BEFORE WE START:

Scores out of 5 (one hand) for how happy you are with:

- intuitionistic logic
- λ-calculus
- combinatory logic
- what a category is + examples
- adjunctions
- universal algebra
- clones / cartesian multicategories

PLAN:

Aim : Introduction to foundations of PL semantics; some indication of how ideas from UA / logic turn up.

Route: how to give a semantic account of <u>functions</u>?
ie. STLC without products.

Aim: trojan horse to highlight some interesting links between CS and logic.

- Introduce STLC as a term language for intuitionistic logic
  Now WANT: semantics / model theory for this language.
  As part of this, will want:
  - every model is sound WRT the eqns pov
  - Want the syntax to form a model → term model (= Lindenbaum-Tarski). For completeness.

- giving a semantic model: think in <u>Set</u>.
  Leads you to ccCs. BUT the term model isn't an example?

- Solution 1: use an idea from universal algebra, namely a <u>clone</u>. BUT can we get a category?

- solution 2: use the equivalence of STLC and CL.
  So λ-clones $\simeq$ SK-clones with extensionality.
  No binding! So can axiomatise in first-order terms, as a kind of closed category.

§1 : MOTIVATION + OUTLINE

As a computer scientist I want to study programs.

What is a program? At a high level, it
1) takes in some inputs;
2) does some work;
3) returns some output.

So when we want study programs and their behaviour, we're commonly led to things that look like functions

We ∴ want both:

① syntax : a way to write down programs / functions, together with some notion of "running" a program

② semantics / model theory : a way to assign "meanings" to programs so we can answer questions about their behaviour

- e.g. is an optimisation safe, or does it change the behaviour in bad ways?
- e.g. how to think about programs with new features?

This at the foundations of a large field called semantics of PLs, which is about using model theory / semantics to study program behaviour and ultimately improve the design of PLs.

Difficulty: the operation of creating a function is a binding one.

This talk: at the foundations, where we get intersections with logic. WANT TO GIVE SOME INDICATION OF THIS PART.

Aim: Show how ideas from logic turn up in CS!

We'll take a simple language for functions and ask how its semantics / models should look.
In each case we're looking for:

① Soundness: if M and N are two programs, and $M = N$ in our lang, then $[\![M]\!] = [\![N]\!]$

② Completeness: if $[\![M]\!] = [\![N]\!]$ in every model, then $M = N$ in the language. So we want term model / Lindenbaum-Tarski / syntactic model.

We'll outline some different approaches:

① Idea 1: using certain categories. Goes back a long way, but we have to add to our language to get completeness. ☹

② Idea 2: using an idea from CA, in the form of clones. Sound and complete, but not so useful as categories — examples harder to find. ☹

③ Idea 3: use an idea from logic, which lets us represent functions in mostly first-order terms. Can then axiomatise a class of categories which *is* sound + complete ☺

WHISTLESTOP TOUR: will be high-level and skip details — you can always ask me after!

I've said category a few times. All you need to know for these is this.

* A category $\mathbb{C}$ is a collection of objects, with maps (or arrows) between them.
  Maps compose like functions, and we have for every object A an identity id_A.
  All subject to associativity + unit axioms.

* We write $\mathbb{C}(A,B)$ for the maps $A \longrightarrow B$.

* Examples to have in mind:
  - Sets + functions $\mathrm{Set}(A,B)$ = all functions $A \to B$;
  - your fav. algebraic structure, eg. $\mathrm{Group}(G,H)$ = all group homos $G \longrightarrow H$;
  - topological spaces + cts maps.
  - any poset, with $X(x,y) = \begin{cases} 1 & \text{if } x \leq y \\ 0 & \text{else} \end{cases}$.

§2: A LANGUAGE FOR FUNCTIONS

This is a logic seminar, so let's start with some rules:

$$\frac{}{A_1, \ldots, A_n \vdash A_i}$$

$$\frac{A_1, \ldots, A_n, B \vdash C}{A_1, \ldots, A_n \vdash B \to C}$$

$$\frac{A_1, \ldots, A_n \vdash B \to C \qquad A_1, \ldots, A_n \vdash B}{A_1, \ldots, A_n \vdash C}$$

What are these? Rules for intuitionistic logic.
But if you add terms, they've also very good rules for functions.

The LHS is a context, telling us what types of variables we have in scope:

$$\frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i}\ \text{var}$$

So we'll always have free vars of RHS $\subseteq$ LHS.

The arrow-formation rule says that you build functions by binding a free variable:

$$\frac{x_1 : A_1, \ldots, x_n : A_n, y : B \vdash M : C}{x_1 : A_1, \ldots, x_n : A_n \vdash \lambda y.M : B \to C}\ \text{abst.}$$

eg. $x+1 \rightsquigarrow x \mapsto x+1$

$\equiv \quad y \mapsto M(y)$

Modus ponens say we can pass arguments to functions:

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash M : B \to C \qquad x_1 : A_1, \ldots, x_n : A_n \vdash N : B}{x_1 : A_1, \ldots, x_n : A_n \vdash M\,N : C}\ \text{app.}$$

eg //

- $\vdash \lambda x.\, x : A \to A \quad \equiv$ identity
- we usually allow ourselves some basic operations and types, so

$$\frac{x : \mathbb{N} \vdash x + 1 : \mathbb{N}}{\vdash \lambda x.\, x + 1 : \mathbb{N} \to \mathbb{N}}$$

- we can iterate function-formation:

$$\frac{\dfrac{x : \mathbb{N},\ y : \mathbb{N} \vdash y + x : \mathbb{N}}{x : \mathbb{N} \vdash \lambda y.\, y + x : \mathbb{N} \to \mathbb{N}}}{\vdash \lambda x.\, \lambda y.\, y + x : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}$$

free: x in $y + x$ (and in $\lambda y.\, y + x$)

bound: y in $\lambda y.\, y + x$

$\equiv$ addition

- $\vdash \lambda f.\, \lambda g.\, \lambda x.\, f\,(g\,x) : (B \to C) \to (A \to B) \to (A \to C)$

$\equiv$ composition. //

To make these behave like functions we need two rules:

① evaluating an argument:

$$\begin{aligned}(x \longmapsto x+1)(3) &= (x+1)[3/x] \\ &= 3+1 \\ &= 4\end{aligned}$$

So we get: $(\lambda x.\, M)\, N =_\beta M[N/x]$.

② extensionality:

given a function f, e.g. $f(x) = x+1$, then $f = (x \longmapsto f\, x)$.

So we get $M = \lambda x.\, M\, x$.

This gives us the simply-typed λ-calculus (with $\beta\eta$).

KEY IDEA: We get a bijection

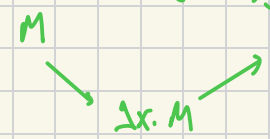
$$\frac{\text{terms } M \text{ s.t. } \Gamma, x:A \vdash M:B}{\text{terms } N \text{ s.t. } \Gamma \vdash N : A \to B}$$

$$\frac{\Gamma, x:A \vdash M:B}{\Gamma \vdash \lambda x.M : A \to B}$$

evaluate at a new variable

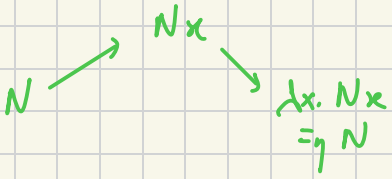
$$\frac{\dfrac{\Gamma \vdash N : A \to B}{\Gamma, x:A \vdash N : A \to B} \qquad \Gamma, x:A \vdash x : A}{\Gamma, x:A \vdash Nx : B}$$

$$(\lambda x.M)\,y = M[y/x] = M$$



$$\frac{z : \mathbb{N} \vdash \lambda y.\, z + y : \mathbb{N} \to \mathbb{N}}{z:\mathbb{N}, x:\mathbb{N} \vdash (\lambda y.\, z+y)\,x : \mathbb{N}}$$

$$= x + y$$



If we can model this bijection, we've pretty much got the semantics sorted.

§3: A FIRST SEMANTICS / MODEL THEORY

Category-theoretic mindset: don't have access to "internals" of objects, only what you can see through maps.

Let's look in a simple example like **Set**.

What about $[\![x : \mathbb{N}, y : \mathbb{N} \vdash x + y : \mathbb{N}]\!]$? This says: if you tell me a value for x and a value for y, you get back their sum. So it has type

$$\begin{aligned} \mathbb{N} \times \mathbb{N} &\longrightarrow \mathbb{N} \\ (x, y) &\longmapsto x + y \end{aligned}$$

In general:

$$[\![x_1 : A_1, \dots, x_n : A_n \vdash M : B]\!] : [\![A_1]\!] \times \dots \times [\![A_n]\!] \longrightarrow [\![B]\!]$$

The meaning of $A \to B$ is all functions which take in an A and return a B. So we want to say

$$[\![A \to B]\!] = ([\![A]\!] \Rightarrow [\![B]\!]) \equiv [\![B]\!]^{[\![A]\!]}$$

So, e.g.:

- $[\![\vdash \lambda x.\, x : \mathbb{N} \to \mathbb{N}]\!] : 1 \longrightarrow (\mathbb{N} \Rightarrow \mathbb{N})$,
- $[\![x : \mathbb{N} \vdash \lambda y.\, y + x : \mathbb{N} \to \mathbb{N}]\!] = \begin{aligned} \mathbb{N} &\longrightarrow (\mathbb{N} \Rightarrow \mathbb{N}) \\ x &\longmapsto (y \mapsto y + x) \end{aligned}$

We need soundness wrt the equations.

Do we get a bijection

$$\frac{\text{maps } [\![\Gamma, x : A \vdash M : B]\!]}{\text{maps } [\![\Gamma \vdash N : A \to B]\!]} \quad ?$$

Unfolding definitions:

$$\frac{\text{maps } [\![\Gamma]\!] \times [\![A]\!] \longrightarrow [\![B]\!]}{\text{maps } [\![\Gamma]\!] \longrightarrow ([\![A]\!] \Rightarrow [\![B]\!])}$$

$(\gamma, a) \mapsto h(\gamma)(a)$ uncurrying $h : \Gamma \to (A \Rightarrow B)$

$f : \Gamma \times A \to B$ ↓ currying

$\Gamma \to (A \Rightarrow B)$

$\gamma \mapsto (a \mapsto f(\gamma, a)) = \lambda a.\, f(\gamma, a)$

— familiar from vector spaces (double dual), modules etc.

So, summing up, what we need to model STLC in a category $\mathcal{C}$ is:

1. for every pair of objects A, B, a "function" object $A \Rightarrow B$
2. some notion of "$\times$": for every Γ, A an object $\Gamma \times A$ (plus some properties so it behaves like cartesian product of sets)
3. a (natural) bijection

$$\frac{\text{maps } \Gamma \times A \longrightarrow B}{\text{maps } \Gamma \longrightarrow (A \Rightarrow B)}$$

for all objects Γ, A, B.

EXAMPLES:

- Sets, with $A \Rightarrow B$ the set of functions;
- any Heyting algebra (i.e. model of intuitionistic logic).

⋮

... but do we get a term model / completeness for our language? NO! The problem is this $\times$ operation: our language has $\to$ but nothing else.

We can get completeness if we add in a product type: i.e. terms for the conjunction:

$$\frac{\Gamma \vdash \quad : A \qquad \Gamma \vdash \quad : B}{\Gamma \vdash \quad : A \times B}$$

$$\frac{\Gamma \vdash \quad : A_1 \times A_2}{\Gamma \vdash \quad : A_i}$$

This works and is well-understood, but is somewhat unsatisfactory. We get a nice class of models, but have had to adjust our syntax to meet it.

We still want a model theory of just functions.

Question: can we find a semantics for the language without products?

§4: A MULTI-ARY COMPLETE SEMANTICS

Problem above: in a category, maps look like $A \to B$ — one-in, one-out. But terms have multiple inputs, eg. addition.

Above we packaged n inputs into one:

$$[\![x_1 : A_1, \dots, x_n : A_n]\!] = \prod_{i=1}^{n} [\![A_i]\!]$$

What if we instead take "maps" that are the same shape as terms? Many-input, one-output.
So let's define a version of categories with this structure:

- objects $A, B, \dots$
- multimaps $A_1, \dots, A_n \xrightarrow{M} B$ = proofs $A_1, \dots A_n \vdash B$ = programs $x_1 : A_1, \dots x_n : A_n \vdash M : B$
  in a set $\mathbb{C}(A_1, \dots A_n ; B)$
- special multimaps corresponding to variables / axioms:
  for every $A_1, \dots A_n$, $p_i : A_1, \dots A_n \longrightarrow A_i$ $(i = 1, \dots n)$

$$= \frac{1 \le i \le n}{A_1, \dots A_n \vdash A_i} \qquad \frac{1 \le i \le n}{x_1 : A_1, \dots x_n : A_n \vdash x_i : A_i}$$

- replace composition with substitution.

$$\frac{x_1 : A_1, \dots x_n : A_n \vdash M : B \qquad (\Delta \vdash N_i : A_i)_{i=1,\dots n}}{\Delta \vdash M[N_1/x_1, \dots, N_n/x_n] : B}$$

becomes

$$\frac{A_1, \dots A_n \xrightarrow{M} B \qquad (\Delta \xrightarrow{N_i} A_i)_{i=1,\dots n}}{M[N_1, \dots N_n] : \Delta \longrightarrow B}$$

Plus we ask for three axioms, which say:

① $x_i[N_1/x_1, \ldots, N_n/x_n] = N_i$;

② $M[x_1/x_1, \ldots, x_n/x_n] = M$;

③ associativity of substitution.

This structure is called a (multisorted) (abstract) clone.
It comes first from UA but is extremely useful in CS because it axiomatizes substitution. Algebraic structure formed by syntax:
So if you want to think about PLs algebraically, this is a good start.

Why do UAists use it?

Take a theory, i.e. some operations and equations, e.g.

- $e^{(0)}$, $\cdot^{(2)}$, $inv(-)^{(1)}$
- unit + assoc. laws + $x \cdot inv(x) = e = inv(x) \cdot x$

Then you can organize all the terms you get as a (single-sorted) clone:

$$\frac{}{x_1, \ldots, x_n \vdash x_i} \qquad \frac{}{x_1, \ldots, x_n \vdash e}$$

$$\frac{x_1, \ldots, x_n \vdash t \quad x_1, \ldots, x_n \vdash u}{x_1, \ldots, x_n \vdash t \cdot u} \qquad \frac{x_1, \ldots, x_n \vdash t}{x_1, \ldots, x_n \vdash inv(t)}$$

$Group(*, \ldots, *; *) = \{\text{terms } x_1, \ldots, x_n \vdash t\}$.

Then you can talk about e.g. comparisons with other theories, algebras (= models) [in this case all groups].

KEY IDEA: can compare different presentations of the same theory by saying their clones are the same.

Note: if you restrict to maps of the form $A \longrightarrow B$ you get exactly a category.

Can we use clones to model STLC? YES! Observed in untyped setting by Hyland; draws on ideas from Pitts, Jacobs, da Paiva, Benton, ...

def: a **λ-clone** is a clone $\mathbb{C}$ equipped with:

- for every $A, B \in \mathbb{C}$ an object $A \Rightarrow B$;
- an "application" operation: for every $M : A_1, \dots, A_n \longrightarrow (B \Rightarrow C)$ an arrow $M\, x^B : A_1, \dots, A_n, B \longrightarrow C$
- an "abstraction" operation: for every $N : A_1, \dots, A_n, B \longrightarrow C$ an arrow $\lambda x^B.\, N : A_1, \dots, A_n \longrightarrow C$

such that we get a bijection

$$\frac{\text{maps } M : A_1, \dots, A_n, B \longrightarrow C}{\text{maps } N : A_1, \dots, A_n \longrightarrow (B \Rightarrow C)}$$

(abstract: ↓ on the left; apply: ↑ on the right)

Thm: λ-clones are sound and complete for STLC.

REMARKS FOR THE COGNOSCENTI:

- if you add products as well, you get a category equivalent to CCCat.
- in this setting we get an "isomorphism" $(A_1 \dots A_n) \cong \prod_i A_i$ saying in the cartesian setting products and contexts must coincide.
- you can add all this structure by looking at CBs in Clone. If you do it in Multicat, you can recover $\otimes$, &, $\multimap$.

//

EVALUATING THE MODEL:

- sound and complete ☺
- useful algebraic description of structure of programs ☺
- not a category — not so clear how much CT will apply to clones, and harder to find examples ☹

Can we think of a class of categories that will be models? Non-trivial!

PROBLEM: binding is not a first-order operation, so it's not so easy to see how to write axioms in the way we do for groups, rings, etc.

§5: A COMPLETE SEMANTICS VIA CL

It's difficult to axiomatise the semantics of STLC directly because it's higher-order: it has binding.

But it is quite easy to define first-order structure, like groups, rings, monoids because these just ask for 1-ary operations with some axioms.

So we look again to logic? Very old: early 20th century!

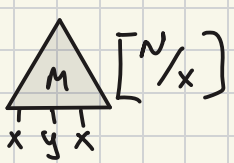
There is a counterpart to λ-calculus (typed or untyped) that has no binding at all. This is combinatory logic.

We'll go CL → cA → categorical models.

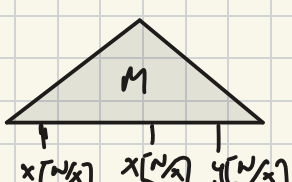
Original motivation: to do for quantifiers like $\forall$ what Schäffer stroke does for propositional logic: reduce to a minimal set of operations. [Schönfinkel]

CL looks a bit mysterious, so some intuition.

Idea (roughly):



= recurse through the tree and replace variables / leaves:



So really what we need is some way of pushing applications through terms, till we get to the leaves. This is what CL does.

CL is a first-order theory with || in the realm of classical HA!
- one binary operation
- three constants = closed terms SK families of
// because of the typing

$$\frac{}{x_1 : A_1, \dots, x_n : A_n \vdash x_i : A_i}\ \text{var} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B}\ \text{app}$$

$$\frac{}{\Gamma \vdash K_{A,B} : A \to (B \to A)}$$

$$\frac{}{\Gamma \vdash S_{A,B,C} : (A \to (B \to C)) \to ((A \to B) \to (A \to C))}$$

We then have the following axioms (left-bracketed), called weak equality:

$$(K\ x)\ y = x \qquad ((S\ x)\ y\ z) = (x\ z)(y\ z)$$

Remarkably, this calculus is equivalent to STLC with β. STANDARD
To get η as well we need to add an extensionality axiom

$$\frac{M\ x_1 \dots x_n = N\ x_1 \dots x_n \qquad (x_1, \dots, x_n \text{ not free in } M, N)}{M = N}$$

STANDARD
With this extra axiom you get something equivalent to STLC with $\beta\eta$.

It's also easy to write down a class of models. For the weak equality it's easy: just use the CCA!

def: an **SK-clone** is a clone $\mathbb{C}$ equipped with

- for every $A, B \in \mathbb{C}$ an object $A \Rightarrow B$
- an application operation

$$\frac{\Gamma \xrightarrow{M} (A \Rightarrow B) \qquad \Gamma \xrightarrow{N} A}{\Gamma \xrightarrow{MN} B}$$

- constants S, K:

$$\frac{}{\Gamma \xrightarrow[K_{A,B}]{} A \Rightarrow (B \Rightarrow A)}$$

$$\frac{}{\Gamma \xrightarrow[S_{A,B,C}]{} (A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))}$$

s.t. the axioms of weak equality hold. //

Prop: SK-clones are sound + complete for CL with weak equality. //

To get extensionality, we note we have the following question for all cartesian:

$$\dfrac{\dfrac{\diamond \xrightarrow{M} [A,[B,C]]}{A.b \xrightarrow{M\,\underline{\ }\,b} [A,[B,C]]}\ \text{weaken}}{A.b \xrightarrow[M\,\underline{\ }\,b\ x^A\ y^B]{} C}\ \text{apply}$$

$\equiv$ $M\ x_1 \dots x_n$ with x_i not free in M

def: an SK-clone is **extensional** if $(-)\,\underline{\ }\,x_1 \dots x_n$ is injective. //

A standard fact about CL is the following:

Prop: (bracket abstraction): for every M in a SK-clone, there exists M^c such that $(M^c)\,\underline{\ }\,x_1 \dots x_n = M$. In other words, $(-)\,\underline{\ }\,x_1 \dots x_n$ has a one-sided inverse. //

So in fact in an extensional SK clone we get $(-)\,\underline{\ }\,x_1 \dots x_n$ is an iso

This does indeed work:

Thm:

1) extensional SK-clones are sound and complete for CL with extensionality;

2) the categories of λ-clones and extensional SKI-clones are equivalent - hence SKI-clones are also sound + complete for STLC. //

a purely algebraic phrasing of the classical result; the proof is also quite algebraic which, depending on your preferences, can be nice!

HOW DOES THIS HELP?

We wanted categorical models, but I've given you another multi-ary one!

BUT: we now have a first-order (mostly) presentation of a class of models. Now easier to write down some equivalent categories!

So: we write down a class of categories and gradually add axioms to get an equivalence with SK-clones:

def: an SK-category is a category $\mathbb{C}$ equipped with:

- a functor $\mathbb{C}^{op} \times \mathbb{C} \longrightarrow \mathbb{C}$ assigning a "function space" $A \Rightarrow B$ to each $A, B \in \mathbb{C}$;
- dinatural transformations with components

$$S : C \Rightarrow (D \Rightarrow E) \longrightarrow (C \Rightarrow D) \Rightarrow (C \Rightarrow E)$$

$$K : D \longrightarrow (C \Rightarrow D)$$

- a functor $U : \mathbb{C} \longrightarrow \mathbf{Set}$ giving the "nullary" maps $\emptyset \longrightarrow A$, and an "application" map

$$\varepsilon : U(C \Rightarrow D) \times UC \longrightarrow UD$$

such that $U(- \Rightarrow -) = \mathbb{C}(-,=)$.

This is subject to 7 axioms, two of which are straightforward translations of weak ext. //

EXPERTS WILL RECOGNISE A VARIANT OF CLOSED CATS. CF KELLY-LAPLAZA.

Thm: the category of SK-categories is equivalent to the category of SK-clones, and hence to the category of λ-clones. So SK-cats are sound + complete for STLC. //

Corollary: TFAE for a category $\mathbb{C}$:

- equipping $\mathbb{C}$ with cc structure;
- equipping $\mathbb{C}$ with SK-structure, and natural isos

$$UC \cong \mathbb{C}(I, C)$$
$$\mathbb{C}(X \otimes A, B) \cong \mathbb{C}(X, [A, B]).$$ //

What actually happens in the model?

$$[\![x_1 : A_1, \ldots, x_n : A_n]\!] = (A_1 \Rightarrow (A_2 \Rightarrow (\cdots \Rightarrow A_n)))$$

so essentially everything gets curried and we use the function space to simulate contexts.

Driven by the algebra but a well-known technique for "decreasing order", cf. encodings of HOATs in FOATs.

§6: Summing up

Main idea of this talk: ideas from logic + CA can be really useful in PL / semantics / TCS!

1) STLC = very simple PL. But with just functions the semantics is hard.

2) Cloves = unifying models adapt very well to give sound + complete models.

3) If you want categories you can do this too, by going via CL to "first-order" things.

Open questions / future work

- extension to linear languages: should recover BCI + closed cats [if you add a rep. for singleton context]
- more generally: understand what MONTs have combinator languages — factors through Λ?
- what about if we drop extensionality, ie. we just do β?
- examples of SK-categories in the wild? Obviously any λ-class or ccc is an example, but does it appear naturally?