On stratified regions

Roberto Amadio

On stratified regions

Roberto M. Amadio
Université Paris Diderot (Paris 7)*

April 14, 2009

Abstract

Type and effect systems are a tool to analyse statically the behaviour of programs with effects. We present a proof based on the so called reducibility candidates that a suitable stratification of the type and effect system entails the termination of the typable programs. The proof technique covers a simply typed, multi-threaded, call-by-value lambda-calculus, equipped with a variety of scheduling (preemptive, cooperative) and interaction mechanisms (references, channels, signals).
Keywords Types and effects. Termination. Reducibility candidates.

1 Introduction

In the framework of functional programs, the relationship between type systems and termination has been extensively studied through the Curry-Howard correspondence. It would be interesting to extend these techniques to programs with *effects*. By effect we mean the possibility of executing operations that modify the state of a system such as reading/writing a reference or sending/receiving a message.

Usual type systems as available, *e.g.*, in various dialects of the ML programming language, are too poor to account for the behaviour of programs with effects. A better approximation is possible if one abstracts the state of a system in a certain number of *regions* and if the types account for the way programs act on such regions. So-called *type and effect* systems [5] are an interesting formalisation of this idea and have been successfully used to analyse statically the problem of heap-memory deallocation [6]. On the other hand, the proof-theoretic foundations of such systems are largely unexplored. Only recently, it has been shown [1] that a *stratification* of the regions entails termination in a certain higher-order language with cooperative threads and references. Our purpose here is to revisit this result trying to clarify and extend both its scope and its proof technique (a more technical comparison is delayed to section 4).

In section 2, we introduce a λ-calculus with *regions*. Regions are an abstraction of dynamically generated values such as references, channels, and signals, and the reduction rules of the calculus are given in such a way that the reduction rules for references, channels, and signals arise as a particular case of those given for regions.

In section 3, we describe a simple *type and effect* system along the lines of [5]. In this discipline, types carry information on the regions on which the evaluated expressions may

*PPS, UMR-CNRS 7126. Work partially supported by ANR-06-SETI-010-02.

read or write. The discipline allows to write in a region r values that have an effect on the region r itself. In turn, this allows to simulate recursive definitions and thus to produce non terminating behaviours.

In section 4, following [1], we describe a stratification of the regions. The idea is that regions are ordered and that a value written in a region may only produce effects in smaller regions. We then propose a new reducibility candidates interpretation (see, *e.g.*, [3] for a good survey) entailing the termination of typable programs.

In section 5, we enrich the language with an operator for reacting to the termination of the computation. The language we consider is then *timed* in the sense that a computation is regarded as a possibly infinite sequence of instants. An instant ends when the calculus cannot progress anymore (cf. synchronous/timed languages such as ESTEREL [2] and Timed CCS [4]). We extend the stratified typing rules to this language and show by means of a translation into the core language that typable programs terminate. Recursive calls in the resulting language are still typable as long as they arise at a *later* instant of the computation.

Appendix A contains the main proofs and appendix B summarizes the type and effect systems considered.

2 A λ-calculus with regions

We consider a λ-calculus with *regions*. Regions are *abstractions* of dynamically generated 'pointers' which, depending on the context, are called references, channels, or signals. Of course, one may consider more concrete calculi where such pointers are explicitly represented, but since the main arguments can be carried on at the level of regions, we prefer to avoid pointers to keep the notation simple.

2.1 Syntax

We consider the following syntactic categories:

$x, y, \ldots$	(variables)
$r, s, \ldots$	(regions)
$e, e', \ldots$	(finite sets of regions)
$A ::= 1 \mid \mathsf{Reg}_r A \mid (A \xrightarrow{e} A)$	(types)
$\Gamma ::= x_1 : A_1, \ldots, x_n : A_n$	(context)
$R ::= r_1 : A_1, \ldots, r_n : A_n$	(region context)
$M ::= x \mid r \mid * \mid \lambda x.M \mid MM \mid !M \mid M := M$	(terms)
$V ::= r \mid * \mid \lambda x.M$	(values)
$v, v', \ldots$	(sets of value)
$S ::= (r \leftarrow v) \mid S, S$	(stores)
$X ::= M \mid S$	(stores or terms)
$P ::= X \mid X, P$	(programs)

We briefly comment the notation: 1 is the terminal (unit) type with value $*$; $\mathsf{Reg}_r A$ is the type of a region r containing values of type A; $A \xrightarrow{e} B$ is the type of functions that when given a value of type A may produce a value of type B and an effect on the regions in e; ! is the operator to read *some* value in a region and := is the operator to *insert* a value in a region.

We write $[N/x]M$ for the substitution of N for x in M. If $R = r_1 : A_1, \ldots, r_n : A_n$ then $dom(R) = \{r_1, \ldots, r_n\}$. We also define the term $\mathsf{reg}_r M$ as an abbreviation for $(\lambda x.r)(r := M)$. Thus the difference between $r := M$ and $\mathsf{reg}_r M$ is that in the first case we return $*$ while in the second we return r. When writing a program $P = X_1, \ldots, X_n$ we regard the symbol ',' as associative and commutative, or equivalently we regard a program as a multi-set of terms and stores. We write $(r \leftarrow V)$ for $(r \leftarrow \{V\})$. We shall identify the store $(r \leftarrow v_1), (r \leftarrow v_1)$ with the store $(r \leftarrow v_1 \cup v_2)$. We denote with $dom(S)$ the set of regions r such that $(r \leftarrow v)$ occurs in S and with $S(r) = \{V \mid (r \leftarrow V) \text{ occurs in } S\}$.

2.2 Reduction

A call-by value *evaluation context* E is defined as:

$$E ::= [\,] \mid EM \mid VE \mid !E \mid E := M \mid V := E$$

An *elementary* evaluation context is defined as:

$$El ::= [\,]M \mid V[\,] \mid ![\,] \mid [\,] := M \mid V := [\,]$$

An evaluation context can be regarded as the finite composition (possibly empty) of elementary evaluation contexts. The *reduction* on programs is defined as follows:

$$\frac{}{E[(\lambda x.M)V] \to E[[V/x]M]} \qquad \frac{}{E[!r], (r \leftarrow V) \to E[V], (r \leftarrow V)}$$

$$\frac{}{E[r := V] \to E[*], (r \leftarrow V)} \qquad \frac{P \to P'}{P, P'' \to P', P''}$$

Note that the semantics of assignment amounts to *add* rather than to *update* a binding between a region and a value. Hence a region can be bound at the same time to several values (possibly infinitely many) and when we dereference a region we select non-deterministically one of them.

As already mentioned, the notion of region is intended to abstract some familiar programming concepts such as references, channels, or signals. Specifically: (i) when writing a reference, we replace the previously written value (if any), (ii) when reading a (unordered, unbounded) channel we consume (remove from the store) the value read, and finally (iii) the values written in a signal persist within an instant and disappear at the end of it.[1]

One can easily formalise the reduction rules for references, channels, and signals, and check that (within an instant) each reduction step is simulated by at least one reduction step in the calculus with regions. Thus, typing disciplines that guarantee termination for the calculus with regions will guarantee the same property when adapted to references, channels, or signals.

3 Types and effects: unstratified case

We introduce a simple *type and effect* system along the lines of [5]. The following rules define when a region context R is *compatible* with a type A (judgement $R \downarrow A$):

$$\frac{}{R \downarrow 1} \qquad \frac{R \downarrow A \quad R \downarrow B \quad e \subseteq dom(R)}{R \downarrow (A \xrightarrow{e} B)} \qquad \frac{r : A \in R}{R \downarrow \mathsf{Reg}_r A}$$

[1] Signals arise in synchronous/timed models where the computation is regulated by a notion of instant or phase (see section 5).

The following rules define when a region context is well formed (judgement $R \vdash$) and when a type and effect is well-formed with respect to a region context (judgements $R \vdash A$ and $R \vdash (A, e)$).

$$\frac{\forall\, r \in dom(R)\;\; R \downarrow R(r)}{R \vdash} \qquad \frac{R \vdash \quad R \downarrow A}{R \vdash A} \qquad \frac{R \vdash A \quad e \subseteq dom(R)}{R \vdash (A, e)}$$

A more informal way to express the condition is to say that a judgement $r_1 : A_1, \ldots, r_n : A_n \vdash B$ is well formed provided that: (1) all the region names occurring in the types $A_1, \ldots, A_n, B$ belong to the set $\{r_1, \ldots, r_n\}$ and (2) all subtypes of the shape $\mathsf{Reg}_{r_i} C$ with $i \in \{1, \ldots, n\}$ and contained in the types $A_1, \ldots, A_n, B$ are such that $C = A_i$. It is easily checked that the following properties hold:

$$\begin{array}{lll} R \vdash 1 & \text{iff} & R \vdash \\ R \vdash \mathsf{Reg}_r A & \text{iff} & R \vdash \text{ and } R(r) = A \\ R \vdash A \xrightarrow{e} B & \text{iff} & R \vdash, R \vdash A, R \vdash B, \text{ and } e \subseteq dom(R) \\ R \vdash & \text{iff} & \forall\, r \in dom(R)\;\; R \vdash R(r) \end{array}$$

The subset relation on effects induces a *subtyping* relation on types and on pairs of types and effects which is defined as follows (judgements $R \vdash A \leq A'$, $R \vdash (A, e) \leq (A', e')$):

$$\frac{R \vdash A}{R \vdash A \leq A} \qquad \frac{\begin{array}{c} R \vdash A' \leq A \quad R \vdash B \leq B' \\ e \subseteq e' \subseteq dom(R) \end{array}}{R \vdash (A \xrightarrow{e} B) \leq (A' \xrightarrow{e'} B')} \qquad \frac{\begin{array}{c} R \vdash A \leq A' \\ e \subseteq e' \subseteq dom(R) \end{array}}{R \vdash (A, e) \leq (A', e')}$$

We notice that the transitivity rule:

$$\frac{R \vdash A \leq B \qquad R \vdash B \leq C}{R \vdash A \leq C}$$

can be derived via a simple induction on the height of the proofs.

We now turn to the typing rules for the terms. We shall write $R \vdash x_1 : A_1, \ldots, x : A_n$ if $R \vdash$ and $R \vdash A_i$ for $i = 1, \ldots, n$. Note that in the following rules we always refer to the *same* region context R.

$$\frac{R \vdash \Gamma \quad x : A \in \Gamma}{R; \Gamma \vdash x : (A, \emptyset)} \qquad \frac{R \vdash \Gamma \quad r : A \in R}{R; \Gamma \vdash r : (\mathsf{Reg}_r A, \emptyset)} \qquad \frac{R \vdash \Gamma}{R; \Gamma \vdash * : (1, \emptyset)}$$

$$\frac{R; \Gamma, x : A \vdash M : (B, e)}{R; \Gamma \vdash \lambda x.M : (A \xrightarrow{e} B, \emptyset)} \qquad \frac{R; \Gamma \vdash M : (A \xrightarrow{e_2} B, e_1) \quad R; \Gamma \vdash N : (A, e_3)}{R; \Gamma \vdash MN : (B, e_1 \cup e_2 \cup e_3)}$$

$$\frac{R; \Gamma \vdash M : (\mathsf{Reg}_r A, e)}{R; \Gamma \vdash !M : (A, e \cup \{r\})} \qquad \frac{R; \Gamma \vdash M : (\mathsf{Reg}_r A, e_1) \quad R; \Gamma \vdash N : (A, e_2)}{R; \Gamma \vdash M := N : (1, e_1 \cup e_2 \cup \{r\})}$$

$$\frac{R; \Gamma \vdash M : (A, e) \quad R \vdash (A, e) \leq (A', e')}{R; \Gamma \vdash M : (A', e')}$$

Finally, the typing rules are extended to stores, and general multi-threaded programs as

follows.[2]

$$\frac{r : A \in R \quad \forall V \in v \;\; R;\Gamma \vdash V : (A, \emptyset)}{R;\Gamma \vdash (r \leftarrow v) : (1, \emptyset)} \qquad \frac{R;\Gamma \vdash X_i : (A_i, e_i) \quad i = 1, \ldots, n \geq 1}{R;\Gamma \vdash X_1, \ldots, X_n : e_1 \cup \cdots \cup e_n}$$

Remark 1 *The derived typing rule for* $\mathsf{reg}_r M$ *is as follows:*

$$\frac{r : A \in R \quad R;\Gamma \vdash M : (A, e)}{R;\Gamma \vdash \mathsf{reg}_r M : (\mathsf{Reg}_r A, e \cup \{r\})}$$

One can derive a more traditional 'effect-free' type system by *erasing all the effects* from the types and the typing judgements. Note that in the resulting system the subtyping rules are useless. We shall write $\vdash^{ef}$ for provability in this system. This 'weaker' type system suffices to state a decomposition property of the terms which is proven by induction on the structure of the term.

Proposition 2 (decomposition) *If* $R;\vdash^{ef} M : A$ *is a well-typed closed term then exactly one of the following situations arises where* E *is an evaluation context:*

1. M *is a value.*
2. $M = E[\Delta]$ *and* Δ *has the shape* $(\lambda x.N)V$, $r := V$, *or* $!r$.

3.1 Basic properties of typing and evaluation

We observe some basic properties: (i) one can weaken both the type and region contexts, (ii) typing is preserved when we replace a variable with an effect-free term of the same type, and (iii) typing is preserved by reduction. If S is a store and e is a set of regions then $S_{|e}$ is the store S restricted to the regions in e.

Proposition 3 (basic properties, unstratified) *The following properties hold:*

weakening *If* $R;\Gamma \vdash M : (A, e)$ *and* $R, R' \vdash \Gamma, \Gamma'$ *then* $R, R';\Gamma, \Gamma' \vdash M : (A, e)$.

substitution *If* $R;\Gamma, x : A \vdash M : (B, e)$ *and* $R;\Gamma \vdash N : (A, \emptyset)$ *then* $R;\Gamma \vdash [N/x]M : (B, e)$.

subject-reduction *Let* $\mathbf{M}$ *denote a sequence* $M_1, \ldots, M_n$. *If* $R, R';\vdash \mathbf{M}, S : e$, $R \vdash e$, *and* $\mathbf{M}, S \to \mathbf{M}', S'$ *then* $R, R';\vdash \mathbf{M}', S' : e$, $S_{|dom(R')} = S'_{|dom(R')}$, *and* $\mathbf{M}, S_{|dom(R)} \to \mathbf{M}', S'_{|dom(R)}$. *Moreover, if* $\mathbf{M} = M$ *and* $R, R' \vdash M : (A, e)$ *then* $\mathbf{M}' = M'$ *and* $R, R' \vdash M' : (A, e)$.

The weakening and substitution properties are shown directly by induction on the proof height. Concerning subject reduction, it is useful to notice that that if a term M, of type and effect (A, e), is ready to read/write the region r then $r \in e$. This follows from an analysis of the evaluation context. Then we prove the assertion by case analysis on the reduction rule applied, relying on the substitution property.

[2]The fact that we attribute the type 1 to a store is a mere convention which allows for a uniform notation for the typing of stores and terms.

Remark 4 *The subject reduction property is formulated so as to make clear that the type and effect system indeed delimits the interactions a term may have with the store. Note that a term may refer to regions which are not explicitly mentioned in its type and effect. For instance, consider* $M = (\lambda f.*)(\lambda x.!rx)$ *and let* $R = r : 1 \xrightarrow{\emptyset} 1$. *Then* $R; \vdash M : (1, \emptyset)$, $\emptyset \vdash (1, \emptyset)$ *but* $\emptyset; \emptyset \not\vdash M : (1, \emptyset)$. *The subject reduction property guarantees that such a term will only read/write regions included in the region context needed to type its type and effect.*

3.2 Recursion

In our (unstratified) calculus, we can write in a region r a functional value $\lambda x.M$ where M reads from the region r itself. For instance, $\mathsf{reg}_r(\lambda x.(!r)x)$.

This kind of circularity leads to diverging computations such as:

$$\begin{array}{lcll} (!(\mathsf{reg}_r \lambda x.(!r)x))* & \to & (!r)*, (r \leftarrow \lambda x.(!r)x) & \to \\ (\lambda x.(!r)x)*, (r \leftarrow \lambda x.(!r)x) & \to & (!r)*, (r \leftarrow \lambda x.(!r)x) & \to \cdots \end{array}$$

It is well known that this phenomena can be exploited to simulate recursive definitions. Specifically, we define:

$$\mathsf{fix}_r f.M = \lambda x.(!(\mathsf{reg}_r(\lambda x.[\lambda x.(!r)x/f]M\ x)))\ x \tag{1}$$

By a direct application of the typing rules and proposition 3(substitution), one can derive a rule to type $\mathsf{fix}_r f.M$.

Proposition 5 (type fixed-point) *The following typing rule for the fixed point combinator is derived:*

$$\frac{r : A \xrightarrow{e} B \in R \quad r \in e \qquad R; \Gamma, f : A \xrightarrow{e} B \vdash M : (A \xrightarrow{e} B, \emptyset)}{R; \Gamma \vdash \mathsf{fix}_r f.M : (A \xrightarrow{e} B, \emptyset)} \tag{2}$$

For a concrete example, assume basic operators on the integer type and let M be the factorial function:

$$M = \lambda x.\mathsf{if}\ x = 0\ \mathsf{then}\ 1\ \mathsf{else}\ x * f(x-1)\ .$$

Then compute $(\mathsf{fix}_r f.M)1$. In this case we have $e = \{r\}$ and $r : \mathit{int} \xrightarrow{r} \mathit{int} \in R$.

4 Types and effects: stratified case

As we have seen, an unstratified simply typed calculus with effects may produce diverging computations. To avoid this, a natural idea proposed by G. Boudol in [1] is to *stratify* regions.

Intuitively, we fix a well-founded order on regions and we make sure that values stored in a region r can only produce effects on smaller regions. For instance, suppose V is a value with type $(1 \xrightarrow{\{r\}} 1)$. Intuitively, this means that when applied to an argument $U : 1$, V may produce an effect on region $\{r\}$. Then the value V can only be stored in regions larger than r. We shall see that this stratification allows for an inductive definition of the values that can be stored in a given region.

The only change in the type system concerns the judgements $R \vdash$, $R \vdash A$, and $R \vdash (A, e)$ whose rules are redefined as follows:

$$\frac{}{\emptyset \vdash} \qquad \frac{R \vdash A \quad r \notin dom(R)}{R, r : A \vdash} \qquad \frac{R \vdash}{R \vdash 1}$$

$$\frac{R \vdash \quad r : A \in R}{R \vdash \mathsf{Reg}_r A} \qquad \frac{R \vdash A \quad R \vdash B \quad e \subseteq dom(R)}{R \vdash A \xrightarrow{e} B} \qquad \frac{R \vdash A \quad e \subseteq dom(R)}{R \vdash (A, e)} \ .$$

Proviso Henceforth we shall use $\vdash$ to refer to provability in the stratified system and $\vdash^u$ for provability in the *unstratified* one. The former implies the latter since $R \vdash$ implies $R \vdash^u$ and $R \vdash A$ implies $R \vdash^u A$, while the other rules are unchanged.

4.1 Basic properties revisited

The main properties we have proven for the unstratified system can be specialised to the stratified one.

Proposition 6 (basic properties, stratified) *The following properties hold in the stratified system.*

weakening *If* $R; \Gamma \vdash M : (A, e)$ *and* $R, R' \vdash \Gamma, \Gamma'$ *then* $R, R'; \Gamma, \Gamma' \vdash M : (A, e)$.

substitution *If* $R; \Gamma, x : A \vdash M : (B, e)$ *and* $R; \Gamma \vdash N : (A, \emptyset)$ *then* $R; \Gamma \vdash [N/x]M : (B, e)$.

subject reduction *If* $R, R'; \vdash \mathbf{M}, S : e$, $R \vdash e$, *and* $\mathbf{M}, S \to \mathbf{M}', S'$ *then* $R, R'; \vdash \mathbf{M}', S' : e$, $S_{|dom(R')} = S'_{|dom(R')}$, *and* $\mathbf{M}, S_{|dom(R)} \to \mathbf{M}', S'_{|dom(R)}$. *Moreover, if* $\mathbf{M} = M$ *and* $R, R'; \vdash M : (A, e)$ *then* $\mathbf{M}' = M'$ *and* $R, R'; \vdash M' : (A, e)$.

4.2 Interpretation

We describe a *reducibility candidates* interpretation that entails that typed programs terminate. We denote with SN the collection of strongly normalising single-threaded programs, *i.e.*, the programs of the shape M, S such that all reduction sequences terminate. We write $(M, S) \Downarrow (N, S')$ if $M, S \xrightarrow{*} N, S'$ and $N, S' \not\to$. We write $R' \geq R$, and say that R' extends R, if $R' \vdash$ and $R' = R, R''$ for some R''.

The starting idea is that the interpretation of $R \vdash$ is a set of stores and the interpretation of $R \vdash (A, e)$ is a set of terms. One difficulty is that the stores and the terms may depend on a region context R' which extends R. We get around this problem, by making the context R' explicit in the interpretation. Then the interpretation can be given directly by induction on the provability of the judgements $R \vdash$ and $R \vdash (A, e)$. This seems an interesting simplification with respect to the the approach taken in [1], where the interpretation is given with respect to a rather technical well-founded order.

A second characteristic of our approach is that the properties a thread must satisfy are specified with respect to a 'saturated' store which intuitively already contains all the values the thread may write into it. This approach provides a simple argument to extend the termination argument from single-threaded to multi-threaded programs. Indeed, if we a have a set of threads which are guaranteed to terminate with respect to a saturated store then their parallel composition will terminate too. To see this, one can reason by contradiction:

if the parallel composition diverges then one thread must run infinitely often and, since the threads cannot modify the saturated store (what they write is already there), this contradicts the hypothesis that all the threads taken alone with the saturated store terminate.

Region-context Let $R = r_1 : A_1, \ldots, r_n : A_n$ and $R_{r_i} = r_1 : A_1, \ldots, r_{i-1} : A_{i-1}$, for $i = 1, \ldots, n$. We interpret a region-context R as a set of pairs $R' \vdash S$ where R' is a region-context which extends R and a S is a 'saturated' store whose domain coincides with R:

$$\begin{array}{rcl} \underline{R} & = & \{\ R' \vdash S \mid R' \geq R, \quad dom(S) = dom(R), \text{ and for } i = 1, \ldots, n \\ & & \qquad S(r_i) = \{V \mid R' \vdash V \in \underline{R_{r_i} \vdash (A_i, \emptyset)}\}\ \} \end{array}$$

If $R' \geq R$ then $\underline{R}(R')$ is defined as the store S such that $R' \vdash S \in \underline{R}$. Note that, for $r \in dom(R)$ and $R = R_1, r : A, R_2$, $V \in \underline{R}(R')(r)$ means $R' \vdash V \in \underline{R_1 \vdash (A, \emptyset)}$.

Type and effect We interpret a type and effect $R \vdash (A, e)$ as the set of pairs $R' \vdash M$ such that R' extends R, and M is a closed term typable with respect to R' and satisfying suitable properties (1-3 below):

$$\begin{array}{rlll} \underline{R \vdash (A, e)} = & \{R' \vdash M \mid & (1) & R' \geq R, \quad R'; \emptyset \vdash M : (A, e), \\ & & (2) & \text{for all } R'' \geq R'\ M, \underline{R}(R'') \in SN, \text{ and} \\ & & (3) & \text{for all } M', S', R'' \geq R' \quad (M, \underline{R}(R'')) \Downarrow (M', S') \\ & & & \quad \text{implies } S' = \underline{R}(R'') \text{ and } \mathcal{C}(A, R, R'', M')\ \} \end{array}$$

where: $\mathcal{C}(A, R, R'', M') \equiv$

$$\begin{array}{l} (A = 1 \quad \supset \quad M' = *) \wedge \\ (A = \mathsf{Reg}_r B \quad \supset \quad M' = r) \wedge \\ (A = A_1 \xrightarrow{e'} A_2 \quad \supset \quad M' = \lambda x.N \quad \wedge \\ \qquad \text{for all } \ R_1 \geq R'', R_1 \vdash V \in \underline{R \vdash (A_1, \emptyset)} \\ \qquad\quad \text{implies } R_1 \vdash M'V \in \underline{R \vdash (A_2, e')}\ ) \ . \end{array}$$

Suppose $R = r_1 : A_1, \ldots, r_n : A_n$. We note that the interpretation of R depends on the interpretation of $r_1 : A_1, \ldots, r_{i-1} : A_{i-1} \vdash A_i$ for $i = 1, \ldots, n$ and the interpretation of $R \vdash (A, e)$ depends on the interpretation of R and, when $A = A_1 \xrightarrow{e'} A_2$, on the interpretation of $R \vdash (A_1, \emptyset)$ and $R \vdash (A_2, e')$. It is easily verified that the definition of the interpretation is well founded by considering as measure the height of the proof of the interpreted judgement. We also note that such a well-founded definition would not be possible in the unstratified system. For instance, the interpretation of $r : A \vdash (A, \emptyset)$ where $A = 1 \xrightarrow{r} 1$ should refer to a store containing values of type A. Finally, we stress that the interpretations of R and $R \vdash (A, e)$ actually contain terms typable in an extension R' of R but that their properties are stated with respect to a store whose domain is $dom(R)$. This is possible because the type and effect system does indeed delimit the effects a term may have when it is executed (cf. remark 4).

4.3 Basic properties of the interpretation

We say that a term M is *neutral* if it is not a λ-abstraction. The following proposition lists some basic properties of the interpretation. Similar properties arise in the reducibility

candidates interpretations used for 'pure' functional languages, but the main point here is that we have to state them relatively to suitable stores. In particular, the extension/restriction property, which is perhaps less familiar, is crucial to prove the following soundness theorem 9.

Proposition 7 (properties interpretation) *The following properties hold.*

Weakening *If* $R'' \geq R' \geq R$, $R \vdash (A, e)$, *and* $R' \vdash M \in \underline{R \vdash (A, e)}$ *then* $R'' \vdash M \in \underline{R \vdash (A, e)}$.

Extension/Restriction *Suppose* $R'' \geq R' \geq R$ *and* $R \vdash (A, e)$. *Then* $R'' \vdash M \in \underline{R \vdash (A, e)}$ *if and only if* $R'' \vdash M \in \underline{R' \vdash (A, e)}$.

Subtyping *If* $R \vdash (A, e) \leq (A', e')$ *then* $\underline{R \vdash (A, e)} \subseteq \underline{R \vdash (A', e')}$.

Strong normalisation *If* $R' \vdash M \in \underline{R \vdash (A, e)}$ *and* $R'' \geq R'$ *then* $M, \underline{R}(R'') \in SN$.

Reduction closure *If* $R' \vdash M \in \underline{R \vdash (A, e)}$, $R'' \geq R'$, *and* $M, \underline{R}(R'') \to M', S'$ *then* $R'' \vdash M' \in \underline{R \vdash (A, e)}$ *and* $S' = \underline{R}(R'')$.

Non-emptiness *If* $R \vdash A$ *then there is a value* V *such that for all* $R' \geq R$ *and* $e \subseteq dom(R)$, $R' \vdash V \in \underline{R \vdash (A, e)}$.

Expansion closure *Suppose* $R \vdash (A, e)$, $R' \geq R$, $R'; \emptyset \vdash M : (A, e)$, *and* M *is neutral. Then* $R' \vdash M \in \underline{R \vdash (A, e)}$ *provided that for all* $R'' \geq R', M', S'$ *such that* $M, \underline{R}(R'') \to M', S'$ *we have that* $R'' \vdash M' \in \underline{R \vdash (A, e)}$ *and* $S' = \underline{R}(R'')$.

Proof hint.

Weakening We rely on proposition 6((syntactic) weakening) and the fact that, the properties the pairs $R' \vdash M$ must satisfy to belong to $\underline{R \vdash (A, e)}$, must hold for all the extensions $R'' \geq R'$.

Extension/Restriction By definition, $\underline{R}(R'')$ coincides with $\underline{R'}(R'')$ on $dom(R)$. On the other hand, the proposition 6(subject reduction) guarantees that the reduction of a term of type and effect (A, e) will not depend and will not affect the part of the store whose domain is $dom(R') \backslash dom(R)$. We then prove the property by induction on the structure of the type A.

Subtyping This is proven by induction on the the proof of $R \vdash A \leq A'$.

Strong normalisation This follows immediately from the definition of the interpretation.

Reduction closure We know that $M, \underline{R}(R'')$ must normalise to a value satisfying suitable properties and the same saturated store $\underline{R}(R'')$. Moreover, we know that the store can only grow during the reduction. We conclude applying the weakening property.

Non-emptiness/Expansion closure These two properties are proven at once, by induction on the proof height of $R \vdash (A, e)$. We take as values: $*$ for the type 1, r for a type of the shape $\mathsf{Reg}_r B$, and the 'constant function' $\lambda x.V_2$ for a type of the shape $A_1 \xrightarrow{e_1} A_2$ where V_2 is the value inductively built for A_2. To prove $\lambda x.V_2 \in \underline{R \vdash (A_1 \xrightarrow{e_1} A_2, e)}$, we use the inductive hypothesis of expansion closure of $\underline{R \vdash (A_2, e_1)}$. □

4.4 Soundness of the interpretation

By definition, if $R \vdash M \in \underline{R \vdash (A, e)}$ then $R; \vdash M : (A, e)$. We are going to show that the converse holds too. First we need to generalise the notion of reducibility to open terms.

Definition 8 (term interpretation) *We write* $R; x_1 : A_1, \ldots, x_n : A_n \models M : (B, e)$ *if whenever* $R' \geq R$ *and* $R' \vdash V_i \in \underline{R \vdash (A_i, \emptyset)}$ *for* $i = 1, \ldots, n$ *we have that* $R' \vdash [V_1/x_1, \ldots, V_n/x_n]M \in \underline{R \vdash (B, e)}$.

As usual, the main result can be stated as the soundness of the interpretation with respect to the typing rules. Since terms in the interpretation are strongly normalising relatively to a saturated store (cf. proposition 7), it follows that typable (closed) terms are strongly normalising.

Theorem 9 (soundness) *If* $R; \Gamma \vdash M : (B, e)$ *then* $R; \Gamma \models M : (B, e)$.

Proof hint. The proof goes by induction on the typing of the the terms and exploits the properties of the interpretation stated in proposition 7. As usual, the case of the abstraction is proven by appealing to expansion closure and the case of application follows from the very interpretation of the functional types and reduction closure. The cases where we write or read from the store have to be handled with some care. We discuss a simplified situation. Suppose $R' \geq R = R_1, r : A, R_2$.

write Suppose $R; \vdash r := V : (1, \{r\})$ is derived from $R; \vdash V : (A, \emptyset)$. Then, by induction hypothesis, we know that $R' \vdash V \in \underline{R \vdash (A, \emptyset)}$. However, for maintaining the invariant that the saturated store is unchanged, we need to show that $R' \vdash V \in \underline{R_1 \vdash (A, \emptyset)}$, and this is indeed the case thanks to proposition 7(restriction).

read Suppose we have $R'; \vdash !r : (A, \{r\})$. Now notice that proposition 7(non-emptiness) guarantees that $\underline{R}(R')(r)$ is not empty. Thus $!r, \underline{R}(R')$ will reduce to $V, \underline{R}(R')$ for some value V such that $R' \vdash V \in \underline{R_1 \vdash (A, \emptyset)}$. However, what we need to show is that $R' \vdash V \in \underline{R \vdash (A, \emptyset)}$ and this is indeed the case thanks to proposition 7(extension). □

Corollary 10 (1) *The judgement* $R; \vdash M : (A, e)$ *is provable if and only if* $R \vdash M \in \underline{R \vdash (A, e)}$.

(2) *Every typable multi-threaded program* $R; \vdash M_1, \ldots, M_n : e$ *terminates.*

Corollary 10(1), follows from theorem 9 taking the context Γ to be empty. Corollary 10(2) follows from the fact that each thread strongly normalizes with respect to a saturated store. Then its execution is not affected by the execution of other threads in parallel: all these parallel threads could do is to write in the saturated store values which are already there.

5 A timed extension

We consider a synchronous/timed extension of our language. Following an established tradition, we consider that the computation is divided into *instants* and that an instant ends when the computation cannot progress. Then we need at least an additional operator that allows to write programs that *react* to the end of the instant by changing their state in the following instant. We shall see that the termination of the typable programs can be obtained by mapping reductions in the extended language into reductions in the core language.

Syntax and Reduction We extend the collection of terms as follows: $M ::= \cdots \mid M \triangleright M$, where the operator *else-next*, written $M \triangleright N$, tries to run M and, if it fails, runs N in the following instant (cf. [4]). We extend the evaluation contexts assuming: $E ::= \cdots \mid E \triangleright M$, and the *elementary* evaluation contexts assuming: $El ::= \cdots \mid [\,] \triangleright M$.

We define a simplification operator *red* that removes from a context all pending branches else-next:

$$red(E) = \begin{cases} [\,] & \text{if } E = [\,] \\ red(E') & \text{if } E = E' \triangleright N \\ El[red(E')] & \text{otherwise, if } E = El[E'] \end{cases}$$

We say that an evaluation context E is time insensitive if $red(E) = E$. We adapt the reduction rules defined in section 2 as follows:

$$\frac{}{E[(\lambda x.M)V] \to red(E)[[V/x]M]} \qquad \frac{}{E[!r], (r \leftarrow V) \to red(E)[V], (r \leftarrow V)}$$

$$\frac{}{E[r := V] \to red(E)[*], (r \leftarrow V)}\ .$$

Further, we have to describe how a program reacts to the end of the computation. This is specified by the relation $\xrightarrow{\mathsf{tick}}$ below:

$$\frac{}{V \xrightarrow{\mathsf{tick}} V} \qquad \frac{M = E[!r] \quad E \text{ time insensitive}}{M \xrightarrow{\mathsf{tick}} M}$$

$$\frac{M = E[E'[\Delta] \triangleright N] \quad E \text{ time insensitive} \quad \Delta ::= V \mid !r}{M \xrightarrow{\mathsf{tick}} E[N]}$$

$$\frac{}{S \xrightarrow{\mathsf{tick}} S} \qquad \frac{P_1, P_2 \not\to \quad P_i \xrightarrow{\mathsf{tick}} P_i' \quad i = 1, 2}{P_1, P_2 \xrightarrow{\mathsf{tick}} P_1', P_2'}\ .$$

For instance, we can write $(\lambda x.M)!r \triangleright N$ for a thread that tries to read a value from the region r in the first instant and if it fails it resumes the computation with N in the following instant. We can also write $* \triangleright N$ for a thread that (unconditionally) stops its computation for the current instant and resumes it with N in the following instant.

Note that $P \xrightarrow{\mathsf{tick}}$ only if $P \not\to$. The converse is in general false, but it will be true for well-typed closed programs (cf. proposition 11). Thus for well-typed closed programs the principle is that time passes (a $\xrightarrow{\mathsf{tick}}$ transition is possible) exactly when the computation cannot progress (a $\to$ transition is impossible). Then termination is obviously a very desirable property of timed/synchronous programs.

Typing The typing rules for the terms are extended as follows:

$$\frac{R; \Gamma \vdash M : (A, e) \quad R; \Gamma \vdash N : (A, e')}{R; \Gamma \vdash M \triangleright N : (A, e)}\ .$$

Note that in typing $M \triangleright N$ we only record the effect of the term M, that is we focus on the effects a term may produce in the first instant while neglecting those that may be produced at later instants.

Reduction The decomposition proposition 2 can be lifted to the extended language.

Proposition 11 (decomposition extended) *If $\vdash^{ef} M : A$ is a well-typed closed thread then exactly one of the following situations arises where E is a time insensitive evaluation context:*

1. *M is a value.*
2. *$M = E[\Delta]$ and Δ has the shape $(\lambda x.N)V$, $r := V$, or $!r$.*
3. *$M = E[E'[\Delta] \triangleright N]$ and Δ has the shape V, $(\lambda x.N)V$, $r := V$, or $!r$.*

The third case in proposition 11 corresponds to the situation where the redex Δ is under the scope of an else-next.

We can adapt the weakening, substitution, and subject reduction properties; we focus here on the stratified case.

Proposition 12 (basic properties, stratified extended) *The following properties hold in the stratified system.*

weakening *If $R; \Gamma \vdash M : (A, e)$ and $R, R' \vdash \Gamma, \Gamma'$ then $R, R'; \Gamma, \Gamma' \vdash M : (A, e)$.*

substitution *If $R; \Gamma, x : A \vdash M : (B, e)$ and $R; \Gamma \vdash N : (A, \emptyset)$ then $R; \Gamma \vdash [N/x]M : (B, e)$.*

context substitution *If $R; \Gamma, x : A \vdash E[x] : (B, e)$ where x is not free in the evaluation context E and $R; \Gamma \vdash N : (A, e')$ then $R; \Gamma \vdash E[N] : (B, e \cup e')$.*

subject reduction *If $R, R'; \Gamma \vdash \mathbf{M}, S : e$, $R \vdash e$, and $\mathbf{M}, S \to \mathbf{M}', S'$ then $R, R'; \Gamma \vdash \mathbf{M}', S' : e$, $S_{|dom(R')} = S'_{|dom(R')}$, and $\mathbf{M}, S_{|dom(R)} \to \mathbf{M}', S'_{|dom(R)}$. Moreover, if $\mathbf{M} = M$ and $R, R' \vdash M : (A, e)$ then $\mathbf{M}' = M'$ and $R, R'; \vdash M' : (A, e)$.*

tick reduction *If $R; \vdash \mathbf{M}, S : e$, and $\mathbf{M}, S \xrightarrow{\mathsf{tick}} \mathbf{M}', S'$ then $S = S'$ and there is an effect e' such that $R; \vdash \mathbf{M}', S : e'$.*

The proofs of weakening and substitution proceed as in proposition 6. The proof of context substitution requires an analysis of the shape of the evaluation context. The reader will notice that now we need to prove that typing is preserved both by ordinary reduction (subject reduction) and by the passage of time (tick reduction). Concerning subject reduction, we have to verify that the operator *red* that removes the pending branches preserves the typing. Concerning the tick reduction, notice that the effect of the reduced term might be incomparable with the effect of the term to be reduced. Still the context substitution property allows to conclude that the resulting term is well-typed.

Translation We consider a translation that removes the else-next operator while preserving typing and reduction. Namely, we define a function $\langle \_ \rangle$ on terms such that $\langle M \triangleright N \rangle = \langle M \rangle$, $\langle x \rangle = x$, $\langle * \rangle = *$, $\langle r \rangle = r$, and which commutes with the other operators (abstraction, application, reading, and writing). Also the translation is extended to stores and programs in the obvious way: $\langle (r \leftarrow V) \rangle = (r \leftarrow \langle V \rangle)$, $\langle X_1, \ldots, X_n \rangle = \langle X_1 \rangle, \ldots, \langle X_n \rangle$.

Proposition 13 (1) *If* $R;\Gamma \vdash M : (A, e)$ *then* $R;\Gamma \vdash \langle M\rangle : (A, e)$.
(2) *If* $R;\Gamma \vdash P : e$ *then* $R;\Gamma \vdash \langle P\rangle : e$.
(3) *If* $R;\vdash P : e$ *and* $P \to P'$ *then* $\langle P\rangle \to \langle P'\rangle$.
(4) *A program* P *terminates if* $\langle P\rangle$ *terminates.*

The proof of this proposition is direct. In particular, to prove (3) we show that the translation commutes with the substitution and that the translation of an evaluation context is again an evaluation context.

Fixed-point, revisited The typing rule (2) proposed for the fixed-point combinator cannot be applied in the stratified system as the condition $r : A \xrightarrow{e} B \in R$ and $r \in e$ cannot be satisfied. However, we can still type recursive calls that happen in a later instant.

Proposition 14 (type fixed-point, revisited) *The following typing rule for the fixed point combinator is derived in the stratified system*

$$\frac{\begin{array}{c} r : A \xrightarrow{e} B \in R \\ R;\Gamma, f : A \xrightarrow{e\cup\{r\}} B \vdash M : (A \xrightarrow{e} B, \emptyset) \end{array}}{R;\Gamma \vdash \mathsf{fix}_r f.M : (A \xrightarrow{e\cup\{r\}} B, \emptyset)} \tag{3}$$

We prove this proposition by a direct application of the typing rules and the proposition 12(substitution). To see a concrete example where the rule can be applied, consider a thread that at each instant writes an integer in a region r' (we assume a basic type *int* of integers):

$$M = \lambda x.(\lambda z. * \triangleright f(x+1))(r' := x)$$

Then, *e.g.*, $(\mathsf{fix}_r f.M)1$ is the infinite behaviour that at the *i-th* instant writes i in region r'. One can check the typability of $\mathsf{fix}_r f.M$ taking as (stratified) region context $R = r' : int, r : int \xrightarrow{\{r'\}} 1$.

6 Conclusion

We have introduced a λ-calculus with regions which is intended to abstract a variety of concrete higher-order concurrent languages with specific scheduling and interaction mechanisms. We have described a stratified type and effect system and provided a new reducibility candidates interpretation for it which entails that typable programs terminate. We have highlighted some relevant properties of the interpretation (proposition 7) which could be taken as the basis for an abstract definition of reducibility candidate. The latter is needed to interpret second-order (polymorphic) types (see, *e.g.*, [3]). We have also lifted our approach to a timed/synchronous framework and derived a form of recursive definition which is useful to define behaviours spanning infinitely many instants. In another direction, one could refine the type and effect system to include *linear information* (in the sense of linear logic) which is relevant both to define deterministic fragments of the calculus and to control better the complexity of the definable programs.

Acknowledgements Thanks to Gérard Boudol for several discussions on [1].

References

[1] G. Boudol. Typing termination in a higher-order concurrent imperative language. In Proc. CONCUR, Springer LNCS 4703:272-286, 2007.

[2] G. Berry and G. Gonthier. The Esterel synchronous programming language. *Science of computer programming*, 19(2):87–152, 1992.

[3] J. Gallier. On Girard's *Candidats de Reductibilité*. In *Logic and Computer Science*, Odifreddi (ed.), Academic Press, 123-203, 1990.

[4] M. Hennessy, T. Regan. A process algebra of timed systems. *Information and Computation*, 117(2):221-239, 1995.

[5] J. Lucassen and D. Gifford. Polymorphic effect systems. In Proc. ACM-POPL, 1988.

[6] M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2): 109-176, 1997.

A Proofs

A.1 Proof of proposition 2

By induction on the structure of M. By the typing hypothesis, M cannot be a variable. If M is a value we are in case 1. Otherwise, M can have exactly one of the following shapes: $M_1 M_2$, $!M_1$, $M_1 := M_2$. We consider in some detail the case for application.

The typing rules force M_1 and M_2 to be typable in an empty context. Moreover M_1 must have a functional type. Because of this, if M_1 is a value then it must be of the shape $\lambda x.M_1'$. Moreover, we can apply the inductive hypothesis to M_2 and suitably compose with the evaluation context $M_1[\,]$. If M_1 is not a value then we apply the inductive hypothesis to M_1 and suitably compose with the evaluation context $[\,]M_2$. □

A.2 Proof of proposition 3

Weakening First prove by induction on the proof height that if $R, R' \vdash$ and $R \vdash A$, ($R \vdash (A, e)$, $R \vdash A \leq B$) then $R, R' \vdash A$ ($R, R' \vdash (A, e)$, $R, R' \vdash A \leq B$). Next, by induction on the proof height, we show how to transform a proof $R; \Gamma \vdash M : (A, e)$ into a proof of $R, R'; \Gamma, \Gamma' \vdash M : (A, e)$. □

Substitution By induction on the proof height of $R; \Gamma, x : A \vdash M : (B, e)$.

Subject reduction First we notice that if a term M, of type and effect (A, e), is ready to interact with the store then the region on which the interaction takes place belongs to e. More formally, if $R; \vdash M : (A, e)$, $M \equiv E[\Delta]$ and Δ has the shape $!r$ or $r := V$ then $r \in e$. To prove these facts we proceed by induction on the structure of the evaluation context E. Then we prove the assertion by case analysis on the reduction rule applied relying on the substitution property. □

A.3 Proof of proposition 5

Suppose $r : A \xrightarrow{e} B \in R$ and $r \in e$. Then $R; \vdash \lambda x.!rx : (A \xrightarrow{e} B, \emptyset)$. By proposition 3(substitution), $R; \Gamma \vdash M' : (A \xrightarrow{e} B, \emptyset)$ where $M' = [\lambda x.!rx/f]M$. From this we derive: $R; \Gamma \vdash M'' : (A \xrightarrow{e} B, \{r\})$ where $M'' = !(\mathsf{reg}_r \lambda x.M'x)$. This judgement can be weakened to $R; \Gamma, x : A \vdash M'' : (A \xrightarrow{e} B, \{r\})$ which combined with $R; \Gamma, x : A \vdash x : (A, \emptyset)$ leads to $R; \Gamma \vdash \lambda x.M''x : (A \xrightarrow{e} B, \emptyset)$ where $\lambda x.M''x = \mathsf{fix}_r f.M$, as required. □

A.4 Proof of proposition 7

Weakening Suppose $R'' \geq R' \geq R$ and $R' \vdash M \in \underline{R \vdash (A, e)}$. Then $R'; \emptyset \vdash M : (A, e)$ and by proposition 6(weakening) we know that $R''; \emptyset \vdash M : (A, e)$. Moreover, an inspection of the definition of $\underline{R \vdash (A, e)}$ reveals that if we take a $R''' \geq R''$ then the required properties are automatically satisfied because $R''' \geq R'$ and $R' \vdash M \in \underline{R \vdash (A, e)}$.

Extension/Restriction Suppose $R'' \geq R' \geq R$ and $R \vdash (A, e)$. We want to show that:

$$R'' \vdash M \in \underline{R \vdash (A, e)} \text{ iff } R'' \vdash M \in \underline{R' \vdash (A, e)}\ .$$

Note that $\underline{R'}(R'')$ coincides with $\underline{R}(R'')$ on $dom(R)$. On the other hand, the proposition 6(subject reduction) guarantees that the reduction of a term of type and effect (A, e) will not depend and will not affect the part of the store whose domain is $dom(R') \backslash dom(R)$.

We proceed by induction on the structure of the type A.

Suppose $A = 1$. If $R'' \vdash M \in \underline{R \vdash (A, e)}$ then we know that for any $R_1 \geq R''$ we have that $M, \underline{R}(R_1)$ strongly normalizes to $*, \underline{R}(R_1)$. By applying subject reduction, we can conclude that $M, \underline{R'}(R_1)$ will also strongly normalize to $*, \underline{R'}(R_1)$. A similar argument applies if we start with $R'' \vdash M \in \underline{R' \vdash (A, e)}$. Also, this proof schema can be repeated if $A = \mathsf{Reg}_r B$.

Suppose now $A = A_1 \xrightarrow{e_1} A_2$. If $R'' \vdash M \in \underline{R \vdash (A, e)}$ then we know that for any $R_1 \geq R''$, $M, \underline{R}(R_1)$ strongly normalizes to $\lambda x.N, \underline{R}(R_1)$, for some $\lambda x.N$. Moreover for any $R_2 \geq R_1$, we have that $R_2 \vdash V \in \underline{R \vdash (A_1, \emptyset)}$ implies $R_2 \vdash (\lambda x.N)V \in \underline{R \vdash (A_2, e_1)}$. By applying subject reduction, we can conclude that $M, \underline{R'}(R_1)$ will also strongly normalize to $(\lambda x.N), \underline{R'}(R_1)$, for some value $\lambda x.N$. Further, by induction hypothesis on A, if $R_2 \geq R_1$ and $R_2 \vdash V \in \underline{R' \vdash (A_1, \emptyset)}$ then $R_2 \vdash (\lambda x.N)V \in \underline{R' \vdash (A_2, e_1)}$.

Again, a similar argument applies if we start with $R'' \vdash M \in \underline{R' \vdash (A, e)}$.

Subtyping Suppose $R \vdash (A, e) \leq (A', e')$. We proceed by induction on the proof of $R \vdash A \leq A'$.

Suppose we use the axiom $R \vdash A \leq A$ and $R' \vdash M \in \underline{R \vdash (A, e)}$. Then we check that $R' \vdash M \in \underline{R \vdash (A, e')}$ since $R'; \emptyset \vdash M : (A, e')$ using the subtyping rule, and the remaining conditions do not depend on e or e'.

Suppose we have $A = A_1 \xrightarrow{e_1} A_2$, $A' = A'_1 \xrightarrow{e'_1} A'_2$, and we derive $R \vdash A \leq A'$ from $R \vdash A'_1 \leq A_1$, $R \vdash A_2 \leq A'_2$, and $e_1 \subseteq e'_1$. Moreover, suppose $R' \vdash M \in \underline{R \vdash (A, e)}$. Then $R'; \emptyset \vdash M : (A', e')$, by the subtyping rule. Moreover, if $R'' \geq R'$ and $M, \underline{R}(R')$ reduces to $\lambda x.N, \underline{R}(R'')$, we can use the induction hypothesis to show that if $R_1 \geq R''$ and $R_1 \vdash V \in \underline{R \vdash (A'_1, \emptyset)}$ then $R_1 \vdash (\lambda x.N)V \in \underline{R \vdash (A'_2, e'_1)}$.

Strong normalisation This follows immediately from the definition of $\underline{R \vdash (A, e)}$.

Reduction closure Suppose $R' \vdash M \in \underline{R \vdash (A, e)}$ and $R'' \geq R'$. We know that $M, \underline{R}(R'')$ strongly normalizes to programs of the shape $M'', \underline{R}(R'')$ where M'' has suitable properties. Then if $M, \underline{R}(R'')$ reduces to M', S' it must be that $S' = \underline{R}(R'')$ since the store can only grow. Moreover, by proposition 6(subject reduction), we know that $R''; \emptyset \vdash M' : (A, e)$. It remains to check conditions (2) and (3) of the interpretation on $R'' \vdash M'$. Let $R''' \geq R''$. We claim $M, \underline{R}(R''')$ reduces to $M', \underline{R}(R''')$ so that M' inherits from M the conditions (2) and (3). To check the claim, recall that $M, \underline{R}(R'')$ reduces to $M', \underline{R}(R'')$. Then we analyse the type of reduction performed. The interesting case arises when M reads a value V from the store $\underline{R}(R'')$ where, say, $R'' \vdash V \in \underline{R_1 \vdash (B, \emptyset)}$ and $R = R_1, r : B, R_2$. But then we can apply weakening to conclude that $\underline{R''' \vdash V \in R_1 \vdash (B, \emptyset)}$.

Non-emptiness/Expansion closure We prove the two properties at once, by induction on the proof height of $R \vdash (A, e)$.

- Suppose $R \vdash (1, e)$. We take $V = *$. Then for $R' \geq R$ we have $R'; \emptyset \vdash * : (1, e)$. Also, for any $R'' \geq R'$, $*, \underline{R}(R'')$ converges to itself and satisfies the required properties. Therefore $R' \vdash * \in \underline{R \vdash (1, e)}$.

This settles non-emptiness. To check expansion closure, suppose $R' \geq R$, $R'; \emptyset \vdash M : (1, e)$, and $R'' \geq R'$. By the decomposition proposition 2, M is either a value or a term of the shape $E[\Delta]$ where Δ is a redex.

If $M, \underline{R}(R'')$ does not reduce then M must be the value $*$. Indeed, by the typing hypothesis it cannot be a region or an abstraction. Also, it cannot be of the shape $E[!r]$. Indeed, suppose $R = R_1, r : B, R_2$, then by induction hypothesis on $R_1 \vdash (B, \emptyset)$, we know that the store $\underline{R}(R'')$ contains at least a value in the region r .

If $M, \underline{R}(R'')$ does reduce then, by hypothesis, for all M', S' such that $M, \underline{R}(R'') \to M', S'$ we have that $R'' \vdash M'$ belongs to $\underline{R \vdash (A, e)}$ and $S' = \underline{R}(R'')$. This is enough to check the conditions (2) and (3) of the interpretation and conclude that $R' \vdash M$ belongs to $\underline{R \vdash (A, e)}$.

- The other basic case is $R \vdash (\mathsf{Reg}_r B, e)$. Then we take as value $V = r$ and we reason as in the previous case.
- Finally, suppose $R \vdash (A_1 \xrightarrow{e_1} A_2, e)$. By induction hypothesis on $R \vdash (A_2, e_1)$, we know that there is a value V_2 such that for any $R' \geq R$ we have $R' \vdash V_2 \in \underline{R \vdash (A_2, e_1)}$. Then we claim that:

$$R' \vdash \lambda x.V_2 \in \underline{R \vdash (A_1 \xrightarrow{e_1} A_2, e)} \ .$$

First, $R'; \vdash \lambda x.V_2 : (A_1 \xrightarrow{e_1} A_2, e)$ is easily derived from the hypothesis that $R'; \vdash V_2 : (A_2, e_1)$. The second property of the interpretation is trivially fulfilled since $\lambda x.V_2$ cannot reduce. For the third property, suppose $R_1 \geq R'' \geq R'$ and $R_1 \vdash V \in \underline{R \vdash (A_1, \emptyset)}$. We have to check that $R_1 \vdash (\lambda x.V_2)V$ belongs to $\underline{R \vdash (A_2, e_1)}$. We observe that $R_1; \vdash (\lambda x.V_2)V : (A_2, e_1)$, and the term $(\lambda x.V_2)V$ is neutral. Moreover, for $R_2 \geq R_1$, $(\lambda x.V_2)V, \underline{R}(R_2) \to V_2, \underline{R}(R_2)$. Thus we are in the situation to apply the inductive hypothesis of expansion closure on $R \vdash (A_2, e_1)$.

This settles non-emptiness at higher-order. To check expansion closure, suppose $R' \geq R$, $R'; \vdash M : (A_1 \xrightarrow{e_1} A_2, e)$, and M neutral. Then M cannot be a value and for any $R'' \geq R'$ the program $M, \underline{R}(R'')$ must reduce. Indeed, M cannot be stuck on a read because if $r \in dom(R)$ then we know, by inductive hypothesis, that $\underline{R}(R'')(r)$ is not-empty. Then we conclude that $R' \vdash M$ satisfies properties (2) and (3) of the interpretation because all the terms it reduces to satisfy them. □

A.5 Proof of theorem 9

We proceed by induction on the proof of $R; \Gamma \vdash M : (B, e)$. We shall write $[\mathbf{V}/\mathbf{x}]$ for $[V_1/x_1, \ldots, V_n/x_n]$. Suppose $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, $R \vdash \Gamma$ and $R' \geq R$. We let $R' \vdash \mathbf{V} \in \underline{R \vdash \Gamma}$ stand for $R' \vdash V_i \in \underline{R \vdash (A_i, \emptyset)}$ for $i = 1, \ldots, n$, where $\mathbf{V} = V_1, \ldots, V_n$.

- Suppose $\Gamma = x_1 : A_1, \ldots, x_i : A_i, \ldots, x_n : A_n$, $R; \Gamma \vdash x_i : (A_i, \emptyset)$, $R' \geq R$, and $R' \vdash \mathbf{V} \in \underline{R \vdash \Gamma}$. Then $[\mathbf{V}/\mathbf{x}]x_i = V_i$ and, by hypothesis, $R' \vdash V_i \in \underline{R \vdash (A_i, \emptyset)}$.
- Suppose $R; \Gamma \vdash * : (1, \emptyset)$, $R' \geq R$, and $R' \vdash \mathbf{V} \in \underline{R \vdash \Gamma}$. Then $[\mathbf{V}/\mathbf{x}]* = *$ and we know that $R' \vdash * \in \underline{R \vdash (1, \emptyset)}$.

- Suppose $R;\Gamma \vdash r : (\mathsf{Reg}_r B, \emptyset)$, $R' \geq R$, and $R' \vdash \mathbf{V} \in \underline{R \vdash \Gamma}$. Then $[\mathbf{V}/\mathbf{x}]r = r$ and we know that $R' \vdash r \in \underline{R \vdash (\mathsf{Reg}_r B, \emptyset)}$.

- Suppose $R;\Gamma \vdash M : (A', e')$ is derived from $R;\Gamma \vdash M : (A, e)$ and $R \vdash (A, e) \leq (A', e')$. Moreover, suppose $R' \geq R$, and $R' \vdash \mathbf{V} \in \underline{R \vdash \Gamma}$. By induction hypothesis, $R' \vdash [\mathbf{V}/\mathbf{x}]M \in \underline{R \vdash (A, e)}$. By proposition 7(subtyping), we conclude that $R' \vdash [\mathbf{V}/\mathbf{x}]M \in \underline{R \vdash (A', e')}$.

- Suppose $R;\Gamma \vdash \lambda x.M : (A \xrightarrow{e} B, \emptyset)$ is derived from $R;\Gamma, x : A \vdash M : (B, e)$. Moreover, suppose $R' \geq R$, and $R' \vdash \mathbf{V} \in \underline{R \vdash \Gamma}$. We need to check that $R' \vdash \lambda x.[\mathbf{V}/\mathbf{x}]M$ belongs to $\underline{R \vdash (A \xrightarrow{e} B, \emptyset)}$. Namely, assuming $R_1 \geq R'' \geq R'$ and $R_1 \vdash V \in \underline{R \vdash (A, \emptyset)}$, we have to show that $R_1 \vdash [\mathbf{V}/\mathbf{x}](\lambda x.M)V \in \underline{R \vdash (B, e)}$. We observe that $R_1;\vdash [\mathbf{V}/\mathbf{x}](\lambda x.M)V : (B, e)$ and that, by weakening R' to R_1 and induction hypothesis, we know that $R_1 \vdash [\mathbf{V}/\mathbf{x}, V/x]M \in \underline{R \vdash (B, e)}$. Then we conclude by applying proposition 7(expansion closure).

- Suppose $R;\Gamma \vdash MN : (B, e_1 \cup e_2 \cup e_3)$ is derived from $R;\Gamma \vdash M : (A \xrightarrow{e_1} B, e_2)$ and $R;\Gamma \vdash N : (A, e_3)$. Moreover, suppose $R' \geq R$, and $R' \vdash \mathbf{V} \in \underline{R \vdash \Gamma}$. By induction hypothesis, we know that $R' \vdash [\mathbf{V}/\mathbf{x}]M \in \underline{R \vdash (A \xrightarrow{e_1} B, e_2)}$ and $R' \vdash [\mathbf{V}/\mathbf{x}]N \in \underline{R \vdash (A, e_3)}$. We have to show that: $R' \vdash [\mathbf{V}/\mathbf{x}](MN) \in \underline{R \vdash (B, e_1 \cup e_2 \cup e_3)}$. Suppose $\overline{R'' \geq R'}$. Then $[\mathbf{V}/\mathbf{x}]M, \underline{R}(R'')$ normalizes to $\lambda x.M', \underline{R}(R'')$ for some value $\lambda x.M'$ and $[\mathbf{V}/\mathbf{x}]N, \underline{R}(R'')$ normalizes to $V, \underline{R}(R'')$ for some value V. Further, by reduction closure, we know that $R'' \vdash \lambda x.M' \in \underline{(A \xrightarrow{e_1} B, e_2)}$ and $R'' \vdash V \in \underline{(A, e_3)}$. It is easily checked that the latter implies $\overline{R'' \vdash V \in (A, \emptyset)}$. By condition (3) of the interpretation, we derive that $R'' \vdash (\lambda x.M')V \in \underline{R \vdash (B, e_1)}$ which suffices to conclude.

- Suppose $R;\Gamma \vdash M := N : (1, e_1 \cup e_2 \cup \{r\})$ is derived from $R;\Gamma \vdash M : (\mathsf{Reg}_r A, e_1)$ and $R;\Gamma \vdash N : (A, e_2)$. Moreover, suppose $R' \geq R$, and $R' \vdash \mathbf{V} \in \underline{R \vdash \Gamma}$. By induction hypothesis, we know that $R' \vdash [\mathbf{V}/\mathbf{x}]M \in \underline{R \vdash (\mathsf{Reg}_r A, e_1)}$ and $R' \vdash [\mathbf{V}/\mathbf{x}]N \in \underline{R \vdash (A, e_2)}$. Then for any $R'' \geq R'$, $[\mathbf{V}/\mathbf{x}]M, \underline{R}(R'')$ normalizes to $r, \underline{R}(R'')$ and $[\mathbf{V}/\mathbf{x}]N, \underline{R}(R'')$ normalizes to $V, \underline{R}(R'')$ where $R'' \vdash V \in \underline{R \vdash (A, \emptyset)}$. Suppose $R = R_1, r : A, R_2$. By definition, $\underline{R}(R'')(r) = \{V' \mid R'' \vdash V' \in \overline{R_1 \vdash (A, \emptyset)}\}$. By proposition 7(restriction), we know that if $R'' \vdash V \in \underline{R \vdash (A, \emptyset)}$ then $\overline{R'' \vdash V \in \underline{R_1 \vdash (A, \emptyset)}}$. Therefore, $V \in \underline{R}(R'')(r)$, and the assignment normalizes to $*, \underline{R}(R'')(r)$. It follows that $R'' \vdash [\mathbf{V}/\mathbf{x}](M = N)$ belongs to $\underline{R \vdash (1, e_1 \cup e_2 \cup \{r\})}$.

- Suppose $R;\Gamma \vdash !M : (A, e \cup \{r\})$ is derived from $R;\Gamma \vdash M : (\mathsf{Reg}_r A, e)$. Moreover, suppose $R' \geq R$, and $R' \vdash \mathbf{V} \in \underline{R \vdash \Gamma}$. By induction hypothesis, we know that $R' \vdash [\mathbf{V}/\mathbf{x}]M \in \underline{R \vdash (\mathsf{Reg}_r A, e)}$. Then for any $R'' \geq R'$, $[\mathbf{V}/\mathbf{x}]M, \underline{R}(R'')$ normalizes to $r, \underline{R}(R'')$. Thus $![\mathbf{V}/\mathbf{x}]M, \underline{R}(R'')$ will reduce to $V, \underline{R}(R'')$ where $V \in \underline{R}(R'')(r)$ which is not empty by proposition 7(not-emptiness). Suppose $R = R_1, r : A, R_2$. We know that $R'' \vdash V \in \underline{R_1 \vdash (A, \emptyset)}$ and by proposition 7(extension) we conclude that $R'' \vdash V \in \underline{R \vdash (A, \emptyset)}$. □

A.6 Proof of corollary 10

(1) By definition, if $R \vdash M \in \underline{R \vdash (A, e)}$ then $R;\vdash M : (A, e)$. On the other hand, as a special case of theorem 9, if $R;\vdash M : (A, e)$ is derivable then $R \vdash M \in \underline{R \vdash (A, e)}$.

(2) Suppose we have $R;\vdash M_1,\ldots,M_n : e$. Then we have $R; M_i : (A_i, e_i)$ for $i = 1,\ldots,n$. By theorem 9, the evaluation of $M_i, \underline{R}(R)$ is guaranteed to terminate in $V_i, \underline{R}(R)$, for some value V_i. Now any reduction starting from $M_1,\ldots,M_n$ can be simulated step by step by a reduction of $M_1,\ldots,M_n,\underline{R}(R)$ and therefore it must terminate. □

A.7 Proof of proposition 11

By induction on the structure of M. We consider in some detail the case for the else-next (cf. proof A.1 for other cases).

$M_1 \triangleright M_2$ We apply the inductive hypothesis to M_1, and we have three cases: (1) M_1 is a value, (2) $M_1 = E_1[\Delta_1]$ with E_1 time insensitive, and (3) $M_1 = E_1[E_2[\Delta_1] \triangleright N]$ with E_1 time insensitive. We note that in each case we fall in case 3 where the insensitive evaluation context is $[\ ]$. □

A.8 Proof of proposition 12

weakening/substitution The proofs of weakening and substitution proceed as in the proof A.2.

context substitution We note that a proof of $R;\Gamma \vdash M : (A, e)$ consists of a proof of $R;\Gamma \vdash M : (A', e')$, where $R \vdash (A', e') \leq (A, e)$, followed by a sequence of subtyping rules. To prove context substitution, we proceed by induction on the proof $R;\Gamma, x : A \vdash E[x] : (B, e)$ and by case analysis on the shape of E.

subject reduction To prove subject reduction, we start by noting that if $R; x : A \vdash E[x] : (B, e)$ then $R; x : A \vdash \mathit{red}(E)[x] : (B, e)$. In other terms, the elimination of the pending else-next branches from the evaluation context preserves the typing. Then we proceed by analysing the redexes as in proof A.2.

tick reduction The interesting case is when $M = E[E'[\Delta] \triangleright N]$, E is time insensitive, Δ has the shape V or $!r$, and $M \xrightarrow{\mathsf{tick}} E[N]$. Suppose $R;\vdash M : (A, e)$. Then the typing of the else-next guarantees that $R;\vdash E'[\Delta] : (B, e_1)$ and $R;\vdash N : (B, e_2)$ for some B, e_1, e_2 where e_1 and e_2 may be incomparable. Then we can conclude $R;\vdash E[N] : (A, e')$ where the effect e' is contained in $\mathit{dom}(R)$ but may be incomparable with e. □

A.9 Proof of proposition 13

(1) A straightforward induction on the typing.

(2) Immediate extension of step (1).

(3) First we check that the translation commutes with the substitution. Also, we extend the translation to evaluation contexts, assuming $\langle[\ ]\rangle = [\ ]$, and check that $\langle E\rangle$ is again an evaluation context. Then we proceed by case analysis on the reduction rule.

(4) Every reduction in P corresponds to a reduction in $\langle P\rangle$. □

Syntactic categories

$x, y, \ldots$	(variables)
$r, s, \ldots$	(regions)
$e, e', \ldots$	(finite sets of regions)
$A ::= 1 \mid \mathsf{Reg}_r A \mid (A \xrightarrow{e} A)$	(types)
$R ::= r_1 : A_1, \ldots, r_n : A_n$	(region context)
$\Gamma ::= x_1 : A_1, \ldots, x_n : A_n$	(context)
$M ::= x \mid r \mid * \mid \lambda x.M \mid MM \mid !M \mid M := M \mid M \triangleright M$	(terms)
$V ::= r \mid * \mid \lambda x.M$	(values)
$v, v', \ldots$	(sets of value)
$S ::= (r \leftarrow v) \mid S, S$	(stores)
$X ::= M \mid S$	(stores or terms)
$P ::= X \mid X, P$	(programs)
$E ::= [\,] \mid EM \mid VE \mid !E \mid E := M \mid r := E \mid E \triangleright M$	(evaluation contexts)

Evaluation rules within an instant

$$\frac{}{E[(\lambda x.M)V] \to \mathit{red}(E)[[V/x]M]} \qquad \frac{}{E[!r], (r \leftarrow V) \to \mathit{red}(E)[V], (r \leftarrow V)}$$

$$\frac{}{E[r := V] \to \mathit{red}(E)[*], (r \leftarrow V)} \qquad \frac{P \to P'}{P, P'' \to P', P''}$$

Rules for the passage of time

$$\frac{}{V \xrightarrow{\mathsf{tick}} V} \qquad \frac{M = E[!r] \quad E \text{ time insensitive}}{M \xrightarrow{\mathsf{tick}} M}$$

$$\frac{M = E[E'[\Delta] \triangleright N] \quad E \text{ time insensitive} \quad \Delta ::= V \mid !r}{M \xrightarrow{\mathsf{tick}} E[N]}$$

$$\frac{}{S \xrightarrow{\mathsf{tick}} S} \qquad \frac{P_1, P_2 \not\to \quad P_i \xrightarrow{\mathsf{tick}} P_i' \quad i = 1, 2}{P_1, P_2 \xrightarrow{\mathsf{tick}} P_1', P_2'}$$

Table 1: Syntactic categories and operational semantics

A.10 Proof of proposition 14

The proof is a variation of the one for proposition 5. Suppose $r : A \xrightarrow{e} B \in R$ (hence $r \notin e$). Then $R; \vdash \lambda x.!rx : (A \xrightarrow{e\cup\{r\}} B, \emptyset)$. By proposition 12(substitution), $R; \Gamma \vdash M' : (A \xrightarrow{e} B, \emptyset)$ where $M' = [\lambda x.!rx/f]M$. From this we derive: $R; \Gamma \vdash M'' : (A \xrightarrow{e} B, \{r\})$ where $M'' = !(\mathsf{reg}_r \lambda x.M'x)$. This judgement can be weakened to $R; \Gamma, x : A \vdash M'' : (A \xrightarrow{e} B, \{r\})$ which combined with $R; \Gamma, x : A \vdash x : (A, \emptyset)$ leads to $R; \Gamma \vdash \lambda x.M''x : (A \xrightarrow{e\cup\{r\}} B, \emptyset)$ where $\lambda x.M''x = \mathsf{fix}_r f.M$, as required. □

B Summary of syntax, operational semantics, and typing rules

Table 1 summarizes the main syntactic categories, the evaluation rules for the computation within an instant (relation $\to$), and the rules for the passage of time (relation $\xrightarrow{\mathsf{tick}}$). Table 2 summarizes the typing rules for the unstratified and stratified systems which differ just in the judgements for region contexts and types.

Unstratified region contexts and types

$$\frac{}{R \downarrow 1} \qquad \frac{R \downarrow A \quad R \downarrow B \quad e \subseteq dom(R)}{R \downarrow A \xrightarrow{e} B} \qquad \frac{r : A \in R}{R \downarrow \mathsf{Reg}_r A}$$

$$\frac{\forall r \in dom(R) \;\; R \downarrow R(r)}{R \vdash} \qquad \frac{R \vdash \quad R \downarrow A}{R \vdash A} \qquad \frac{R \vdash A \quad e \subseteq dom(R)}{R \vdash (A, e)}$$

Stratified region contexts and types

$$\frac{}{\emptyset \vdash} \qquad \frac{R \vdash A \quad r \notin dom(R)}{R, r : A \vdash} \qquad \frac{R \vdash}{R \vdash 1}$$

$$\frac{R \vdash \quad r : A \in R}{R \vdash \mathsf{Reg}_r A} \qquad \frac{R \vdash A \quad R \vdash B \quad e \subseteq dom(R)}{R \vdash A \xrightarrow{e} B} \qquad \frac{R \vdash A \quad e \subseteq dom(R)}{R \vdash (A, e)}$$

Subtyping rules

$$\frac{R \vdash A}{R \vdash A \leq A} \qquad \frac{R \vdash A' \leq A \quad R \vdash B \leq B' \quad e \subseteq e' \subseteq dom(R)}{R \vdash (A \xrightarrow{e} B) \leq (A' \xrightarrow{e'} B')} \qquad \frac{R \vdash A \leq A' \quad e \subseteq e' \subseteq dom(R)}{R \vdash (A, e) \leq (A', e')}$$

Terms, stores, and programs

$$\frac{R \vdash \Gamma \quad x : A \in \Gamma}{R; \Gamma \vdash x : (A, \emptyset)} \qquad \frac{R \vdash \Gamma \quad r : A \in R}{R; \Gamma \vdash r : (\mathsf{Reg}_r A, \emptyset)} \qquad \frac{R \vdash \Gamma}{R; \Gamma \vdash * : (1, \emptyset)}$$

$$\frac{R; \Gamma, x : A \vdash M : (B, e)}{R; \Gamma \vdash \lambda x.M : (A \xrightarrow{e} B, \emptyset)} \qquad \frac{R; \Gamma \vdash M : (A \xrightarrow{e_2} B, e_1) \quad R; \Gamma \vdash N : (A, e_3)}{R; \Gamma \vdash MN : (B, e_1 \cup e_2 \cup e_3)}$$

$$\frac{R; \Gamma \vdash M : (\mathsf{Reg}_r A, e)}{R; \Gamma \vdash !M : (A, e \cup \{r\})} \qquad \frac{R; \Gamma \vdash M : (\mathsf{Reg}_r A, e_1) \quad R; \Gamma \vdash N : (A, e_2)}{R; \Gamma \vdash M := N : (1, e_1 \cup e_2 \cup \{r\})}$$

$$\frac{R; \Gamma \vdash M : (A, e) \quad R; \Gamma \vdash N : (A, e')}{R; \Gamma \vdash M \triangleright N : (A, e)} \qquad \frac{R; \Gamma \vdash M : (A, e) \quad R \vdash (A, e) \leq (A', e')}{R; \Gamma \vdash M : (A', e')}$$

$$\frac{r : A \in R \quad \forall V \in v \;\; R; \Gamma \vdash V : (A, \emptyset)}{R; \Gamma \vdash (r \leftarrow v) : (1, \emptyset)} \qquad \frac{R; \Gamma \vdash X_i : (A_i, e_i) \quad i = 1, \ldots, n \geq 1}{R; \Gamma \vdash X_1, \ldots, X_n : e_1 \cup \cdots \cup e_n}$$

Table 2: Typing systems