Introduction

to Programming

Language

Semantics

Ohe Kammav

1st Joint Category Theory &
Computer Science Seminar

Cambridge Saturday 17th Nov
2012
Warning:

Careful! Loose math ahead!
Goals

The audience will know the various application domains of CT in semantics.

Subjects
- The audience will know the various use of CT in shaping generational semantics.
- The audience will know the use of CT in domain theory.
- The audience will know the use of CT in specifying language structures.
- The audience will know the use of CT in type theory.

PL's → "easy" part, syntax. (Not really easy, but well-said)

Example syntax

```
\( M : = \text{true} | \text{false} | n | \text{if } n \lt k \text{ then } M \text{ else } N \)
```

```
\( \text{let } c \text{ in } c \)
```

```
\( \text{let } x \text{ in } c \)
```

```
\( \text{let } x \text{ in } c \)
```

What does it mean?

1. Fundamental way: show the box it compiles!
2. Write a compiler/interpreter. Meaning = how we run the box.

Problems:

The meaning becomes GCC running on Ubuntu 14.04 on an Intel x86 processor...

```
\( \text{not } x86 \text{ or } \text{not } \text{let } x \text{ in } c \)
```

Complete implementations are complicated - we must be creative.
Instead: 

\[ \langle \text{Program, Conf} \rangle \rightarrow \langle \text{Program, Conf} \rangle \]

Near Finally: Define a relation \( \rightarrow \) over configurations \( \times \) progress inductively over the syntax:

\[ \langle 1, \text{z+2, Conf} \rangle \rightarrow \langle 1, 0 \rangle \]

\[ \langle \text{if true then } M_1 \text{ else } M_2, C \rangle \rightarrow \langle M_1, C \rangle \]

\[ \text{if false } \rightarrow \langle M_2, C \rangle \]

\[ \text{case } C = \text{ heaps } = \text{ loc } \rightarrow \] with finite support.

\[ \langle 0, C \rangle \]

\[ \langle l := n, C \rangle \rightarrow \langle n, C[l \rightarrow n] \rangle \]

\[ \langle l !, C \rangle \rightarrow \langle C(l), C \rangle \]

\[ \langle M, C \rangle \rightarrow \langle M', C \rangle \]

\[ \langle l := M, C \rangle \rightarrow \langle l := M, C' \rangle \]

(As the de facto method in the PL Community. Not yet in the industry.)
Statements and types

Programs can get stuck:

\[ \text{true} = 5 \quad \text{or} \quad \text{error} \]

E.g., segmentation fault, program crash, blue screen (of death)

Introduce type systems:

\[ \text{vars} \rightarrow \text{types} \quad \text{types: A ::= bool \mid Nat \mid \text{A} \rightarrow \text{B}} \]

\[ \Gamma \vdash M : A \quad \text{Loc} \]

\[ \Gamma \vdash x : A \quad \text{Loc} \]

\[ \Gamma \vdash M, M_2 : \text{int} \quad \Gamma \vdash M_1, M_2 : \text{int} \]

\[ \Gamma \vdash M + M_2 : \text{int} \quad \Gamma \vdash M, M_2 : \text{bool} \]

\[ \Gamma, x : A \vdash M : B \quad \Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A \]

\[ \Gamma \vdash \lambda x. M : A \rightarrow B \quad \Gamma \vdash M(N) : B \]

Etc.

Predicate if \( \Gamma \vdash M : A \) and \( C \rightarrow x.C \) is total then:

\[ \begin{align*}
&\text{when } A = \text{bool} \quad \langle M, C \rangle \rightarrow x.C, \xi.C \\
&\text{when } A = \text{int} \quad \langle M, C \rangle \rightarrow x.C, \xi.C \\
&\text{when } A = A \rightarrow B \quad \langle M, C \rangle \rightarrow x.C \\
\end{align*} \]
Great! We sort just like operational semantics are we're let.

well, not exactly...

E.g. equivalence

\[
\begin{align*}
\text{temp} &= x_i \\
& \text{if} \quad x &= x_i \text{XOR} y_i \\
& \text{then}
\end{align*}
\]

\[
\begin{align*}
x &= y_i \\
& \text{if} \quad y &= x_i \text{XOR} y_i \\
y &= \text{temp} \\
& \text{then}
\end{align*}
\]

Name: VC: \( \langle \text{LHS}, C \rangle \rightarrow^* \langle n, C \rangle \leftrightarrow \langle \text{RHS}, C \rangle \rightarrow^* \langle n, C \rangle \)

No... LHS uses or an extra memory location.

So add: temp = 0 temp = 0 at the end.

Useful: for all CE = J

define operational observation contexts:

\[
C ::= \begin{cases} 
M + C & \text{if } C \text{ th. } m \text{ else } M \\
M & C \text{ else } M \\
m & M \text{ else } C \\
C = M \\
M : = C \cdot M_n \cdot M_j C \cdot (x, C) \cdot C(M) & M(C) 
\end{cases}
\]

extend types: \( \Lambda \vdash A \rightarrow \Sigma \cdot B \)

Observational

Contextual equivalence

\( m \equiv m \) for all contexts CE = J and \( \mu \) conf C: \( \langle M, J, C \rangle \rightarrow^* \langle b, C \rangle \rightarrow^* \)

How to prove LHs = LHs? Similar same as reduction for all possible interactions.

Very sensitive to language changes, e.g. if we add the ability to run in parallel.

CE\( \times \) Parallel; ie \( (\text{temp} = 1, -) \)

meaning of each program phrase depends heavily on other phases.
Denotational semantics

Assign meaning to each term:
  \( \forall x. \text{true} := 1 \)
  \( \forall x. \text{false} := 0 \)
  \( \forall x. \text{BEEF} := 48,879 \)

Compositionally:
  \( \llbracket (x+5) \times 7 \rrbracket := \llbracket (x+5) \rrbracket \times 7 \)

of course, only for well-typed! \( \llbracket \text{the} := 5 \rrbracket = ? \)

So first we define semantics for types in the

- \( \llbracket \text{bool} \rrbracket := \{ T, F \} \quad \llbracket \text{int} \rrbracket := \mathbb{Z} \quad \llbracket \text{loc} \rrbracket \ (\text{eg.} \ := \ 2^{64}) \)

we intent:

\( \forall A \rightarrow B \rrbracket := [A] \rightarrow [B] \quad \llbracket \Gamma \vdash M : A \rrbracket := \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket \)

and indeed we can interpret:

\[ M_{\text{true}} \]
\[ M_{\text{false}} \]

but what about \( \llbracket M := N \rrbracket \)?

instant pass state around:
 \( \llbracket \text{loc} \rrbracket := \text{Int} \)

define \( \llbracket \text{int} \rrbracket := \text{Int} \rightarrow \text{Int} \) with finite support.

and thus \( \llbracket \Gamma \vdash M : A \rrbracket = \llbracket \Gamma \rrbracket \times [A] \rightarrow \llbracket \Gamma \rrbracket \rightarrow \llbracket [A] \rrbracket \)

Similarly:

\( \llbracket A \rightarrow B \rrbracket := \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket \)

and now we can give semantics:

\( \llbracket (\Gamma) \rrbracket (\pi) = \langle n, n' \rangle \)

\( \llbracket \text{true} \rrbracket (\pi) = \langle n, n' \rangle \)

\( \llbracket \text{false} \rrbracket (\pi) = \langle n, n' \rangle \)

\( \llbracket \text{BEEF} \rrbracket (\pi) = \langle n, n' \rangle \)

\( \llbracket (x+5) \times 7 \rrbracket (\pi) = \langle n, n' \rangle \)

\( \llbracket (x+5) \rrbracket (\pi) = \langle n, n' \rangle \)

\( \llbracket 0 \rrbracket (\pi) = \langle n, n' \rangle \)

\( \llbracket 1 \rrbracket (\pi) = \langle n, n' \rangle \)

\( \llbracket \text{the} := 5 \rrbracket (\pi) = \langle n, n' \rangle \)

but now:

\( \llbracket M_{\text{loc}} \rrbracket (\pi, n) := (n, n') \text{ if } \pi \rightarrow n \)

\( \llbracket M_{\text{int}} \rrbracket (\pi, n) = (\pi, n') \)

\( \llbracket M_{\text{true}} \rrbracket (\pi, n) = (\pi, n') \)

\( \llbracket M_{\text{false}} \rrbracket (\pi, n) = (\pi, n') \)

\( \llbracket M_{\text{BEEF}} \rrbracket (\pi, n) = (\pi, n') \)

\( \llbracket (x+5) \rrbracket (\pi, n) = (\pi, n') \)

\( \llbracket (x+5) \times 7 \rrbracket (\pi, n) = (\pi, n') \)

\( \llbracket 0 \rrbracket (\pi, n) = (\pi, n') \)

\( \llbracket 1 \rrbracket (\pi, n) = (\pi, n') \)

\( \llbracket \text{the} := 5 \rrbracket (\pi, n) = (\pi, n') \)
Theorem (Soundness of the den. semantics):

if $M \vdash M : A$ and $\langle M, v \rangle \rightarrow^{*} \langle V, v' \rangle$ then $[M](v)(v') = (\langle V, v' \rangle, v')$

Proof: By induction on terms, but strengthening the hypo:

Given $\Gamma \vdash M : A$ and $\sigma$ for all $x \in \text{dom} \sigma$, $\sigma(x)$ is a value $V : A$
and $[M]_{\Gamma} M_\sigma = \Gamma [M_{\sigma[x]}]$. Define $\langle \sigma \rangle := \langle \sigma(x) \rangle_{x \in \text{dom} \sigma}$

Then the new hypo is:

if $\Gamma \vdash M : A$, $\sigma$ a value subst for $\Gamma$ and $\langle M_\sigma, v \rangle \rightarrow^{*} \langle V, v' \rangle$
then $[M]_{\Gamma} \sigma \langle v \rangle = (\langle V, v' \rangle, v')$

(by induction on $\rightarrow^{*}$)

(Conclusion: determinacy ...)

Theorem (Equivalence) if $[M] = [M']$ then $M \equiv M'$.

Example (as before)

Proof: Using logical relations.

Let $\text{Terms}_A := \{ \Gamma \vdash M : A \}$, $\text{Values}_A := \{ \Gamma \vdash V : A \}$

We define relations $R_A \subseteq \Pi A \times \text{Values}_A / \equiv$, $R_A^{\equiv} \subseteq (A \times A)^{\equiv} \times \text{Terms}_A / \equiv$
\[ R_{\text{val}} := \{ \langle 0, \text{false} \rangle, \langle 1, \text{true} \rangle \} \]

\[ R_{\text{int}}^{\text{val}} := \{ (n, \text{cn}) \mid n \in \mathbb{Z} \} \]

\[ R_{\text{int}}^{\text{val}} := \{ (f, [M]) \mid \text{for all } (a, N) \in R_{\text{int}}^{\text{val}}, (f(a), [M(a)]) \in R_{\text{comp}} \} \]

\[ R_{\text{int}}^{\text{comp}} := \{ (\nu, (a, \overline{\nu}), [N]) \mid \text{for all } \nu, \lambda \cdot < M, N > \rightarrow^* (\nu, \overline{\nu}) \text{ and } (a, N) \in R_{\text{int}}^{\text{val}} \} \]

\[ R_{\text{int}}^{\text{val}} := \{ \langle \sigma, \sigma \rangle \mid \text{for all } \sigma \text{ is a substitution for } M \text{ and for all x-domain } (\sigma(x), \sigma(x)) \in R_{\text{comp}} \} \]

**Basic lemma:** if \( \Gamma \vdash M : A \) and \( (r, \sigma) \in R_{\text{int}}^{\text{val}} \), then \( [M](r), [M\sigma] \in R_{\text{comp}} \)

**Proof:** by induction on \( \Gamma \vdash M : A \), for example:

**for \( \Gamma \vdash \text{int} \quad [M] = n \cdot < a, n > \) \( \quad \text{it } (n, \sigma) \in R_{\text{int}}^{\text{val}} \), and \( \text{well-defined} \)

\( \quad \text{then indeed } \quad (a, [M]) \in R_{\text{int}}^{\text{val}} \)

**More interestingly for** \( \Gamma \vdash M : A \rightarrow B \\ \Gamma \vdash N : A \\ \text{true } (r, \sigma) \in R_{\text{int}}^{\text{val}}) \quad \text{then } \quad [M\sigma](r), [M\sigma](r) \in R_{\text{comp}} \quad (\nu, [M\sigma]) \in R_{\text{comp}} \quad (\nu, [M\sigma]) \in R_{\text{comp}} \)

\( \quad \text{take any } N \) \( \text{then } \quad [M\sigma](r)(N) = f, N \) \( \text{and we have } \quad [M\sigma], \rightarrow^{*} (f, N) \) \( \text{and } (f, [V]) \)

\( \quad \text{for } N \text{ we then have } \quad [M\sigma](r)(N) = (a, N \nu) \text{ and we have } [M\sigma], \rightarrow^{*} (V, a, N \nu) \) \( \text{and } (a, [V]) \)

\( \quad \text{as we also} \quad \text{as } \text{we have } \quad f_{\nu}, [V_{\text{fun}}(N)] \in R_{\text{int}}^{\text{val}}, \text{fun } \text{for } \Gamma \vdash M : A \in R_{\text{comp}} \quad (a, [V_{\text{fun}}(N)]) \in R_{\text{comp}} \text{, but } \text{here:} \)

\( \quad \text{for all } n: \quad \left( [M \rightarrow^{*} N]_{\text{fun}}(a, N) \rightarrow^{*} (V_{\text{fun}}(N), N \nu) \right) \quad \text{and we have} \quad V_{\text{fun}}(V) = (M \rightarrow^{*} N) \quad \text{here:} \)

\( \quad [M(N)]_{\text{int}}(r), [M(N)]_{\text{int}}(r) \in R_{\text{comp}} \quad (f_{\nu}, [V_{\text{fun}}(V)]) \in R_{\text{comp}} \)
Probably no time, but also:

\[
\Gamma \vdash \text{loc : } \text{int} \quad \Gamma \vdash \text{int : } \text{int}
\]

\[
\Gamma \vdash \text{loc : } \text{int} \quad \Gamma \vdash \text{int : } \text{int}
\]

take \( (s, \delta) \in R^\text{comp} \) and any \( \nu \in N \).

let \( \text{loc} : \text{int} \overset{\nu}{\rightarrow} \delta : \text{int} \), \( \nu' \)

let \( \text{loc} : \text{int} \overset{\nu}{\rightarrow} \delta : \text{int} \), \( \nu' \)

hence: \( (\text{loc} : \text{int} \overset{\nu}{\rightarrow} \delta : \text{int}, \nu') \rightarrow (\text{loc} : \text{int} \overset{\nu}{\rightarrow} \delta : \text{int}, \nu') \)

hence: recall:

\[
\begin{align*}
\Gamma \vdash \text{loc : } \text{int} \overset{\nu}{\rightarrow} \delta : \text{int} \\
\Gamma \vdash \text{loc : } \text{int} \overset{\nu}{\rightarrow} \delta : \text{int} \\
\end{align*}
\]

so we indeed have:

\[
(\Gamma M (\nu), \Gamma M (\nu)) \in R^\text{comp}
\]

And now we can prove adequacy:

assume \( \Gamma M = \Gamma M' \) for any \( \Gamma \vdash \text{C} \) and \( (\Gamma M, \nu) \rightarrow^* (\nu, \nu') \).

By compositionality \( \Gamma \text{C} M = \Gamma \text{C} M' \) so by the basic lemma

we have:

\[
(\Gamma \text{C} M (\nu), \Gamma \text{C} M (\nu)) \in R^\text{comp}
\]

By soundness, \( \Gamma \text{C} M (\nu) (\nu) = (\Gamma \text{C} M (\nu), \nu') \) hence by \( R^\text{comp} \) 3 def:

\[
(\Gamma \text{C} M (\nu), \nu) \rightarrow^{*} (\nu, \nu') \quad \text{and we have adequacy.}
\]

So now we can continue the 'meaning of programs' by just calculating \( \Gamma M \), as

the adequacy theorem guarantees correct, semantics-comparing with all the other contexts,

But what would happen if we changed the language? What would change, what would

stay the same? This is where CT enters the arena.
For example, we replace memory accesses with non-determinism. We have an operation: $AK(M, N)$ that magically chooses between doing $M$ and doing $N$. Operationally, $AK(M, N) \to M$ and $AK(M, N) \to N$ (each conts) a relation and becomes finite. Now $I \to M; A$ becomes $[I] : I \to P^\times(\#A)$ the non-empty powerset.

and similarly the function space $[I \to A] = I \to A \to P^\times(\#B)$. 

$$\llbracket AK(M, N) \rrbracket_{fs}(r) := \llbracket M \rrbracket_{fs}(r) \cup \llbracket N \rrbracket_{fs}(r)$$

The logical relation then becomes: $\llbracket R^\times \rrbracket_{fs}(r) := \{(X, [M]) | \forall x : M \to \nu, (a, e) \in R^\times \}$

But of course, all the proofs need to be reiterated! And we have other languages: Exceptions, IP, and combinations of them! $\Sigma = 32$ away!

So categorically: we define a model of a language as we need:

A category $C$ which has finite products, a sum $1+1$,

a strong monad $T : C \to C$ and $C$ has kleisli exponentials.

and interpretations:

And maybe: $\llbracket \text{int} \rrbracket$, $\llbracket \text{loc} \rrbracket$, $\llbracket \text{IL} \rrbracket$, $\llbracket \text{IL} \rrbracket$

For arrowiness and for the effects, e.g.,

For example: for memory, $TA := (A \times A)$ with $\mid := 0$ and $\| := 0$ as before for $\eta$, $TA := P^\times(\#A)$

For exceptions: $TA := A + E$ with $[\text{raise}]_{M} := \text{ins}[M]$

The rest of the semantics is standard, with; for example:

$$\llbracket \text{Bool} \rrbracket := \{0, 1\} \quad \llbracket \text{Int} \rrbracket := \mathbb{Z} \quad \llbracket \text{M} \rrbracket := \{I \to TA\} \quad \llbracket \text{M} \rrbracket := \{I \to \#\llbracket TA \rrbracket\}$$

Weiski arrows

$\llbracket r : M \rrbracket := \llbracket r : M \rrbracket \quad \llbracket r : M \rrbracket := \llbracket r : M \rrbracket \to \llbracket TA \rrbracket$

Weiski composition

etc.
objects: types, \( A \), with finite products.

morphisms: sequence of equivalence classes:

\[
\begin{align*}
\pi' \circ \pi & \to \tau \pi \cdot i \\
\pi' @ \cdot \pi & \equiv i 
\end{align*}
\]

with values as below:

\[
\begin{align*}
\tau & \equiv [V_j]_{i \in F} \\
\pi' & \equiv [V_j]_{i \in F} \circ [V_i]_{i \in F} \\
\pi & \equiv [V_j]_{i \in F} \cdot [V_i]_{i \in F} \\
\end{align*}
\]

Composition is given by substitution.

monad:

\[
\begin{align*}
\tau A & := (1 \to A) \\
\eta A & := a : A \mapsto a, a : 1 \to A \\
\mu & := m : 1 \to (1 \to A) \\
\end{align*}
\]

str: \( a : A, m : 1 \to B \mapsto \lambda x. (a, (\pi \circ m)(x)) : 1 \to A \times B \)

reasonable operational semantics will now turn into a model, and we will have a model \( \text{Syn} \).

We then construct another model:

\[
\begin{array}{c}
\xymatrix{L \ar[r] & \text{Pred} \\
\downarrow & \\
\text{Set} \times \text{Syn} \ar[r]^-{\times \text{Id}(\text{Syn})} & \text{Set}
}
\end{array}
\]

\( L \) is the category of logical relations, and we can construct a model in \( L \)

by specifying:

\( \text{int} \) := \{ ([\text{fl}], [\text{en}]) \}

\( \text{loc} \) := \{ \text{fl}, \text{en} \}

and lifting the monad structure \( T \) to some \( R^\text{conf} \subseteq \tau \text{fl} \times \text{en} \times A \)

If we give such a model we automatically get a model \( L \), and it being a model is precisely a restatement of the basic lemma.

The fact that \( L \) has all the required structure follows from Joachim's abstract nonsense (e.g., \( T \) being a bi-libration or \( L \) being arising out of a factorisation system of \( \text{Set} \)).
And the story continues...

For some languages, $c = \text{Set}$ is not enough, and we have to choose a suitable category different from $\text{Set}$. Let’s talk about different categories of domains, which we need to model recursion (while loops, for example) and algebraic datatypes.

If we want to model locality (local memory for example), we need to switch to richer functor categories $c = \text{Set}^I$ on the category of nominal sets.

If we want to model concurrency, we need even a category of event structures, etc. (domains and event structures also merit CT inside themselves, for other reasons...)

The story is far from over... for example, full abstraction in game semantics.

Another way that we must mention is type languages. Syntax is used to reason about our programs. As minimally as possible, we can make our programs easier to understand.

This is the idea behind monads in a PL. Dominic will talk about that, hopefully.

To summarize:

- We focused on operational denotational semantics, covering typed lambda calculus, denotational semantics, adequacy, logical relations.
- We’ve seen how CT helps the Meta-theory by giving a uniform notion of semantic structure.
Recommended reading

Probably covers all topics:


Operational semantics


http://homepages.inf.ed.ac.uk/gdp/publications/SOS.ps

Perhaps more contemporary * Cambridge CS Tripos course, e.g.:

http://www.cl.cam.ac.uk/teaching/1112/Semantics/

And the recommended reading:

  Out of print, but available on the web at

http://www.scss.tcd.ie/Matthew.Hennessy/slexternal/reading.php

Denotational semantics

See Winskel’s book above, but also the CS Tripos course “Denotational Semantics”

http://www.cl.cam.ac.uk/teaching/1112/DenotSem/

The recommended reading from it:

Categorical models of computation

I’m not sure what’s the best place to read about categorical models. You can try the Part III course here:

http://www.cl.cam.ac.uk/teaching/1213/L24/

And chase the recommended reading...

It runs on Lent term, so perhaps you want to attend?