Announcements

- Homework 0 grades posted.
- Homework 1 explanations posted
- Time Spent
  - HW0: mean 8.4 hrs, std dev 4.5, median 8
  - HW1: mean 7.9 hrs, std dev 4.7, median 7.5
- Homework 2 due Thu at 11:55pm

HW1 Feedback

- Interactivity++++
  - “initial skepticism but now find it beneficial and should increase it even more”
- Resources well-organized
- High-level of energy
- Connection between theory and applications
- Like forums+++ 
- Dislike forums---
- Want better closure on forums
- Dislike SLAM/BLAST classes (Like)

Questions?

Review of Contextual Operational Semantics

Context Decomposition Theorem

- If c is not "skip" then there exist unique H and r such that c is H[r]
  - What does this mean?

- unique = deterministic
  - event = progress
**Unique Next Redex:** \( \langle H, c, \sigma \rangle \rightarrow \langle H[c], \sigma \rangle \)

"Proof" By Handwaving Examples

- e.g., \( c = c_1 ; c_2 \) – either
  - \( c_1 = \text{skip} \) and then \( c = H[\text{skip}; c_2] \) with \( H = \bullet \)
  - or \( c_1 \neq \text{skip} \) and then \( c_1 = H[r] \); so \( c = H[r] \) with \( H' = H; c_2 \)

- e.g., \( c = "\text{if } b \text{ then } c_1 \text{ else } c_2" \)
  - either \( b = \text{true} \) or \( b = \text{false} \) and then \( c = H[r] \) with \( H = \bullet \)
  - or \( b \) is not a value and \( b = H[r] \); so \( c = H[r] \) with \( H' = \text{if } H \text{ then } c_1 \text{ else } c_2 \)

What if we want short-circuit evaluation of \( \land \)?

- Define the following contexts, redexes and local reduction rules:

\[
H ::= \ldots \mid H \land b \\
r ::= \ldots \mid \text{true} \land b \mid \text{false} \land b \\
<\text{true} \land b, \sigma> \rightarrow <b, \sigma> \\
<\text{false} \land b, \sigma> \rightarrow <\text{false}, \sigma>
\]

- the local reduction kicks in before \( b_2 \) is evaluated

Summary:

Contextual Operational Semantics

- Can view \( \bullet \) as representing the program counter

  - The advancement rules for \( \bullet \) are non-trivial

  - This makes contextual semantics inefficient to implement directly

- The major advantage of contextual semantics: allows a mix of local and global reduction rules

  - For IMP we have only local reduction rules: only the redex is reduced

  - Sometimes it is useful to work on the context too

End of Review

Questions?
“Real World”

Reading Real-World Examples

- Cobbe and Felleisen, POPL 2005
- Small-step contextual op. sem. for Java
- Their rule for object field access:

\[
P \vdash <E[obj.fd], S> \rightarrow <E[F(fd)], S> \]

where \( F = \text{fields}(S(obj)) \) and \( fd \in \text{dom}(F) \)

- They use “E” for context, we use “H”
- They use “S” for state, we use “\( \sigma \)”

Lost In Translation

- \( P \vdash <H[obj.fd], \sigma> \rightarrow <H[F(fd)], \sigma> \)
  where \( F = \text{fields}(\sigma(obj)) \) and \( fd \in \text{dom}(F) \)

- They have “\( P \vdash \)” but that just means “it can be proved in our system given \( P \)”

- \( <H[obj.fd], \sigma> \rightarrow <H[F(fd)], \sigma> \)
  where \( F = \text{fields}(\sigma(obj)) \) and \( fd \in \text{dom}(F) \)

Lost In Translation

- \( <H[obj.fd], \sigma> \rightarrow <H[F(fd)], \sigma> \)
  where \( F = \text{fields}(\sigma(obj)) \) and \( fd \in \text{dom}(F) \)

- They model objects (like \( obj \)), but we do not (yet); let’s just make \( fd \) a variable:
  - \( <H[fd], \sigma> \rightarrow <H[F(fd)], \sigma> \)
    where \( F = \sigma \) and \( fd \in L \)

- That’s just our variable-lookup rule:
  - \( <H[fd], \sigma> \rightarrow <H[\sigma(fd)], \sigma> \) (when \( fd \in L \))

On to Proof Techniques

Plan

- Why Bother?
- Mathematical Induction
- Well-Founded Induction
- Structural Induction
  - “By induction on the structure of the derivation \( D \)”
### One-Slide Summary

- **Mathematical induction** is a proof technique. For what?
  - For all $n: P(n)$, if $P(0)$ and $P(n) \Rightarrow P(n+1)$, then you can conclude that for all natural numbers $n$, $P(n)$ holds.

- Why does induction work?
  - Induction works because the natural numbers are well-founded: there are no infinite descending chains $n > n-1 > n-2 > ... > ...$.

- **Structural induction** is induction on a formal structure, like an abstract syntax. The base cases use the leaves, the inductive steps use the inner nodes.

- **Induction on a derivation** is structural induction applied to a derivation $D$ (e.g., $D :: <c, \sigma \sigma \sigma \sigma > \downarrow \downarrow \downarrow \downarrow \sigma \sigma \sigma \sigma'$).

### Why Bother?

- I must convince you op. sem. proof techniques are useful.

- Recall class goals:
  - Understand PL research techniques and
  - Apply them to your research

### Classic Example (Schema)

- "A well-typed program cannot go wrong.
- Robin Milner"
- When you design a new type system, you must show that it is safe
  - that the type system is sound with respect to the operational semantics

- **A Syntactic Approach to Type Soundness.** Andrew K. Wright, Matthias Felleisen, 1992.
  - Preservation: "if you have a well-typed program and apply an opsem rule, the result is well-typed."
  - Progress: "a well-typed program will never get stuck in a state with no applicable opsem rules"
- Done for real languages: SML, SPARK ADA, Java
  - plus basically every toy PL research language ever

### Instances

- **CCured Project (Berkeley)**
  - A program that is instrumented with CCured run-time checks ("adherence to the CCured type system") will not segfault (<"the x86 op. sem. rules will never get stuck">

- **Vault Language (Microsoft Research)**
  - A well-typed Vault program does not leak any tracked resources and invokes tracked APIs correctly (e.g., handles DRQ, correctly in asynchronous Windows device drivers; cf. Capability Calculus)

- **RC - Reference-Counted Regions For C (Intel Research)**
  - A well-typed RC program gains the speed and convenience of region-based memory management but need never worry about freeing a region too early (run-time checks).

- **Typed Assembly Language (Cornell)**
  - Reasonable C programs (e.g., device drivers) can be translated to TALx86. Well-typed TALx86 programs are type- and memory-safe.

### Induction!

- Probably most important technique for studying the formal semantics of PLs
  - To perform or understand PL research, you must grok this!

- **Mathematical Induction (simple)**
- **Well-Founded Induction (general)**
- **Structural Induction (widely used in PL)**
Mathematical Induction

• Goal: prove $\forall n \in \mathbb{N}. P(n)$

• **Base Case:** prove $P(0)$

• **Inductive Step:**
  - Prove "For all $n$, if $P(n)$, then $P(n+1)$"
  - "Pick arbitrary $n$, assume $P(n)$, prove $P(n+1)$"

• Why does induction work?

Why does it work?

• There are no infinite descending chains of natural numbers

• For any $n$, $P(n)$ can be obtained by starting from the base case and applying $n$ instances of the inductive step

Example (With IMP Eval. Semantics)

• Prove that if $\sigma(x) \leq 6$ then $\langle \text{while } x \leq 5 \text{ do } x := x + 1, \sigma \rangle \triangleright \sigma[x := 6]$

• Reformulate the claim:
  - Let $W = \text{while } x \leq 5 \text{ do } x := x + 1$
  - Let $\sigma_i = \sigma[x := 6 - i]$
  - Claim: For all $i \in \mathbb{N} \times W, \sigma_i \triangleright \sigma_0$

• Now looks provable by mathematical induction on $i$

Evaluation Example (Base Case)

• Base case: $i = 0$ or $\langle W, \sigma_0 \triangleright \sigma_0 \rangle$
  - To prove an evaluation judgment, construct a derivation tree:

Evaluation Example (Inductive Case)

• Must prove for all $i \in \mathbb{N}$, if $\langle W, \sigma_i \triangleright \sigma_0 \rangle$, then $\langle W, \sigma_{i+1} \triangleright \sigma_0 \rangle$
  - Pick an arbitrary $i \in \mathbb{N}$
  - Assume that $\langle W, \sigma_i \triangleright \sigma_0 \rangle$
  - Now prove that $\langle W, \sigma_{i+1} \triangleright \sigma_0 \rangle$
  - Must construct a derivation tree: