CS 611
Advanced Programming Languages

Andrew Myers
Cornell University

Lecture 28

Strong Normalization, Logical relations

1 Nov 00

Soundness for SOS

- Last time: soundness of typing rules for structural operational semantics
- "e is well-typed" $\vdash e : \tau$
- "e does not get stuck": $\forall e' \,.\, e \to^* e' \Rightarrow e \in Value \lor \exists e''.\, e' \to e''$
- Soundness: "e is typable" $\Rightarrow$ "e does not get stuck"
- Three parts to proof:
  - Preservation/Subject reduction $\vdash e : \tau \land e \to e' \Rightarrow \vdash e' : \tau$
  - Progress $\vdash e : \tau \Rightarrow (e \in Value \lor \exists e''.\, e' \to e'')$
  - Induction on number of steps (generic)
- New tool: induction on type derivation
- Real languages much harder...

Strong normalization

- Every program in $\lambda^{\to}$ terminates. Obvious?
  - Reduction can increase size of an expression
  - Reduction can increase number of contained lambda expressions
    $((\lambda\, f : \text{int}\to\text{int} \,.\, (+\ (f\ 0)\ (f\ 1)))\ (\lambda\, y : \text{int} \,.\, (*\ y\ 2)))$
  - Untyped lambda calculus is *not* strongly normalizing
- Idea: size of *types* decreases
- Proof strategy: define set of strongly normalizing expressions SN_τ for every type τ, show by induction on type derivation that expression of type t is a member of SN_τ.
- Method of *logical relations* : relations on expressions indexed by types

Stable expressions

- Problem: induction hypothesis is not strong enough to handle application expressions.
- Strengthen induction hypothesis: define subset of strongly normalizing expressions (the *stable* expressions); show all expressions in $\lambda^{\to}$ are stable.
- Stable expressions are strongly normalizing and result in strongly normalizing expressions when applied to other strongly normalizing expressions.
- T_τ is the set of stable expressions of type τ.
- Define inductively (note $e \Downarrow v \iff e \to^* v$)

$$T_{\text{int}} = \{\, e \mid \vdash e : \text{int} \land e \Downarrow n \,\}$$
$$T_{\tau\to\tau'} = \{\, e \mid \vdash e : \tau\to\tau' \land e \Downarrow v \land (\forall e' \in T_\tau \,.\, (e\ e') \in T_{\tau'})\}$$

- Goal: $\vdash e : \tau \Rightarrow e \in T_\tau$

Strategy

$$T_{\text{int}} = \{\, e \mid \vdash e : \text{int} \land e \Downarrow n \,\}$$
$$T_{\tau\to\tau'} = \{\, e \mid \vdash e : \tau\to\tau' \land e \Downarrow v \land (\forall e' \in T_\tau \,.\, (e\ e') \in T_{\tau'})\,\}$$

Goal: $\vdash e : \tau \Rightarrow e \in T_\tau$ (since $T_\tau \subseteq SN_\tau$)

- Will use induction on type derivation for e
- Problem: rule for typing λ exprs adds to type context Γ. Need to extend goal to allow it to be proved inductively: use substitution operators
- Introduce function γ mapping variables to expressions. $\gamma : \text{Var} \to \text{Exp}$
- γ only substitutes stable expressions of the right type: $\gamma \vDash \Gamma \iff \forall x \in \text{dom}(\Gamma) \,.\, \gamma(x) \in T_{\Gamma(x)}$

Substitution function

- Given any function γ, we can define a related function $\underline{\gamma}$ mapping *Expr*$\to$*Expr* and performing all the substitutions specified by γ:

$$\underline{\gamma}[\![x]\!] = \gamma(x) \quad \text{if } x \in \text{dom}(\gamma)$$
$$\underline{\gamma}[\![x]\!] = x \quad \text{if } x \notin \text{dom}(\gamma)$$
$$\underline{\gamma}[\![n]\!] = n$$
$$\underline{\gamma}[\![e_0\ e_1]\!] = \underline{\gamma}[\![e_0]\!]\ \underline{\gamma}[\![e_1]\!]$$
$$\underline{\gamma}[\![\lambda\, x : \tau \,.\, e]\!] = \lambda\, x : \tau \,.\, \underline{\gamma'}[\![e]\!]$$

γ' is identical to γ except that it does not map x

Refined goal

- Original goal: show all expressions are stable
  $\vdash e : \tau \Rightarrow e \in T_\tau$
- Suppose we can prove the following goal:
  $\Gamma \vdash e : \tau \Rightarrow \forall \gamma \vDash \Gamma \,.\, \gamma[\![e]\!] \in T_\tau$
- Now consider $\Gamma = \varnothing$. The only γ satisfying this type context is the identity mapping. Therefore, our refined goal becomes our original goal.
- Substitution Lemma: $\Gamma \vdash e : \tau \Rightarrow \forall \gamma \vDash \Gamma. \vdash \gamma[\![e]\!] : \tau$
  - Generalization of proof from last class
- Now we turn the inductive crank.

Part I

To show: $\Gamma \vdash e : \tau \Rightarrow \forall \gamma \vDash \Gamma \,.\, \gamma[\![e]\!] \in T_\tau$

- Integers: $\Gamma \vdash n : \text{int} \Rightarrow \forall \gamma \vDash \Gamma \,.\, n \in T_{\text{int}}$
- Variables: $\Gamma \vdash x : \Gamma(x) \Rightarrow \forall \gamma \vDash \Gamma \,.\, \gamma[\![x]\!] \in T_{\Gamma(x)}$
  - if $\gamma \vDash \Gamma$ then $\gamma[\![x]\!] = \gamma(x) \in T_{\Gamma(x)}$ by definition.
- Application: $\Gamma \vdash (e_0\ e_1) : \tau$
  - Consider a γ such that $\gamma \vDash \Gamma$
  - $\gamma[\![e_0\ e_1]\!] = \gamma[\![e_0]\!]\ \gamma[\![e_1]\!]$
  - From type judgement: $\Gamma \vdash e_0 : \tau_1 \rightarrow \tau,\quad \Gamma \vdash e_1 : \tau_1$
  - inductive hypothesis gives us $\gamma[\![e_0]\!]$ and $\gamma[\![e_1]\!]$ are stable; therefore their application is too

  $T_{\tau\rightarrow\tau'} = \{\, e \mid \vdash e : \tau\rightarrow\tau' \wedge e \Downarrow v \wedge (\forall e' \in T_\tau \,.\, (e\ e') \in T_{\tau'})\}$

Part II

To show: $\Gamma \vdash e : \tau \Rightarrow \forall \gamma \vDash \Gamma \,.\, \gamma[\![e]\!] \in T_\tau$

$\Gamma \vdash (\lambda x{:}\tau \,.\, e) : \tau\rightarrow\tau' \Rightarrow \forall \gamma \vDash \Gamma. \gamma[\![\lambda x{:}\tau \,.\, e]\!] \in T_{\tau\rightarrow\tau'}$

- Assume LHS, consider arbitrary $\gamma \vDash \Gamma$

Recall $T_{\tau\rightarrow\tau'} = \{\, e \mid \vdash e : \tau\rightarrow\tau' \wedge e \Downarrow v \wedge (\forall e' \in T_\tau \,.\, (e\ e') \in T_{\tau'})\}$

- Substitution Lemma: $\vdash \gamma[\![(\lambda\, x : \tau \,.\, e)]\!] : \tau\rightarrow\tau'$
- $\gamma[\![\lambda\, x : \tau \,.\, e]\!]$ is already a value so $\Downarrow v$
- Need $\forall e' \in T_\tau \,.\, (\gamma[\![\lambda\, x : \tau \,.\, e]\!]\ e') \in T_{\tau'}$
  - $\gamma[\![\lambda\, x : \tau \,.\, e]\!]\ e' = (\lambda\, x : \tau \,.\, \gamma[\![e]\!])e' = \gamma[\![e]\!]\{e'/x\} = \gamma'[\![e]\!]$ where $\gamma' = \gamma[x \mapsto e']$
  - From typing rule: $\Gamma[x \mapsto \tau] \vdash e' : \tau'$
  - Apply induction hypothesis, instantiate on γ': $\gamma' \vDash \Gamma[x \mapsto \tau] \Rightarrow \gamma'[\![e]\!] \in T_{\tau'} \qquad \gamma' \vDash \Gamma[x \mapsto \tau]\ \ (e' \in T_\tau)$

QED

Agreement

- Would like to know that denotational semantics and operational semantics agree

Operational evaluation

$e \rightarrow e' \rightarrow e'' \rightarrow \ldots \rightarrow v : \tau$

$\mathcal{C}[\![\vdash e : \tau]\!]\rho_0 \in \mathcal{T}[\![\tau]\!]$

Denotational model

Adequacy

- Denotational semantics are *adequate* with respect to operational semantics if:
- Operational evaluation produces one of the values allowed by denotational semantics

  $e \rightarrow^* v \wedge \vdash e : \tau \Rightarrow \mathcal{C}[\![\vdash e : \tau]\!]\rho_0 = \mathcal{C}[\![\vdash v : \tau]\!]\rho_0$
- They agree on observable results: divergence

  $\exists\, v \,.\, e \rightarrow^* v \wedge \vdash e : \tau \Leftrightarrow \mathcal{C}[\![\vdash e : \tau]\!]\rho_0 \neq \bot$
- and also on ground types (e.g. int)

  $e \rightarrow^* v \wedge \vdash e : \text{int} \Leftrightarrow \mathcal{C}[\![\vdash e : \text{int}]\!]\rho_0 = v$

$e \rightarrow^* v \wedge \vdash e : \tau \Leftrightarrow \mathcal{C}[\![\vdash e : \tau]\!]\rho_0 = \mathcal{C}[\![\vdash v : \tau]\!]\rho_0$?

Coming soon: richer types

- Recursive types
- Polymorphic types
- Subtyping
- Objects