Δενοτατιοναλ Σεμαντιχσ
(Denotational Semantics)

Formal Semantics

`What does        y:= f(x) + x        mean?`

- y is assigned the value of f(x) + x
- y becomes a pointer to the result of f(x) + x
- f(x) may or may not have side effects
- statement is undefined if types aren't equivalent
- statement is undefined if types aren't compatible
- etc

• Need formal semantics to make meanings of programs unambiguous.

Utility of Formal Semantics

- Handy for:
  - language design
  - proofs of correctness
  - language implementation
  - reasoning about programs
  - providing a clear specification of behavior

Formal Semantics (continued)

Tennent: " ... a precise specification of the meanings of programs for use by programmers, language designers and implementers, and in the theoretical investigations of language properties.”

- Three major approaches:

1) Denotational: define functions that map syntactic structures into mathematical objects (e.g. numbers, truth values & functions)

(Algebraic) - considered a component of denotational

2) Operational: formal virtual machine description (VDL, H-Graphs)

3) Axiomatic: development of axioms defining meanings of classic statement types. (Dijkstra, Hoare)

Uses

- Denotational: Ashcroft and Wadge argue best use is language *design*. (as opposed to retrofit, as attempted with Ada). Used some for formal verification.
- Operational: Best for implementation description.
- Axiomatic: Most often used for formal verification.

Axiomatic Semantics

Axioms:

a: antecedent
c: consequent

null: {P} skip {P}

assignment: $\{P^x_E\}$ x:= E {P} where P^x_E is the assertion formed by replacing every occurrence of x in P by E.

alternation:
$$\frac{\{P \wedge B\}\ S_1\ \{Q\},\ \{P \wedge \neg B)\ \ S_2\ \ \{Q\}}{\{P\}\ \text{if } B \text{ then } S_1 \text{ else } S_2\ \{Q\}}$$

iteration:
$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \text{while } B \text{ do } S\ \{P \wedge \neg B\}}$$

composition:
$$\frac{\{P_1\}\ S_1\ \{P_2\},\{P_2\}\ S_2\ \{P_3\},\ldots,\{P_n\}\ S_n\ \{P_{n+1}\}}{\{P_1\}\ \text{begin } S_1, S_2, \ldots, S_n \text{ end } \{P_{n+1}\}}$$

rules of inference

Axiomatic Semantics

More axioms:

consequence: $\dfrac{\{P_1\}\ S\ \{Q_1\},\ \ P \vdash P_1,\ Q_1 \vdash Q}{\{P\}\ S\ \{Q\}}$

await: $\dfrac{\{P \wedge B\}\ S\ \{Q\}}{\{P\}\ \text{await } B \text{ then } S\ \{Q\}}$

cobegin: $\dfrac{\{P_1\}\ S1\ \{Q_1\}, \ldots, \{P_n\}\ S_n\ \{Q_n\} \text{ are interference free}}{\{P_1 \wedge \ldots \wedge P_n\}\ \text{cobegin } S_1 // \ldots // S_n \text{ coend } \{Q_1 \wedge \ldots \wedge Q_n\}}$

Uses: Dijkstra's weakest preconditions
Temporal logic

Using Axiomatic Semantics

Prove noninterference in the following:

```
{x=0 and y = 0}
S: cobegin

     s1:  await true then y:= y + 1
    //

     s2:  await true then x:= x + 2
    //

     s3:  await  y>0  then x:= x + y

  coend
{x=3 and y=1}
```

Denotational Semantics

- Assigning denotations to language constructs
- Utilizes *domains* and functions over domains
  - domains are sets with properties that allow us to deal with questions regarding
    - recursive definitions of functions (over domains)
    - recursive definitions of domains

Mathematical recursion

e.g. consider (recursive function over domain)

f: Num $\rightarrow$ Num -- f: maps numbers into numbers

Candidates f's

- Two candidate "defining" functions for f:

  (i) fx = (fx) + 1

  (ii) fx = fx

- Assuming Num = {0,1,2,....}, there is no f for (i) and every f satisfies (ii).

- In contrast:

  (iii) fx = (x=0) $\rightarrow$ 1, x * f(x-1)

  uniquely defines f as factorial

Scott’s Theory

- Scott's (1969) theory of domains ensures every definition is good by:
  - requiring all domains to have an "implicit structure." This requirement guarantees that all equations (e.g. i, ii and iii) have at least one solution.
  - providing direction, using implicit structure, for choosing an "intended" solution from the solutions guaranteed by (a).
    - based on lattices and fixed point theory.

- e.g. Num consists of 0, 1, 2, ... and *undefined*

  - $Num_{\perp}$ is called a *lifted domain*

Defining Moment

- Thus,

  (i) and (ii) define f to be undefined and (iii) defines f as fx = x! if x=0, 1, 2, ...

  and f undefined = undefined
- Using $\perp$ as a value is an alternative to using partial functions.
- With $\perp$, all elements in domain have a value.
  - e.g. f undefined = undefined

- Scott's theory applies as well to recursive definitions of domains.
  - e.g. lists defined in terms of lists

On Defining a Language's Denotational Semantics

Three components:

- Abstract syntax (syntactic domain)
  - list of syntactic categories
  - list of syntactic clauses (a mapping onto *immediate constituents*)
- Semantic Domain (Semantic Algebras)
  - domain equations: provide framework for defining denotations
  - sets that are used as value spaces in PL semantics
- Semantic functions
  - functions that define denotation of constructs
  - semantic clauses

Terms

- $\lambda x.e$: Church's lambda notation (seen before)
- $\underline{\lambda}x.e : A_\perp \rightarrow B_\perp$ ::= $(\underline{\lambda}x.e)\perp = \perp$

  $(\underline{\lambda}x.e)a = [a/x]e$ for $a \neq \perp$

  ^-"proper element"
  - $\underline{\lambda}$x.e is e.g. of a *strict* operation
  - non-strict operations allow $\perp$ to be mapped to proper elements
- (let x = e_1 in e_2) is a syntactic substitute for $(\underline{\lambda}x.e_2)e_1$
- **diverge**: statement that goes into an infinite loop

More Terms

- $x \rightarrow e_1 \mid e_2$: syntactic form for conditional
  e.g. **C[**If B THEN C_1 ELSE C_2**]** = $\underline{\lambda}$s.**B[**B**]**s $\rightarrow$ **C[**C_1**]**s $\mid$ **C[**C_2**]**s
- Expressions in mini-language assumed to have *no* side effects.
  - e.g. no reads in expressions.
- $[i \rightarrow n]s$ is a function updating expression

  $([i \rightarrow n]s)(i) = n$

  $([i \rightarrow n]s)(j) = s(j) \quad \forall\, j \neq i$

  - useful for reflecting effects of updating the i^{th} component of a store: i^{th} component changes; rest stays the same
  - update's signature: Id x Nat x Store $\rightarrow$ Store

Even More Terms

- Interpretation of:

  P**[**C.**]** = λn**.** let s = (update **[**A**]** n newstore) in
  let s' = **C[**C**]**s in (access **[**Z**]**s')

  - input number is associated with identifier **[**A**]** in a new store
  - then program body is evaluated
  - then answer is extracted from store at **[**Z**]**

  (program mapping: Nat $\rightarrow$ $Nat_{\perp}$ **--** $\perp$ is possible because **diverge** is possible)
- Clauses for **C** are all strict in use of store
- **E** does not modify store; expression evaluation order is not specified
  - e.g. **E[**E_1**]**s plus **E[**E_2**]**s
- Same for Booleans

A Small Imperative Language

- Abstract Syntax

$P \in$ Program

$C \in$ Command

$E \in$ Expression

$B \in$ Boolean-expr

$N \in$ Numeral

Syntactic categories

P ::= C**.**

C ::= $C_1;C_2$ | if B then C | if B then C_1 else C_2 | I:=E | **diverge**

E ::= $E_1 + E_2$ | I | N

B ::= $E_1 = E_2$ | $\neg B$

Syntactic clauses

A Small Imperative Language (cont)

- Semantic domain

. . .

- Semantic Functions

functions clauses

P: Program $\rightarrow$ *Nat* $\rightarrow$ $Nat_\perp$

P[C**.]** = λn**.** let s = (update**[**A**]** n newstore) in
let s' = **C[**C**]**s in (access**[**Z**]** s')

C: Command $\rightarrow$ $Store_\perp$ $\rightarrow$ $Store_\perp$

C[$C_1;C_2$**]** = $\underline{\lambda}$s**. C[**C_2**]** (**C[**C_1**]**s)

C[if B then C**]** = $\underline{\lambda}$s**. B[**B**]**s $\rightarrow$ **C[**C**]**s | s

C[if B then C_1 else C_2**]** = $\underline{\lambda}$s**. B[**B**]**s $\rightarrow$ **C[**C_1**]**s | **C[**C_2**]**s

C[I:=E**]** = $\underline{\lambda}$s**.** *update* **[**I**]** (**E[**E**]**s) s

C[diverge] = $\underline{\lambda}$s**.** $\perp$

A Small Imperative Language (cont)

- Semantic Functions (cont)

  E: Expression $\rightarrow$ *Store* $\rightarrow$ *Nat*

  E[E_1+E_2**]** = λs**. E[**E_1**]**s *plus* **E[**E_2**]**s

  E[I**]** = λs**.** *access* **[**I**]** s

  E[N**]** = λs**. N**[N]

  note

  B: Boolean-expr $\rightarrow$ *Store* $\rightarrow$ *Tr*

  B[E_1=E_2**]** = λs**. E[**E_1**]**s *equals* **E[**E_2**]**s

  B[$\neg$B**]** = λs**.** *not* **B[**B**]**s

  N: Numeral $\rightarrow$ *Nat* (omitted)

Semantic Domain

- Truth Values

  Domain $t \in Tr = \mathbf{B}$

  Operations

  true, *false*: *Tr*

  not: *Tr* $\rightarrow$ *Tr*
- Identifiers

  Domain $i \in Id$ = Identifier
- Natural numbers

  Domain $n \in Nat = \mathbf{N}$

  Operations

  zero, *one*, …: *Nat*

  Zero order function

  plus: Nat x Nat $\rightarrow$ *Nat*

  equals: Nat x Nat $\rightarrow$ *Tr*

Semantic Domain (cont)

- Store

  Domain $s \in Store = Id \rightarrow Nat$

  Operations

  $newstore$: $Store$

  $newstore = \lambda i.\ zero$

  $access$: $Id \rightarrow Store \rightarrow Nat$

  $access = \lambda i.\ \lambda s.\ s(i)$

  $update$: $Id \rightarrow Nat \rightarrow Store \rightarrow Store$

  $update = \lambda i.\ \lambda n.\ \lambda s.\ [i \rightarrow n]s$