Computing and Software Department, McMaster University

Normalization and References

Wen Yu

October, 2006

Today's Agenda

- Normalization
  - Logical Relations
  - Proof Outline
- References
  - Introduction
  - Typing
  - Evaluation
  - Store Typings
  - Safety

Normalization

Introduction

- Evaluation of a well-typed program is guaranteed to halt in a finite number of steps — that is, every well-typed term is *normalizable*
- The simply typed lambda–calculus over a single base type A is considered here.
- *Logical relations* is used for proving normalization.

Problems with Induction on the Size

Example: proof that $t_1\ t_2$ is normalizing.

- Assume both t_1 and t_2 are normalized and have normal forms v_1 and v_2 respectively.
- By the inversion lemma: v_1 has type $T_{11} \to T_{12}$ for some T_{11} and T_{12}.
- By the canonical forms lemma: v_1 has the form $\lambda x : T_{11}.t_{12}$
- Then, we get $[x \mapsto v_2]t_{12}$.
- However, if there are more than on occurrences of x in t_{12}, $[x \mapsto v_2]t_{12}$ is bigger than the original term $t_1\ t_2$.
- We get stuck.

Logical Relations

Prove some property P of all closed terms of type A by induction on types

- all terms of type A *possess* property P
- all terms of type $A \rightarrow A$ *preserve* property P
- all terms of type $(A \rightarrow A) \rightarrow (A \rightarrow A)$ *preserve the property of preserving* property P
- and so on

Definitions

For each type T, define a set R_T of closed terms of type T, written as $R_T(t)$ for $t \in R_T$.

- $R_A(t)$ iff t halts.
- $R_{T_1 \to T_2}(t)$ iff t halts and, whenever $R_{T_1}(s)$, we have $R_{T_2}(ts)$.

Proof Outline

- Theorem [Normalization]: If $\vdash t : T$, then t is normalizable.
- Steps of Proof
  1. Every element of every set R_T is normalizable
  2. Every well-typed term of type T is an element of R_T.

Proof Outline (Cont.) I

1. The first step is immediate from the definition of R_T.

   Lemma: If $R_T(t)$, then t halts.

2. The second step is broken into two lemmas.

   Lemma: If $t : T$ and $t \rightarrow t'$, then $R_T(t)$ iff $R_T(t')$

   Proof: by induction on the structure of the type T.
   For "only if" direction ($\Longrightarrow$):

   - If $T = A$, there is nothing more to show.
   - Suppose that $T = T_1 \rightarrow T_2$ for some T_1 and T_2, and that $R_T(t)$ and that $R_{T_1}(s)$ for some arbitrary $s : T_1$.
     - By definition: $R_{T_2}(t\ s)$
     - By induction hypothesis: $R_{T_2}(t'\ s)$ since $t\ s \rightarrow t'\ s$

Proof Outline (Cont.) II

Since this holds for an arbitrary s, we have $R_T(t')$.
The proof of "if" direction ($\Longleftarrow$) is similar.

> *Lemma: if $x_1 : T_1, \ldots, x_n : T_n \vdash t : T$ and $v_1, \ldots, v_n$ are closed values of types $T_1, \ldots, T_n$ with $R_{T_i}(v_i)$ for each i, then $R_T([x_1 \mapsto v_1] \cdots [x_n \mapsto v_n]t)$.*

Proof: by induction on a derivation of $x_1 : T_1, \ldots, x_n : T_n$.
(See the proof of *Lemma 12.1.5*.)

References

Introduction

- Basics

  The basic operations on references are *allocation*, *dereferencing*, and *assignment*.

  - To allocate a reference, we use the **ref** operator, providing an initial value for the new cell.

    $r = ref\ 5;$

    $\triangleright\ r\!:\ Ref\ Nat$

  - To change the value stored in the cell, we use the assignment operator.

    $r := 7;$

    $\triangleright\ unit\!:\ Unit$

  - If we dereference r again, we see the updated value.

    $!r;$

    $\triangleright\ 7 : Nat$

Introduction (Cont.)

- Side Effects and Sequencing

  The fact that the result of an assignment expression is that the trivial value *unit* fits nicely with the sequencing notation.

$$\frac{t_1 \longrightarrow t_1'}{t_1; t_2 \longrightarrow t_1'; t_2}$$

$$unit; t_2 \longrightarrow t_2$$

  We can write $(r := succ(!r); !r)$; instead of the equivalent, but more cumbersome, $(\lambda\_ : Unit.!r)(r := succ(!r))$; to evaluate two expressions in order and return the value of the second.

Introduction (Cont.)

- References and Aliasing
  If we make a cope of r ($s = r$), what gets copied if only the reference, not the cell.
  The references r and s are said to be *aliases* for the same cell.
- Shared State
  For example,

$$
\begin{aligned}
&c = ref\ 0;\\
&incc = \lambda x{:}Unit.\ (c := succ\ (!c);\ !c);\\
&decc = \lambda x{:}Unit.\ (c := pred\ (!c);\ !c);\\
&o = \{\ i = incc,\ d = decc\ \}\ ;
\end{aligned}
$$

  The whole structure can be passed around as a unit. Its components can be used to perform incrementing and decrementing operations on theshared piece of state in c.

Introduction (Cont.)

- References to Compound Types
  An example: an implementation of arrays of numbers

$$
\begin{array}{l}
\mathit{NatArray} = \mathit{Ref}\ (\mathit{Nat} \to \mathit{Nat});\\
\mathit{newarray} = \lambda\_{:}\mathit{Unit}.\ \mathit{ref}\ (\lambda n{:}\mathit{Nat}.0);\\
\mathit{lookup} = \lambda a{:}\mathit{NatArray}.\ \lambda n{:}\mathit{Nat}.\ (!a)\ n;\\
\mathit{update} = \lambda a{:}\ \mathit{NatArray}.\ \lambda m{:}\mathit{Nat}.\ \lambda v{:}\mathit{Nat}\\
\quad \mathit{let}\ \mathit{oldf} = !a\ \mathit{in}\\
\quad a := (\lambda a{:}\mathit{NatArray}.\ \mathit{if}\ \mathit{equal}\ m\ n\ \mathit{then}\ v\\
\quad \mathit{else}\ \mathit{oldf}\ n);
\end{array}
$$

- No garbage collection primitives for freeing reference cells

Typing Rules for **ref**, :=, and !

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \mathit{ref}\ \ t_1 : \mathit{Ref}\ \ T_1}$$

$$\frac{\Gamma \vdash t_1 : \mathit{Ref}\ \ T_1}{\Gamma \vdash\ !t_1 : T_1}$$

$$\frac{\Gamma \vdash t_1 : \mathit{Ref}\ \ T_1 \qquad \Gamma \vdash\ \ t_2 : T_1}{\Gamma \vdash t_1 := t_2 : \mathit{Unit}}$$

Evaluation

In most programming language implementations

- The run-time store is a big array of bytes.
- A new reference cell is a large enough segment form the free region of the store(4 bytes for integer cells, 8 bytes for cells storing **Float**, tec.)
- A reference is the index of the start of the newly allocated region

Abstraction

- The store is an array of *values*.
- Each value is a reference cell.
- A reference is an element of some uninterpreted set L of *store locations*.
- A store becomes a partial function from locations l to values.
- The metavariable μ is used to range over stores

Change of syntax

$$
\begin{aligned}
v ::= {} & \cdots \\
& l \\[1em]
t ::= {} & \cdots \\
& ref\ t \\
& !t \\
& t := t \\
& l
\end{aligned}
$$

Augmenting existing evaluation rules

$$(\lambda x : T_{11}.t_{12})v_2|\mu \rightarrow [x \mapsto v_2]t_{12}|\mu$$

$$\frac{t_1|\mu \rightarrow t_1'|\mu'}{t_1\,t_2|\mu \rightarrow t_1'\,t_2|\mu'}$$

$$\frac{t_2|\mu \rightarrow t_2'|\mu'}{v_1\,t_2|\mu \rightarrow v_1\,t_2'|\mu'}$$

New Evaluation rules

$$\frac{t_1|\mu \rightarrow t_1'|\mu'}{!t_1|\mu \rightarrow !t_1'|\mu'}$$

$$\frac{\mu(l) = v}{!l|\mu \rightarrow v|\mu}$$

$$\frac{t_1|\mu \rightarrow t_1'|\mu'}{t_1 := t_2|\mu \rightarrow t_1' := t_2|\mu'}$$

McMaster University

New Evaluation rules (Cont.)

$$\frac{t_2|\mu \rightarrow t_2'|\mu'}{v_1 := t_2|\mu \rightarrow v_1 := t_2'|\mu'}$$

$$l := v_2|\mu \rightarrow unit|[l \mapsto v_2]\mu$$

$$\frac{t_1|\mu \rightarrow t_1'|\mu'}{ref\ \ t_1|\mu \rightarrow ref\ \ t_1'|\mu'}$$

$$\frac{l \notin dom(\mu)}{ref\ \ v_1|\mu \rightarrow l|(\mu, l \mapsto v_1)}$$

Store Typings

- First attempt

$$\frac{\Gamma \vdash \mu(l) : T_1}{\Gamma \vdash l : Ref\ \ T_1}$$

- Second attempt

$$\frac{\Gamma|\mu \vdash \mu(l) : T_1}{\Gamma|\mu \vdash l : Ref\ \ T_1}$$

Store Typings (Cont.)

- Problems
  - Inefficient

$$
\begin{aligned}
&(l_1 \mapsto \lambda x : Nat.\ 999,\\
&l_2 \mapsto \lambda x : Nat.\ (!l_1),\\
&l_3 \mapsto \lambda x : Nat.\ (!l_2),\\
&l_4 \mapsto \lambda x : Nat.\ (!l_3),\\
&l_5 \mapsto \lambda x : Nat.\ (!l_4)),
\end{aligned}
$$

  - The store may contains *cycle*

$$
\begin{aligned}
&(l_1 \mapsto \lambda x : Nat.\ (!l_2)\ ,\\
&(l_2 \mapsto \lambda x : Nat.\ (!l_1)),
\end{aligned}
$$

Store Typings (Cont.)

- Solution
  - Every location has a single, definite type in the store.
  - Store typing Σ is defined as a finite function mapping locations to types.

Typing Rules

Typing rules

$$\frac{\Sigma(l) = T_1}{\Gamma|\Sigma \vdash l : Ref \;\; T_1}$$

$$\frac{\Gamma|\Sigma \vdash t_1 : T_1}{\Gamma|\Sigma \vdash ref \;\; t_1 : Ref \;\; T_1}$$

$$\frac{\Gamma|\Sigma \vdash t_1 : Ref \;\; T_{11}}{\Gamma|\Sigma \vdash !t_1 : T_{11}}$$

$$\frac{\Gamma|\Sigma \vdash t_1 : Ref \;\; T_{11} \qquad \Gamma|\Sigma \vdash \;\; t_2 : T_{11}}{\Gamma|\Sigma \vdash t_1 := t_2 : Unit}$$

Safety

- Defintion: A store μ is said to be *well typed* with respect to a typing context Γ and a store typing Σ, written $\Gamma | \Sigma \vdash \mu$, if $dom(\mu) = dom(\Sigma)$ and $\Gamma | \Sigma \vdash \mu(l) : \Sigma(l)$ for every $l \in dom(\mu)$.
- Lemma [Substitution]: If $\Gamma, x : S | \Sigma \vdash t : T$ and $\Gamma | \Sigma \vdash s : S$, then $\Gamma | \Sigma \vdash [x \mapsto s]t : T$.
- Lemma: If

$$\begin{array}{l} \Gamma | \Sigma \vdash \mu \\ \Sigma(l) = T \\ \Gamma | \Sigma \vdash v : T \end{array}$$

  then, $\Gamma | \Sigma \vdash [l \mapsto v]\mu$
- Lemma: If $\Gamma | \Sigma \vdash t : T$ and $\Sigma' \supseteq \Sigma$, then $\Gamma | \Sigma' \vdash t : T$.

Safety (Cont.)

- Theorem [Preservation]: If

  $$\begin{array}{l} \Gamma|\Sigma \vdash t : T \\ \Gamma|\Sigma \vdash \mu \\ t|\mu \to t'|\mu' \end{array}$$

  then, for some $\Sigma' \supseteq \Sigma$,

  $$\begin{array}{l} \Gamma|\Sigma' \vdash t' : T \\ \Gamma|\Sigma' \vdash \mu' \end{array}$$

- Theorem [Progress]: Suppose t is a closed, well-typed term (that is $\emptyset|\Sigma \vdash t : T$ for some T and Σ). Then either t is a value or else, for any store μ such that $\emptyset|\Sigma \vdash \mu$, there is some term t' and store μ' with $t|\mu \to t'|\mu'$.