Type Safety

February 12, 2001

CS 131: Programming Languages

Review: Formal Semantics

- In the past two classes we have looked at
  - Dynamic Semantics
    - Definition of evaluation of programs
  - Static Semantics
    - Definition of the type system
- How can we say whether we've gotten the *right* dynamic or static semantics?

Dynamic Semantics

- Many, many possible ways to formalize this.
  - Big-step or small-step
  - Stores or substitution
  - SOS or contextual semantics
  - etc.
- Choice largely a matter of convenience.
  - Ease of definition
  - Ease of understanding
  - Ease of use
- May even formalize twice

Example: Arithmetic Again

$$\frac{}{n_1+n_2 \rightarrow n_1 \oplus n_2}$$

$$\frac{e_1 \rightarrow e_1'}{e_1+e_2 \rightarrow e_1'+e_2} \qquad \frac{e_2 \rightarrow e_2'}{e_1+e_2 \rightarrow e_1+e_2'}$$

$$\frac{}{n \Downarrow n}$$

$$\frac{e_1 \Downarrow n_1 \quad e_2 \Downarrow n_2}{e_1+e_2 \Downarrow n_1 \oplus n_2}$$

Equivalence

- Theorem
  - If $e \Downarrow v$ then $e \rightarrow^* v$.
  - If $e \rightarrow e'$ and $e' \Downarrow v$ then $e \Downarrow v$.
  - If $e \rightarrow^* v$ then $e \Downarrow v$.
- Corollary: $e \rightarrow^* v$ iff $e \Downarrow v$.
- Proofs: By induction on *proofs*.

Type Safety

- We introduced a static semantics to prevent run-time errors.
  - Slogan: "Well-typed programs don't go wrong"
- What does type safety mean?
  - How is safety defined?
  - How can we tell if we have the right type system?

Type Soundness

- We say that a type system is *sound* for a small-step operational semantics if evaluation of a well-typed program can never "get stuck"
  - That is, either it evaluates to a value
  - Or, there is an infinite sequence of evaluation steps
  - (Or both, if the semantics is not deterministic)
- This can be broken down into two properties
  - Type Preservation:

    If $\vdash e : t$ and $e \rightarrow e'$ then $\vdash e' : t$
  - Progress:

    $\vdash e : t$ then e is a value or $e \rightarrow e'$ for some e'.

Type Soundness

- What does type soundness tell us?
  - Operations are performed only on values of the right sort
    - E.g.,never going to add two functions.
    - Proves that bad typecasts never occur
    - Means that we don't have to do run-time checks for the types of arguments (hence values need not be self-describing)
- Notes:
  - Type soundness reflects the interactions between the static and dynamic semantics
  - We're proving something about the run-time behavior of *every* well-typed program.