Termination of Threads with Shared Memory via Infinitary Choice

Paolo Tranquilli

Termination of Threads with Shared Memory via Infinitary Choice

Paolo Tranquilli

paolo.tranquilli@ens-lyon.fr
LIP, CNRS UMR 5668, INRIA,
ENS de Lyon, Université Claude Bernard Lyon 1, France

Abstract. We present a static type discipline on an extension of λ-calculus with threads and shared memory ensuring termination. This discipline is based on a type and effects system, and is a condition forced on regions. It generalizes and clarifies the stratification discipline previously proposed in the literature with the same objective, and is directly inspired by positive recursive types. The proof is carried out by translating the calculus with memory reference into an extension of lambda-calculus with a non-deterministic infinitary choice, whose strong normalization is in turn proved by a standard reducibility method.

1 Introduction

Mainstream programming paradigms are pervaded with side effects. The great majority of programs do not simply calculate a function, but carry out a whole lot of other actions that may influence the result: interacting with the user or with other processes, jumping to particular parts of its code, accessing memory, etc. There is a lot of research in computer science that goes towards controlling such side effects. Indeed programs that make large, uncontrolled use of side effects are harder to understand, verify or optimize.

Among the abstract tools that have been developed to this end and that are of interest to this work are *types and effects* systems [7] and *monads* [9]. The objective of the former is to analyze statically side effects by annotating in some way the ordinary types of programs. A typical way to analyze memory access is to abstract memory into different entities called regions. One then decorates types with the set of regions which the typed program can access, possibly specifying what kind of access it needs. The annotated types become then informative on what and where can something happen when calling the function. Such a level of abstraction from the actual workings of memory management allows to carry out static analysis. For example such an approach has been successfully used to analyze the problem of heap memory deallocation ([11], leading to the so-called region based memory management).

Monads are a tool directly coming from category theory which envisages to encapsulate and abstract away the details of side effects while remaining in a "clean" typed world. The idea is that a monad T can be seen as a type constructor

modelling a computational paradigm where (effect-less) *values* of type A are separated from *computations* of type TA. Since their inception they made it to be a highlight of Haskell's type system and way of programming.

Both approaches rely on a common ground: types as a tool to study and/or discipline programs. Relating to memory access, the typical result of using such systems is either to allow effective parallelization of the computation (like in the original type and effect proposal [7]), or ensure type safety (e.g. no "wrong" operation occurs during execution), or allow timely memory deallocation as already mentioned. However there is another property that in general type systems have been studied to deliver: *termination*, i.e. a certificate that the program will eventually yield a result.

Until recently this particular aspect has not been much studied in the presence of side effects involving memory access, especially when higher order types are possibly referenced. Indeed it was long known [6] that apart from the classical way of obtaining a (diverging) fix-point operator through self application, which is easily forbidden by types, such a term can be encoded through well-typed self reference. A diverging term can be easily written following this idea[1]:

$$\texttt{new}\ x := \lambda y.!x\, y\ \texttt{in}\ !x\,\langle\rangle\,.$$

Such a term can be read as "store in x the higher order function that reads from x what it should do, then apply it". Indeed, setting $M := \lambda y.!x\, y$, and executing the program against an empty memory store yields (see the rules in Table 1):

$$\emptyset, \texttt{new}\ x := M\ \texttt{in}\ !x\,\langle\rangle \xrightarrow{2} \emptyset, l := M; !l\,\langle\rangle \xrightarrow{2} \\ [l{\mapsto}M]\,, !l\,\langle\rangle \to [l{\mapsto}M]\,, (\lambda y.!l\, y)\,\langle\rangle \to [l{\mapsto}M]\,, !l\,\langle\rangle \to \cdots$$

In a standard type system with a type $\texttt{ref}A$ for references to values of type A, we can type the above term with 1 if we type the reference x with $\texttt{ref}(1 \to 1)$. We may get a hint as to why the program loops (but a priori no solution) by annotating types with regions, and seeing that in fact x with region r stores functions of type $1 \xrightarrow{\{r\}} 1$: the set added to the arrow indicates that those functions may access locations in region r, so circularity may ensue.

Recent works explored the idea of *stratification* of regions to avoid such circularities ([2,1] and recently in [4]) and yield termination not only for sequential but also for cooperative multithreading programs. The idea is that of imposing an ordering on regions so that, intuitively, a region may affect or read only regions that are strictly smaller. This has a distinct logical scent to it: even more so when one sees the proof techniques employed in [2,1], i.e. reducibility candidates.

In this work we set out to deepen this intuition. We will see how in fact the type and effect system can be translated to usual λ-calculus if we allow certain forms of *recursive types*, i.e. roughly types that can contain themselves in their definition. The translation is roughly one in memory passing style, where the

[1] The constructor $\texttt{new}\ x := M\ \texttt{in}\ N$ can be constructed from the primitives we will show in Table 1 as $(\lambda x.x := M; N)\texttt{new}$.

annotated type $A \xrightarrow{\{r_1,\ldots,r_k\}} B$ gets translated to $A^\bullet \to X_{r_1} \to \ldots \to X_{r_k} \to B^\bullet$, where X_r is to denote the translation of the type assigned to the region r. This is neither new nor surprising: memory read access can be modelled by additional inputs to the procedure via the memory access monad $TA := S \to A$. Returning to our diverging example, we get that the region r assigned to the location x must be typed with $1 \to X_r \to 1$, where in fact X_r is itself the type assigned to r. This leads to the recursive type equation $X_r \doteq 1 \to X_r \to 1$.

As already mentioned, this translation is not surprising. However within the framework it provides, we see that the stratification proposed in [2,1] corresponds exactly to *not* needing recursive types, i.e. using regular simple types. Apart from providing yet another proof that stratification yields termination, this clear view allows to easily generalize the result in two directions. On the one hand, it becomes clear that write operations can be completely ignored, so that a higher-order memory location that can write to itself poses no problems as to termination. On the other hand, while general recursive types break termination[2], it is known that *positive* recursive types ensure strong normalization [8]. The condition states that the self-references of recursive types should appear in positive position, i.e. on the left of an even number of function arrows: examples are $X \doteq 1 \to X$ or the system $X \doteq Y \to 1$, $Y \doteq X \to 1$. The general translation using recursive types allows to lift the positiveness condition to a condition on regions which is more general than stratification, and which we also call positiveness (subsection 2.1).

The way in which we are able to disregard write operations and include multithreading in the picture deserves a highlight. In fact we do not actually use regular λ-calculus to simulate the language with references $\Lambda_{\mathtt{ref}}$: we use Λ_∞, an extension of simply typed λ-calculus with a choice operator picking from a possibly *infinite* list of possible terms. This operator is used to simulate read access, while write operations are replaced by dummy ones. This allows to approximate (really coarsely) memory operations: we could say that one contracts both the *space* and the *time* of such operations. The space, as the contents of different memory locations abstracted with the same region are mixed together; the time, as the translated term could possibly read a value that is not (yet) written in the memory. In fact the translation allows *all* terms of the correct type to be present in memory at the same time. Among all the possible computations of the translation of the term, there is one following its intended semantics.

By completely ignoring the actual write operations a program performs before (or after) the read ones, multithreading is trivially included in the termination argument. In fact two parallel programs are translated as non-communicating programs, both having access to a pool of non-deterministically chosen data, incidentally also containing what the other program could have possibly written.

In a way, the coarseness of this approximation shows the limits of this technique: would a finer static analysis still ensure termination while allowing to type more programs? We could say that the argument used for termination has hardly anything to do with actual memory operations. It says roughly that enriching

[2] For example untyped λ-calculus can be "typed" with the recursive type $X \doteq X \to X$.

simply typed λ-calculus with a typed oracle giving any term of its type each time that it is called preserves normalization. In the literature we can find a roughly similar concept in the $\epsilon(A)$ construct of [5], akin to Hilbert's ϵ in the ϵ calculus. Finite non-deterministic choice on the other hand has been extensively studied (e.g. [10,3]). The technique we use to prove termination of the language is with reducibility candidates, as non-deterministic choice in general, and the infinite one in particular, really gives no sweat to the technique. The proof is therefore standard, following from what already done in [8].

Outline. In the upcoming section 2 we introduce $\Lambda_{\mathtt{ref}}$, the calculus on which we base our work. It is a standard λ-calculus endowed with references *à la* ML, with read operations $!M$ and write ones $M := N$, and an allocation operator `new`. On top of it, a parallel composition $M|N$ allows separate threads to communicate via shared memory. We end the section by defining the positiveness condition on region contexts, and the stronger stratification one in order to discuss and subsume the previous work found in [2,1].

In the following section 3 we prepare the ground for the definitions of the following section by presenting systems of (possibly mutually recursive) type equations of the form $X \doteq A$. The section is in fact a revision of the work done in [8], where we mainly introduce our notation for the subsequent exposition.

Next in section 4 we introduce the infinitary choice calculus Λ_∞, which extends regular λ-calculus with the choice $\{M_i\}_{i\in I}$ which can reduce to any of its terms non-deterministically. The typing is defined as to allow type conversions dictated by a type equation system. We then proceed to prove that if E is positive then Λ_∞ is strongly normalizing (Theorem 1), via an easy adaptation of Mendler's proof in [8].

Finally in section 5 we discuss the translation of $\Lambda_{\mathtt{ref}}$ to Λ_∞, which is already presented in Table 1. We prove that this translation yields a simulation (Theorem 2), and that positiveness and stratification of region contexts correspond to positiveness and *solvability* of the systems of equations translating such contexts (Corollary 1). This in turn leads to the result stating that positive region contexts ensure termination (Theorem 3).

Notation. We will denote usual substitution (in types and terms) by $[-/-]$. In types the arrow constructor $\to$ associates to the right, as in $A \to B \to C = A \to (B \to C)$. If $(A_i)_{i\in k}$ is a sequence of types, then $(A_i)_{i\in k} \to C$ denotes $A_0 \to \ldots \to A_{k-1} \to C$. In λ-calculus we allow to bunch together repeated abstractions or applications, as in $\lambda xy.x$ or zMN. We use the standard Church's encoding of pairing $\langle M, N\rangle := \lambda z.zMN$. Finally, we denote by $M; N$ the term $(\lambda d.N)M$ with d fresh for N.

2 The higher level: $\Lambda_{\mathtt{ref}}$

Table 1 shows the typing rules of $\Lambda_{\mathtt{ref}}$, and the translation to Λ_∞ that will be discussed in the section 5. As is usual (see for example [7]), we have **locations**

$l, m, \ldots$ as syntactic entities, denoting memory cells. These are to be considered as runtime entities not available at the time of programming.

Typing judgements are of the form $R; L; \Gamma \vdash M : A, e \mapsto M'$ where:

- R is a **region context**, a function with finite domain from regions to types;
- L is a function mapping locations to regions; this information can possibly be expanded during reduction (namely when allocating memory via the `new` construct), as we will see in Theorem 2;
- Γ and $M : A$ are the usual type assignments for free variables and the typed term respectively;
- e denotes the regions that M may read from;
- M' is a regular λ-calculus term: indeed we hard-code the translation to Λ_∞ in the type derivations to facilitate the reasoning on it.

The final rule is to type the interaction of a term with a memory store. The translation attached is the only place where we actually use Λ_∞'s infinite choice operator, indexed with all type derivations for $\Lambda_{\mathtt{ref}}$'s values with a given type. These indices are clearly countable. We denote by $\Lambda^R_{\mathtt{ref}}$ the set of store/term pairs typable with the region context R. Throughout this and the following section, we suppose we have a fixed order on regions, so that a sequence like $(x_r)_{r\in e}$ is uniquely determined by e.

The following example shows the dynamic threading feature of $\Lambda_{\mathtt{ref}}$:

$$\mathtt{spawn} := \lambda n, p.n\,(\lambda x, d.p\langle\rangle | x\langle\rangle)\,(\lambda d.\langle\rangle)\,\langle\rangle : \mathtt{nat}_{1\xrightarrow{c}\top} \to (1 \xrightarrow{e} \top) \to \top.$$

If M is a program of type $\top$ with effects e, and n is a Church integer, then $\mathtt{spawn}\,n\,(\lambda d.M)$ reduces to n parallel copies of M.

2.1 The positiveness condition

Mimicking the positiveness condition of recursive types, we will present in this section the positiveness condition of region contexts. This condition will assure termination via a suitable translation in positively recursive type equations (section 5). On the other hand, this condition is more general than the stratification one presented in [2,1].

First, let us define by mutual recursion the **positive** and **negative effects** of a type, as follows.

$$\mathrm{eff}_+(1) = \mathrm{eff}_-(1) = \mathrm{eff}_+(\top) = \mathrm{eff}_-(\top) = \mathrm{eff}_+(\mathtt{ref}_r)) = \mathrm{eff}_-(\mathtt{ref}_r) := \emptyset,$$

$$\mathrm{eff}_+(A \xrightarrow{e} B) := \mathrm{eff}_-(A) \cup \mathrm{eff}_+(B), \qquad \mathrm{eff}_-(A \xrightarrow{e} B) := \mathrm{eff}_+(A) \cup e \cup \mathrm{eff}_-(B).$$

Notice how encapsulated effects contribute to the *negative* effects of a type, as morally we must think read effects to be as additional inputs to the procedure, therefore occupying a negative position.

Next, we define **positive** (resp. **negative**) **dependence in** R, written $r \succ_+ s\ (R)$ (resp. $r \succ_- s\ (R)$) as follows:

$$\frac{s \in \mathrm{eff}_\varepsilon(R(r))}{r \succ_\varepsilon s\ (R)} \qquad \frac{r \succ_{\varepsilon_1} s\ (R) \qquad s \succ_{\varepsilon_2} t\ (R)}{r \succ_{\varepsilon_1\varepsilon_2} t\ (R)}$$

Syntax

$$
\begin{array}{lll}
x, y, \ldots & \text{(variables)} & l, m, \ldots \quad \text{(locations)} \\
U, V, \ldots & ::= x \mid l \mid \langle\rangle \mid \lambda x.M & \text{(values)} \\
M, N, \ldots & ::= V \mid MN \mid \mathtt{new} \mid !M \mid M := N \mid M|N & \text{(terms)} \\
E, F, \ldots & ::= [\,] \mid EM \mid VE \mid !E \mid E := M \mid V := E \mid E|M \mid M|E & \text{(contexts)} \\
S, T, \ldots & & \text{(stores, functions from locations to values)} \\
r, s, \ldots & \text{(regions)} & e, f, \ldots \quad \text{(sets of regions)} \\
A, B, \ldots & ::= 1 \mid A \xrightarrow{e} B \mid \mathtt{ref}_r & \text{(types)}
\end{array}
$$

Reduction

$$
\frac{}{S, E[(\lambda x.M)V] \to S, E[M\,[V/x]]} \qquad \frac{l \text{ fresh for } S \text{ and } E}{S, E[\mathtt{new}] \to S, E[l]}
$$

$$
\frac{l \in \mathrm{dom}(S)}{S, E[!l] \to S, E[S(l)]} \qquad \frac{}{S, E[l := V] \to S\,[l \mapsto V]\,, E[S(l)]}
$$

Typing rules and translation to Λ_∞

$$
\frac{}{R; L; \Gamma \vdash x : \Gamma(x), \emptyset \mapsto x} \qquad \frac{}{R; L; \Gamma \vdash l : \mathtt{ref}_{L(l)}, \emptyset \mapsto I} \qquad \frac{}{R; L; \Gamma \vdash \langle\rangle : 1, \emptyset \mapsto I}
$$

$$
\frac{R; L; \Gamma, x : A \vdash M : B, e \mapsto M'}{R; L; \Gamma \vdash \lambda x.M : A \xrightarrow{e} B, \emptyset \mapsto \lambda x.\lambda(x_r)_{r \in e}.M'}
$$

$$
\frac{R; L; \Gamma \vdash M : A \xrightarrow{e} B, f \mapsto M' \qquad R; L; \Gamma \vdash N : A, f \mapsto N' \qquad e \subseteq f}{R; L; \Gamma \vdash MN : B, f \mapsto M'N'(x_r)_{r \in e}}
$$

$$
\frac{}{R; L; \Gamma \vdash \mathtt{new} : \mathtt{ref}_r, \emptyset \mapsto I; I} \qquad \frac{R; L; \Gamma \vdash M : \mathtt{ref}_r, e \mapsto M' \qquad r \in e}{R; L; \Gamma \vdash !M : R(r), e \mapsto M'; x_r}
$$

$$
\frac{R; L; \Gamma \vdash M : \mathtt{ref}_r, e \mapsto M' \qquad R; L; \Gamma \vdash N : R(r), e \mapsto N'}{R; L; \Gamma \vdash M := N : 1, e \mapsto M'; N'; I}
$$

$$
\frac{R; L; \Gamma \vdash M : A, e \mapsto M' \qquad e \subseteq f}{R; L; \Gamma \vdash M : A, f \mapsto M'} \qquad \frac{R; L; \Gamma \vdash M : A, e \mapsto M'}{R; L; \Gamma \vdash M : \top, e \mapsto M'}
$$

$$
\frac{R; L; \Gamma \vdash M : \top, e \mapsto M' \qquad R; L; \Gamma \vdash N : \top, e \mapsto N'}{R; \Gamma \vdash M|N : \top, e \mapsto \langle M', N'\rangle}
$$

$$
\frac{R; L; \Gamma \vdash M : A, e \mapsto M' \qquad \forall l \in \mathrm{dom}\, S : R; ; \Gamma \vdash S(l) : R(L(l)), \emptyset}{R; L; \Gamma \vdash S, M : A, e \mapsto M' \left[\{V'\}_{R;;\Gamma \vdash V : R(r) \mapsto V'}/r\right]_{r \in e}}
$$

Table 1. Typing rules of $\Lambda_{\mathtt{ref}}$ and translation to Λ_∞ (which will be presented in section 4).

where the signs $\{+,-\}$ are endowed with the natural multiplication $\varepsilon_1\varepsilon_2 = +$ iff $\varepsilon_1 = \varepsilon_2$. We will say that R is **positive** if for all $r \in \operatorname{dom} R$ we *do not* have that $r \succ_- r\ (R)$. We say R is **stratified** if in R the relation $\succ = \succ_+ \cup \succ_-$ is a strict order. Clearly stratification implies positiveness, though not the contrary, as exemplified by the region context $r : 1 \overset{\{s\}}{\to} 1, s : 1 \overset{\{r\}}{\to} 1$, or $r : (1 \overset{\{r\}}{\to} 1) \to 1$.

Notice that the positiveness condition can be checked by a simple variation of the path finding algorithm on directed graphs, and as such is a problem in the polynomial complexity class.

3 Mutually Recursive Type Equations

We will here introduce our notations for mutually recursive sets of type equations. In this and the following section we will consider types to be the usual simple types of λ-calculus, i.e. generated by a countable set of type variables (with meta-variables $X, Y, \ldots$) and the arrow constructor, plus the $\top$ type:

$$A, B, \ldots ::= X \mid \top \mid A \to B.$$

The sets $\mathrm{V}_+(A)$ and $\mathrm{V}_-(A)$ of **positive** (resp. **negative**) **variables** of a type A are defined by mutual recursion as follows:

$$\begin{array}{rlrl} \mathrm{V}_+(X) &:= \{X\}, & \mathrm{V}_+(A \to B) &:= \mathrm{V}_-(A) \cup \mathrm{V}_+(B), \\ \mathrm{V}_+(\top) = \mathrm{V}_-(\top) = \mathrm{V}_-(X) &:= \emptyset, & \mathrm{V}_-(A \to B) &:= \mathrm{V}_+(A) \cup \mathrm{V}_-(B), \end{array}$$

Clearly the set of variables occurring in A is $\mathrm{V}(A) := \mathrm{V}_+(A) \cup \mathrm{V}_-(A)$.

A **type equation** is an ordered pair of a type variable and a non-variable type[3], written $X \doteq A$. We say that $X \doteq A$ **is on** the variable X. A **system of equations** E is a finite set of equations on distinct variables. A system E can be equivalently seen as a function with finite domain from type variables to types. Consequently we denote by $\operatorname{dom} E$ the projection of E on variables and by $E(X)$ the type associated to X in E. We define **positive** (resp. **negative**) **dependency in** E, written $X \succ_+ Y\ (E)$ (resp. $X \succ_- Y\ (E)$) via the following rules:

$$\frac{Y \in \mathrm{V}^{\varepsilon}(E(X))}{X \succ_{\varepsilon} Y\ (E)} \qquad \frac{X \succ_{\varepsilon_1} Y\ (E) \quad Y \succ_{\varepsilon_2} \gamma\ (E)}{X \succ_{\varepsilon_1\varepsilon_2} \gamma\ (E)}$$

We denote plain dependecy by $X \succ Y\ (E)$, which happens iff $X \succ_{\varepsilon} Y\ (E)$ for a sign ε.

We say that a system of equations E is **circular** if for every $X \neq Y \in \operatorname{dom} E$ we have $X \succ Y \succ X\ (E)$. Notice that every single equation $X \doteq A$ is considered a circular system. Given two subsystems $E_1, E_2 \subseteq E$ with disjoint domains we write $E_1 \succ E_2\ (E)$ if $\exists X \in \operatorname{dom} E_1, Y \in \operatorname{dom} E_2 : X \succ Y\ (E)$. An easy exercise shows that in case E_1 and E_2 are non-empty and circular the latter condition is equivalent with turning the existential into a universal quantifier.

Lemma 1. *Every system of equations E is uniquely decomposable in a partition $E = E_1 + \cdots + E_k$ such that every E_i is circular and $\succ$ on $\{E_i\}$ is an order.*

→ proof in tech. app.

[3] We explicitly prevent aliasing, i.e. equations of the form $X \doteq Y$.

We say a sequence $E_1, \dots, E_k$ of systems is **canonic** if it is a partition of $E_1 \cup \dots \cup E_k$ satisfying the conditions of the above lemma and if $E_i \succ E_j$ implies $i > j$. Clearly every system E admits a canonic (not necessarily unique) decomposition in circular subsystems.

We say a system of equations is **positive** if for every $X \in \operatorname{dom} E$ it is never the case that $X \succ_- X \ (E)$.

proof in tech. app. ←

Lemma 2. *If $E = E_1 + \dots + E_k$ is a partition satisfying the conditions of Lemma 1, then E is positive iff every E_i is positive.*

We write $A = B \ (E)$ for the contextual, symmetric, transitive and reflexive closure of $\doteq$ as prescribed by E. Namely:

$$\frac{X \doteq A \in E}{X = A \ (E)} \qquad \frac{}{A = A \ (E)} \qquad \frac{A = B \ (E)}{B = A \ (E)}$$

$$\frac{A = B \ (E) \quad B = C \ (E)}{A = C \ (E)} \qquad \frac{A = B \ (E) \quad C = D \ (E)}{A \to C = B \to D \ (E)}$$

We further write that a system E is **solvable** if for every $X \in \operatorname{dom} E$ there is a type A_X such that $X = A_X \ (E)$ and $\mathrm{V}(A_X) \cap \operatorname{dom} E = \emptyset$.

proof in tech. app. ←

Lemma 3. *E is solvable iff $\succ$ in E is a strict order on $\operatorname{dom} E$. In particular if E is solvable then it is positive.*

4 The lower level: Λ_∞

Table 2 shows the syntax, the reduction and the typing rules of the infinite choice λ-calculus Λ_∞. Typing judgments are of the form $E; \Gamma \vdash M : A$, where as usual Γ is the context mapping term variables to their types, and E is a system of type equations whose induced equality may be used while typing the term. Given a system of type equations E, we denote by Λ_∞^E the set of terms M typable with judgments $E; \Gamma \vdash M$ for some context Γ.

The only difference from regular λ-calculus is the choice operator $\{M_i\}_{i \in I}$. In it M_i is a family of terms indexed by I, where I is a set of arbitrary cardinality, though bounded by a cardinal fixed once and for all for all terms[4]. Such an operator is just a non-deterministic choice done anywhere within a term and picking from a possibly infinite list of terms (typed with the same type).

Before going on we stress some aspects of the calculus.

No reduction inside the choice operator. Reduction does not happen inside the choice operator, as prescribed by the definition of context. This avoids simple infinite reductions: take $M_{h,k}$ to be I if $h < k$ and II otherwise (with I the classic identity $\lambda x.x$). Then with a reduction happening inside a choice operator we would have $\{M_{h,k}\}_{h \in \mathbb{N}} \to \{M_{h,k+1}\}_{h \in \mathbb{N}}$ for all k.

[4] Letting I's cardinality be arbitrary for any choice operator would lead to a bad inductive definition, as taking as I the set of terms themselves would give a paradox.

Syntax

$$\begin{array}{lll} x, y, \ldots & & \text{(variables)} \\ M, N, \ldots & ::= x \mid \lambda x.M \mid MN \mid \{M_i\}_{i\in I} & \text{(terms)} \\ E, F, \ldots & ::= [\,] \mid EM \mid ME \mid \lambda x.E & \text{(contexts)} \end{array}$$

Reduction

$$\frac{}{(\lambda x.M)N \to M\,[N/x]} \qquad \frac{j \in I}{\{M_i\}_{i\in I} \to M_j} \qquad \frac{M \to N}{E[M] \to E[N]}$$

Typing rules

$$\frac{x : A \in \Gamma}{E;\Gamma \vdash x : A} \qquad \frac{E;\Gamma, x : A \vdash M : B}{E;\Gamma \vdash \lambda x.M : A \to B} \qquad \frac{E;\Gamma \vdash M : A \to B \qquad E;\Gamma \vdash N : A}{E;\Gamma \vdash MN : B}$$

$$\frac{\forall i \in I : E;\Gamma \vdash M_i : A}{E;\Gamma \vdash \{M_i\}_{i\in I} : A} \qquad \frac{E;\Gamma \vdash M : A}{E;\Gamma \vdash M : \top} \qquad \frac{E;\Gamma \vdash M : A \qquad A = B\ (E)}{\Gamma \vdash M : B}$$

Table 2. Grammar, reduction and typing rules of Λ_∞. In the choice operator $\{M_i\}_{i\in I}$ the set I of indexes is arbitrary but with bounded cardinality fixed for all terms.

No confluence. The calculus is not confluent, as can be expected from a non-deterministic reduction. However the situation is even worse than that. We could consider the reduction as happening between sets of possible computations, where internal choice is the only responsible of forking. However, even with such a formalism confluence would fail. For example taking any family N_i the term $(\lambda x.\langle x, x\rangle)\{N_i\}_{i\in I}$ reduces either to $\{\,\langle N_i, N_j\rangle \mid i, j \in I\,\}$ if first reducing the β-redex, or to $\{\,\langle N_i, N_i\rangle \mid i \in I\,\}$ if first firing the internal choice. This is a standard problem encountered in non-deterministic calculi. However we are looking for termination of this language as a tool for the termination of another extension of λ-calculus, so we really do not care about any form of confluence.

The top type. We add a $\top$ type whose semantics is all typable terms. This poses no problems whatsoever and is needed to reflect the thread behaviour used by $\Lambda_{\mathtt{ref}}$ (see Table 1).

Subject reduction. Preservation of typing by reduction is easily obtainable by a standard proof.

4.1 Candidates of reducibility

In this subsection we sketch the termination of Λ^E_∞ for a positive system E of type equations. The technique is exactly the same as is found in [8], with a trivial extension needed for the infinite choice operator. Let SN denote the set of strongly normalizing Λ_∞ terms.

We say that a subset $\mathcal{X} \subseteq \mathsf{SN}$ is **saturated** if for every terms $\vec{P}$ in SN:

S1) for every variable x we have $x\vec{P} \in \mathcal{X}$;

S2) if $Q \in \mathsf{SN}$ we have that $M[Q/x]\,\vec{P} \in \mathcal{X}$ entails $(\lambda x.M)Q\vec{P} \in \mathcal{X}$;
S3) we have that if for all $i \in I$ we have $M_i\vec{P} \in \mathcal{X}$ then $\{M_i\}_{i\in I}\,\vec{P} \in \mathcal{X}$.

A *valuation* is any map $\rho : \mathcal{V} \to \mathsf{SAT}$. We denote by $\rho[X \mapsto \mathcal{X}]$ the valuation acting as ρ on $Y \neq X$ and assigning $\mathcal{X}$ to X. Similarly we will write $\rho[X_i \mapsto \mathcal{X}_i]_{i\in I}$ for indexed families of variables and saturated sets. For every valuation ρ, we extend it on all types by setting $\rho(\top) := \mathsf{SN}$ and $\rho(A \to B) := \rho(A) \to \rho(B) = \{\, M \mid \forall N \in \rho(A) : MN \in \rho(B) \,\}$.

Lemma 4. *Given a valuation ρ and a type A we have $\rho(A) \in \mathsf{SAT}$.*

proof in tech. app. ←

Lemma 5 ([8]). *If E is positive and circular, then for every ρ the function $F_{E,\rho} : \langle \mathcal{X}_X \rangle_{X\in\operatorname{dom} E} \mapsto \langle \rho[Y \mapsto \mathcal{X}_Y]_{Y\in\operatorname{dom} E}(E(X)) \rangle_{X\in\operatorname{dom} E}$ has a fixpoint $T_{E,\rho}$.*

Let us fix for every E and ρ the fixpoint $T_{E,\rho}$ whose existence is assured by the lemma above. Given a valuation ρ and a circular positive system E let us denote by $\rho[E]$ the valuation defined by $\rho[E] = \rho\left[X \mapsto \pi_X^{\operatorname{dom} E} T_{E,\rho}\right]_{X\in\operatorname{dom} E}$. Abusing the notation, we will denote with $\rho[E]$ for any positive system E the valuation $\rho[E_1]\cdots[E_k]$ for a canonic decomposition into circular subsystems $E = E_1 + \cdots + E_k$, even if there might be several such decompositions.

proof in tech. app. ←

Lemma 6 ([8]). *If E is positive and $A = B$ (E) then $\rho[E](A) = \rho[E](B)$.*

If σ is a partial function $\sigma : \mathcal{V} \to \Lambda_\infty$ with finite domain, let $M\sigma$ stand for the simultaneous substitution $M[\sigma(x)/x]_{x\in\operatorname{dom}\sigma}$. If moreover ρ is an evaluation let $\rho, \sigma, E \vDash M : A$ stand for $M\sigma \in \rho[E](A)$, and $\rho, \sigma, E \vDash \Gamma$ be $\rho, \sigma, E \vDash x : A$ for every $x : A \in \Gamma$. The adequacy of the interpretation can now be stated.

proof in tech. app. ←

Lemma 7. *If E is positive, $E; \Gamma \vdash M : A$ and $\rho, \sigma, E \vDash \Gamma$ then $\rho, \sigma, E \vDash M : A$.*

Theorem 1. *Λ_∞^E with E positive is strongly normalizable.*

Proof. Immediate from the above lemma, as if M belongs to a saturated set then it is in SN by definition.

5 Translation

In this section we will discuss the translation from $\Lambda_{\mathtt{ref}}$ to Λ_∞ which we already presented as a decoration of $\Lambda_{\mathtt{ref}}$'s typing rules. We will in particular show that it is a simulation ensuring that $\Lambda_{\mathtt{ref}}$ is terminating when the region context is positive.

As can be seen in Table 1, to each regular typing judgement derivable for $\Lambda_{\mathtt{ref}}$, there is a term M' attached, as in $R; L; \Gamma \vdash M : A, e \mapsto M'$. This term is a regular λ-calculus term, which has new term variables x_r for each $r \in e$ (which are considered fresh). Those variables are bound when typing an abstraction, and when finally the interaction of a term and a store is typed, those variables are instantiated with an infinite choice.

We will see that if $R; L; \Gamma \vdash S, M : A, e \mapsto M'$, then M' non-deterministically approximates the behaviour of S, M. However we can see how such an approximation is coarse: indeed the definition of M' does not take into account S, and it just instantiates the variables x_r (which stands for any location of region r) with the translation of *all* the possible terms typed with the correct type. This is why the write operation is ignored by the effects system. In fact what we show in this paper is that the stratification discipline (or the finer positive one) ensures something much stronger than termination of programs with shared memory. In fact the termination argument does not depend in any way of what is written in memory, by just using the fact that it is typed of the correct type.

By hard-coding the translation in the typing rules, we can indeed prove subject reduction of $\Lambda_{\mathtt{ref}}$ and simulation in one go. First we need a substitution lemma, which has an easy proof by induction. Then we can prove Theorem 2 by an induction too.

Lemma 8. *If $R; L; \Gamma \vdash V : A, \emptyset \mapsto V'$ and $R; L; \Gamma, x : A \vdash M : B, e \mapsto M'$ then $R; L; \Gamma \vdash M\,[V/x] \mapsto M'\,[V'/x]$.*

Theorem 2. *If $R; L; \Gamma \vdash S, M : A, e \mapsto M'$ and $S, M \to T, N$, then there is $L' \supseteq L$ and N' with $R; L'; \Gamma \vdash T, N : A, e \mapsto N'$, such that $M' \xrightarrow{+} N'$ in Λ_∞.*

→ proof in tech. app.

Next, we show how the translation respects typing. In order to do so we provide the following translation of $\Lambda_{\mathtt{ref}}$ types into regular types, $\Lambda_{\mathtt{ref}}$ region contexts into type equations and $\Lambda_{\mathtt{ref}}$ type assignements into regular type assignements:

$$1^\bullet := \top \to \top, \qquad (A \xrightarrow{e} B)^\bullet := A^\bullet \to (X_r)_{r\in e} \to B^\bullet, \qquad (\mathtt{ref}_r)^\bullet := \top \to \top,$$
$$\top^\bullet := \top, \qquad R^\bullet(X_r) := (R(r))^\bullet, \qquad \Gamma^\bullet(x) := (\Gamma(x))^\bullet.$$

Notice how we are just using a memory monad $TA = S \to A$ where S is the type of memory [9]. The monad is however localized in the regions effectively used by the procedures. We could define an indexed monad $T_e A = (X_r)_{r\in e} \to A$ and thus having the usual monadic translation of $A \xrightarrow{e} B$ as $A^\bullet \to T_e B^\bullet$. The translation's adequacy is stated as follows.

Lemma 9. *If $R; L; \Gamma \vdash M : A, e \mapsto M'$ (resp. $R; L; \Gamma \vdash S, M : A, e \mapsto M'$), then in Λ_∞ we have $R^\bullet; \Gamma^\bullet, (x_r : X_r)_{r\in e} \vdash M' : A^\bullet$ (resp. $R^\bullet; \Gamma^\bullet \vdash M' : A^\bullet$).*

→ proof in tech. app.

Notice that if we abstracted the x_r's after each rule, we would get a monadic typing in the form of $R^\bullet; \Gamma^\bullet \vdash M' : T_e A^\bullet$ when starting from $R; L; \Gamma \vdash M : A, e \mapsto M'$ in $\Lambda_{\mathtt{ref}}$.

Next, we show how the positiveness condition is invariant for the translation, and how it relates stratification with solvability of type systems.

Lemma 10. *$r \succ_\varepsilon s$ (R) iff $X_r \succ_\varepsilon X_s$ $(R^\bullet)$.*

Proof. The statement follows from the observation that $X_s \in \mathrm{V}_\varepsilon(R^\bullet(X_r))$ if and only if $s \in \mathrm{eff}_\varepsilon(R(r))$, and the rules for transitivity are the same. □

Corollary 1. *We have that*

- *R is positive iff $R^\bullet$ is positive.*
- *R is stratified iff $R^\bullet$ is solvable.*

Proof. The first point is a direct consequence of the above lemma. The second is equally immediate by using Lemma 3. □

Theorem 3. *If R is a positive region context, then $\Lambda^R_{\mathtt{ref}}$ is strongly normalizing.*

Proof. Immediate consequence of Theorems 1 and 2 and of Corollary 1. □

References

1. Amadio, R.M.: On stratified regions. In: Hu, Z. (ed.) APLAS. Lecture Notes in Computer Science, vol. 5904, pp. 210–225. Springer (2009)
2. Boudol, G.: Fair cooperative multithreading. In: CONCUR. Lecture Notes in Computer Science, vol. 4703, pp. 272–286. Springer (2007)
3. de'Liguoro, U., Piperno, A.: Non deterministic extensions of untyped λ-calculus. Info. and Comp 122, 149–177 (1995)
4. Demangeon, R., Hirschkoff, D., Sangiorgi, D.: Termination in impure concurrent languages. In: Gastin, P., Laroussinie, F. (eds.) CONCUR. Lecture Notes in Computer Science, vol. 6269, pp. 328–342. Springer (2010)
5. de Groote, P.: Denotations for classical proofs - preliminary results. In: Nerode, A., Taitslin, M.A. (eds.) LFCS. Lecture Notes in Computer Science, vol. 620, pp. 105–116. Springer (1992)
6. Landin, P.J.: The mechanical evaluation of expressions. The Computer Journal 6(4), 308–320 (January 1964), http://dx.doi.org/10.1093/comjnl/6.4.308
7. Lucassen, J.M., Gifford, D.K.: Polymorphic effect systems. In: POPL '88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 47–57. ACM, New York, NY, USA (1988)
8. Mendler, N.P.: Inductive types and type constraints in the second-order lambda calculus. Ann. Pure Appl. Logic 51(1-2), 159–172 (1991)
9. Moggi, E.: Notions of computation and monads. Information and Computation 93(1), 55–92 (Jul 1991)
10. Saheb-Djahromi, N.: Probabilistic lcf. In: Winkowski, J. (ed.) MFCS. Lecture Notes in Computer Science, vol. 64, pp. 442–451. Springer (1978)
11. Tofte, M., Talpin, J.P.: Region-based memory management. Inf. Comput. 132(2), 109–176 (1997)

A Technical Proofs

Lemma 1. *Every system of equations E is uniquely decomposable in a partition $E = E_1 + \cdots + E_k$ such that:*

- *every E_i is circular;*
- *$\succ$ on $\{E_i\}$ is an order.*

Proof. Existence is by induction on $\#\operatorname{dom} E$. If there is $X \in \operatorname{dom} E$, we can take the maximal circular subsystem $E_1 \subseteq E$ with $X \in \operatorname{dom} E_1$, then apply the inductive hypothesis on $E \setminus E_1$ obtaining $E = E_1 + E_2 + \cdots + E_k$. Then $\succ$ is an order on $\{E_i\}_{i\geq 2}$. Now suppose for a contradiction that $E_1 \succ E_j$ and $E_j \succ E_1$: it would follow that $E_1 \cup E_j$ is circular which contradicts maximality of E_1.

For uniqueness, suppose without loss of generality that $E = E_1 + \cdots + E_k = E'_1 + \cdots + E'_{k'}$ with $E_1 \cap E'_1 \neq \emptyset$ and $E_1 \setminus E'_1 \neq \emptyset$. We can take $X \in \operatorname{dom} E_1 \cap \operatorname{dom} E'_1$ and $Y \in \operatorname{dom} E_1 \setminus \operatorname{dom} E'_1$. Suppose $Y \in \operatorname{dom} E'_i$. As $X \neq Y \in \operatorname{dom} E_1$ we have $X \succ Y \succ X$ (E) by circularity of E_1, so $E'_1 \succ E'_i \succ E'_1$ which is a contradiction. □

Lemma 2. *If $E = E_1 + \cdots + E_k$ is a partition satisfying the conditions of Lemma 1, then E is positive iff every E_i is positive.*

Proof. For only if part, it suffices to see that if $X \succ_- X$ (E_i) then the same holds in E. For the if part: the only way in which one would have $X \succ_- X$ (E) but not $X \succ_- X$ (E_i) with $X \in \operatorname{dom} E_i$ would be to have $X \succ_{\varepsilon_1} Y$ (E) and $Y \succ_{\varepsilon_2} X$ (E) with $Y \notin \operatorname{dom} E_i$ and $\varepsilon_1 \neq \varepsilon_2$. But then one would have $E_i \succ E_j \succ E_i$ with $Y \in \operatorname{dom} E_j$, which is a contradiction. □

Lemma 11. *$X \succ_\varepsilon Y$ (E) iff there is a non-variable type A with $X = A$ (E) and $b \in \mathrm{V}_\varepsilon(A)$.*

Proof. Immediate induction. □

Lemma 12. *If A and B are such that $\mathrm{V}(A) \cap \operatorname{dom} E = \mathrm{V}(B) \cap \operatorname{dom} E$ and $A = B$ (E), then they are the same type.*

Proof. Immediate induction. □

Lemma 3. *E is solvable iff $\succ$ in E is a strict order on $\operatorname{dom} E$. In particular if E is solvable then it is positive.*

Proof. For the only if part, suppose E is solvable with solution A_Z and $X \succ X$ (E) for an X. Then by Lemma 11 we would have a type B with $X = B$ (E), $X \in \mathrm{V}(B)$ and B not a variable. Then if we let B' be the type $B\,[A_Z/Z]_{Z\in\operatorname{dom} E}$, we would have $A_X = B'$ (E), which reverts to an actual equality by Lemma 12 as $\mathrm{V}(B') \cap \operatorname{dom} E = \emptyset$. This is a contradiction as $X \in \mathrm{V}(B)$ and B not a variable would imply that A_X is a proper subterm of B', i.e. itself.

For the if part, one proceeds by induction on the cardinality of $\operatorname{dom} E$. If it is not empty by its finitess there must be a variable X in $\operatorname{dom} E$ which is minimal

with respect to $\succ$. In particular it must be the case that $\mathrm{V}(E(X)) \cap (\operatorname{dom} E \setminus \{X\}) = \emptyset$. We then can solve $(E \setminus \{X \doteq E(X)\})[E(X)/X]$ with the inductive hypothesis, and adding $A_X = E(X)$ to that solution obtain one for the whole of E. □

Lemma 13. *For every $X \notin \mathrm{V}^-(A)$ (resp. $X \notin \mathrm{V}^+(A)$) we have that if $\mathcal{X} \subseteq \mathcal{Y}$ then $\rho[X \mapsto \mathcal{X}](A) \subseteq \rho[X \mapsto \mathcal{Y}](A)$ (resp. $\rho[X \mapsto \mathcal{X}](A) \supseteq \rho[X \mapsto \mathcal{Y}](A)$).*

Lemma 5. *Given a positive circular system of equations E, then for every ρ the function*

$$F_{E,\rho} : \langle \mathcal{X}_X \rangle_{X \in \operatorname{dom} E} \mapsto \langle \rho[Y \mapsto \mathcal{X}_Y]_{Y \in \operatorname{dom} E}(E(X)) \rangle_{X \in \operatorname{dom} E}$$

has a fixpoint $T_{E,\rho}$.

Proof. By positiveness and circularity for every $X, Y \in \operatorname{dom} E$ it is not possible that $X \succ_+ Y$ and $X \succ_- Y$ at the same time. If it were the case as $Y \succ X$, we would have $X \succ_- X$ whatever the sign of the dependency of Y on X. So, by Lemma 13 we know that $\pi_X^{\operatorname{dom} E} F_{E,\rho}(\langle \mathcal{X}_Y \rangle_{Y \in \operatorname{dom} E}$ is increasing on those $\mathcal{X}_Y$'s with $X \succ_+ Y$ (as $Y \notin \mathrm{V}^-(E(X))$) and decreasing on the others.

Let us fix any $X_0 \in \operatorname{dom} E$. We define the order $\sqsubseteq$ on $\mathsf{SAT}^{\operatorname{dom} E}$ by

$$\langle \mathcal{X}_X \rangle_{X \in \operatorname{dom} E} \sqsubseteq \langle \mathcal{Y}_X \rangle_{X \in \operatorname{dom} E} \iff \forall X \in \operatorname{dom} E : \begin{cases} \mathcal{X}_X \subseteq \mathcal{Y}_X & \text{if } X \succ_+ X_0 \ (E), \\ \mathcal{X}_X \supseteq \mathcal{Y}_X & \text{otherwise.} \end{cases}$$

Clearly from completeness of $\subseteq$ and $\supseteq$ derives completeness of $\sqsubseteq$ Then we can see that $F_{E,\rho}$ is monotone increasing with respect to $\sqsubseteq$. Indeed suppose $\langle \mathcal{X}_X \rangle_{X \in \operatorname{dom} E} \sqsubseteq \langle \mathcal{Y}_X \rangle_{X \in \operatorname{dom} E}$. Then take any $X \in \operatorname{dom} E$ with $X \succ_{\varepsilon_1} X_0$, and consider $\pi_X^{\operatorname{dom} E} F_{E,\rho}$. If $Y \succ_{\varepsilon_2} X_0$ by circularity $X_0 \succ Y$, and by positiveness it must be $X_0 \succ_{\varepsilon_2} Y$ so in fact $X \succ_{\varepsilon_1 \varepsilon_2} Y$. So in fact by inspecting all the combinations of ε_1 and ε_2 and exploiting the above fact on monotonicity we see that indeed $\pi_X^{\operatorname{dom} E} F_{E,\rho}$ maps $\sqsubseteq$ to $\subseteq$ if $X \succ_+ X_0$ and to $\supseteq$ otherwise. By completeness of $\sqsubseteq$ we can then take for $T_{E,\rho}$ the least (for example) fixpoint of $F_{E,\rho}$ with respect to $\sqsubseteq$. □

Fact 4. *If $\mathrm{V}(A) \cap \operatorname{dom} E = \emptyset$ then $\rho[E](A) = \rho(A)$.*

Lemma 6. *If E is positive and $A = B \ (E)$ then $\rho[E](A) = \rho[E](B)$.*

Proof. Suppose $E = E_1 + \cdots + E_k$ is the canonic partition into positive subsystems underlying the definition of $\rho[E]$. We proceed by induction on the derivation of $A = B \ (E)$, the only difficult case being proving that $\rho[E_1] \cdots [E_k](X) = \rho[E_1] \cdots [E_k](E(X))$. We can suppose $X \in \operatorname{dom} E_k$. If indeed $X \in \operatorname{dom} E_i$ with $i < k$ then on one side $\rho[E_1] \cdots [E_k](X) = \rho[E_1] \cdots [E_{k-1}](X)$ by definition, while on the other we can see that $\operatorname{dom} E_k \cap \mathrm{V}(E(X)) = \emptyset$ (otherwise $E_i \succ E_k$ which contradicts canonicity), so $\rho[E_1] \cdots [E_k](E(X)) = \rho[E_1] \cdots [E_{k-1}](E(X))$ by Fact 4.

So if $X \in \operatorname{dom} E_k$, we let $\rho' = \rho[E_1] \cdots [E_{k-1}]$. The fact that $\rho[E](X) = \rho'[E_k](X) = \pi_X^{\operatorname{dom} E_k} T_{E_k,\rho'} = \rho'[E_k](E(X))$ is a direct consequence of $T_{E_k,\rho'}$ being a fixpoint of $F_{E_k,\rho'}$ as defined in Lemma 13. □

Lemma 7. *If E is positive, $E; \Gamma \vdash M : A$ and $\rho, \sigma, E \vDash \Gamma$ then $\rho, \sigma, E \vDash M : A$.*

Proof. By induction on the length of the derivation of $E; \Gamma \vdash M : A$. Here are the cases for the three rules differing from usual λ-calculus (taken as in Table 2).

$\{\,\}$-intro: We have that for all i it is the case that $M_i\sigma \in \rho[E](A)$ by inductive hypothesis. Then by S3 $\{M_i\sigma\} = \{M_i\}\,\sigma \in \rho[E](A)$.
$\top$: From inductive hypothesis follows $M\sigma \in \rho[E](A) \subseteq \mathsf{SN} = \rho[E](\top)$.
type equality: The soundness of this rule is a direct consequence of Lemma 6. □

The next lemma is a consequence of how values are typed and translated.

Lemma 14. *If $R; L; \Gamma \vdash V : A, e \mapsto V'$, then $R; L; \Gamma \vdash V : A, \emptyset \mapsto V'$.*

Theorem 2. *If $R; L; \Gamma \vdash S, M : A, e \mapsto M'$ and $S, M \to T, N$, then there is $L' \supseteq L$ and N' with $R; L'; \Gamma \vdash T, N : A, e \mapsto N'$, such that $M' \xrightarrow{+} N'$ in Λ_∞.*

Proof. We proceed by induction on the size of the derivation, reasoning by cases on the second last rule (i.e. the rule just preceding the one for the interaction with the store). Let σ_e be the substitution $\left[\{V'\}_{R;;\Gamma \vdash V : R(r), \emptyset \mapsto V'} / x_r\right]_{r \in e}$, so that $R; L; \Gamma \vdash M : A, e \mapsto M''$ for some M'' with $M' = M''\sigma_e$. Notice how σ_e does not depend in any way on the store.

The passing to context of the reduction is easily handled by applying inductive hypotheses. As an example, take $S, M|P \to T, N|P$, with $\langle M', P' \rangle$ translating $M|P$. Then we have $R; L; \Gamma \vdash M : \top, e \mapsto M'$ and $S, M \to T, N$, so inductive hypothesis gives L' and N' with $R; L'; \Gamma \vdash N \mapsto N'$ and $M'\sigma_e \to N'\sigma_e$, so that

$$\langle M', P' \rangle \sigma_e = \langle M'\sigma_e, P'\sigma_e \rangle \to \langle N'\sigma_e, P'\sigma_e \rangle = \langle N', P' \rangle \sigma_e,$$

and indeed $R; L'; \Gamma \vdash N|P : \top \mapsto \langle N', P' \rangle \sigma_e$.

We are therefore left with the cases of immediate redexes.

Case 1 (β-redex). let $R; L; \Gamma \vdash (\lambda x.M)V : B, e \mapsto P$, with $S, (\lambda x.M)V \to S, M\,[V/x]$ and application as the last rule of the derivation. By Lemma 14 we can restrict to the case where $R; L; \Gamma, x : A \vdash M : B, e \mapsto M'$ and $R; L; \Gamma \vdash V : A, \emptyset \mapsto V'$. Then Lemma 8 entails this case.

Case 2 (new). let $R; L; \Gamma \vdash \mathtt{new} : \mathtt{ref}_r, \emptyset \mapsto I; I$, with $S, \mathtt{new} \to S, l$, l a fresh location. By setting $L' = L\,[l \mapsto r]$, we immediately get $R; L'; \Gamma \vdash l : \mathtt{ref}_r \mapsto I$, with $I; I \to I$.

Case 3 (dereferencing). by Lemma 14 we can reduce to the case where $R; L; \Gamma \vdash !l : R(r), \{r\} \mapsto x_r$, with $L(l) = r$ and $S, !l \to S, S(l)$. By typing of the store have $R; ; \Gamma \vdash S(l) : R(r), \emptyset \mapsto U'$, which can become $R; L; \Gamma \vdash S(l) : R(r), \{r\} \mapsto U'$. We conclude by observing that $I; x_r\sigma_e = I; \{V'\}_{R;;\Sigma \vdash V : R(r) \mapsto V'} \xrightarrow{2} U'$.

Case 4 (assignment). again by Lemma 14, we can suppose we have $R; L; \Gamma \vdash l : \mathtt{ref}_{L(l)}, \emptyset \mapsto I$ and $R; L; \Gamma \vdash V : R(L(l)), \emptyset \mapsto V'$, giving $R; L; \Gamma \vdash l := V : 1 \mapsto I; V'; I$, and $S, l := V \to S\left[l \mapsto V\right], \langle\rangle$. Indeed $I; V'; I$ reduces in two steps to I, where $R; L; \Gamma \vdash \langle\rangle : 1, \emptyset \mapsto I$. □

Lemma 9. *If $R; L; \Gamma \vdash M : A, e \mapsto M'$ (resp. $R; L; \Gamma \vdash S, M : A, e \mapsto M'$), then in Λ_∞ we have $R^\bullet; \Gamma^\bullet, (x_r : X_r)_{r \in e} \vdash M' : A^\bullet$ (resp. $R^\bullet; \Gamma^\bullet \vdash M' : A^\bullet$).*

Proof. Once the statement for typing terms is obtained, the one for the interactions with stores follows. Indeed if we have $R; L; \Gamma \vdash S, M : A, e \mapsto M'$ then we would have $R^\bullet; \Gamma^\bullet, (x_r : X_r)_{r \in e} \vdash M'' : A^\bullet, e \mapsto M''$ with $M' = M''\sigma_e$, with $\sigma_e = \left[\{V'\}_{R;;\Gamma \vdash V : R(r) \mapsto V'} / x_r\right]_{r \in e}$. But then for every $r \in e$ we have

$$\dfrac{\dfrac{\forall (V', R;; \Gamma \vdash V : R(r), \emptyset \mapsto V') : \quad R^\bullet; \Gamma^\bullet \vdash V' : R^\bullet(X_r)}{R^\bullet; \Gamma^\bullet \vdash \{V'\}_{R;;\Gamma \vdash V : R(r), \emptyset \mapsto V'} : R^\bullet(X_r)} \qquad R^\bullet(X_r) = X_r \ \ (R^\bullet)}{R^\bullet; \Gamma^\bullet \vdash \{V'\}_{R;;\Gamma \vdash V : R(r), \emptyset \mapsto V'} : X_r}$$

so that the desired typing for M' is obtained by a standard substitution argument.

So we reason by induction on the typing, splitting on the last rule used. We refer to the same terminology of Table 1. Let $\Gamma_e = (x_r : X_r)_{r \in e}$.

Case 1 (axioms). These cases are trivial. In particular notice how in typing the translation of locations any reference to their actual type is lost. Indeed we have $R; L; \Gamma \vdash l : \mathtt{ref}_{L(l)}, \emptyset \mapsto I$ and indeed $R^\bullet; \Gamma^\bullet \vdash I : \top \to \top = (\mathtt{ref}_{L(l)})^\bullet$.

Case 2 (abstraction). By inductive hypothesis we have $R^\bullet; \Gamma^\bullet, x : A^\bullet, \Gamma_e \vdash M' : B^\bullet$, so that repeated abstraction rules give

$$R^\bullet; \Gamma^\bullet, \Gamma_\emptyset \vdash \lambda x.\lambda (x_r)_{r \in e}.M' : A^\bullet \to (X_r)_{r \in e} \to B^\bullet = (A \xrightarrow{e} B)^\bullet.$$

Case 3 (application). We have $R^\bullet; \Gamma^\bullet, \Gamma_f \vdash M' : A^\bullet \to (X_r)_{r \in e} \to B^\bullet$ and $R^\bullet; \Gamma^\bullet, \Gamma_f \vdash N' : A^\bullet$ with $e \subseteq f$, so repeated applications with suitable axioms on the x_r's give $R^\bullet; \Gamma^\bullet, \Gamma_f \vdash M'N'(x_r)_{r \in e} : B^\bullet$.

Case 4 (dereferencing). M' is typable, but we do not need its actual type. As $r \in e$ we immediately get $R^\bullet; \Gamma^\bullet, \Gamma_e \vdash M'; x_r : X_r = (R(r))^\bullet \ \ (R^\bullet)$, i.e. we use a type equality in the type system $R^\bullet$.

Case 5 (assignment). M' and N' are typable, but then $M'; N'; I$ is trivially typable with $1^\bullet = \top \to \top$.

Case 6 (dummy effects). This case follows from a standard weakening property of Λ_∞ type assignment.

Case 7 (top). Λ_∞'s $\top$ rule fits the bill.

Case 8 (parallel composition). If M' and N' are both typed with $\top$, we clearly have $R^\bullet; \Gamma^\bullet, \Gamma_e \vdash \langle M', N' \rangle : (\top \to \top \to \top) \to \top$, which can be again typed as $\top = \top^\bullet$. □