Operational Semantics
Small-Step Semantics

Today’s Cunning Plan
• Review, Truth, and Provability
• Large-Step Opsem Commentary
• Small-Step Contextual Semantics
  - Reductions, Redexes, and Contexts
• Applications
• (Induction)

Summary - Semantics
• A **formal semantics** is a system for assigning meanings to programs.
• For now, programs are IMP commands and expressions
• In **operational semantics** the meaning of a program is “what it evaluates to”
• Any opsem system gives **rules of inference** that tell you how to evaluate programs

Summary - Judgments
• Rules of inference allow you to derive **judgments** (“something that is knowable”) like
  \(<e, \sigma> \Downarrow n\)
  - In state \(\sigma\), expression \(e\) evaluates to \(n\)
  \(<c, \sigma> \Downarrow \sigma'\)
  - After evaluating command \(c\) in state \(\sigma\) the new state will be \(\sigma'\)
• State \(\sigma\) maps variables to values (\(\sigma : L \rightarrow Z\))
• Inferences equivalent up to variable renaming:
  \(<c, \sigma> \Downarrow \sigma' \iff <c', \sigma> \Downarrow \sigma'_a\)

Summary - Rules
• **Rules of inference** list the hypotheses necessary to arrive at a conclusion
  \(<x, \sigma> \Downarrow \sigma(x)\)
  \(<e_1, \sigma> \Downarrow n_1, <e_2, \sigma> \Downarrow n_2\)
  \(<e_1 - e_2, \sigma> \Downarrow n_1 \text{ minus } n_2\)
• A **derivation** involves interlocking instances of rules of inference
  \(<4, \sigma> \Downarrow 4\)
  \(<2, \sigma> \Downarrow 2\)
  \(<4*2, \sigma> \Downarrow 8\)
  \(<6, \sigma> \Downarrow 6\)
  \(<4*2 - 6, \sigma> \Downarrow 2\)

Provability
• Given an opsem system, \(<e, \sigma> \Downarrow n\) is **provable** if there exists a well-formed derivation with \(<e, \sigma> \Downarrow n\) as its conclusion
  - "well-formed" = "every step in the derivation is a valid instance of one of the rules of inference for this opsem system"
  - "\(<e, \sigma> \Downarrow n\) = "it is provable that \(<e, \sigma> \Downarrow n\)"
• We would like truth and provability to be closely related
Truth?

- “A Vorlon said understanding is a three-edged sword. Your side, their side and the truth.”
  - Sheridan, *Into The Fire*
- We will not formally define “truth” yet
- Instead we appeal to your intuition
  - \(<2+2, \sigma> \downarrow 4\) -- should be true
  - \(<2+2, \sigma> \downarrow 5\) -- should be false

Completeness

- A proof system (like our operational semantics) is complete if every true judgment is provable.
- If we replaced the subtract rule with:
  \[
  \frac{\langle e_1, \sigma \rangle \downarrow n \quad \langle e_2, \sigma \rangle \downarrow 0}{\langle e_1 - e_2, \sigma \rangle \downarrow n}
  \]
  - Our opsem would be incomplete:
    - \(<4-2, \sigma> \downarrow 2\) -- true but not provable

Consistency

- A proof system is consistent (or sound) if every provable judgment is true.
- If we replaced the subtract rule with:
  \[
  \frac{\langle e_1, \sigma \rangle \downarrow n_1 \quad \langle e_2, \sigma \rangle \downarrow n_2}{\langle e_1 - e_2, \sigma \rangle \downarrow n_1 + 3}
  \]
  - Our opsem would be inconsistent (or unsound):
    - \(<6-1, \sigma> \downarrow 9\) -- false but provable

Desired Traits

- Typically a system (of operational semantics) is always complete (unless you forget a rule)
- If you are not careful, however, your system may be unsound
- Usually that is very bad
  - A paper with an unsound type system is usually rejected
  - Papers often prove (sketch) that a system is sound
  - Recent research (e.g., Engler, ESP) into useful but unsound systems exists, however
- In this class your work should be complete and consistent (e.g., on homework problems)

With That In Mind

- We now return to opsem for IMP
  \[
  \frac{\langle e, \sigma \rangle \downarrow n}{\langle x := e, \sigma \rangle \downarrow \sigma[x := n]}
  \]
  - Def: \(\sigma[x := n](x) = n\)
  - \(\sigma[x := n](y) = \sigma(y)\)

  \[
  \frac{\langle b, \sigma \rangle \downarrow false}{\langle while b do c, \sigma \rangle \downarrow \sigma'}
  \]
  \[
  \frac{\langle b, \sigma \rangle \downarrow true \quad \langle c; while b do c, \sigma \rangle \downarrow \sigma'}{\langle while b do c, \sigma \rangle \downarrow \sigma'}
  \]

Command Evaluation Notes

- The order of evaluation is important
  - \(c_i\) is evaluated before \(c_j\) in \(c_i; c_j\)
  - \(c_j\) is not evaluated in "if true then \(c_i\) else \(c_j\)"
  - \(c\) is not evaluated in "while false do \(c\)"
  - \(b\) is evaluated first in "if \(b\) then \(c_i\) else \(c_j\)"
  - this is explicit in the evaluation rules
- Conditional constructs (e.g., \(b_1 \lor b_2\)) have multiple evaluation rules
  - but only one can be applied at one time
Command Evaluation Trials

- The evaluation rules are **not syntax-directed**
  - See the rules for while, ∧
  - The evaluation **might not terminate**
- Recall: the evaluation rules suggest an interpreter
- Natural-style semantics has two big disadvantages (continued ...)

Disadvantages of Natural-Style Operational Semantics

- It is hard to talk about commands whose evaluation does **not terminate**
  - i.e., when there is no \( \sigma' \) such that \(<c, \sigma> \Downarrow \sigma'\)
  - But that is true also of ill-formed or erroneous commands (in a richer language!)
- It does not give us a way to talk about **intermediate states**
  - Thus we cannot say that on a parallel machine the execution of two commands is interleaved

Semantics Solution

- **Small-step semantics** addresses these problems
  - Execution is modeled as a (possible infinite) sequence of states
- Not quite as easy as large-step natural semantics, though
- **Contextual semantics** is a small-step semantics where the atomic execution step is a rewrite of the program

Contextual Semantics

- We will define a relation \(<c, \sigma> \rightarrow <c', \sigma'>\)
  - \(c'\) is obtained from \(c\) via an **atomic rewrite step**
  - Evaluation terminates when the program has been rewritten to a **terminal program**
    - one from which we cannot make further progress
  - For IMP the terminal command is "skip"
  - As long as the command is not "skip" we can make further progress
    - some commands never reduce to skip (e.g., "while true do skip")

Contextual Derivations

- In small-step contextual semantics, derivations are not tree-structured
- A **contextual semantics derivation** is a sequence (or list) of atomic rewrites:

  \(<x+(7-3),\sigma> \rightarrow <x+(4),\sigma> \rightarrow <5+4,\sigma> \rightarrow <9,\sigma>\)

What is an Atomic Reduction?

- **What is an atomic reduction step?**
  - Granularity is a choice of the semantics designer
- **How to select the next reduction step**, when several are possible?
  - This is the **order of evaluation** issue
Redexes

- A redex is a syntactic expression or command that can be reduced (transformed) in one atomic step.
- Defined as a grammar:
  \[ r ::= x \quad (x \in L) \]
  | \[ n_1 + n_2 \]
  | \[ x := n \]
  | \[ skip; c \]
  | \[ if \text{true} \then c \else c_2 \]
  | \[ if \text{false} \then c \else c_2 \]
  | \[ \text{while} \ b \then c \]
- For brevity, we mix exp and command redexes.
- Note that \((1 + 3) + 2\) is not a redex, but \(1 + 3\) is.

Local Reduction Rules for IMP

- One for each redex: \(<r, \sigma> \rightarrow <e, \sigma'>\)
  - means that in state \(\sigma\), the redex \(r\) can be replaced in one step with the expression \(e\).
  \[<x, \sigma> \rightarrow <\sigma(x), \sigma>\]
  \[<n_1 + n_2, \sigma> \rightarrow <n, \sigma>\quad \text{where } n = n_1 + n_2\]
  \[<n_1 = n_2, \sigma> \rightarrow <\text{true}, \sigma>\quad \text{if } n_1 = n_2\]
  \[<x := n, \sigma> \rightarrow <\text{skip}, \sigma[x := n]>\]
  \[<\text{skip}; c, \sigma> \rightarrow <c, \sigma>\]
  \[<\text{if true} \then c \else c_2, \sigma> \rightarrow <c_1, \sigma>\]
  \[<\text{if false} \then c \else c_2, \sigma> \rightarrow <c_2, \sigma>\]
  \[<\text{while} \ b \then c \text{do } c, \sigma> \rightarrow <\text{if } b \then c; \text{while } b \text{do } c \text{ else skip}, \sigma>\]

The Global Reduction Rule

- General idea of contextual semantics
  - Decompose the current expression into the redex-to-reduce-next and the remaining program.
  - The remaining program is called a context.
  - Reduce the redex "\(r\)" to some other expression "\(e\)".
  - The resulting (reduced) expression consists of "\(e\)" with the original context.

As A Picture (1)

( Context )
...
  x := 2+2
...

Step 1: Find The Redex

As A Picture (2)

( Context )
...
  x := 2+2 (redex)
...

Step 1: Find The Redex
Step 2: Reduce The Redex

As A Picture (3)

( Context )
...
  x := 2+2 (redex)
...

4 (reduced)

Step 1: Find The Redex
Step 2: Reduce The Redex
As A Picture (4)

( Context )
...  x := 4  
...

Step 1: Find The Redex
Step 2: Reduce The Redex
Step 3: Replace It In The Context

Contextual Analysis

• We use H to range over contexts
• We write H[r] for the expression obtained by placing redex r in context H
• Now we can define a small step

If <r, σ> → <e, σ'>
then <H[r], σ> → <H[e], σ'>

Contexts

• A context is like an expression (or command) with a marker • in the place where the redex goes
• Examples:
  - To evaluate "(1 + 3) + 2" we use the redex 1 + 3 and the context "• + 2"
  - To evaluate "if x > 2 then c₁ else c₂" we use the redex x and the context "if • > 2 then c₁ else c₂"

Context Terminology

• A context is also called an "expression with a hole"
• The marker • is sometimes called a hole
• H[r] is the expression obtained from H by replacing • with the redex r

Contextual Semantics Example

• x := 1 ; x := x + 1 with initial state [x:=0]

<table>
<thead>
<tr>
<th>Context</th>
<th>Redex</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;x := 1; x := x + 1, [x := 0]&gt; x := 1</td>
<td>•; x := x + 1</td>
<td></td>
</tr>
<tr>
<td>&lt;skip; x := x + 1, [x := 1]&gt; skip; x := x + 1</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>&lt;x := x + 1, [x := 1]&gt; x</td>
<td>x := • + 1</td>
<td></td>
</tr>
</tbody>
</table>

What happens next?

Contextual Semantics Example

• x := 1 ; x := x + 1 with initial state [x:=0]

<table>
<thead>
<tr>
<th>Context</th>
<th>Redex</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;x := 1; x := x + 1, [x := 0]&gt; x := 1</td>
<td>•; x := x + 1</td>
<td></td>
</tr>
<tr>
<td>&lt;skip; x := x + 1, [x := 1]&gt; skip; x := x + 1</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>&lt;x := x + 1, [x := 1]&gt; x</td>
<td>x := • + 1</td>
<td></td>
</tr>
<tr>
<td>&lt;x := 1 + 1, [x := 1]&gt; 1 + 1</td>
<td>x := •</td>
<td></td>
</tr>
<tr>
<td>&lt;x := 2, [x := 1]&gt; x := 2</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>&lt;skip, [x := 2]&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More On Contexts
- Contexts are defined by a grammar:
  \[ H ::= \bullet | n + H | H + e | x := H | \text{if } H \text{ then } c_1 \text{ else } c_2 | H; c \]
- A context has exactly one \( \bullet \) marker
- A redex is never a value

What’s In A Context?
- Contexts specify precisely how to find the next redex
  - Consider \( e_1 + e_2 \) and its decomposition as \( H[r] \)
  - If \( e_1 \) is \( n_1 \) and \( e_2 \) is \( n_2 \) then \( H = \bullet \) and \( r = n_1 + n_2 \)
  - If \( e_1 \) is \( n_1 \) and \( e_2 \) is not \( n_2 \) then \( H = n_1 + H_2 \) and \( e_2 = H_2[r] \)
  - If \( e_1 \) is not \( n_1 \) then \( H = H_1 + e_2 \) and \( e_1 = H_1[r] \)
  - In the last two cases the decomposition is done recursively
  - Check that in each case the solution is unique

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

Context Decomposition
- Decomposition theorem:
  If \( c \) is not "skip" then there exist unique \( H \) and \( r \) such that \( c = H[r] \)
  - "Exist" means progress
  - "Unique" means determinism

Short-Circuit Evaluation
- What if we want to express short-circuit evaluation of \( \land \) ?
  - Define the following contexts, redexes and local reduction rules
    \[ H ::= \ldots | H \land b_2 \]
    \[ r ::= \ldots | \text{true} \land b | \text{false} \land b \]
    \[ <\text{true} \land b, \sigma> \rightarrow <b, \sigma> \]
    \[ <\text{false} \land b, \sigma> \rightarrow <\text{false}, \sigma> \]
  - the local reduction kicks in before \( b_2 \) is evaluated

Contextual Semantics Summary
- One can think of the \( \bullet \) as representing the program counter
- The advancement rules for \( \bullet \) are non trivial
  - At each step the entire command is decomposed
  - This makes contextual semantics inefficient to implement directly
- The major advantage of contextual semantics is that it 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
Real-World Example
- Cobbe and Felleisen, POPL 2005
- Small-step contextual opsem for Java
- Their rule for object field access:
  \[ P \vdash <E[\text{obj.fd}],S> \rightarrow <E[F(fd)],S> \]
  - Where \( F = \text{fields}(S(\text{obj})) \) 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
- \( P \vdash <H[\text{obj.fd}],\sigma> \rightarrow <H[F(fd)],\sigma> \)
  - Where \( F = \text{fields}(\sigma(\text{obj})) \) and \( fd \in \text{dom}(F) \)

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

Lost In Translation 2
- \( <H[\text{obj.fd}],\sigma> \rightarrow <H[F(fd)],\sigma> \)
  - Where \( F = \text{fields}(\sigma(\text{obj})) \) and \( fd \in \text{dom}(F) \)

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

  Which is really just our rule:
  - \( <H[fd],\sigma> \rightarrow <H[\sigma(fd)],\sigma> \) (when \( fd \in L \))

“Sleep On It”

“The Semantics Pillow”

“Learn while you sleep!”

Only $19.95

---

Homework
- Straw Poll
- Homework 2 Out Today
  - Due Thursday, Feb 02
- Read Winskel Chapter 3
- Want an extra opsem review?
  - Natural deduction article
  - Plotkin Chapter 2
- Optional Philosophy of Science article