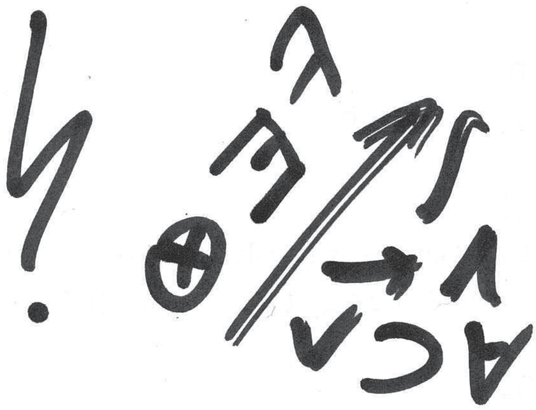
Introduction to Programming Language Semantics

Ohad Kammar

1st Joint Category Theory & Computer Science Seminar

Cambridge Sunday 18th Nov 2012

Warning:



Careful! Loose math ahead!

Goals

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

Subgoals
- The audience will know the ~~previous~~ use of CT in shaping denotational semantics.
- ~~The audience will know the use of CT in domain theory.~~
- The audience will know the use of CT in programming language structures.
- ~~The audience will know the use of CT in tops theory in CT???~~

PL's → "easy" part, syntax. (Not really easy, but well-studied and famous, make it into commercial use)

example syntax

$\cancel{E ::= E}$

$M ::= x \mid true \mid false \mid n \mid + \mid < \mid l \mid \text{if } M \text{ then } M' \text{ else } M''$

$\langle M_1, M_2 \rangle$

$M+M' \mid M<M'$

eg 64-bit addresses such as 0xBEEF

$\mid M := M \mid M; M' \mid \lambda x.M \mid M(M)$

$\cancel{C := C}$
$\cancel{C; C}$
$\lambda x. C$

What does it mean?

(I) - fundamental way: show me how it computes!

(i) write a compiler / interpreter. Meaning = how machine behave.

Problems:

The meaning becomes "gcc running on Ubuntu 11.04 on a 32 bit x86 intel machine ... on Wednesday 24th October / Sunday 18th November ...

(Not great at all).

Complete implementations are complicated - we want to understand.

Instead: ~~structural~~ o

~~operational~~ semantic

$\langle \text{Program}, \text{Conf} \rangle \longrightarrow \langle \text{Program}, \text{Conf} \rangle$

~~More~~ Formally: Define a relation $\longrightarrow$ over Configurations × Programs inductively over the syntax:

~~$\langle n_1 + n_2$~~ $\langle 3+7, \overset{c}{\text{conf}} \rangle \longrightarrow \langle 7, c \rangle$

$\langle \text{if true then } M_1 \text{ else } M_2, c \rangle \longrightarrow \langle M_1, c \rangle$

$\langle \text{if false} \quad " \quad M_1 \quad " \quad " \rangle \longrightarrow \langle M_2, c \rangle$

Conf: c = heaps = $\text{Loc} \longrightarrow \mathbb{Z}$ with finite support.

$\langle M, c \rangle$

$\langle l := n, c \rangle \longrightarrow \langle n, c[l \mapsto n] \rangle \qquad \langle l!, c \rangle \longrightarrow \langle c(l), c \rangle$

$$\frac{\langle M, c \rangle \longrightarrow \langle M', c' \rangle}{\langle l := M, c \rangle \longrightarrow \langle l := M', c' \rangle}$$

(By now the de facto method in the PL Community. Not yet in the industry...)

Soundness and types

Programs can get stuck:

Ah $\langle true := 5, c \rangle \longrightarrow ???$

(e.g., segmentation fault, program crashes, blue screen (if kernel code))

Introduce type systems:

$\Gamma ::=$ vars $\longrightarrow$ types

$\Gamma \vdash M : A$

types: $A ::= Bool \mid int \mid A \to B \mid Loc$

$$\frac{}{\Gamma \vdash x : A}\ \Gamma(x) = A$$

$$\frac{\Gamma \vdash M_1, M_2 : int}{\Gamma \vdash M_1 + M_2 : int} \qquad \frac{\Gamma \vdash M_1, M_2 : int}{\Gamma \vdash M_1 < M_2 : Bool}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x . M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M(N) : B}$$

etc.

Soundness if $\Gamma \vdash M : A$ and c is total then:

when $A = Bool$ $\langle M, c \rangle \longrightarrow^* \langle true/false, c' \rangle$

$A = int$ $\langle M, c \rangle \longrightarrow^* \langle n, c' \rangle$

$A = A \to B$ $\langle M, c \rangle \longrightarrow^* \langle \lambda x . M', c' \rangle$

Great! We should just define operational semantics and we're set.

Well, not exactly...

e.g. equivalence

$$\begin{array}{lcl} temp := x!; & & x := x! \text{ XOR } y!; \\ x := y!; & \overset{?}{=} & y := x! \text{ XOR } y!; \\ y := temp! & \sim & x := x! \text{ XOR } y! \end{array}$$

Naive: $\forall c: \langle LHS, c\rangle \longrightarrow^* \langle n, c_1\rangle \iff \langle RHS, c\rangle \longrightarrow^* \langle n, c_2\rangle$
$(\sim)$

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

So add: $temp := 0$ $\quad$ $temp := 0$ at the end.

Useful: ~~for all $C[-]$~~

define ~~contexts~~ observational contexts:

$$C ::= - \mid C+M \mid M+C \mid \text{if } C \text{ then } M \text{ else } M$$
$$\text{" } M \text{ " } C \text{ else } M$$
$$\text{" } M \text{ " } M \text{ else } C$$

$$\mid C := M \mid M := C \mid C;M \mid M;C \mid \lambda x.C \mid C(M)$$
$$M(C)$$

extend types: $\Gamma \mid -:A \vdash C[-]:B$

observational
/Contextual equivalence:

$M_1 \cong M_2$ for all contexts $(-:A) \vdash C[-] : Bool$ and conf c: $\langle C[M_1], c\rangle \longrightarrow^* \langle \cancel{bool}\ b, c'\rangle$ iff $\langle C[M_2], c\rangle \longrightarrow^* \langle b, c''\rangle$

How to prove $LHS_1 \cong LHS_2$? Involves some kind of induction over all possible interactions.

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

$C[-] = Parallelize(temp := 1, -);$

meaning of each program phrase depends heavily on other (call!) phrases

Denotational ~~semantics~~ approach

Assign meaning to each term:

easy: $\mathcal{V}[\![\text{true}]\!] := 1$ $\mathcal{V}[\![\text{false}]\!] := 0$ $[\![\text{0x BEEF}]\!] := 48{,}879$

compositional: $[\![(3+5)*7]\!] := [\![3+5]\!] * [\![7]\!]$

Of course, only for well-typed! : $[\![\text{true} := 5]\!] = ???$

So first we define semantics for types: in turn

$[\![\text{bool}]\!] := \mathbb{2}$ $[\![\text{int}]\!] := \mathbb{Z}$ $[\![\text{loc}]\!]$ (e.g. $:= \mathbb{2}^{64}$)

$[\![\Gamma]\!] := \prod_{x \in \text{dom}\,\Gamma} [\![\Gamma(x)]\!]$

~~we want~~

Naive: $[\![A \to B]\!] := [\![A]\!] \to [\![B]\!]$ $[\![\Gamma \vdash M : A]\!] : [\![\Gamma]\!] \to [\![A]\!]$

and indeed we can interpret:

$$[\![\text{if } M_{bool} \text{ then } M_{true} \text{ else } M_{false}]\!](\gamma) := \begin{cases} [\![M_{true}]\!](\gamma) & [\![M_{bool}]\!](\gamma) = 1 \\ [\![M_{false}]\!](\gamma) & [\![M_{bool}]\!](\gamma) = 0 \end{cases}$$

but what about $[\![M := M']\!]$???

instead, pass state around: ~~$[\![\Gamma \vdash M : A]\!] : [\![\Gamma]\!] \times$~~

define ~~Mem~~ $\mathbb{M} := [\![\text{Loc}]\!] \to [\![\text{int}]\!]$ with finite support.

and then $[\![\Gamma \vdash M : A]\!] : [\![\Gamma]\!] \times \mathbb{M} \to [\![A]\!] \times \mathbb{M} \cong [\![\Gamma]\!] \to ([\![A]\!] \times \mathbb{M})^{\mathbb{M}}$

similarly: $[\![A \to B]\!] := [\![A]\!] \to ([\![B]\!] \times \mathbb{M})^{\mathbb{M}}$

and now we can give semantics:

$\gamma \in [\![\Gamma]\!], \mu \in \mathbb{M}$ $[\![n]\!](\gamma)(\mu) = \langle n, \mu \rangle$ $[\![\text{if } M_{bool} \text{ then } M_{true} \text{ else } M_{false}]\!](\gamma)(\mu)$

$$= \begin{cases} [\![M_{true}]\!](\gamma)(\mu') & [\![M_{bool}]\!](\gamma)(\mu) = \langle 1, \mu' \rangle \\ [\![M_{false}]\!](\gamma)(\mu') & [\![M_{bool}]\!](\gamma)(\mu) = \langle 0, \mu' \rangle \end{cases}$$

but now: $[\![M_{loc} := M_{int}]\!](\gamma, \mu) := (n, \mu''[\ell \mapsto n])$

where $[\![M_{loc}]\!](\gamma)(\mu) = (\ell, \mu')$

$[\![M_{int}]\!](\gamma)(\mu') = (n, \mu'')$

and similarly: $[\![M_{loc}!]\!](\gamma, \mu) := (\mu'(\ell), \mu')$

where $[\![M_{loc}]\!](\gamma, \mu) = (\ell, \mu')$.

Theorem (~~denotational~~ Soundness of the den. semantics):

if $\vdash M:A$ and $\langle M, n\rangle \longrightarrow^{*} \langle V, n'\rangle$ then $[\![M]\!](\ast)(n) = ([\![V]\!], n')$

Proof: By induction on terms, but strengthening the hyp.:

a value substitution is a ~~function~~ map σ s.t.

Given ~~Γ⊢~~ $\Gamma \vdash M:A$ and ~~σ~~ for all $x \in dom\,\Gamma$, $\sigma(x)$ is a value $V : \Gamma(x)$

and $M\sigma = M[{}^{\sigma(x)}/x]$. define $[\![\sigma]\!] := \langle [\![\sigma(x)]\!] \rangle_{x \in dom\,\Gamma}$

then the new hyp. is:

if $\Gamma \vdash M:A$, σ a value subst. for Γ and $\langle M\sigma, n\rangle \longrightarrow^{*} \langle V, n'\rangle$

then $[\![M]\!][\![\sigma]\!](n) = ([\![V]\!], n')$

By induction on $\rightarrow^{*}$

(Corollary: determinacy ...)

Theorem: (adequacy) if $[\![M]\!] = [\![M']\!]$ then $M \cong M'$.

Example: (as before)

Proof: Using logical relations ~~define relations:~~

Let $Terms_A := \{\vdash M:A\}$ $Values_A := \{\vdash V:A\}$

we define relations $R^{Val}_A \subseteq [\![A]\!] \times Values_A/\cong)$ $R^{Comp}_A \subseteq ([\![A]\!] \times \mathbb{N})^{\mathbb{N}} \times Terms_A/\cong$

$R^{val}_{bool} := \{(0,[false]),(1,[true])\}$

$R^{val}_{int} := \{(n,[n]) \mid n \in \mathbb{Z}\}$

$R^{val}_{Loc} = \dots$

$R^{val}_{A\to B} := \{(f,[M]) \mid \text{for all } (a,N)\in R^{val}_A, (f(a),[M(N)])\in R^{comp}_B\}$

$R^{comp}_A := \{(\lambda N.(a_N,\bar{N}_N),[M]) \mid \text{for all } N\in|K|, \langle M,N\rangle \to^* (V,\bar{N}_N) \text{ and } (a_N,[V])\in R^{val}_A\}$

$R^{val}_\Gamma := \{\langle\gamma,\sigma\rangle \mid \text{dom}\,\sigma \text{ is a substitution for } \Gamma \text{ and for all } x\in \text{dom}\,\Gamma,\ (\gamma(x),\sigma(x))\in R^{val}_{\Gamma(x)}\}$

Basic lemma: if $\Gamma\vdash M:A$ and $(\gamma,\sigma)\in R^{val}_\Gamma$ then $([\![M]\!](\gamma),[M\sigma])\in R^{comp}_A$

Proof: By induction on $\Gamma\vdash M:A$, for example:

for $\Gamma\vdash n:int$ $\quad [\![n]\!] = \lambda N.\langle n,N\rangle$ if $(\gamma,\sigma)\in R^{val}_\Gamma$, and $N\in|K|$

then indeed the $\langle n,N\rangle \to^* \langle n,N\rangle$ and indeed $(n,[n])\in R^{val}_{int}$.

More interestingly, for $\dfrac{\Gamma\vdash M:A\to B \qquad \Gamma\vdash N:A}{\Gamma\vdash M(N):B}$.

take $(\gamma,\sigma)\in R^{val}_\Gamma$ then $([\![M]\!](\gamma),[M\sigma])\in R^{comp}_{A\to B}$ $\quad ([\![N]\!](\gamma),N\sigma)\in R^{comp}_A$

take any $N\in|K|$ then (denote) $[\![M]\!](\gamma)(N) = (f_N,N')$ and we have $[M\sigma]_N \to^* (V_{fun},N')$ and $(f_N,[V_{fun}]) \in R^{val}_{A\to B}$

for N' we thus have $[\![N]\!](\gamma)(N') = (a,N'')$ and we have $[N\sigma]_{N'} \to^* (V_A,N'')$ and $(a,[V_A]) \in R^{val}_A$

But then ~~$(M(N))\sigma,_N \to^* (V_{fun}\, V_A, N'')$~~
as we also

As we have $(f_N,[V_{fun}])\in R^{val}_{A\to B}$, then for all $(a,[V_A])\in R^{val}_A$ we have
$(f(a),[V_{fun}(V_A)])\in R^{comp}_B$. But then:

for all N: $\quad ((M(N))\sigma,N) \to^* (V_{fun}(N\sigma),N') \to^* (V_{fun}(V_A),N'')$ and hence

$V_{fun}\,V_A \cong (M(N))\sigma$ hence:

$([\![M(N)]\!](\gamma),[(M\,N)\sigma]) = (f_N(a),[V_{fun}(V_A)]) \in R^{comp}_B$

Probably no time, but also:

$$\frac{\Gamma \vdash M_{loc} : loc \qquad \Gamma \vdash M_{int} : int}{\Gamma \vdash M_{loc} := M_{int} : int}$$

take $(\gamma, \sigma) \in \mathcal{R}^{val}_{\Gamma}$ and any $N \in |\mathcal{S}|$.

let $[\![M_{loc}]\!](\gamma)(N) = (l, N')$ by induction, $\langle M_{loc}\sigma, N\rangle \longrightarrow^* \langle l, N'\rangle$

let $[\![M_{int}]\!](\gamma)(N') = (n, N'')$ by induction $\langle M_{int}\sigma, N'\rangle \longrightarrow^* \langle n, N''\rangle$

hence: $\langle (M_{loc} := M_{int})\sigma, N\rangle \longrightarrow^* \langle l := M_{int}\sigma, N'\rangle \longrightarrow^* \langle l := n, N''\rangle \longrightarrow \langle n, N''[l \mapsto n]\rangle$

~~hence:~~ recall:

$[\![M_{loc} := M_{int}]\!](\gamma)(N) = (n, N''[l \mapsto n])$ so we indeed have:

$([\![M]\!](\gamma), [M\sigma]) \in \mathcal{R}^{comp}_{int}$ $\square$

And now we can prove adequacy:

Assume $[\![M]\!] = [\![M']\!]$ Take any $C[-]$ and $(C[M], N) \longrightarrow^* \langle V, N'\rangle$.

By compositionality $[\![C[M]]\!] = [\![C[M']]\!]$ So by the basic lemma

we have: $([\![C[M']]\!](*), [C[M']]) \in R^{comp}_{bool}$

By Soundness, $[\![C[M]]\!](*)(N) = ([\![V]\!](*), N')$ hence; by R^{comp}_{bool}'s def:

$$(C[M'], N) \longrightarrow^* (V, N')$$ and we have adequacy. $\blacksquare$

So now we can capture the "meaning" of programs M by just calculating $[\![M]\!]$, and the adequacy theorem guarantees ~~___~~ somehow 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 $AK(M,N) \to N$ (empty cont's) and $\to$ becomes a relation.

Now $[\![\Gamma \vdash M : A]\!]$ becomes $[\![M]\!] : [\![\Gamma]\!] \to \mathcal{P}_0^{\aleph_0}([\![A]\!])$ the finite nonempty-powerset.

and similarly the function space $[\![A \to B]\!] := [\![A]\!] \to \mathcal{P}_0^{\aleph_0}([\![B]\!])$

$$[\![AK(M,N)]\!](\gamma) := [\![M]\!](\gamma) \cup [\![N]\!](\gamma)$$

The logical relation then becomes: $X \in \mathcal{P}_0^{\aleph_0}([\![A]\!])$ finite

$$R_A^{comp} := \{(X, [M]) \mid \forall a \in X : M \to^* V,\ (a, [V]) \in R_A^{val}\}.$$

randomness ✓

But of course, all the proofs need to be reiterated! And we have other languages: Exceptions, I/O, and combinations of them! $2^5 = 32$ already!

So categorically: To give a model of a language, we need:

A category $\mathcal{C}$ which has finite products, the distributive sum $1+1$, a strong monad $T : \mathcal{C} \to \mathcal{C}$ and $\mathcal{C}$ has Kleisli exponentials.

and interpretations:

$[\![int]\!]$, $[\![loc]\!]$, $[\![n]\!]$, $[\![l]\!]$

and for the effects. e.g.

For our semantics $\mathcal{C} = Set$ and:

For example: for memory, $TA := (A \times H)^H$ with $[\![:=]\!]$ and $[\![!]\!]$ as before

for ND, $TA := \mathcal{P}_0^{\aleph_0}(A)$

for exceptions: $TA := A + E$ with $[\![raise_M]\!] := inj_2\ [\![M]\!]$

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

$[\![Bool]\!] := 1+1$ $\quad [\![\Gamma \vdash M : A]\!] : [\![\Gamma]\!] \to T[\![A]\!]$ Kleisli arrows

$[\![\Gamma \vdash n : int]\!] := \eta_{[\![int]\!]}(n)$ $\quad [\![M ; M']\!] : [\![\Gamma]\!] \to T[\![\Gamma]\!] \to T[\![A]\!]$

Kleisli composition

etc.

__Objects:__ types, A, with finite products.

__Morphisms:__ ~~Sequent~~ ~~a~~ equivalence classes:

$$\prod_{x \in \mathrm{Dom}\,\Gamma} \Gamma(x) \longrightarrow \prod_{i \in I} A_i$$

~~with values as no~~ $\langle [V_i] \rangle_{i \in I}$ with

$\hat{\Gamma} \vdash V_i : A_i$.

Composition is given by substitution.

__Monad:__ $TA := (1 \to A)$

$\eta_A := a : A \vdash \lambda x. a : 1 \to A$

$\mu := m : 1 \to (1 \to A) \vdash \lambda x.\ m(x)(x) : 1 \to A$

$\mathrm{Str}: a : A, m : 1 \to B \vdash \lambda x. (\lambda b. \langle a, b \rangle)\ m(x)) : 1 \to A \times B$

reasonable operational semantics will make this into a monad, and we will have a model Syn.

We then construct another model:

$$\begin{tikzcd} \mathcal{L} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathrm{Pred} \arrow[d, "\pi_2"] \\ \mathrm{Set} \times \mathrm{Syn} \arrow[r, "\times \mathrm{Syn}(1,-)"] & \mathrm{Set} \end{tikzcd}$$

$\mathcal{L}$ is the category of logical relations, and we can construct a model in $\mathcal{L}$ by specifying: $\mathcal{L}[\![\mathrm{int}]\!] := \{([\![n]\!], [n])\}$ $\mathcal{L}[\![\mathrm{loc}]\!] = \{[\![\ell]\!], [\ell]\}$

and lifting the monad structure T to some $R_A^{\mathrm{Comp}} \subseteq T[\![A]\!] \times \mathrm{Terms}\, A$

If we give such a model we automatically get a model $\mathcal{L}$, and it being a model is __precisely__ a restatement of the __basic lemma.__

The fact that $\mathcal{L}$ has all the other required structure follows from fibration abstract nonsense (e.g. π_2 being a bi-fibration or for Pred arising out of a factorisation system of Set)

And the story continues... ~~[illegible]~~

For some languages, $C = \text{set}$ is not enough, and we have to choose a suitable ~~category~~ different category.

Steffan will talk about ~~[illegible]~~ categories of domains which we need ~~to~~ in order to model recursion (while loops, for example) and ~~[illegible]~~ algebraic datatypes.

If we want to model locality (local memory for example) we need to switch to ~~Presh~~ functor categories $C = \text{Set}^{I}$, or ~~[illegible]~~ to the category of Nominal Sets.

If we want to model (the) concurrency, we need ~~[illegible]~~ 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 & game semantics.

Another key idea that we must mention is ~~to~~ new languages.

Semantics are used to reason about our programs. By minimising the semantic gap, we can make our programs easier to understand.

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

To summarise:

we've looked at operational & denotational semantics, covering type soundness, denotational soundness, adequacy, logical relations

we've seen how CT helps the Meta-theory by giving a uniform ~~sem~~ notion of semantic structure.

~~[illegible]~~

Recommended reading

Probably covers all topics:

- Winskel, G. (1993). The formal semantics of programming languages. MIT Press.

Operational semantics

- Classical reference on structural operational semantics: (∼1981 iirc) Gordon Plotkin: A Structural Approach to 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:

- Pierce, B.C. (2002). Types and programming languages. MIT Press.
- Hennessy, M. (1990). The semantics of programming languages. Wiley. 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:

- Winskel, G. (1993). The formal semantics of programming languages: an introduction. MIT Press.
- Gunter, C. (1992). Semantics of programming languages: structures and techniques. MIT Press.
- Tennent, R. (1991). Semantics of programming languages. Prentice Hall.

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?