Where we're at

Compositionality: $[\![t]\!] = [\![t']\!] \Rightarrow [\![\mathcal{C}[t]]\!] = [\![\mathcal{C}[t']]\!]$. ✓

Soundness: for any type τ, $t \Downarrow_\tau v \Rightarrow [\![t]\!] = [\![v]\!]$. ✓

Adequacy: for $\gamma = \texttt{bool}$ or $\texttt{nat}$, if $t \in \mathrm{PCF}_\gamma$ and $[\![t]\!] = [\![v]\!]$ then $t \Downarrow_\gamma v$. ✓

Where we're at

Compositionality: $[\![t]\!] = [\![t']\!] \Rightarrow [\![\mathcal{C}[t]]\!] = [\![\mathcal{C}[t']]\!]$. ✓

Soundness: for any type τ, $t \Downarrow_\tau v \Rightarrow [\![t]\!] = [\![v]\!]$. ✓

Adequacy: for $\gamma = \texttt{bool}$ or $\texttt{nat}$, if $t \in \mathrm{PCF}_\gamma$ and $[\![t]\!] = [\![v]\!]$ then $t \Downarrow_\gamma v$. ✓

Full Abstraction: $t_1 \cong_{\mathrm{ctx}} t_2 : \tau \Rightarrow [\![t_1]\!] = [\![t_2]\!] \in [\![\tau]\!]$!!

Full abstraction

Beyond full abstraction failure

Interpreting full abstraction failure

- PCF is not expressive enough to present the model?
- Contexts are too weak: they do not distinguish enough programs?
- The model does not adequately capture PCF?

$\boxed{\Gamma \vdash t : \tau}$

...

$$\textsc{Por}\ \frac{\Gamma \vdash t_1 : \tau \qquad \Gamma \vdash t_2 : \tau}{\Gamma \vdash \mathtt{por}(t_1, t_2) : \tau}$$

$\boxed{t \Downarrow_\tau v}$

$$\textsc{PorL}\ \frac{t_1 \Downarrow_{\mathtt{bool}} \mathtt{true}}{\mathtt{por}(t_1, t_2) \Downarrow_{\mathtt{bool}} \mathtt{true}} \qquad \textsc{PorR}\ \frac{t_2 \Downarrow_{\mathtt{bool}} \mathtt{true}}{\mathtt{por}(t_1, t_2) \Downarrow_{\mathtt{bool}} \mathtt{true}}$$

$$\textsc{PorF}\ \frac{t_1 \Downarrow_{\mathtt{bool}} \mathtt{false} \qquad t_2 \Downarrow_{\mathtt{bool}} \mathtt{false}}{\mathtt{por}(t_1, t_2) \Downarrow_{\mathtt{bool}} \mathtt{false}}$$

Full abstraction for PCF+por

If we extend the semantics of PCF to PCF+$\mathtt{por}$ with

$$[\![\mathtt{por}]\!] = \mathrm{por}$$

the resulting denotational semantics is fully abstract.

Full abstraction for PCF+por

If we extend the semantics of PCF to PCF+$\mathtt{por}$ with

$$[\![\mathtt{por}]\!] = \mathrm{por}$$

the resulting denotational semantics is fully abstract...

but is PCF+$\mathtt{por}$ still a reasonable model of programming language?

Stronger contexts

```
let taste (f : (bool -> bool) -> bool) : int =
let r = ref 0 in
let _  = f (fun x -> incr r ; x) in
!r
;;

print_int (taste (fun f -> f true)) (* 1 *) ;;
print_int (taste (fun f -> (f true) && (f true))) (* 2 *)
```

```
let taste (f : (bool -> bool) -> bool) : int =
let r = ref 0 in
let _  = f (fun x -> incr r ; x) in
!r
;;

print_int (taste (fun f -> f true)) (* 1 *) ;;
print_int (taste (fun f -> (f true) && (f true))) (* 2 *)
```

With more contexts, you can distinguish more programs

Real languages have effects

If you add effects (references, control flow…) to a language, you can distinguish more programs

Real languages have effects

If you add effects (references, control flow…) to a language, you can distinguish more programs

Full abstraction becomes different: somewhat easier…

Real languages have effects

If you add effects (references, control flow…) to a language, you can distinguish more programs

Full abstraction becomes different: somewhat easier…
but is contextual equivalence still a reasonable notion?

Fully abstract semantics

- dI-domains & stable functions $\rightarrow$ no $\mathbf{por}$ any more, but still not fully abstract...

Fully abstract semantics

- dI-domains & stable functions $\rightarrow$ no $\mathbf{por}$ any more, but still not fully abstract...

- only proper answers in the late 90s (!): logical relations and game semantics

Loader's undecidability result

PCF $_{\texttt{bool}}$:

- variables, $\rightarrow$, application
- `true`, `false`, `if`
- a primitive undefined $\texttt{raise} : \texttt{bool}$

Extremely simple

Loader's undecidability result

PCF $_{\texttt{bool}}$:

- variables, $\rightarrow$, application
- `true`, `false`, `if`
- a primitive undefined $\texttt{raise} : \texttt{bool}$

Extremely simple

Yet, definability and contextual equivalence are undecidable...

Where to go from here?

Towards full abstraction

Source of a very rich literature:

- linear logic
- logical relations
- game semantics
- bisimulations techniques
- ...

Separate

1. the structure needed to interpret a language (generic)
2. how to construct this structure in particular examples (specific)

Categorical semantics

Separate

1. the structure needed to interpret a language (generic)
2. how to construct this structure in particular examples (specific)

Example:

1. λ-calculus → cartesian closed categories
2. domains and continuous functions are a CCC

Categorical semantics

Separate

1. the structure needed to interpret a language (generic)
2. how to construct this structure in particular examples (specific)

Example:

1. λ-calculus $\longrightarrow$ cartesian closed categories
2. domains and continuous functions are a CCC

Interpret:

- a type τ as an object in a category;
- a term $\Gamma \vdash t : \tau$ as a morphism/arrow $[\![t]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.

OCaml's ADT:

$'a\ \text{tree} = 1 + a' * 'a\ \text{tree} * 'a\ \text{tree}$

```
type 'a tree =
  | Leaf
  | Node of 'a * 'a tree * 'a tree
```

It is a fixed point equation! We can use domain theory to solve it.

Beyond pure languages

Effects: control flow (errors), mutability/state, input-output, etc.
An important aspect of programming languages!

Beyond pure languages

Effects: control flow (errors), mutability/state, input-output, etc.
An important aspect of programming languages!

Modelled as a monad T (example: $T(A) \stackrel{\text{def}}{=} (A \times \text{State})^{\text{State}}$)

Beyond pure languages

Effects: control flow (errors), mutability/state, input-output, etc.
An important aspect of programming languages!

Modelled as a monad T (example: $T(A) \stackrel{\text{def}}{=} (A \times \text{State})^{\text{State}}$)

Denotation of a computation: $[\![\Gamma]\!] \to T([\![\tau]\!])$

$T(A) = E + A$

Beyond pure languages

Effects: control flow (errors), mutability/state, input-output, etc.
An important aspect of programming languages!

Modelled as a monad T (example: $T(A) \stackrel{\text{def}}{=} (A \times \text{State})^{\text{State}}$)

Denotation of a computation: $[\![\Gamma]\!] \to T([\![\tau]\!])$

And more: adjunctions, effect handlers...

Easter: axiomatic semantic (Hoare Logic and Model Checking)

More semantics

Easter: axiomatic semantic (Hoare Logic and Model Checking)

In the end, the most interesting aspects of semantics is in the interaction between different approaches.