Denotational Semantics

Ioannis Markakis

Lectures for Part II CST 2025/2026

Beyond full abstraction failure

Recap

We have related operational semantics and denotational semantics:

Recap

We have related operational semantics and denotational semantics:

- Soundness: $t \Downarrow_\tau v \implies [\![t]\!] = [\![v]\!]$ for every type τ
- Adequacy: $[\![t]\!] = [\![v]\!] \implies t \Downarrow_\tau v$ for $\tau \in \{\mathtt{nat}, \mathtt{bool}\}$

Recap

We have related operational semantics and denotational semantics:

- Soundness: $t \Downarrow_\tau v \implies [\![t]\!] = [\![v]\!]$ for every type τ
- Adequacy: $[\![t]\!] = [\![v]\!] \implies t \Downarrow_\tau v$ for $\tau \in \{\texttt{nat}, \texttt{bool}\}$

We have also related contextual with denotational equivalence:

Recap

We have related operational semantics and denotational semantics:

- Soundness: $t \Downarrow_\tau v \implies [\![t]\!] = [\![v]\!]$ for every type τ
- Adequacy: $[\![t]\!] = [\![v]\!] \implies t \Downarrow_\tau v$ for $\tau \in \{\mathtt{nat}, \mathtt{bool}\}$

We have also related contextual with denotational equivalence:

- Compositionality: $[\![t]\!] = [\![t']\!] \implies [\![\mathcal{C}[t]]\!] = [\![\mathcal{C}[t']]\!]$ for all terms and contexts
- Using the above gives $[\![t]\!] = [\![t']\!] \implies t \cong_{\mathrm{ctx}} t' : \tau$

Recap

We have related operational semantics and denotational semantics:

- Soundness: $t \Downarrow_\tau v \implies [\![t]\!] = [\![v]\!]$ for every type τ
- Adequacy: $[\![t]\!] = [\![v]\!] \implies t \Downarrow_\tau v$ for $\tau \in \{\mathtt{nat}, \mathtt{bool}\}$

We have also related contextual with denotational equivalence:

- Compositionality: $[\![t]\!] = [\![t']\!] \implies [\![\mathcal{C}[t]]\!] = [\![\mathcal{C}[t']]\!]$ for all terms and contexts
- Using the above gives $[\![t]\!] = [\![t']\!] \implies t \cong_{\mathrm{ctx}} t' : \tau$

- Failure of full abstraction:

$$t \cong_{\mathrm{ctx}} t' : \tau \not\Rightarrow [\![t]\!] = [\![t']\!]$$

Recap

We have related operational semantics and denotational semantics:

- Soundness: $t \Downarrow_\tau v \implies [\![t]\!] = [\![v]\!]$ for every type τ
- Adequacy: $[\![t]\!] = [\![v]\!] \implies t \Downarrow_\tau v$ for $\tau \in \{\mathtt{nat}, \mathtt{bool}\}$

We have also related contextual with denotational equivalence:

- Compositionality: $[\![t]\!] = [\![t']\!] \implies [\![\mathcal{C}[t]]\!] = [\![\mathcal{C}[t']]\!]$ for all terms and contexts
- Using the above gives $[\![t]\!] = [\![t']\!] \implies t \cong_{\mathrm{ctx}} t' : \tau$

- Failure of full abstraction:

$$t \cong_{\mathrm{ctx}} t' : \tau \not\Rightarrow [\![t]\!] = [\![t']\!]$$

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?

PCF+por

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

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

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

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

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

Full abstraction for PCF+por

Theorem

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

Theorem

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?

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…

Towards fully abstract semantics for Pcf

- Berry introduced dI-domains & stable functions
  - removed $\mathbf{por}$ from the model
  - still not fully abstract

Towards fully abstract semantics for Pcf

- Berry introduced dI-domains & stable functions
  - removed $\mathbf{por}$ from the model
  - still not fully abstract
- O'Hearn and Riecke used logical relations
  - characterised the definable elements
  - fully abstract but not as insightful

Towards fully abstract semantics for Pcf

- Berry introduced dI-domains & stable functions
  - removed **por** from the model
  - still not fully abstract
- O'Hearn and Riecke used logical relations
  - characterised the definable elements
  - fully abstract but not as insightful
- Game semantics give another fully abstract model
  - program execution $\mapsto$ two-player game

Loader's undecidability result

Let PCF $_{\texttt{bool}}$ be the restriction of PCF consisting of:

- variables, abstraction, application
- `true`, `false`, `if`
- a primitive divergent $\Omega : \texttt{bool}$

Loader's undecidability result

Let PCF $_{\texttt{bool}}$ be the restriction of PCF consisting of:

- variables, abstraction, application
- `true`, `false`, `if`
- a primitive divergent $\Omega : \texttt{bool}$

This is a very minimal language with semantics in finite domains. However...

Loader's Theorem

Definability and contextual equivalence in PCF $_{\texttt{bool}}$ are undecidable.

Overview: Where to go from here?

Towards full abstraction

Source of a very rich literature:

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

Categorical semantics

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 → cartesian closed categories
2. domains and continuous functions are a CCC

Generally we interpret:

- a type τ as an object in a category;
- a context Γ as the product of its types;
- a term $\Gamma \vdash t : \tau$ as an arrow $[\![t]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.
- parallel substitution $\Gamma \vdash \sigma : \Delta$ as an arrow $[\![\sigma]\!] : [\![\Gamma]\!] \to [\![\Delta]\!]$

Domain theory for abstract datatypes

OCaml's ADT:

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

Domain theory for abstract datatypes

OCaml's ADT:

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

$$\mathrm{Tree}(A) = 1 + A \times \mathrm{Tree}(A) \times \mathrm{Tree}(A)$$

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: $[\![\Gamma]\!] \to T([\![\tau]\!])$

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...

More semantics

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.