Denotational vs Operational Approaches

COS 441
Princeton University
Fall 2004

Operational Semantics

It is the purpose of these notes to develop a simple and direct method for specifying the semantics of programming languages. Very little is required in the way of mathematical background all that will be involved is “symbol-pushing” of one kind or another …

A Structural Approach to Operational Semantics - Gordon Plotkin


• What does it mean for two functions to be “equal”?  
• Same syntax?  
• Always reduce to the same value given same inputs?  
• One-to-one relationship between their reductions?

Strengths of DS

• Every functions has a precise meaning two functions are equal if they are semantically equal  
• Completely hides details of how the functions compute values  
• Treats all non-terminating functions as equivalents

What’s Hard About DS

• DS deals simply with non-looping functions  
• When we introduce non-terminating functions naïve interpretation of functions as sets is no longer valid  
• Must introduce new theory of domains to account for technicalities

Contextual Equivalence

• Op. Sem. has one natural equivalence on functions that gets at the essences of the natural equivalences

Contexts for $\lambda$-calculus

$e ::= x | \lambda x.e | (e_1 e_2)$  
$v ::= \lambda x.e$  
$C ::= [] | (C e) | (v C)$
**Contexts and Stacks**

Contexts are just a reformulation of the control stack. $(\nu ([\square] e))$ is $(\nu [\square] e) \triangleright (\nu [\square] e) \triangleright \cdot$

We say $e_1$ and $e_2$ are contextually equivalent $(e_1 \equiv_{\text{ctx}} e_2)$ when:

$$\forall k. \ (k,e_1) \mapsto_{c}^* \cdot, v \iff (k,e_2) \mapsto_{c}^* \cdot, v$$

Intuitively swapping one expression for the other doesn’t change the result of any possible evaluation under any context.

---

**Op Sem vs DS**

- Op has simple mathematical foundations
  - Requires sophisticated statements to get at program equality
  - With enough cleverness DS theorems can be restated in terms of Op Sem definitions
- DS has simple notions of program equivalence
  - Requires sophisticated mathematical foundations

---

**When to use DS**

- For languages that do not admit non-terminating computations DS approach is simple to use
- Consider our DS for pictures
  - Elementary ideas with simple notion of equality
- What would a Op Sem. for pictures look like?

---

**When to use Op Sem**

- When dealing with non-terminating computations and don’t care so much about a rich theory of equalities
  - e.g. type safety
- When you care about how the computation takes place
  - Sketching out details of implementation for things like exceptions and the like