1/3 November
Fall 2004
Software Foundations
CIS 500

References

Mutability

Announcements

Homework 7 out.

No office hours for Professor this week.


Incorrect answers: Plus = \|a\|, \|a\| \|b\|. In \{a, b\} since x.

Extra credit problem 8(i) of the exam.

In most programming languages, variables are mutable — i.e., a variable
value is immutable.

Variables are only for numbers — the builder between a variable and its
value is immutable.

In some languages (e.g. OCaml), these two features are kept separate.

Variables are only for numbers — the builder between a variable and its
value is immutable.

In some languages (e.g. OCaml), these two features are kept separate.
So we can change \( x \) by assigning to \( \mathbf{S} \):

\[
\begin{array}{c}
\mathbf{S} & = x \quad \mathbf{L} \\
\end{array}
\]

\[= \mathbf{S} \]

A value of type \( \mathbf{L} \) is a pointer to a cell holding a value of type \( \mathbf{L} \).
Example

```c
{ (t = trace, d = dec)
  dec w.
  trace w.
  dec = x: z:=. (c := read (ic) ; i)
  trace = x: z:=. (c := succ (ic) ; i)
  c = read 0
}
```

The difficulties of aliasing

- c++
- shared resources (c++), threads (in concurrent systems)
- exception handling (c++), stack-based (C, C++)
- callback (C++ (2013+)

The benefits of aliasing

- There are good reasons why most languages do provide constructs like
  `new`, `delete` (C++)
- But there are also reasons why some languages do not provide constructs like
  `new`, `delete` (C++)
Syntax

Another example

Ⅹ : T
Ⅹ := T
Ⅹ +Ⅹ
Ⅹ −Ⅹ
Ⅹ ×Ⅹ
Ⅹ ÷Ⅹ
Ⅹ =Ⅹ
Ⅹ ≠Ⅹ
Ⅹ <Ⅹ
Ⅹ ≤Ⅹ
Ⅹ >Ⅹ
Ⅹ ≥Ⅹ

Type Rules

When is the value of the expression T of type T₁?

Evaluation

Reference to T

function

unit

Reference creation

assignment
dereference

Reference creation

application

vardecl

unit construct

let

... plus other similar types in examples.
So what is a reference (or pointer) to that store? Specifically, evaluating \texttt{x} should allocate some storage and yield a

\begin{align*}
& i = s \\
& o \text{ ret } i
\end{align*}

and

\begin{align*}
& 0 \text{ ret } o \\
& 0 \text{ ret } o
\end{align*}

Otherwise,

\begin{align*}
& \text{ret } 0 \\
& \text{ret } 0
\end{align*}

Where is the value of the expression \texttt{ret o}?

\begin{align*}
\text{ret } 0
\end{align*}

\begin{align*}
\text{ret } 0
\end{align*}

\begin{align*}
\text{ret } 0
\end{align*}

\begin{align*}
\text{ret } 0
\end{align*}

A reference names a location in the store (also known as the heap or just the store).

\textbf{Evaluation}

\textbf{Evaluation}
We use the measurable $n$ to range over scores.

$$n \leftarrow \sum$$

Let the evaluation function map a term and a score to a reduced term and a new score.

The result of evaluating a term now depends on the score in which it is evaluated. All results of evaluating a term not just a value — we evaluate the result of evaluating a term not just a value — we evaluate the result of evaluating a term now depends on the score in which it is.

**Evaluation**

**Syntax of Terms**

1. `store-location` 
2. `assignment` 
3. `reference-citation` 
4. `application` 
5. `expression`

```
let x * x
```

**Aside**

Set of terms: locations to the set of values (results of evaluation) without adding them to the set as new values. When we add an implicit evaluation in the expression, the value can add as an additional evaluation to the implicit evaluation of source terms.

Aside: if we formulate evaluation in the expression, we can add evaluation as a relation between source terms.

Includes some primitive structures so that we can continue to formulate evaluation as a relation between source terms. When we add an implicit evaluation in the expression, the value can add as an additional evaluation to the implicit evaluation of source terms.

Does this mean we are going to allow programs to write explicit locations?
(E-APPL2)

\[ \frac{\pi \vdash f \cdot t \quad \pi \vdash x \quad \pi \vdash \lambda x. t}{\pi \vdash \lambda f. t} \]

(2) (E-APPL)

\[ \frac{\pi \vdash f \quad \pi \vdash t}{\pi \vdash f \cdot t} \]

(3) (E-APPL)

\[ \frac{\pi \vdash f \quad \pi \vdash t}{\pi \vdash f \cdot t} \]

(Aside: Garbage collection)

Aside: Garbage collection — the store just grows

without bound

Note that we are not modifying structures in the store just grows...
Q: What is the type of a location?

Typing Locations

Aside: Pointer arithmetic

int *p; // p is a pointer

p = &x; // p now points to x

*p = 5; // Dereference p before assigning

storeTyping

But in the store (if uint, i.e. x: uint x), the term i\_2 has type

E.g., in the store (if uint, i.e. x: uint), the term i\_2 has type uint

A: It depends on the store

Q: What is the type of a location?
Then how is the typing derivation for $\pi_2$?

\[
\begin{array}{c}
\Gamma \vdash x : \text{Nat} \\
\Gamma \vdash \pi_2 (x) : \text{Nat}
\end{array}
\]

Problem

Unlike derivations not labeled, however, this rule is not completely syntactically for one thing, it can make

The typing derivations very large.

\[
\begin{array}{c}
\Gamma \vdash x : \text{Nat} \\
\Gamma \vdash \pi_1 (x) : \text{Nat}
\end{array}
\]

Example: (continued)

\[
\begin{array}{c}
\Gamma \vdash x : \text{Nat} \\
\Gamma \vdash \pi_1 (x) : \text{Nat}
\end{array}
\]

Example: (continued)

\[
\begin{array}{c}
\Gamma \vdash x : \text{Nat} \\
\Gamma \vdash \pi_1 (x) : \text{Nat}
\end{array}
\]

Example: (continued)

\[
\begin{array}{c}
\Gamma \vdash x : \text{Nat} \\
\Gamma \vdash \pi_1 (x) : \text{Nat}
\end{array}
\]

Example: (continued)

\[
\begin{array}{c}
\Gamma \vdash x : \text{Nat} \\
\Gamma \vdash \pi_1 (x) : \text{Nat}
\end{array}
\]
**Typing Terms and Types**

Let τ be now a four-place relation between contexts, score, and types.

\[
\begin{align*}
&\text{L.TYP} : 1 \vdash \tau \\
&\text{L.TDEF} : \text{L.TYP} \vdash \text{L.TYP} \\
&\text{L.TDER} : \text{L.TYP} \vdash \text{L.TYP} \\
&\text{L.TREV} : \text{L.TYP} \vdash \text{L.TYP} \\
&\text{L.TLOC} : \text{L.TYP} \vdash \tau
\end{align*}
\]

These intended types can be collected into a score τ瞬 — a partial function.

**Score Typing**

A reasonable score τ瞬 would be

\[
\begin{align*}
\text{L.TYP} & \vdash \tau \\
\text{L.TDEF} & \vdash \tau \\
\text{L.TDER} & \vdash \tau \\
\text{L.TREV} & \vdash \tau \\
\text{L.TLOC} & \vdash \tau
\end{align*}
\]
Safety

WHERE do these score 5's come from?

appropria te:
we can observe the type of \( \mathcal{L} \) and extend the "current score 5 prints"

(E-REV) \[ \begin{align*}
\text{let } \pi_1 & \leftarrow \pi, \\
\text{ret } \pi_1 & \\
\text{dom } \pi & \exists \, 1
\end{align*} \]

So when a new location is created during evaluation,
we can use an empty score 5 for
A: When we first encounter a program, there will be no explicit locations so