CS 611
Advanced Programming Languages

Andrew Myers
Cornell University

Lecture 16
Operational vs. Denotational Semantics
29 Sep 00

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

$$\mathcal{C}[\![e]\!]\mathcal{D}[\![d]\!]\varnothing = \lfloor n \rfloor \Leftrightarrow (d,e) \Downarrow n$$
$$\Leftrightarrow (d,e) \rightarrow^* (d,n)$$

Strategy

- Define $\delta \triangleq \mathcal{D}[\![d]\!]$
- For every closed term e, prove
  $\mathcal{C}[\![e]\!]\delta\varnothing = \lfloor n \rfloor \Rightarrow (d,e) \rightarrow^* (d,n)$
  - Technique: induction on structure of e
- For every closed term e, prove
  $(d,e_1) \rightarrow (d,e_2) \Rightarrow \mathcal{C}[\![e_1]\!]\delta\varnothing = \mathcal{C}[\![e_2]\!]\delta\varnothing$
  - Technique: induction on derivation of $(d,e_1) \rightarrow (d,e_2)$
  - Then, induction on # of steps ($\rightarrow$):
    $(d,e) \rightarrow^* (d,n) \Rightarrow \mathcal{C}[\![e_1]\!]\delta\varnothing = \mathcal{C}[\![n]\!]\delta\varnothing = n$
- Equivalence of small-step & denotational:
  $\mathcal{C}[\![e]\!]\delta\rho = \lfloor n \rfloor \Leftrightarrow (d,e) \rightarrow^* (d,n)$ (e closed)

$\Rightarrow$

- Show $\mathcal{C}[\![e]\!]\delta\varnothing = \lfloor n \rfloor \Rightarrow (d,e) \rightarrow^* (d,n)$

$\Leftarrow \mathcal{C}[\![e]\!]\delta\{x_k \mapsto n_k{}^{k\in 1..K}\} = \lfloor n \rfloor \Rightarrow (d,e\{n_k/x_k{}^{k\in 1..K}\}) \rightarrow^* (d,n)$ where all free variables of e are in $\{x_1,\ldots x_K\}$

- Base cases:
  $\mathcal{C}[\![n]\!]\delta\{x_k \mapsto n_k\} = \lfloor n \rfloor \Rightarrow (d,\ n\{n_k/x_k\}) \rightarrow^* (d,n)$
  $\mathcal{C}[\![x]\!]\delta\{x_k \mapsto n_k\} = \lfloor n \rfloor \Rightarrow (d,\ x\{n_k/x_k\}) \rightarrow^* (d,n)$

Easy cases

$\mathcal{C}[\![e_1 \oplus e_2]\!]\delta\{x_k \mapsto n_k\} = \lfloor n \rfloor \Rightarrow$
$(d,\ (e_1 \oplus e_2)\{n_k/x_k\}) \rightarrow^* (d,n)$

Assume LHS: $\mathcal{C}[\![e_1]\!]\delta\{x_k \mapsto n_k\} = \lfloor n_1 \rfloor$,
$\mathcal{C}[\![e_2]\!]\delta\{x_k \mapsto n_k\} = \lfloor n_2 \rfloor$

Ind.Hyp.: $(d,e_1\{n_k/x_k\}) \rightarrow^* (d,n_1)$,
$(d,e_2\{n_k/x_k\}) \rightarrow^* (d,n_2)$

$(d,(e_1 \oplus e_2)\{n_k/x_k\}) \rightarrow^* (d,n_1 \oplus e_2\{n_k/x_k\})$
$\rightarrow^* (d,n_1 \oplus n_2) \rightarrow (d,n)$

Function call

$\mathcal{C}[\![f_i(e_1,\ldots,e_{ai})]\!]\delta\{x_k \mapsto n_k\} = \lfloor n \rfloor \Rightarrow$
$(d,\ f_i(e_1,\ldots,e_{ai})\{n_k/x_k\}) \rightarrow^* (d,n)$

Ind. Hyp: $\mathcal{C}[\![e_j]\!]\delta\{x_k \mapsto n_k\} = \lfloor n_j \rfloor^{j\in 1..n}$
$(d,e_j\{n_k/x_k\}) \rightarrow^* (d,n_j)$

$(d,\ f_i(e_1,\ldots,e_{ai})\{n_k/x_k\}) \rightarrow^* (d,\ f_i(n_1,\ldots,n_{ai}))$

Need to show:
$(\pi_i\ \delta)(n_1,\ldots,n_{ai}) = n \Rightarrow e_i\{n_j/x_j{}^{j\in 1..ai}\} \rightarrow^* n$

So far: have used no properties of δ

Handling recursive fns

- Plan : prove statement holds for function environment ϕ:

$$\pi_i\,\phi\,(n_1,\ldots,n_{ai}) = \begin{cases} \lfloor n \rfloor & \text{if } e_i\{n_j/x_j\} \rightarrow^* n \\ \bot & \text{otherwise} \end{cases}$$

- Bootstrap to show it holds for real δ

Need to show:

$(\pi_i\,\phi)(n_1,\ldots,n_{ai})=n \Rightarrow e_i\{n_j/x_{j\in 1..ai}\}\rightarrow^* n$

- Follows from defn of ϕ

Applying induction...

$\mathcal{C}[\![e]\!]\phi\{x_k \mapsto n_{k\in 1..K}\} = \lfloor n \rfloor$

$\Rightarrow (d,e\{n_k/x_{k\in 1..K}\})\rightarrow^*(d,n)$

- Apply to function body e_i:

$\mathcal{C}[\![e_i]\!]\phi\{x_j \mapsto n_{j\in 1..ai}\} = \lfloor n \rfloor$

$\Rightarrow (d, e_i\{n_j/x_j\})\rightarrow^*(d,n)$

- $\mathcal{F}_i(\phi)(n_j)= \mathcal{C}[\![e_i]\!]\phi\{x_j \mapsto n_{j\in 1..ai}\} \sqsubseteq \pi_i\phi(n_j)$
- ϕ is a prefixed point of $\lambda\phi.(\mathcal{F}_1\phi,\ldots,\mathcal{F}_n\phi)$
- δ is least prefixed point: $\delta \sqsubseteq \phi$
- $\mathcal{C}[\![e]\!]\delta\{x_k \mapsto n_k\} = \lfloor n \rfloor \Rightarrow (d,e\{n_k/x_k\})\rightarrow^*(d,n)$

$\therefore\ \mathcal{C}[\![e]\!]\delta\varnothing = \lfloor n \rfloor \Rightarrow (d,e)\rightarrow^*(d,n)$

$\Leftarrow$

- Show $(d,e_1)\rightarrow(d,e_2) \Rightarrow \mathcal{C}[\![e_1]\!]\delta\varnothing = \mathcal{C}[\![e_2]\!]\delta\varnothing$
- Technique: induction on height of derivation of $(d,e_1)\rightarrow(d,e_2)$
- $(d,e_1 \oplus e_2)\rightarrow(d,e'_1 \oplus e_2)$:
  - Ind.Hyp.: $(d,e_1)\rightarrow(d,e'_1), \mathcal{C}[\![e_1]\!]\delta\varnothing = \mathcal{C}[\![e'_1]\!]\delta\varnothing$
  - $\mathcal{C}[\![e_1 \oplus e_2]\!]\delta\varnothing = \mathcal{C}[\![e_1]\!]\delta\varnothing \oplus_\bot \mathcal{C}[\![e_2]\!]\delta\varnothing = \mathcal{C}[\![e'_1]\!]\delta\varnothing \oplus_\bot \mathcal{C}[\![e_2]\!]\delta\varnothing = \mathcal{C}[\![e'_1 \oplus e_2]\!]\delta\varnothing$
- $(d,n_1 \oplus n_2)\rightarrow(d,n)$: trivial
- Interesting case:

$(d, f_i(n_1,\ldots,n_{ai}))\rightarrow(d,e_i\{n_j/x_{j\in 1..ai}\})$

Function call

$(d, f_i(n_1,\ldots,n_{ai}))\rightarrow(d,e_i\{n_j/x_{j\in 1..ai}\})$

Ind.Hyp.:

$\mathcal{C}[\![f_i(n_1,\ldots,n_{ai})]\!]\delta\varnothing = \mathcal{C}[\![e_i\{n_j/x_{j\in 1..ai}\}]\!]\delta\varnothing$?

$= (\pi_i\,\delta)(n_1,\ldots,n_{ai}) = \mathcal{C}[\![e_i]\!]\delta\{x_j \mapsto n_{j\in 1..ai}\}$

- Need *substitution lemma* to complete proof: $\mathcal{C}[\![e]\!]\delta\rho[x \mapsto n] = \mathcal{C}[\![e\{n/x\}]\!]\delta\rho$
- Proof: structural induction on e

$\therefore$ *denotational semantics adequately model operational semantics*

Comparison

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

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 | #t | #f | #u | x | $e_1 \oplus e_2$ | if e_0 e_1 e_2 |
| fn x e | rec x e_r | $\langle e_1, e_2 \rangle$ | first e | rest e

e_r ::= fn x e

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

Operational Semantics

e ::= n | #t | #f | #u | x | $e_1 \oplus e_2$ | if e_0 e_1 e_2 | e_1 e_2 | fn x e | $\langle e_1, e_2 \rangle$ | first e | rest e | rec x e_r

v ::= n | #t | #f | #u | fn x e | $\langle v_1, v_2 \rangle$

if #t e_1 e_2 → e_1 if #f e_1 e_2 → e_2

(fn x e) v → $e\{v/x\}$

first $\langle v_1, v_2 \rangle$→v_1 rest $\langle v_1, v_2 \rangle$→v_2

rec x e_r→$e_r\{$rec x $e_r/x\}$

C ::= $[\cdot]$ | $[\cdot] \oplus e$ | $v \oplus [\cdot]$ | if $[\cdot]$ e_1 e_2 | $[\cdot]$ e | v $[\cdot]$ | $\langle [\cdot], e \rangle$ | $\langle v, [\cdot] \rangle$ | first $[\cdot]$ | rest $[\cdot]$

$C[e]$→$C[e']$ if e→e'

let

- Consider uF+let language
  e ::= ... | let x=e_1 in e_2
  C ::= ... | let x = $[\cdot]$ in e
  let x = v in e → $e\{v/x\}$
- Don't need let x=e_1 in e_2 in uF; instead, write

  (fn x e_2) e_1
- Define desugaring translation:
  $\mathcal{D}[\![e]\!] = e'$ transforms uF+let term e into uF term e'

let elimination

e ::= n | #t | #f | #u | x | $e_1 \oplus e_2$ | if e_0 e_1 e_2 | e_1 e_2 | fn x e | $\langle e_1, e_2 \rangle$ | first e | rest e | rec x e_r

$\mathcal{D}[\![c]\!] = c$

$\mathcal{D}[\![e_1 \oplus e_2]\!] = \mathcal{D}[\![e_1]\!] \oplus \mathcal{D}[\![e_2]\!]$

$\mathcal{D}[\![$if e_0 e_1 $e_2]\!] =$ if $\mathcal{D}[\![e_0]\!]$ $\mathcal{D}[\![e_1]\!]$ $\mathcal{D}[\![e_2]\!]$

$\mathcal{D}[\![e_1$ $e_2]\!] = \mathcal{D}[\![e_1]\!]$ $\mathcal{D}[\![e_2]\!]$

...

$\mathcal{D}[\![$let x=e_1 in $e_2]\!] = (\text{fn } x\ \mathcal{D}[\![e_2]\!])\ \mathcal{D}[\![e_1]\!]$

- Translation defines uF+let in terms of uF
- *Definitional semantics* for uF+let
- Equivalence of definitional, operational semantics? $e \rightarrow^* v \Leftrightarrow \mathcal{D}[\![e]\!] \rightarrow^* \mathcal{D}[\![v]\!]$