

# Strong Normalisation in $\lambda$ -calculi with References

Romain Demangeon, Daniel Hirschhoff, Davide Sangiorgi

**Matthew Hennessy**

## Contributions of this work

- ▶ adding references to the simply-typed  $\lambda$ -calculus is known to bring divergences

“Landin’s trick”:

```
let u = ref ( $\lambda x. x$ ) in  allocate a new cell pointing to the identity...
u :=  $\lambda y. (deref(u) y)$ ;  ... make a knot ...
deref(u) 32              ... loop along the knot
                        deref(u) 32  $\Leftrightarrow \lambda y. (deref(u) y)$  32
```

## Contributions of this work

- ▶ adding references to the simply-typed  $\lambda$ -calculus is known to bring divergences

“Landin’s trick”:

```
let u = ref ( $\lambda x. x$ ) in   allocate a new cell pointing to the identity...
u :=  $\lambda y. (\text{deref}(u) y)$ ; ... make a knot ...
deref(u) 32              ... loop along the knot
                         $\text{deref}(u) 32 \rightleftharpoons \lambda y. (\text{deref}(u) y) 32$ 
```

- ▶ in this work, we exploit (a minor adaptation of) a *type and effect system* first introduced by G. Boudol to rule out diverging terms
- ▶ the main focus is on the *proof strategy* we use
  - ▶ different from the proof in [Boudol07]
  - ▶ comes from a study of termination in the  $\pi$ -calculus

[DHS Concur10]

# Outline of the talk

1. Boudol's system of types and effects for  $\lambda_{\text{ref}}$ , a call-by-value  $\lambda$ -calculus with references
  - ▶ rather standard; think, e.g., of ML plus effects
  - ▶ key point: stratification of the store
2. soundness proof
  - ▶ overall strategy
  - ▶ a pruning function mapping  $\lambda_{\text{ref}}$  terms to  $\lambda$ -terms

*First part*

*The setting*

## $\lambda_{\text{ref}}$ , a $\lambda$ -calculus with references

$x, y, z, \dots$   $\lambda$ -calculus variables

$u_{(n,T)}, \dots$  references

$M ::= \lambda x:T. M \mid M M \mid x \mid \star$  functions ( $\star:\text{unit}$ )  
 $\mid \text{ref}_n M \mid \text{deref}_n(M) \mid M :=_n M \mid u_{(n,T)}$  store manipulations

- ▶ an *annotated* language
  - ▶ references are stored in *regions*
    - ▶ a region is a natural number (*stratification of the store*)
    - ▶  $u_{(n,T)}$ : reference to a value of type  $T$ , stored in region  $n$
  - ▶ operations involving the store are annotated with a region
    - ▶ note that values of different types may be stored in different references hosted by the same region
  - ▶ we ignore for now questions related to inference of types and regions
- ▶ example: the knot from Landin's trick

$$u_{(n,T)} :=_n \lambda y : \text{unit}. (\text{deref}_n(u_{(n,T)}) y)$$

# $\lambda_{\text{ref}}$ : types and effects (1)

## Types

$T ::= \text{unit} \mid T \text{ ref}_n \mid T \rightarrow^n T$

## Type judgements

$\Gamma \vdash M : (T, k)$

- ▶  $M$  is well-typed of type  $T$  and effect  $k \in \mathbb{N}$
- ▶ intuition:  $M$  will not access regions  $> k$  when evaluated

## $\lambda_{\text{ref}}$ : types and effects (2)

### Types and effects for operations involving the store

$$\frac{}{\Gamma \vdash u_{(n,T)} : (T \text{ ref}_n, 0)} \quad \frac{\Gamma \vdash M : (T, p)}{\Gamma \vdash \text{ref}_n M : (T \text{ ref}_n, \max(p, n))}$$

$$\frac{\Gamma \vdash M : (T \text{ ref}_n, p)}{\Gamma \vdash \text{deref}_n(M) : (T, \max(p, n))}$$

$$\frac{\Gamma \vdash M : (T \text{ ref}_n, p) \quad \Gamma \vdash N : (T, k)}{\Gamma \vdash M :=_n N : (\text{unit}, \max(p, n, k))}$$

## $\lambda_{\text{ref}}$ : types and effects (3)

### Types and effects for lambda-terms

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : (T, 0)} \quad \frac{}{\Gamma \vdash \star : (\text{unit}, 0)} \quad \frac{\Gamma, x : T_1 \vdash M : (T_2, n)}{\Gamma \vdash \lambda x : T_1. M : (T_1 \rightarrow^n T_2, 0)}$$
$$\frac{\Gamma \vdash M : (T_1 \rightarrow^n T_2, p) \quad \Gamma \vdash N : (T_1, k)}{\Gamma \vdash M N : (T_2, \max(p, n, k))}$$

- ▶ values have effect 0 (three topmost rules)
- ▶  $T_1 \rightarrow^n T_2$  :  $n$  is the effect of the body (of type  $T_2$ ) of the function
- ▶ typing application: the effect is the maximum between
  - ▶ the effect of the function  $p$
  - ▶ the effect of the argument  $k$
  - ▶ the effect of the body of the function  $n$

# Reduction

- ▶ the execution of programs involves values and stores

- ▶  $V ::= \lambda x:T.M \mid x \mid \star \mid u_{(n,T)}$  values

- ▶  $\delta$  stores

$\delta \langle u_{(n,T)} \rightsquigarrow V \rangle$  store update or extension

- ▶ two kinds of reduction steps, decorated with their region

- ▶ functional steps  $\beta$ -reductions

$$\frac{}{(\lambda x:T.M \ V, \delta) \xrightarrow{n} (M\{V/x\}, \delta)} \quad \text{with } \lambda x:T.M : T \rightarrow^n T'$$

- ▶ imperative steps accesses to the store

$$\frac{u_{(n,T)} \notin \text{supp}(\delta) \quad \Gamma \vdash V : (T, k)}{(\text{ref}_n \ V, \delta) \xrightarrow{n} (u_{(n,T)}, \delta \langle u_{(n,T)} \rightsquigarrow V \rangle)}$$

$$\frac{\delta(u_{(n,T)}) = V}{(\text{deref}_n(u_{(n,T)}), \delta) \xrightarrow{n} (V, \delta)}$$

$$\frac{\Gamma \vdash V : (T, k)}{(u_{(n,T)} :=_n V, \delta) \xrightarrow{n} (\star, \delta \langle u_{(n,T)} \rightsquigarrow V \rangle)}$$

# Insuring termination

- ▶ **Definition:**  $M$  is **terminating** if there is no infinite sequence of (functional or imperative) reductions starting from  $(M, \emptyset)$

$$(M, \emptyset) \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \dots$$

- ▶ key ingredient for termination: **well-formedness of types**
  - ▶ a type  $T$  is *well-formed* if for all its subtypes of the form

$$T' \text{ ref}_n ,$$

$T'$  mentions only regions  $< n$

- ▶ “almost accurate” definition: see the true story in the paper
- ▶ intuitively, a term acting at region  $n$  must be stored at region at least  $n + 1$

## Consequences of well-formedness of types

- ▶ consider for instance the following reduction steps

$$\left( \text{deref}_n \left( \underbrace{V}_{(T_1 \rightarrow^p T_2)} \right) \right) \underbrace{V'}_{T_1}$$

necessarily,  
 $V = u_{(n, (T_1 \rightarrow^p T_2))}$

$$\begin{array}{c} \downarrow n \\ \underbrace{(\lambda x. M)}_{T_1 \rightarrow^p T_2} V' \end{array}$$

we have  $M : (T_2, p)$   
and  $p < n$  (well-formedness)

$$\begin{array}{c} \downarrow p \\ M[V'/x] \end{array}$$

property of “subject substitution”:  
 $M[V'/x] : (T_2, p)$

- ▶ more generally, by well-formedness of types, imperative reduction steps at region  $n$  can only trigger reduction steps at region  $p < n$

*Second part*

*The soundness proof*

# Soundness of the type system

- ▶ by contradiction: consider an **infinite sequence** of reductions
  - ▶ starting from a well-typed term, and
  - ▶ involving **functional steps** ( $\beta$ -reductions) or **imperative steps**

→ → → → → → → → → → → . . .

annotated with their regions

5 3 7 2 5 6 2 5 3 5 5  
→ → → → → → → → → → → . . .

# Soundness of the type system

- ▶ by contradiction: consider an **infinite sequence** of reductions
  - ▶ starting from a well-typed term, and
  - ▶ involving **functional steps** ( $\beta$ -reductions) or **imperative steps**

→ → → → → → → → → → → ...

annotated with their regions

5 3 7 2 5 6 2 5 3 5 5  
→ → → → → → → → → → → ...

- ▶ we want to forget about the imperative parts, and deduce

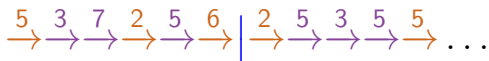
→ → → → → → → → → ... ,

an infinite derivation starting from a simply-typed  $\lambda$ -calculus term **contradiction**

(the **purely functional** calculus is strongly normalising)

## First step: maximal region

from



to



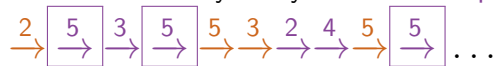
- ▶ there exists a maximal region (5) which appears an infinite number of times (because a given term is typed using a *finite* number of regions)
- ▶ start the sequence after all reductions  $> 5$  have been done

## Second step: infinitely many functional reductions

analysing



there are infinitely many **functional steps** at region 5



indeed:

- ▶ a given term contains a finite number  $C_5$  of occurrences of constructs acting on the store (`deref`, `ref`, `:=`) at region 5
- ▶  $\xrightarrow{5}$  makes  $C_5$  strictly decrease
- ▶ by well-formedness of types,  $\xrightarrow{n}$  and  $\xrightarrow{n}$  leave  $C_5$  unchanged for  $n < 5$
- ▶ therefore, there are infinitely many  $\xrightarrow{5}$ 
  - ▶ otherwise there are infinitely many  $\xrightarrow{5}$  and no  $\xrightarrow{5}$  after some point, which is impossible

## Last step: pruning

from



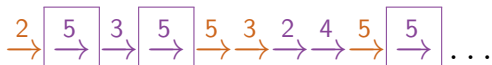
to



- ▶ transforming **impure terms** into **pure ones** (in simply-typed  $\lambda$ )
- ▶ main goal: define a pruning function in such a way that “no divergence is left out”

## Last step: pruning

from



to



- ▶ transforming **impure terms** into **pure ones** (in simply-typed  $\lambda$ )
- ▶ main goal: define a pruning function in such a way that “no divergence is left out”
- ▶ some clauses defining  $\mathcal{P}r(\cdot)$ : (*omitting technical details*)
  - ▶  $\mathcal{P}r(M N) \stackrel{\text{def}}{=} \mathcal{P}r(M) \mathcal{P}r(N)$
  - ▶  $\mathcal{P}r(\text{deref}_n(M))$ : (with  $\vdash M : T \text{ ref}_n$ )
    - ▶ if  $\text{deref}_n(M)$  interacts at region 5,  
(because  $n = 5$ , or  $M$  does something at region 5)  
execute the pruning of  $M$ , and then return a dummy value of type  $T$
    - ▶ otherwise, simply return a dummy value of type  $T$
  - ▶ hence,  $\mathcal{P}r((\text{deref}_n(M_1)) M_2)$  will execute (the pruning of)  $M_1$ , then execute (the pruning of)  $M_2$ , and finally return  $V_T$

## Pruning – properties

**Lemma:** The pruning of a term in  $\lambda_{\text{ref}}$  yields a term in the simply typed  $\lambda$ -calculus.

**Proposition [Simulation]:** pruning maps

$$\blacktriangleright \xrightarrow{5} \text{ into } \rightarrow$$

$$\blacktriangleright \xrightarrow{5} \text{ into } \rightarrow^+ \quad (\text{at least one } \rightarrow)$$

$$\blacktriangleright \xrightarrow{n} \text{ into } = \quad (\text{for } n < 5)$$

$$\blacktriangleright \xrightarrow{n} \text{ into } = \quad (\text{for } n < 5)$$



## Related studies

- ▶ G. Boudol, *Fair Cooperative Multithreading* Proc. of CONCUR 2007
  - ▶ introduction of the type and effect system based on a stratification of the store
  - ▶ proof using the realisability technique (interpretation of types as sets of terms)
- ▶ R. Amadio, *On stratified regions* Proc. of APLAS 2009
  - ▶ a more abstract and general (references, channels, signals) presentation of Boudol's proposal
  - ▶ proof based on reducibility candidates
  - ▶ further developed in R. Amadio, P. Baillot, A. Madet, *An affine-intuitionistic system of types and effects: confluence and termination*, LOLA 2010
- ▶ P. Tranquilli, *Termination of Threads with Shared Memory via Infinitary Choice* (submitted)
  - ▶ revisits [Boudol07] by studying a monadic transformation from  $\lambda_{\text{ref}}$  to the  $\lambda$ -calculus with (certain kinds of) recursive types
  - ▶ apparently no connection with our transformation  $\mathcal{Pr}(\cdot)$

## Conclusion: a flexible approach

- ▶ a similar approach is at work for  $\pi$  in [DHS Concur10]
  - ▶ channel-based formalism
  - ▶ two kinds of channels:
    - ▶ **functional channels**: specific usage discipline
    - ▶ **imperative channels**: stratified into regions

## Conclusion: a flexible approach

- ▶ a similar approach is at work for  $\pi$  in [DHS Concur10]
  - ▶ channel-based formalism
  - ▶ two kinds of channels:
    - ▶ **functional channels**: specific usage discipline
    - ▶ **imperative channels**: stratified into regions
- ▶ in our proof, termination of the core functional calculus (simply-typed  $\lambda$ -calculus) is treated like a black box
  - ▶ variations are possible (e.g. polymorphism)

## Conclusion: a flexible approach

- ▶ a similar approach is at work for  $\pi$  in [DHS Concur10]
  - ▶ channel-based formalism
  - ▶ two kinds of channels:
    - ▶ **functional channels**: specific usage discipline
    - ▶ **imperative channels**: stratified into regions
- ▶ in our proof, termination of the core functional calculus (simply-typed  $\lambda$ -calculus) is treated like a black box
  - ▶ variations are possible (e.g. polymorphism)
- ▶ future work: implicit computational complexity (more than plain termination)

when going from  $\xrightarrow{5} \xrightarrow{3} \xrightarrow{7} \xrightarrow{2} \xrightarrow{5} \xrightarrow{6} \xrightarrow{2} \xrightarrow{5} \xrightarrow{3} \xrightarrow{5} \xrightarrow{5} \dots$   
to  $= \boxed{\rightarrow^+} = \boxed{\rightarrow^+} \rightarrow \dots \rightarrow \boxed{\rightarrow^+} \dots$

guarantee complexity bounds on the maximum length of a derivation, through a careful analysis of what happens when establishing the simulation property