CS152: Programming Languages

Lecture 4 — Proofs About Operational Semantics

Dan Grossman
Spring 2011

This lecture

Carefully consider two proofs using our IMP operational semantics

- First emphasizes need to strengthen an induction hypothesis
  - An art: We will purposely take several wrong turns
- Second shows a property is *preserved* using induction over:
  - Length of execution sequence
  - Inner: (Height of) statement-evaluation derivation (tree)
  - Inner: (Height of) expression-evaluation derivation (tree)

Much of this lecture is in writing the proofs out "live"

On Assignment 1, "the big proof" requires most of the second proof's form *and* strengthening the induction hypothesis appropriately

First proof

Key points from the proof of the barely-interesting fact that **while 1 skip** diverges:

- First carefully state theorem in terms that can be proven by induction
  - In this case on n, the number of steps taken
- Must get induction hypothesis "just right"
  - Not too weak: proof doesn't go through
  - Not too strong: can't prove something false
- Stronger induction hypothesis means implies the original claim
  - Often obvious

Second proof

Key points from the proof of the "no negatives" property:

- Showing a property is *preserved* is about *invariants*, a fundamental software-development concept
- Can define a program property via judgments and prove it holds after every step
- "Inverting assumed derivations" gives you necessary facts for smaller expressions/statements (e.g., the while case)
  - *Inversion* is the technique of saying, "By *assumption*, there is a derivation of X. In *this case* of the proof, such a derivation *must* end with rule Y (no other rule matches). Therefore the hypotheses of that rule must hold here."
  - A common pattern: induction, cases, inversion in each case

Motivation of non-negatives

While "no negatives preserved" boils to down to properties of blue-+ and blue-*, writing out the whole proof ensures our language has no mistakes or bad interactions

- Like a system test for the semantics

The theorem is false if we have:

- Overly flexible rules, e.g.:

$$\frac{}{H\ ;\ c \Downarrow c'}$$

- An "unsafe" language like C:

$$\frac{H(x) = \{c_0, \ldots, c_{n-1}\} \qquad H\ ;\ e \Downarrow c \qquad c \geq n}{H\ ;\ x[e] := e' \rightarrow H'\ ;\ s'}$$

Even more general proofs to come

We *defined* the semantics.

Given our semantics, we established properties of programs and sets of programs

More interesting is having multiple semantics—for what program states are they equivalent? (For what notion of equivalence?)

Or having a more abstract semantics (e.g., a type system) and asking if it is preserved under evaluation. (If e has type τ and e becomes e', does e' have type τ?)

But first a one-lecture detour to "denotational" semantics.