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 $\vdash e : \tau \land e \rightarrow e' \Rightarrow \vdash e' : \tau$
 - Progress $\vdash e: \tau \Rightarrow (e \in Value \lor \exists e''. e' \rightarrow e'')$

2

- Induction on number of steps (generic)
- New tool: induction on type derivation
- Real languages much harder...
 CS 611 Fall 00 -- Andrew Myers, Cornell University

Strong normalization

- Every program in λ^{\rightarrow} terminates. Obvious?
 - Reduction can increase size of an expression
 Reduction can increase number of contained lambda expressions
 - $((\lambda f: int \rightarrow int . (+ (f 0) (f 1))) (\lambda y: int . (* y 2)))$
 - 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 λ^{\rightarrow} 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 $\tau.$
- Define inductively (note $e \Downarrow v \iff e \rightarrow^* v$)

$T_{\text{int}} = \{ e \mid \vdash e : \text{int} \land e \Downarrow n \}$ $T_{\tau \to \tau'} = \{ e \mid \vdash e : \tau \to \tau' \land e \Downarrow v$ $\land (\forall e' \in T_{\tau} . (e e') \in T_{\tau}) \}$ • Goal: $\vdash e : \tau \Rightarrow e \in T_{\tau}$ CS 601 Fall TO - Andrew Myers, Formell University

Strategy

$$\begin{split} \mathbf{T}_{\mathrm{int}} &= \{ \, e \mid \vdash e : \mathrm{int} \ \land e \Downarrow n \, \} \\ \mathbf{T}_{\tau \to \tau'} &= \{ \, e \mid \vdash e : \tau \to \tau' \land e \Downarrow v \land (\forall e' \in \mathbf{T}_{\tau} . \, (e \, e') \in \mathbf{T}_{\tau}) \, \} \end{split}$$

$\text{Goal:} \vdash e : \tau \Rightarrow e \in \mathsf{T}_{\tau} \qquad (\text{since } \mathsf{T}_{\tau} \subseteq SN_{\tau})$

- 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. γ : Var \rightarrow Exp
- γ only substitutes stable expressions of the right type: $\gamma \models \Gamma \Leftrightarrow \forall x \in \text{dom}(\Gamma) \cdot \gamma(x) \in T_{\Gamma(x)}$ CS 611 Fall '00 - Andrew Myers, Cornell University 5

Substitution function

• Given any function γ , we can define a related function γ mapping *Expr* \rightarrow *Expr* and performing all the substitutions specified by γ :

$\underline{\gamma}[x] = \gamma(x) \quad \text{if } x \in \operatorname{dom}(\gamma)$ $\underline{\gamma}[x] = x \quad \text{if } x \notin \operatorname{dom}(\gamma)$

$\gamma[n] = n$

 $\gamma [\![e_0 \ e_1]\!] = \gamma [\![e_0]\!] \gamma [\![e_1]\!]$

$$\llbracket \lambda x : \tau \cdot e \rrbracket = \lambda x : \tau \cdot \cancel{\llbracket e \rrbracket}$$

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

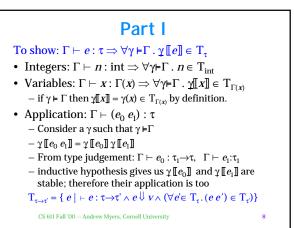
CS 611 Fall '00 -- Andrew Myers, Cornell University

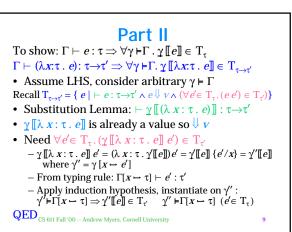
Refined goal

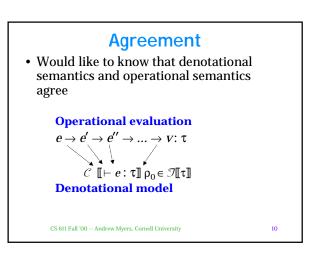
• Original goal: show all expressions are stable

 $\vdash e : \tau \Longrightarrow e \in \mathbf{T}_{\tau}$

- Suppose we can prove the following goal: $\Gamma \vdash e : \tau \Rightarrow \forall \gamma \models \Gamma . \ \gamma \llbracket e \rrbracket \in T_{\tau}$
- Now consider $\Gamma = \emptyset$. The only γ satisfying this type context is the identity mapping. Therefore, our refined goal becomes our original goal.
- Substitution Lemma: Γ⊢ e: τ ⇒ ∀γ⊨Γ. ⊢ γ [[e]] : τ
 Generalization of proof from last class
- Now we turn the inductive crank. CS 611 Fall '00 -- Andrew Myers, Cornell University







Adequacy al semantics are *adeq*

- Denotational semantics are *adequate* with respect to operational semantics if:
- Operational evaluation produces one of the values allowed by denotational semantics
- $e \to^* v \land \vdash e : \tau \implies \mathcal{C}\llbracket \vdash e : \tau \rrbracket \rho_0 = \mathcal{C}\llbracket \vdash v : \tau \rrbracket \rho_0$
- They agree on observable results: divergence
- $\exists v. e \to^* v \land \vdash e : \tau \Leftrightarrow \mathcal{C}\llbracket \vdash e : \tau \rrbracket \rho_0 \neq \bot$
- and also on ground types (e.g. int) $e \rightarrow^* v \land \vdash e : int \Leftrightarrow C[\![\vdash e : int]\!] \rho_0 = v$

 $e \to^{*} v \land \vdash e : \tau \Leftrightarrow \mathcal{C}\llbracket \vdash e : \tau \rrbracket \rho_{0} = \mathcal{C}\llbracket \vdash v : \tau \rrbracket \rho_{0} ?$ CS 611 Fall '00 -- Andrew Myers, Cornell University 11

