mathbf{w}\,(p,s') & \text{if } n<\omega \\ \mathbf{done} & \text{elsewise} \end{cases} \end{cases} \tag{e}$$

Since (d) gives that $(\gamma'_i)_{i<p(n)} : \Gamma\ @\ p(n)$, and (a) gives that M is well-typed – the induction hypothesis applies on (e) and gives:

$$(\delta'_i)_{i<p(n)} : A\ @\ p(n)$$
$$s' : S\ \mathfrak{G}\ p(n)$$

Immediate application of IWarp, and SWarp if $n<\omega$ or SDone if $n=\omega$ yield:

$$\left(\triangleright(\delta'_j)_{p(i)<j\leqslant p(i+1)}\right)_{i<n} : p \Rrightarrow A\ @\ n$$

$$\begin{cases} \mathbf{w}\,(p,s') : p \Rrightarrow S\ \mathfrak{G}\ n & \text{if } n<\omega \\ \mathbf{done} : p \Rrightarrow S\ \mathfrak{G}\ \omega & \text{elsewise} \end{cases}$$

Substitution of both equalities in (e) yield what we wanted.

- Case Delay. We have $\Gamma \vdash \mathsf{delay}^{q\leqslant p}(M) : q \Rrightarrow A \mid S \times \int(p \Rrightarrow A)$. By inversion of Delay, we obtain:

$$\Gamma \vdash M : p \Rrightarrow A \mid S \text{ and } q \leqslant p \tag{a}$$

If $n=0$, Lemma A.8 proves what we want. Thus, suppose that $n>0$. Let:

$$(\gamma_i)_{i<n} : \Gamma\ @\ n \tag{b}$$
$$\mathsf{delay}^{q\leqslant p}(M) \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s\,;(\delta_i)_{i<n} \tag{c}$$

Application of Lemma A.3 yields:

$$\exists s'.\ \exists (\delta'_i)_{i<p(n)}.\ \begin{cases} M \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s'\,;\left(\triangleright(\delta'_i)_{p(i)<j\leqslant p(i+1)}\right)_{i<n} \\ \delta_i = \triangleright(\delta'_j)_{q(i)<j\leqslant q(i+1)} \text{ for all } i<n \\ s = \begin{cases} \left(s', \mathbf{inc}\left(\left(\triangleright(\delta'_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}\right)\right) & \text{if } n<\omega \\ \mathbf{done} & \text{otherwise} \end{cases} \end{cases} \tag{d}$$

Since (b) gives that $(\gamma_i)_{i<n} : \Gamma\ @\ n$, and (a) gives that M is well-typed – the induction hypothesis applies on (d) and gives:

$$\left(\triangleright(\delta'_i)_{p(i)<j\leqslant p(i+1)}\right)_{i<n} : p \Rrightarrow A\ @\ n \tag{e}$$
$$s' : S\ \mathfrak{G}\ n \tag{f}$$

Inversion of IWarp in (e) gives that $(\delta'_i)_{i<p(n)} : A \mathbin{@} p(n)$. Since the typing of increments is monotonic (cf. Theorem A.1), and $q(n) \leqslant p(n)$ (for we have $q \leqslant p$ by (a)) – we have that $(\delta'_i)_{i<q(n)} : A \mathbin{@} q(n)$.

Immediate application of IWarp proves that:

$$\left(\triangleright(\delta'_i)_{q(i)<j\leqslant q(i+1)}\right)_{i<n} : q \Rrightarrow A \mathbin{@} n \tag{g}$$

If $n = \omega$, then $s = \mathbf{done}$, and SDone immediately gives that $s : S \mathfrak{G} \omega$. Otherwise, if $n < \omega$, application of SIncr on (e) followed immediate application of SPair with (f) give:

$$\left(s', \mathbf{inc}\left(\left(\triangleright(\delta'_i)_{p(i)<j\leqslant p(i+1)}\right)_{i<n}\right)\right) : S \times \textstyle\int (p \Rrightarrow A) \mathfrak{G} n \tag{h}$$

Substitution of s and δ_i in (h) and (g) yield what we wanted.

□

A.2.4 Totality

Theorem A.4 (Totality)**.** *Let $\Gamma \vdash M : A \mid S$, and $n \leqslant \omega$. For all $(\gamma_i)_{i<n} : \Gamma \mathbin{@} n$, there exists s and $(\delta_i)_{i\leq n}$ such that $M \parallel \mathbf{stop} ; (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s ; (\delta_i)_{i<n}$.*

Proof. Let M be a well-typed term such that $(\pi) : \Gamma \vdash M : A \mid S$. By induction over (π).

- Var. Inversion of this rule gives $M = x$ and $x \in \mathrm{dom}(\Gamma)$ – (π) rewrites to :

$$\underbrace{\Gamma', x{:}A}_{\Gamma} \vdash x : A \mid \mathbb{1} \tag{a}$$

  We proceed by case over n.

  - $\underline{n = 0}$. By immediate application of EZero.
  - $\underline{n > 0}$. Suppose we have $(\gamma_i)_{i<n} : \Gamma \mathbin{@} n$. Inversion of IEnv yields that for all $i < n$, we have $\mathrm{dom}(\gamma_i) = \mathrm{dom}(\Gamma)$.
    Since (a) gives $x \in \mathrm{dom}(\Gamma)$, we have that for all $i < n$, $x \in \mathrm{dom}(\gamma_i)$. Thus, Lemma A.4 applies, which proves what we wanted.

- Rec. We have $\Gamma \vdash \mathsf{rec}\, x^A.\, M : A \mid S \times \int A$. Immediate inversion of Rec gives:

$$\Gamma, x{:}\mathbf{lat} \Rrightarrow A \vdash M : A \mid S \tag{a}$$

  By induction over n.

  - $\underline{n = 0}$. By immediate application of EZero.
  - $\underline{1 \leqslant n < \omega}$. Suppose we have $(\gamma_i)_{i<n} : \Gamma \mathbin{@} n$. Since increment typing is monotonic (cf. Theorem A.1), it also gives $(\gamma_i)_{i<n-1} : \Gamma \mathbin{@} n - 1$. Thus, the induction hypothesis over n applies and gives:

$$\exists s_I.\ \exists (\delta_i)_{i<n-1}.\ \mathsf{rec}\, x^A.\, N \parallel \mathbf{stop} ; (\gamma_i)_{i<n-1} \overset{n-1}{\Longrightarrow} s_I ; (\delta_i)_{i<n-1} \tag{b}$$

By applying subject reduction (cf. Theorem A.3) on (b), then immediate application of IWarp and IEnv, we respectively obtain:

$$(\delta_i)_{i<n-1} : A \mathbin{@} n-1$$
$$(\triangleright\varepsilon) \oplus (\triangleright(\delta_{i-1}))_{1\leqslant i<n} : \mathbf{lat} \Rrightarrow \Gamma \mathbin{@} n$$
$$\left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<n} : \Gamma, x{:}\mathbf{lat} \Rrightarrow A \mathbin{@} n$$

This allows us to apply on the subterm N the induction hypothesis over (π):

$$N \parallel \mathbf{stop} \,;\, \left(\gamma_0\left[\triangleright\varepsilon/x\right]\right) \oplus \left(\gamma_i\left[\triangleright(\delta_{i-1})/x\right]\right)_{1\leqslant i<n} \overset{n}{\Longrightarrow} s' \,;\, (\delta_i)_{i<n}$$

Immediate application of Lemma A.6 proves what we wanted:

$$\mathsf{rec}\, x^A.\, N \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} \left(s', \mathbf{inc}\left((\delta_i)_{i<n}\right)\right) \,;\, (\delta_i)_{i<n} \tag{c}$$

– $\underline{n = \omega}$. Suppose we have $(\gamma_i)_{i<\omega} : \Gamma \mathbin{@} \omega$. Inversion of IBox and IEnv gives that for all $m < \omega$, we have $(\gamma_i)_{i<m} : \Gamma \mathbin{@} m$. Thus, the induction hypothesis over n applies and gives:

$$\forall m < \omega. \quad \exists s'_m. \quad \mathsf{rec}\, x^A.\, M \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<m} \overset{m}{\Longrightarrow} s'_m \,;\, (\delta_i)_{i<m}$$

Immediate application of EOmega yields what we wanted.

• $\boxed{\text{By}}$. We have $p \Rrightarrow \Gamma \vdash M \mathsf{\ by\ } p : p \Rrightarrow A \mid p \Rrightarrow S$. Immediate inversion of By gives us:

$$\Gamma \vdash M : A \mid S \tag{a}$$

We proceed by case over n.

– $\underline{n = 0}$. By immediate application of EZero.

– $\underline{n > 0}$. Suppose we have $(\gamma_i)_{i<n} : p \Rrightarrow \Gamma \mathbin{@} n$. Inversion of IWarp and IEnv gives:

$$\exists (\gamma'_i)_{i<p(n)}. \quad \begin{cases} (\gamma'_i)_{i<p(n)} : A \mathbin{@} p(n) \\ \gamma_i = \triangleright\left(\gamma'_j\right)_{p(i)\leqslant j<p(i+1)} \text{ for all } i < n \end{cases} \tag{b}$$

Thus, the induction hypothesis applies on (a) and yields:

$$M \parallel \mathbf{stop} \,;\, (\gamma'_i)_{i<p(n)} \overset{p(n)}{\Longrightarrow} s' \,;\, (\delta_i)_{i<p(n)}$$

Applying Lemma A.5 proves what we wanted:

$$M \mathsf{\ by\ } p \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} \mathbf{w}\,(p, s') \,;\, \left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}$$

□

A.3 Correctness

A.3.1 Preliminaries

Recall Lemma A.9, presented in Section 3.3.

Lemma A.9. *If $M\,;E\,;x\,;V \Uparrow_i^k V'$ then for all j such that $i \leqslant j \leqslant k$, there is V'' such that $M\,;\lfloor E\rfloor_j\,;x\,;V \Uparrow_i^j V''$ and $M\,;E\,;x\,;V'' \Uparrow_j^k V'$.*

Proof. Let's proceed by induction over $k-i$.

- Base case. We have $k-i=0$, which in turn gives gives $i=j=k$.
  We have $N\,;E\,;x\,;V \Uparrow_i^k V'$. By PIFinish, this gives $V=V'$. Set $V''=V$. Since $i=j$, PIFinish yields $N\,;E\,;x\,;V \Uparrow_i^j V''$. Since $j=k$, PIFinish yields $N\,;E\,;x\,;V \Uparrow_j^k V'$. This concludes our proof for $k=0$

- Inductive case. Set $k+1-i=n$. Suppose the theorem is true for every value of $k-i \leqslant n$. If $i=j$, the theorem is trivially true, since rule PIFinish gives $V''=V$. Thus, in all the following, we will suppose that $j>i$.

  Suppose that $N\,;E\,;x\,;V \Uparrow_i^{k+1} V'$. By inversion of rule PIIter, we obtain:

  $$\begin{cases} N\,;\lfloor E\rfloor_{i+1}\left[\mathbf{w}\,(\mathbf{lat},V)\,/x\right] \Downarrow_{i+1} V_0 \\ N\,;E\,;x\,;V_0 \Uparrow_{i+1}^{k+1} V' \end{cases}$$

  Since $i+1 \leqslant j$ and $(k+1)-(i+1)=n$, applying the induction hypothesis on $N;E;x;V_0 \Uparrow_{i+1}^{k+1} V'$ is allowed, and yields:

  $$\begin{cases} N\,;E\,;x\,;V_0 \Uparrow_{i+1}^{j} V'' \\ N\,;E\,;x\,;V'' \Uparrow_j^{k+1} V' \end{cases}$$

  Notice that $i+1 \leqslant j$ gives that $i<j$. Now, consider the following proof tree:

  $$\frac{i<j \qquad N\,;\lfloor E\rfloor_{i+1}\left[\mathbf{w}\,(\mathbf{lat},V)\,/x\right] \Downarrow_{i+1} V_0 \qquad N\,;E\,;x\,;V_0 \Uparrow_{i+1}^{j} V''}{N\,;E\,;x\,;V \Uparrow_i^j V''}$$

  Since all hypotheses hold, this proves that $N\,;E\,;x\,;V \Uparrow_i^j V''$.
  Since we also had $N\,;E\,;x\,;V'' \Uparrow_j^{k+1} V'$, this concludes the proof. □

This lemma immediately implies the following result:

Corollary A.4.1. *If we have $M\,;E\,;x\,;\mathbf{stop} \Uparrow_0^{n+1} V$, then there exists a value V' such that $M\,;\lfloor E\rfloor_n\,;x\,;\mathbf{stop} \Uparrow_0^n V'$ and $M\,;E\left[\mathbf{w}\,(\mathbf{lat},V')\,/x\right] \Downarrow_{n+1} V$.*

Proof. By Lemma A.9, with $i=0$, $j=n$, $k=n+1$. Then, notice that the iteration judgement over $n \leqslant n+1$ is equivalent to performing just one step of the reduction $\Rightarrow$. □

We then give the proof of Lemma A.11, which we will need in order to prove the full adequacy lemma. In order to do so, we need the following lemma, which handles the case of truncating a coherent slicing to a size of zero.

Lemma A.10 (Coherence of truncation at $n=0$)**.** *If $(\delta_i)_{i<n} \models_n^A V$, then $(\delta_i)_{i<0} \models_0^A \lfloor V\rfloor_0$.*

Proof. Since $\lfloor V\rfloor_0 = \mathbf{stop}$ and $(\delta_i)_{i<0}=\varepsilon$, the proof immediately follows from the definition of the logical relation. □

We now prove the general case.

Lemma A.11 (Coherence of truncation)**.** *If* $(\delta_i)_{i<n} \models^A_n V$ *and* $m \leqslant n$*, then* $(\delta_i)_{i<m} \models^A_m \lfloor V \rfloor_m$.

Proof. We proceed by induction over the pair (A, n) using the well-founded lexicographical product order $<_{\mathsf{ty}} \times_{\mathsf{lex}} <_{\omega+1}$ between the strict subtree ordering on types $<_{\mathsf{ty}}$ and the canonical strict ordering $<_{\omega+1}$ on $\omega + 1$.

- Case $(A, 0)$. Let V, and $(\delta_i)_{i<n}$ such that $(\delta_i)_{i<n} \models^A_n V$. Since $n = 0$, we have $V = \mathbf{stop}$ and $(\delta_i)_i = \varepsilon$. Let $m \leqslant n$. This means that $m = 0$, which in turn implies that $\lfloor V \rfloor_m = \mathbf{stop}$. Thus, $(\delta_i)_{i<m} \models^A_m V$.

- Inductive case (A, n) with $1 \leqslant n < \omega$. The induction hypothesis gives us that the property holds for each (A', n') such that $A' <_{\mathsf{ty}} A \vee (A' = A \wedge n' < n)$.

  Let V, and $(\delta_i)_{i<n}$ such that $(\delta_i)_{i<n} \models^A_n V$. We now proceed by case over A:

  - $\boxed{A = \nu}$. By definition of the logical relation in Figure 12, we have $V = c$, $\delta_0 = c \in S_\nu$, and $\delta_i = \mathbf{nil}$ for all $i > 0$. Let $m \leqslant n$. If $m = 0$, Lemma A.10 proves what we wanted. Thus, suppose that $1 \leqslant m < \omega$. By definition, we have that:

    $$(c) \oplus (\mathbf{nil})_{1 \leqslant i < m} \models^\nu_m c$$

    Since $\lfloor V \rfloor_m = \lfloor c \rfloor_m = c$, we obtained what we wanted.

  - $\boxed{A = A_1 \times A_2}$. By definition of the logical relation in Figure 12, we have:

    $$\left.\begin{array}{r} \forall i < n.\ \delta_i = (\delta^1_i, \delta^2_i) \\ V = (V_1, V_2) \end{array}\right\} \text{ st. } \begin{cases} (\delta^1_i)_{i<n} \models^{A_1}_n V_1 \\ (\delta^2_i)_{i<n} \models^{A_2}_n V_2 \end{cases}$$

    Let $m \leqslant n$. If $m = 0$, Lemma A.10 proves what we wanted. Thus, suppose that $1 \leqslant m < \omega$. Since A_1 and A_2 are structural subtypes of A, the induction hypothesis applies and gives:

    $$(\delta^1_i)_{i<m} \models^{A_1}_m \lfloor V_1 \rfloor_m \qquad (\delta^2_i)_{i<m} \models^{A_2}_m \lfloor V_2 \rfloor_m$$

    By definition of the logical relation, this gives:

    $$(\delta^1_i, \delta^2_i)_{i<m} \models^{A_1 \times A_2}_m (\lfloor V_1 \rfloor_m, \lfloor V_2 \rfloor_m)$$

    Substitution of $\lfloor V \rfloor_m = (\lfloor V_1 \rfloor_m, \lfloor V_2 \rfloor_m)$ and $\delta_i = (\delta^1_i, \delta^2_i)$ in the above proves what we wanted.

  - $\boxed{A = A_1 + A_2}$. By definition of the logical relation in Figure 12, we have:

    $$\left.\begin{array}{r} \forall i < n.\ \delta_i = \mathbf{inj}_k(\delta'_i) \\ V = \mathbf{inj}_k\,(V') \end{array}\right\} \text{ st. } (\delta'_i)_{i<n} \models^{A_k}_n V'$$

    Let $m \leqslant n$. If $m = 0$, Lemma A.10 proves what we wanted. Thus, suppose that $1 \leqslant m < \omega$. Since A_1 and A_2 are structural subtypes of A, the induction hypothesis applies and gives:

    $$(\delta'_i)_{i<m} \models^{A_k}_m \lfloor V' \rfloor_m$$

By definition of the logical relation, this gives:

$$(\mathbf{inj}_k(\delta'_i))_{i<m} \models_m^{A_1+A_2} \lfloor V' \rfloor_m$$

Substitution of $\lfloor V \rfloor_m = \mathbf{inj}_k(\lfloor V' \rfloor_m)$ and $\delta_i = \mathbf{inj}_k(\delta'_i)$ in the above proves what we wanted.

– $\boxed{A = p \Rrightarrow A'}$ By definition of the logical relation in Figure 12, we have:

$$\left.\begin{array}{r} \forall i < n.\ \delta_i = \triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)} \\ V = \mathbf{w}\,(p, V') \end{array}\right\} \text{ st. } (\delta'_i)_{i<p(n)} \models_{p(n)}^{A'} V'$$

Let $m \leqslant n$. Notice that $p(m) \leqslant p(n)$. If $m = 0$, Lemma A.10 proves what we wanted. Thus, suppose that $1 \leqslant m < \omega$. Since A_1 and A_2 are structural subtypes of A, the induction hypothesis applies and gives:

$$(\delta'_i)_{i<p(m)} \models_{p(m)}^{A'} \lfloor V' \rfloor_{p(m)}$$

By definition of the logical relation, this gives:

$$\left(\triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<m} \models_m^{p \Rrightarrow A'} \mathbf{w}\left(p, \lfloor V' \rfloor_{p(m)}\right)$$

Substitution of $\lfloor V \rfloor_m = \mathbf{w}\left(p, \lfloor V \rfloor_{p(m)}\right)$ and $\delta_i = \triangleright(\delta_j)_{p(i)\leqslant j<p(i+1)}$ in the above proves what we wanted.

– $\boxed{A = \mathsf{Str}\,A'}$. By definition of the logical relation in Figure 12, we have:

$$\left.\begin{array}{r} \forall i < n.\ \delta_i = \delta'_i :: \triangleright(\delta''_j)_{i-1\leqslant j<i} \\ V = V' :: V'' \end{array}\right\} \text{ st. } \begin{cases} (\delta'_i)_{i<n} \models_n^{A'} V' \\ (\delta''_i)_{i<n-1} \models_{n-1}^{\mathsf{Str}\,A'} V'' \end{cases}$$

Let $m \leqslant n$. If $m = 0$, Lemma A.10 proves what we wanted. Thus, suppose that $1 \leqslant m < \omega$. Since $(A', n) <_{\mathsf{lex}} (\mathsf{Str}\,A', n)$ and $(\mathsf{Str}\,A', n-1) <_{\mathsf{lex}} (\mathsf{Str}\,A', n)$, the induction hypothesis applies and gives:

$$(\delta'_i)_{i<m} \models_m^{A'} \lfloor V' \rfloor_m \qquad (\delta''_i)_{i<m-1} \models_{m-1}^{\mathsf{Str}\,A'} \lfloor V'' \rfloor_{m-1}$$

By definition of the logical relation, this gives:

$$\left(\delta'_i :: \triangleright(\delta''_j)_{i-1\leqslant j<i}\right)_{i<m} \models_m^{\mathsf{Str}} \lfloor V' \rfloor_m :: \lfloor V'' \rfloor_{m-1}$$

Substitution of $\lfloor V \rfloor_m = \lfloor V' \rfloor_m :: \lfloor V'' \rfloor_{m-1}$ and $\delta_i = \delta'_i :: \triangleright(\delta''_j)_{i-1\leqslant j<i}$ in the above proves what we wanted.

– $\boxed{A = A_1 \overset{S}{\Rightarrow} A_2}$. By definition of the logical relation in Figure 12, we have:

$$\begin{array}{r} \forall i < n.\ \delta_i = (x.M)\{\gamma_i\} \\ V = (x.N)\{E\} \end{array}$$

Such that for all $k \leqslant n$:

$$\forall (\delta'_i)_{i<k} \models_k^{A_1} V', \exists s, (\delta''_i)_{i<k}, V'', \begin{cases} M \parallel \mathbf{stop} \,;\, \left(\gamma_i\left[\delta'_i/x\right]\right)_{i<k} \overset{k}{\Longrightarrow} s \,;\, (\delta''_i)_{i<k} \\ N \,;\, \lfloor E \rfloor_k \left[V'/x\right] \Downarrow_k V'' \\ (\delta''_i)_{i<k} \models_k^{A_2} V'' \end{cases} \tag{a}$$

Let $m \leqslant n$. If $m = 0$, Lemma A.10 proves what we wanted. Thus, suppose that $1 \leqslant m < \omega$. Since $m \leqslant n$, if the property (a) holds for all $k \leqslant n$, it also holds for all $k \leqslant m$. Due to the functoriality of truncation, we have that $\lfloor \lfloor E \rfloor_m \rfloor_k = \lfloor E \rfloor_{\min(k,m)}$. However, when $k \leqslant m$, this rewrites as $\lfloor \lfloor E \rfloor_m \rfloor_k = \lfloor E \rfloor_k$. All of these arguments thus yield that, for all $k \leqslant m$:

$$\forall (\delta'_i)_{i<k} \models_k^{A_1} V', \exists s, (\delta''_i)_{i<k}, V'', \begin{cases} M \parallel \mathbf{stop} \,;\, \left(\gamma_i\left[\delta'_i/x\right]\right)_{i<k} \overset{k}{\Longrightarrow} s \,;\, (\delta''_i)_{i<k} \\ N \,;\, \lfloor \lfloor E \rfloor_m \rfloor_k \left[V'/x\right] \Downarrow_k V'' \\ (\delta''_i)_{i<k} \models_k^{A_2} V'' \end{cases} \tag{b}$$

This is precisely the right hand side of the definition of the logical relation on functions. Thus, applying said relation to the above yields:

$$((x.M)\{\gamma_i\})_{i<m} \models_m^{A \overset{S}{\Rightarrow} B} (x.N)\{\lfloor E \rfloor_m\}$$

Immediate substitution of $\lfloor V \rfloor_m = (x.M)\{\lfloor E \rfloor_m\}$ and $\delta_i = (x.M)\{\gamma_i\}$ in the above proves what we wanted.

- Inductive case (A, ω). The induction hypothesis gives us that the property holds for each (A', n') such that $A' <_{\mathsf{ty}} A \vee (A' = A \wedge n' < \omega)$.

  Let V, and $(\delta_i)_{i<\omega}$ such that $(\delta_i)_{i<\omega} \models_\omega^A V$. By definition of the logical relation, we have that:

  $$V = \mathbf{box}(M)\{E\} \text{ st. for all } m < \omega,\ (\delta_i)_{i<m} \models_m^A \lfloor \mathbf{box}(M)\{E\} \rfloor_m$$

  Since $m < \omega$ implies that $m \leqslant n$, the above equation proves what we wanted.

□

A.3.2 Adequacy lemma

Lemma A.12 (Adequacy)**.** *If* $\Gamma \vdash M : A \mid S$*, then* $M \models^{\Gamma \vdash A} M$*.*

Proof. Let M be a well-typed term such that $(\pi) : \Gamma \vdash M : A \mid S$. By induction over (π).

- VAR. Immediate inversion of (π) yields:

  $$\underbrace{\Gamma', x{:}A}_{\Gamma} \vdash x : A \mid S \tag{a}$$

  Let $n \leqslant \omega$ — let $(\gamma_i)_{i<n} : \Gamma \ @\ n$, and $E : \Gamma \ @\ n$. Suppose that:

  $$x \parallel \mathbf{stop} \,;\, (\gamma_i)_{i<n} \overset{n}{\Longrightarrow} - \,;\, (\delta_i)_{i<n} \tag{b}$$

  $$x \,;\, E \Downarrow_n V \tag{c}$$

  $$(\gamma_i)_{i<n} \models_n^\Gamma E \tag{d}$$

Notice that if $n = 0$, then by PZero and EZero, we have $V = \mathbf{stop}$ and $(\delta_i)_{i<n} = \varepsilon$. Therefore, by definition of the logical relation, we automatically have $(\delta_i)_{i<n} \models^A_0 V$. Consequently, we will suppose that $n > 0$ in all the following.

By hypothesis, we have $(\gamma_i)_{i<n} : \Gamma \ @ \ n$, and $E : \Gamma \ @ \ n$. Inversion of IEnv and VEnv yield:

$$\begin{aligned} &\mathrm{dom}(E) = \mathrm{dom}(\Gamma) \\ &\mathrm{dom}(E) = \mathrm{dom}(\gamma_i), \text{ for all } i < n \end{aligned}$$

By (a), we have that $x \in \mathrm{dom}(\Gamma)$. Thus, we have $x \in \mathrm{dom}(\Gamma)$ and $\forall i < n.\ x \in \mathrm{dom}(\gamma_i)$. From this, Lemma A.4 to (b) and inversion of PVar in (c) respectively yield:

$$\forall i < n.\ \delta_i = \gamma_i(x) \qquad\qquad V = E(x) \tag{e}$$

Since x is in the domain of both environments, unfolding Definition 3.1 in (d) yields:

$$(\gamma_i(x))_{i<n} \models^A_n E(x)$$

Immediate substitution of (e) concludes our proof.

- By. We have $(\pi) : p \Rrightarrow \Gamma \vdash M \text{ by } p : p \Rrightarrow A \mid p \Rrightarrow S$. Immediate inversion of (π) yields:

$$\Gamma \vdash M : A \mid S \tag{a}$$

Let $n \leqslant \omega$ — let $(\gamma_i)_{i<n} : p \Rrightarrow \Gamma \ @ \ n$, and $E : p \Rrightarrow \Gamma \ @ \ n$. Thus, there exists $(\gamma'_i)_{i<p(n)}$ and E' such that :

$$\begin{cases} (\gamma'_i)_{i<p(n)} : \Gamma \ @ \ p(n) \\ \forall i < n.\ \gamma_i = \triangleright(\gamma'_j)_{p(i)\leqslant j<p(i+1)} \end{cases} \qquad \begin{cases} E' : \Gamma \ @ \ p(n) \\ E = \mathbf{w}\,(p, E') \end{cases} \tag{b}$$

Now, suppose that:

$$M \text{ by } p \parallel \mathbf{stop}\ ;\ \underbrace{\left(\triangleright(\gamma'_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n}}_{=\gamma_i} \overset{n}{\Longrightarrow} -\ ;(\delta_i)_{i<n} \tag{c}$$

$$M \text{ by } p\ ;\underbrace{\mathbf{w}\,(p, E')}_{=E} \Downarrow_n V \tag{d}$$

$$\left(\triangleright(\gamma'_j)_{p(i)\leqslant j<p(i+1)}\right)_{i<n} \models^{p \Rrightarrow \Gamma}_n \mathbf{w}\,(p, E') \tag{e}$$

Notice that if $n = 0$, then by PZero and EZero, we have $V = \mathbf{stop}$ and $(\delta_i)_{i<n} = \varepsilon$. Therefore, by definition of the logical relation, we have $(\delta_i)_{i<n} \models^A_0 V$. We will thus suppose that $n > 0$ in all the following.

By application of Lemma A.1 to (c), inversion of PWarp in (d), and unfolding the definition of the logical relation in (e), we obtain:

$$M \parallel \mathbf{stop}\ ;(\gamma'_i)_{i<p(n)} \overset{p(n)}{\Longrightarrow} -\ ;(\delta'_i)_{i<p(n)} \text{ st. } \delta_i = \triangleright(\delta'_j)_{p(i)\leqslant j<p(i+1)} \text{ for all } i < n \tag{f}$$

$$M\ ;E' \Downarrow_{p(n)} V' \text{ st. } V = \mathbf{w}\,(p, V') \tag{g}$$

$$(\gamma'_i)_{i<p(n)} \models^{\Gamma}_{p(n)} E' \tag{h}$$

The induction hypothesis gives:

$$M \models^{\Gamma \vdash A} M$$

Its immediate application to (f), (g), and (h) yields:

$$(\delta'_i)_{i<p(n)} \models^{A}_{p(n)} V'$$

By definition of the logical relation, this gives:

$$\Big((\triangleright(\delta'_j)_{p(i)\leqslant j<p(i+1)}) \Big)_{i<n} \models^{p \Rrightarrow A}_{n} \mathbf{w}\,(p, V')$$

By substition of the equalities given in (f) and (g), we obtain the following, which concludes our proof:

$$(\delta_i)_{i<n} \models^{p \Rrightarrow A}_{n} V$$

- Rec. We have $(\pi) : \Gamma \vdash \mathsf{rec}\, x^A.\, M : A \mid S \times \int A$. Immediate inversion of (π) yields:

$$\Gamma,\, x : \mathbf{lat} \Rrightarrow A \vdash M : A \mid S \tag{a}$$

We want to prove that $\mathsf{rec}\, x^A.\, M \models^{\Gamma \vdash A} \mathsf{rec}\, x^A.\, M$, whose definition is that for all $n \leqslant \omega$,

$$\left.\begin{array}{c} \forall E, V. \\ \forall (\gamma_i)_{i<n}, (\delta_i)_{i<n}. \\ \forall s. \end{array}\right. \quad \left.\begin{array}{r} \mathsf{rec}\, x^A.\, M \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s\,;(\delta_i)_{i<n} \\ \mathsf{rec}\, x^A.\, M\,;E \Downarrow_n V \\ (\gamma_i)_{i<n} \models^{\Gamma}_{n} E \end{array}\right\} \implies (\delta_i)_n \models^{A}_{n} V$$

We prove the property by induction over n.

- Case $n = 0$. By PZero and EZero, we have $V = \mathbf{stop}$ and $(\delta_i)_{i<n} = \varepsilon$. Therefore, by definition of the logical relation, we have $(\delta_i)_{i<n} \models^{A}_{0} V$.
- Case $1 \leqslant n < \omega$. Let $(\gamma_i)_{i<n} : \Gamma \mathbin{@} n$, and $E : \Gamma \mathbin{@} n$. Now, suppose that:

$$\mathsf{rec}\, x^A.\, M \parallel \mathbf{stop}\,;(\gamma_i)_{i<n} \overset{n}{\Longrightarrow} s\,;(\delta_i)_{i<n} \tag{b}$$

$$\mathsf{rec}\, x^A.\, M\,;E \Downarrow_n V \tag{c}$$

$$(\gamma_i)_{i<n} \models^{\Gamma}_{n} E \tag{d}$$

By application of Lemma A.2 to (b), we obtain that there exists s' such that:

$$\begin{array}{c} M \parallel \mathbf{stop}\,;\big(\gamma_0[\triangleright\varepsilon/x]\big) \oplus \big(\gamma_i[\triangleright(\delta_{i-1})/x]\big)_{1\leqslant i<n} \overset{n}{\Longrightarrow} s'\,;(\delta_i)_{i<n} \\ s = \big(s', \mathbf{inc}\,\big((\delta_i)_{i<n}\big)\big) \end{array} \tag{e}$$

By inversion of EStep in (e), we split the many step reduction and obtain that there exists s'_I such that:

$$M \parallel \mathbf{stop}\,;\big(\gamma_0[\triangleright\varepsilon/x]\big) \oplus \big(\gamma_i[\triangleright(\delta_{i-1})/x]\big)_{1\leqslant i<n-1} \overset{n-1}{\Longrightarrow} s'_I\,;(\delta_i)_{i<n-1} \tag{f}$$

$$M \parallel s'_I\,;\gamma_{n-1}[\triangleright(\delta_{n-2})/x] \Longrightarrow s'\,;\delta_{n-1} \tag{g}$$

Inversion of PRec in (c) gives us that $M\,;E\,;x\,;\mathbf{stop} \Uparrow^{n}_{0} V$. Immediate application

of Corollary A.4.1 gives us that there exists a value V_I such that:

$$M\,;\lfloor E\rfloor_{n-1}\,;x\,;\mathbf{stop}\Uparrow_0^{n-1}V_I \tag{h}$$

$$M\,;E\left[\mathbf{w}\,(\mathbf{lat},V_I)\,/_x\right]\Downarrow_n V \tag{i}$$

We turn back (f) and (h) into reductions over a rec by respectively applying to them Lemma A.6 and PRec:

$$\mathsf{rec}\,x^A.\,M\parallel\mathbf{stop}\,;(\gamma_i)_{i<n-1}\overset{n-1}{\Longrightarrow}\left(s'_I,\mathbf{inc}\left((\delta_i)_{i<n-1}\right)\right);(\delta_i)_{i<n-1} \tag{j}$$

$$\mathsf{rec}\,x^A.\,M\,;\lfloor E\rfloor_{n-1}\Downarrow_{n-1}V_I \tag{k}$$

Since Lemma A.11 gives us that the logical relation preserves truncation, (d) gives us that $(\gamma_{i<n-1})\models^{\Gamma}_{n-1}\lfloor E\rfloor_{n-1}$. Thus, since (j) and (k) also hold, we can apply the induction hypothesis given by our induction over n, and we obtain:

$$(\delta_i)_{i<n-1}\models^A_{n-1}V_I$$

By definition of the logical relation, this entails:

$$(\triangleright\varepsilon)\oplus(\triangleright\delta_{i-1})_{1\leqslant i<n}\models^{\mathbf{lat}\Rightarrow A}_n\mathbf{w}\,(\mathbf{lat},V_I)$$

Combined with (d), this allows us to conclude:

$$\left(\gamma_0\left[\triangleright\varepsilon/_x\right]\right)\oplus\left(\gamma_i\left[\triangleright(\delta_{i-1})/_x\right]\right)_{1\leqslant i<n}\models^{\Gamma,\,x:\mathbf{lat}\Rightarrow A}_n E\left[\mathbf{w}\,(\mathbf{lat},V_I)\,/_x\right] \tag{l}$$

The induction hypothesis given by our induction over (π) gives that:

$$M\models^{\Gamma,\,x:\mathbf{lat}\Rightarrow A\vdash A}M$$

Thus, since we have that the reductions (e) and (i) hold, and that their environments are coherent slicings by (l), we obtain that:

$$(\delta_i)_{i<n}\models^A_n V$$

Which is what we wanted.

– Case $n=\omega$. Let $(\gamma_i)_{i<\omega}:\Gamma\ @\ \omega$, and $E:\Gamma\ @\ \omega$. Now, suppose that:

$$\mathsf{rec}\,x^A.\,M\parallel\mathbf{stop}\,;(\gamma_i)_{i<\omega}\overset{\omega}{\Longrightarrow}s\,;(\delta_i)_{i<\omega} \tag{m}$$

$$\mathsf{rec}\,x^A.\,M\,;E\Downarrow_\omega V \tag{n}$$

$$(\gamma_i)_{i<\omega}\models^{\Gamma}_{\omega}E \tag{o}$$

By inversion of POmega in (n), we obtain:

$$V=\mathbf{box}(\mathsf{rec}\,x^A.\,M)\{E\} \tag{p}$$

Since $\mathsf{rec}\,x^A.\,M$ is well typed, by totality (Theorem 2.3) and determinism (Theorem 2.4) of the prefix reduction, we have:

$$\forall m<\omega.\ \ \mathsf{rec}\,x^A.\,M\,;\lfloor E\rfloor_m\Downarrow_m\lfloor\mathbf{box}(\mathsf{rec}\,x^A.\,M)\{E\}\rfloor_m \tag{q}$$

Then, by inversion of EOmega in (m) and application of Lemma A.11 in (o) we

obtain:

$$\forall m < \omega.\ \mathsf{rec}\, x^A.\, M \,\|\, \mathbf{stop}\,;(\gamma_i)_{i<m} \overset{m}{\Longrightarrow} s_m\,;(\delta_i)_{i<m} \tag{r}$$

$$\forall m < \omega.\ (\gamma_i)_{i<m} \models^{\Gamma}_m \lfloor E \rfloor_m \tag{s}$$

The induction hypothesis obtained by our induction over n applied on the three above equations yields:

$$\forall m < \omega.\ (\delta_i)_{i<m} \models^{A}_m \lfloor \mathbf{box}(\mathsf{rec}\, x^A.\, M)\{E\} \rfloor_m$$

By the definition of the logical relation given in Figure 12, this gives what we wanted:

$$(\delta_i)_{i<\omega} \models^{A}_{\omega} \mathbf{box}(\mathsf{rec}\, x^A.\, M)\{E\}$$

□