Typed Lambda Calculus (2/3)

Eduardo Bonelli

"There may, indeed, be other applications of the system other than its use as a logic"

Alonzo Church, 1932

February 14, 2019

Two More Extensions

- References

  Imperative Programming = Functional Programming + Effects

- Recursion
  - None of the extensions seen up until now allow the definition of recursive functions
  - All functions currently definable are total

References - Motivation

- In an expression such as $let\ x = \underline{2}\ in\ M$
  - x is a variable declared to have value 2
  - The value of x remains unaltered during the evaluation of M
  - In this sense x is immutable: there is no assignment operation
- In imperative programming every variable is mutable
- We extend $\lambda_V^{\mathbb{B},\rightarrow}$ with mutable variables

Basic Operations

Allocation

ref M generates a fresh reference whose contents is the value of M

Dereference

$!x$ dereferences reference x and returns its contents

Assignment

$x := M$ updates the contents of reference x with the value of M

Examples

- $let\ x = ref\ \underline{2}\ in\ (\lambda\_ : unit.!x)\,(x := succ(!x))$ evaluates to $\underline{3}$
- What does $let\ x = ref\ \underline{2}\ in\ x$ evaluate to?
- $let\ x = \underline{2}\ in\ x$ evaluates to $\underline{2}$
- $let\ x = ref\ \underline{2}\ in\ let\ y = x\ in\ (\lambda\_ : unit.!x)\,(y := succ(!y))$ evaluates to $\underline{3}$
  - x and y are aliases for the same memory cell

Commands = Expressions with Effects

- Consider $\mathit{let}\ x = \mathit{ref}\ \underline{2}\ \mathit{in}\ x := \mathit{succ}(!x)$. What is its value?
- Assignment is evaluated to cause an effect, not return a value
  - Hence it makes little sense to consider the value of assignment
  - It does make sense to ask about its effect

Command

Expression that is evaluated to cause an effect; we set *unit* to be its value

- Pure Functional Language: one in which all expressions are pure in the sense of causing no effects

Type Expressions for $\lambda_V^{\mathbb{B},\mathbb{U},\to,\mathit{Ref}}$

The type expressions are extended as follows:

$$\sigma \quad ::= \quad \ldots \mid \mathit{Ref}\ \sigma$$

Informally:

- $\mathit{Ref}\ \sigma$ is the type of references to values of type σ
- Eg. $\mathit{Ref}\ (\mathbb{B} \to \mathbb{N})$ is type of references to functions from $\mathbb{B}$ to $\mathbb{N}$

Terms

$$
\begin{array}{rcl}
M & ::= & x \\
 & | & \lambda x : \sigma . M \\
 & | & M\ N \\
 & | & unit \\
 & | & ref\ \ M \\
 & | & !M \\
 & | & M := N
\end{array}
$$

The type system must exclude "illegal" terms such as:

- $!2$
- $2 := 3$

Typing Rules

- Presented in two stages
- First presentation:
  - Preliminary
  - based on syntax of terms introduced up until now
- Second presentation:
  - Definitive
  - when presenting the operational semantics the need will arise to extend the syntax
  - based on this extended syntax

Typing Rules - Preliminary

$$\frac{\Gamma \rhd M_1 : \sigma}{\Gamma \rhd \mathit{ref}\ \ M_1 : \mathit{Ref}\ \ \sigma}\ (\textsc{T-Ref})$$

$$\frac{\Gamma \rhd M_1 : \mathit{Ref}\ \ \sigma}{\Gamma \rhd\ !M_1 : \sigma}\ (\textsc{T-DeRef})$$

$$\frac{\Gamma \rhd M_1 : \mathit{Ref}\ \ \sigma_1 \quad \Gamma \rhd M_2 : \sigma_1}{\Gamma \rhd M_1 := M_2 : \mathbb{U}}\ (\textsc{T-Assign})$$

Examples

- $let\ x = ref\ \underline{2}\ in\ (\lambda\_ : unit.!x)\,(x := succ(!x))$
- $let\ x = ref\ \underline{2}\ in\ x$
- $let\ x = \underline{2}\ in\ x$
- $let\ x = ref\ \underline{2}\ in\ let\ y = x\ in\ (\lambda\_ : unit.!x)\,(y := succ(!y))$

Note: the item in the first bullet can also be written as follows:

$$let\ x = ref\ \underline{2}\ in\ (x := succ(!x)); !x$$

Operational Semantics

- What are the values of type $Ref\ \sigma$?
- How may we model evaluation of the term $ref\ M$?

The answers depend on another question:

What is a reference?

A. It is an abstraction of a portion of memory

Memory or Store

- We use symbolic addresses or "locations" $l, l_i \in \mathcal{L}$ to model references

  Memory (or store): partial function from locations to values

- We use letters μ, μ' for stores
- Notation:=
  - $\mu[l \mapsto V]$ is the store obtained from μ by overriding $\mu(l)$ with V
  - $\mu \oplus (l \mapsto V)$ is an extended store resulting from extending μ with a new association $l \mapsto V$ (we assume $l \notin Dom(\mu)$)

Evaluation judgements:

$$M \mid \mu \rightarrow M' \mid \mu'$$

Values

Intuition:

$$\frac{l \notin Dom(\mu)}{ref\ \ V \mid \mu \rightarrow l \mid \mu \oplus (l \mapsto V)}\ (\text{E-RefV})$$

Possible values now include locations

$$V \quad ::= \quad unit \mid \lambda x : \sigma.M \mid l$$

Given that values are a subset of the terms,

- we must add locations to terms
- note that these are not meant for programmers to manipulate explicitly

Terms

$$
\begin{array}{rcl}
M & ::= & x \\
& | & \lambda x : \sigma . M \\
& | & M\, N \\
& | & \mathit{unit} \\
& | & \mathit{ref}\ \ M \\
& | & !M \\
& | & M := N \\
& | & l
\end{array}
$$

Typing Judgement

$$\Gamma \triangleright l : \textbf{?}$$

- Depends on the values stored in l
- Similar situation to that of free variables
- We need a "typing context" for locations:
  - Σ partial function from locations to types

New Typing Judgement

$$\Gamma | \Sigma \triangleright M : \sigma$$

Typing Rules - Definitive

$$\frac{\Gamma|\Sigma \rhd M_1 : \sigma}{\Gamma|\Sigma \rhd \mathit{ref}\ \ M_1 : \mathit{Ref}\ \ \sigma}\ (\textsc{T-Ref})$$

$$\frac{\Gamma|\Sigma \rhd M_1 : \mathit{Ref}\ \ \sigma}{\Gamma|\Sigma \rhd\ !M_1 : \sigma}\ (\textsc{T-DeRef})$$

$$\frac{\Gamma|\Sigma \rhd M_1 : \mathit{Ref}\ \ \sigma_1 \quad \Gamma|\Sigma \rhd M_2 : \sigma_1}{\Gamma|\Sigma \rhd M_1 := M_2 : \mathbb{U}}\ (\textsc{T-Assign})$$

$$\frac{\Sigma(l) = \sigma}{\Gamma|\Sigma \rhd l : \mathit{Ref}\ \ \sigma}\ (\textsc{T-Loc})$$

Evaluation Judgement - Small-Step

- We now return to the operational semantics
- We introduce axioms and inference rules that define the meaning of evaluation judgements

$$M \mid \mu \to M' \mid \mu'$$

- Recall the set of values:

$$V \quad ::= \quad unit \mid \lambda x : \sigma.M \mid l$$

Evaluation Judgement (1/4)

$$\frac{M_1 \mid \mu \to M_1' \mid \mu'}{M_1\ M_2 \mid \mu \to M_1'\ M_2 \mid \mu'}\ (\textsc{E-App1})$$

$$\frac{M_2 \mid \mu \to M_2' \mid \mu'}{V_1\ M_2 \mid \mu \to V_1\ M_2' \mid \mu'}\ (\textsc{E-App2})$$

$$\frac{}{(\lambda x : \sigma.M)\ V \mid \mu \to M\{x := V\} \mid \mu}\ (\textsc{E-AppAbs})$$

Note: These rules do not modify the store

Evaluation Judgement (2/4)

$$\frac{M_1 \mid \mu \to M_1' \mid \mu'}{!M_1 \mid \mu \to !M_1' \mid \mu'} \text{(E-DEREF)}$$

$$\frac{\mu(l) = V}{!l \mid \mu \to V \mid \mu} \text{(E-DEREFLOC)}$$

Evaluation Judgement (3/4)

$$\frac{M_1 \mid \mu \to M_1' \mid \mu'}{M_1 := M_2 \mid \mu \to M_1' := M_2 \mid \mu'} \text{ (E-ASSIGN1)}$$

$$\frac{M_2 \mid \mu \to M_2' \mid \mu'}{V := M_2 \mid \mu \to V := M_2' \mid \mu'} \text{ (E-ASSIGN2)}$$

$$\frac{}{l := V \mid \mu \to unit \mid \mu[l \mapsto V]} \text{ (E-ASSIGN)}$$

Evaluation Judgement (4/4)

$$\frac{M_1 \mid \mu \to M_1' \mid \mu'}{\mathit{ref}\ \ M_1 \mid \mu \to \mathit{ref}\ \ M_1' \mid \mu'}\ (\text{E-Ref})$$

$$\frac{l \notin \mathit{Dom}(\mu)}{\mathit{ref}\ \ V \mid \mu \to l \mid \mu \oplus (l \mapsto V)}\ (\text{E-RefV})$$

Example

$$
\begin{array}{ll}
 & let\ x = ref\ \underline{2}\ in\ (\lambda\_ : \mathbb{U}.!x)\,(x := succ(!x)) \\
\rightarrow & let\ x = l_1\ in\ (\lambda\_ : \mathbb{U}.!x)\,(x := succ(!x)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\rightarrow & (\lambda\_ : \mathbb{U}.!l_1)\,(l_1 := succ(!l_1)) \\
\rightarrow & (\lambda\_ : \mathbb{U}.!l_1)\,(l_1 := succ(\underline{2})) \\
\rightarrow & (\lambda\_ : \mathbb{U}.!l_1)\ unit \mid \mu[l_1 := \underline{3}] \\
\rightarrow & !l_1 \\
\rightarrow & \underline{3}
\end{array}
$$

Example

Sea

$$
\begin{array}{rcl}
M & = & \lambda r : Ref(\mathbb{U} \to \mathbb{U}). \\
& & \quad let\ f = !r \\
& & \quad\ in\ (r := \lambda x : \mathbb{U}.f\ x); (!r)\ unit
\end{array}
$$

$$
\begin{array}{rl}
& M\,(ref\ (\lambda x : \mathbb{U}.x)) \\
\to & M\ l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : \mathbb{U}.x) \\
\to & let\ f = !l_1\ in\ (l_1 := \lambda x : \mathbb{U}.f\ x); (!l_1)\ unit \\
\to & let\ f = \lambda x : \mathbb{U}.x\ in\ (l_1 := \lambda x : \mathbb{U}.f\ x); (!l_1)\ unit \\
\to & (l_1 := \lambda x : \mathbb{U}.(\lambda x : \mathbb{U}.x)\ x); (!l_1)\ unit \\
\to & unit; (!l_1)\ unit \\
\to & (!l_1)\ unit \mid \mu \oplus (l_1 \mapsto \lambda x : \mathbb{U}.(\lambda x : \mathbb{U}.x)\ x) \\
\to & (\lambda x : \mathbb{U}.(\lambda x : \mathbb{U}.x)\ x)\ unit \\
\to & (\lambda x : \mathbb{U}.x)\ unit \\
\to & unit
\end{array}
$$

Example

Sea

$$\begin{aligned} M &= \lambda r : Ref(\mathbb{U} \to \mathbb{U}). \\ & \quad let\ f = !r \\ & \quad\ in\ (r := \lambda x : \mathbb{U}.f\ x); (!r)\ unit \end{aligned}$$

We replace f with $!r$ obtaining

$$\begin{aligned} M &= \lambda r : Ref\ (\mathbb{U} \to \mathbb{U}). \\ & \quad (r := \lambda x : \mathbb{U}.(!r)\ x); (!r)\ unit \end{aligned}$$

Let us evaluate the new M and see what happens...

Example

$$
\begin{array}{rcl}
M & = & \lambda r : \mathit{Ref}\ (\mathbb{U} \to \mathbb{U}). \\
 & & \quad (r := \lambda x : \mathbb{U}.(!r)\, x); (!r)\, \mathit{unit}
\end{array}
$$

$$
\begin{array}{rl}
 & M\, (\mathit{ref}\ (\lambda x : \mathbb{U}.x)) \\
\to & M\, l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : \mathbb{U}.x) \\
\to & (l_1 := \lambda x : \mathbb{U}.(!l_1)\, x); (!l_1)\, \mathit{unit} \\
\to & \boxed{(!l_1)\, \mathit{unit}} \mid \mu \oplus (l_1 \mapsto \lambda x : \mathbb{U}.(!l_1)\, x) \\
\to & (\lambda x : \mathbb{U}.(!l_1)\, x)\, \mathit{unit} \\
\to & \boxed{(!l_1)\, \mathit{unit}} \\
\to & \ldots
\end{array}
$$

Note: Not every typable term in $\lambda_V^{\mathbb{B},\mathbb{U},\to,\mathit{Ref}}$ is terminating

Safety - Last Class

Safety = Progress + Preservation

Progress

If M is closed and typable, then

1. M is a value; or
2. there exists M' s.t. $M \to M'$

Evaluation cannot get stuck on closed, typable terms that are not values

Preservation

If $\Gamma \triangleright M : \sigma$ and $M \to N$, then $\Gamma \triangleright N : \sigma$

Evaluation preserves types

Must revise!

Preservation - Naive Formulation

The naive formulation is incorrect:

$$\Gamma|\Sigma \triangleright M : \sigma \text{ and } M \,|\, \mu \rightarrow M' \,|\, \mu' \text{ imply } \Gamma|\Sigma \triangleright M' : \sigma$$

- The problem: evaluation may not respect the types for the locations assumed by the type system (i.e. Σ)
- Example

Preservation - Naive Formulation

$$\Gamma|\Sigma \rhd M : \sigma \text{ and } M \mid \mu \to M' \mid \mu' \text{ implies } \Gamma|\Sigma \rhd M' : \sigma$$

Suppose

- $M = !l$
- $\Gamma = \emptyset$
- $\Sigma(l) = \mathbb{N}$
- $\mu(l) = \mathit{true}$

Note that

- $\Gamma|\Sigma \rhd M : \mathbb{N}$; and
- $M \mid \mu \to \mathit{true} \mid \mu$;
- but $\Gamma|\Sigma \rhd \mathit{true} : \mathbb{N}$ fails

Preservation - Naive Formulation

$$\Gamma|\Sigma \triangleright M : \sigma \text{ and } M \mid \mu \to M' \mid \mu' \text{ implies } \Gamma|\Sigma \triangleright M' : \sigma$$

Suppose

- $M = !l$
- $\Gamma = \emptyset$
- $\Sigma(l) = \boxed{\mathbb{N}}$
- $\mu(l) = \boxed{\mathit{true}}$

Note that

- $\Gamma|\Sigma \triangleright M : \mathbb{N}$; and
- $M \mid \mu \to \mathit{true} \mid \mu$
- but $\Gamma|\Sigma \triangleright \mathit{true} : \mathbb{N}$ fails

Preservation - Revisited

- We need a notion of compatibility between the store and the typing context for stores
  - We have to type stores
- New typing judgement

$$\Gamma | \Sigma \triangleright \mu$$

- Defined as follows:

$$\Gamma | \Sigma \triangleright \mu \text{ iff}$$

1. $Dom(\Sigma) = Dom(\mu)$; and
2. $\Gamma | \Sigma \triangleright \mu(l) : \Sigma(l)$ for all $l \in Dom(\mu)$

Preservation - Revisited

Si $\Gamma|\Sigma \rhd M : \sigma$ y $M \to N$ y $\Gamma|\Sigma \rhd \mu : \Sigma$, entonces $\Gamma|\boxed{\Sigma} \rhd N : \sigma$

- Almost correct
- Does not cater for the possibility that the domain of Σ might have grown
  - Due to possible allocations

Preservation - Revisited

- $\Gamma | \Sigma \triangleright M : \sigma$;
- $M \to N$; and
- $\Gamma | \Sigma \triangleright \mu : \Sigma$

imply there exist $\Sigma' \supseteq \Sigma$ s.t. $\Gamma | \Sigma' \triangleright N : \sigma$

Progress - Revisited

If M is closed and typed (i.e. $\emptyset|\Sigma \triangleright M : \sigma$ for some Σ, σ) then

1. M is a value; or
2. for any store μ s.t. $\emptyset|\Sigma \triangleright \mu$, there exists M' and μ' s.t. $M \,|\, \mu \rightarrow M' \,|\mu'$

References

- Motivation
- Types and Typing
- Operational Semantics
- Safety
- Termination for $\lambda_V^{\mathbb{B},\mathbb{U},\rightarrow,\mathit{Ref}}$

Recursion

- Introduction
- Syntax, Typing, Semantics

References

- G. Boudol. Typing termination in a higher-order concurrent imperative language. In Proc. CONCUR, Springer LNCS 4703:272-286, 2007.
- Roberto M. Amadio: On Stratified Regions. APLAS 2009: 210-225

Recursion

Recursive equation

$$f = \ldots f \ldots f \ldots$$

Two explanations

- Denotational
  - Limit of a sequence of approximations
- Operational
  - The "unfolder" and fixed-points

Note: On the board

Terms and Types

$$M \quad ::= \quad \ldots \mid \mathit{fix}\ M$$

- No new types
- New typing rule

$$\frac{\Gamma \rhd M : \sigma_1 \to \sigma_1}{\Gamma \rhd \mathit{fix}\ M : \sigma_1}\ (\textsc{T-Fix})$$

Semántica operacional small-step

- No new values
- New evaluation rules

$$\frac{M_1 \to M_1'}{\mathit{fix}\ M_1 \to \mathit{fix}\ M_1'}\ (\text{E-Fix})$$

$$\frac{}{\mathit{fix}\ (\lambda x : \sigma . M) \to M\{x := \mathit{fix}\ (\lambda x : \sigma . M)\}}\ (\text{E-FixBeta})$$

Examples

Let M be the term

$$\begin{array}{l}\lambda f : \mathbb{N} \to \mathbb{N}. \\ \ \lambda x : \mathbb{N}. \\ \ \ \mathit{if}\ x = 0\ \mathit{then}\ \underline{1}\ \mathit{else}\ x * f(\mathit{pred}(x))\end{array}$$

in

$$\mathit{let}\ \mathit{fact} = \mathit{fix}\ M\ \mathit{in}\ \mathit{fact}\,\underline{3}$$

Examples

$\mathit{fix}\,(\lambda x : \mathbb{N}.x + 1)$

Examples

Let M be the term

$$\begin{array}{l}\lambda s : \mathbb{N} \to \mathbb{N} \to \mathbb{N}. \\ \ \lambda x : \mathbb{N}. \\ \ \ \lambda y : \mathbb{N}. \\ \ \ \ \mathit{if}\ \ x = 0\ \ \mathit{then}\ \ y\ \ \mathit{else}\ \ \mathit{succ}(s\, \mathit{pred}(x)\, y)\end{array}$$

in

$$\mathit{let}\ \ \mathit{sum} = \mathit{fix}\, M\ \ \mathit{in}\ \ \mathit{sum}\, \underline{2}\, \underline{3}$$

Letrec

Alternative primitive for defining recursive functions

$$\mathit{letrec}\ f : \sigma = \lambda x : \tau . M\ \mathit{in}\ N$$

For example,

$$\begin{array}{l} \mathit{letrec} \\ \ \mathit{fact} : \mathbb{N} \to \mathbb{N} = \lambda x : \mathbb{N}.\mathit{if}\ x = 0\ \mathit{then}\ \underline{1}\ \mathit{else}\ x * f(\mathit{pred}(x)) \\ \ \ \mathit{in}\ \mathit{fact}\,\underline{3} \end{array}$$

letrec can be encoded using *fix*:

$$\mathit{let}\ f = \mathit{fix}\,(\lambda f : \sigma \to \sigma . \lambda x : \tau . M)\ \mathit{in}\ N$$