Strategy

1. Define $\delta \triangleq \Downarrow [d]$
2. For every closed term $e$, prove $C[[e]] \Downarrow = [n] \Rightarrow (d,e) \rightarrow^* (d,n)$
   - Technique: induction on structure of $e$
3. For every closed term $e$, prove $(d,e_1) \rightarrow (d,e_2) \Rightarrow C[[e_1]] \Downarrow = C[[e_2]] \Downarrow$
   - Technique: induction on derivation of $(d,e_1) \rightarrow (d,e_2)$
   - Then, induction on $\#$ of steps $(\rightarrow)$:
     $(d,e) \rightarrow^* (d,n) \Rightarrow C[[e]] \Downarrow = C[[n]] \Downarrow = n$
4. Equivalence of small-step & denotational:
   $C[e] \Downarrow = [n] \Rightarrow (d,e) \rightarrow^* (d,n)$ (closed)

Easy cases

$C[[e_1 \otimes e_2]] \Downarrow x_k \rightarrow n_k = [n] \Rightarrow$
$(d,(e_1 \otimes e_2)(n_k/x_k)) \rightarrow^* (d,n)$
Assume LHS: $C[[e_1]] \Downarrow x_k \rightarrow n_k = [n_1]$, $C[[e_2]] \Downarrow x_k \rightarrow n_k = [n_2]$
Ind.Hyp.: $(d,e_1(n_k/x_k)) \rightarrow^* (d,n_1)$, $(d,e_2(n_k/x_k)) \rightarrow^* (d,n_2)$
$(d,(e_1 \otimes e_2)(n_k/x_k)) \rightarrow^* (d,(n_1 \otimes n_2)(n_k/x_k)) \rightarrow^* (d,n)$

Equivalence

- Have 2-3 different semantics for REC$^*$ language: large-step SOS, small-step SOS, denotational
- Do they describe the same language?
- Winskel Ch. 9.4, 9.6: equivalence of large-step SOS and denotational semantics
  $C[e] \Downarrow \Downarrow = [n] \Rightarrow (d,e) \Downarrow n$
  $\Leftarrow (d,e) \rightarrow^* (d,n)$

Function call

$C[[f(e_1,\ldots,e_m)] \Downarrow x_k \rightarrow n_k = [n] \Rightarrow$
$(d,f(e_1,\ldots,e_m)(n_k/x_k)) \rightarrow^* (d,n)$
Ind. Hyp.: $C[[e]] \Downarrow x_k \rightarrow n_k = [n_1] \Rightarrow$
$(d,e(n_k/x_k)) \rightarrow^* (d,n_1)$
$(d,f(e_1,\ldots,e_m)(n_k/x_k)) \rightarrow^* (d,(f(n_1,\ldots,n_n)))$
Need to show:
$(\pi,\delta)(n_1,\ldots,n_n) = n \Rightarrow e(n_k/x_k) \rightarrow^* n$
So far: have used no properties of $\delta$
Handling recursive fns

- Plan: prove statement holds for function environment $\phi$:
  $$\pi_i (n_1, ..., n_m) = \begin{cases} 
  \{ n \} & \text{if } e_i(n_j/x_j) \rightarrow^* n \\
  \bot & \text{otherwise}
  \end{cases}$$
- Bootstrap to show it holds for real $\delta$
  Need to show:
  $$(\pi, \phi)(n_1, ..., n_m) = \tau \Rightarrow e_i(n_j/x_j) \rightarrow^* n$$
- Follows from defn of $\phi$

Applying induction...

- $C[e_\phi(x_j \rightarrow_n \lambda^{\downarrow K})] = \{ n \} \Rightarrow (d, e_i(n_j/x_j)) \rightarrow^* (d, n)$
- Apply to function body $e_i$:
  $$C[e_\phi(x_j \rightarrow_n \lambda^{\downarrow K})] = \{ n \} \Rightarrow (d, e_i(n_j/x_j)) \rightarrow^* (d, n)$$
- $\tau(\phi)(n_i) = C[e_\phi(x_j \rightarrow_n \lambda^{\downarrow K})] \subseteq \pi_i(n_j)$
- $\phi$ is a prefixed point of $\lambda_\phi, \tau(\phi), ..., \tau_i(\phi)$
- $\delta$ is least prefixed point: $\delta \subseteq \phi$
- $C[e_\phi(x_j \rightarrow_n) = \{ n \} \Rightarrow (d, e_i(n_j/x_j)) \rightarrow^* (d, n)$
  $$\therefore C[e_\phi \delta \phi = \{ n \} \Rightarrow (d, e) \rightarrow^* (d, n)$$

Function call

$$(d, f_i(n_1, ..., n_m)) \rightarrow (d, e_i(n_j/x_j) \lambda^{\downarrow K})$$

Ind. Hyp.:

$$C[e_\phi(x_j \rightarrow_n \lambda^{\downarrow K})] \delta \phi \Rightarrow (d, f(x_1, ..., x_m)) \Rightarrow C[e_\phi(x_j \rightarrow_n \lambda^{\downarrow K})] \delta \phi$$

- Need substitution lemma to complete proof:
  $$C[e_\phi \delta \phi[x \rightarrow n] = C[e_\phi(n/x)] \delta \phi$$
- Proof: structural induction on $e$
  $$\therefore \text{denotational semantics adequately model operational semantics}$$

Comparison

<table>
<thead>
<tr>
<th>Operational</th>
<th>Denotational (fixed-pt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed transitions between syntactic forms</td>
<td>translates syntax into a model</td>
</tr>
<tr>
<td>results are just syntax: easy to define</td>
<td>results are domain elements: harder to define</td>
</tr>
<tr>
<td>easily express simple concurrency and non-determinism (am-step)</td>
<td>non-det.: powerdomains concurrency: scheduling</td>
</tr>
<tr>
<td>termination behavior not obvious</td>
<td>termination behavior implicit in domain</td>
</tr>
<tr>
<td>does not explain compilation</td>
<td>gives insight: e.g., fixed points signal extra pass or back-patch</td>
</tr>
</tbody>
</table>

Upcoming attractions

- We will explore language features and design issues
- Will build on a simple untyped functional language: uF
  - similar to Scheme, Turbak & Gifford FLK
  - will look at typed version later
- Use both operational and denotational (fixed-point) semantics
uF syntax
\[ e ::= n \mid \#t \mid \#f \mid \#u \mid x \mid e_1 \oplus e_2 \mid \text{if } e_0 \text{ } e_1 \text{ } e_2 \mid \text{fn } x \text{ } e \mid \text{rec } x \text{ } e_r \mid \text{first } e \mid \text{rest } e \]
\[ e_r ::= \text{fn } x \text{ } e \mid \text{rec } x \text{ } e_r \]

- Call-by-value, left-to-right eager evaluation (by default)

Operational Semantics
\[ e ::= n \mid \#t \mid \#f \mid \#u \mid \#r \mid x \mid e_1 \oplus e_2 \mid \text{if } e_0 \text{ } e_1 \text{ } e_2 \mid \text{fn } x \text{ } e \mid \text{rec } x \text{ } e_r \]
\[ e_r ::= \text{fn } x \text{ } e \mid \text{rec } x \text{ } e_r \]

- \[ \text{if } e_0 \text{ } e_1 \text{ } e_2 \to e_1 \]
- \[ \text{fn } x \text{ } e \to \text{fn } x \{v/x\} \]
- \[ \text{first } e \to e_1 \]
- \[ \text{rest } e \to e_2 \]

let

- Consider uF+let language
\[ e ::= \ldots \mid \text{let } x = e_1 \text{ in } e_2 \]
\[ C ::= \ldots \mid \text{let } x = [\cdot] \text{ in } e \]
\[ \text{let } x = v \text{ in } e \to e[v/x] \]

- Don’t need \text{let } x = e_1 \text{ in } e_2 \text{ in uF}; instead, write
\[ (\text{fn } x \text{ } e_2) \text{ } e_1 \]

- Define desugaring translation:
\[ \Delta[e] = e' \] transforms uF+let term \( e \) into uF term \( e' \)

let elimination

- Translation defines uF+let in terms of uF
- Definitional semantics for uF+let
- Equivalence of definitional, operational semantics? \[ e \to v \Leftrightarrow \Delta[e] \to \Delta[v] \]