We have a denotational semantics for types $[\tau]$ and terms $[t]$ such that:

**Compositionality:** $[t] = [t'] \Rightarrow [c[t]] = [c[t']]$. ✓

**Soundness:** For any type $\tau$, $t \Downarrow_\tau v \Rightarrow [t] = [v]$. ✓

**Adequacy:** For $\gamma = \text{bool}$ or $\text{nat}$, if $t \in \text{PCF}_\gamma$ and $[t] = [v]$ then $t \Downarrow_\gamma v$. ✓
We have a denotational semantics for types $[\tau]$ and terms $[t]$ such that:

**Compositionality:** $[t] = [t'] \implies [C[t]] = [C[t']]$. ✓

**Soundness:** for any type $\tau$, $t \Downarrow_{\tau} v \implies [t] = [v]$. ✓

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

From this we can show

$$[t] = [u] \in [\tau] \implies t \equiv_{\text{ctx}} u : \tau$$

What about the converse implication?
FULL ABSTRACTION
FULL ABSTRACTION
FAILURE OF FULL ABSTRACTION
A denotational model is **fully abstract** if

\[ t_1 \cong_{\text{ctx}} t_2 : \tau \implies [t_1] = [t_2] \in [\tau] \]
A denotational model is **fully abstract** if

\[ t_1 \equiv_{\text{ctx}} t_2 : \tau \Rightarrow \llbracket t_1 \rrbracket = \llbracket t_2 \rrbracket \in \llbracket \tau \rrbracket \]

A form of **completeness** of semantic equivalence wrt. program equivalence.
A denotational model is **fully abstract** if

\[
t_1 \cong_{\text{ctx}} t_2 : \tau \Rightarrow \llbracket t_1 \rrbracket = \llbracket t_2 \rrbracket \in \llbracket \tau \rrbracket
\]

A form of **completeness** of semantic equivalence wrt. program equivalence.

The domain model of PCF is **not** fully abstract.
The *parallel or* function $\text{por} : \mathbb{B}_\bot \times \mathbb{B}_\bot \to \mathbb{B}_\bot$ is defined as given by the following table:

<table>
<thead>
<tr>
<th>por</th>
<th>true</th>
<th>false</th>
<th>$\bot$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>true</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>
The (left) sequential or function \( \text{or} : \mathbb{B}_{\bot} \times \mathbb{B}_{\bot} \rightarrow \mathbb{B}_{\bot} \) is defined as

\[
\text{or} \overset{\text{def}}{=} \left[ \text{fun } x: \text{bool}. \text{ fun } y: \text{bool}. \text{ if } x \text{ then true else } y \right]
\]

It is given by the following table:

<table>
<thead>
<tr>
<th>or</th>
<th>true</th>
<th>false</th>
<th>( \bot )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>( \bot )</td>
</tr>
<tr>
<td>( \bot )</td>
<td>( \bot )</td>
<td>( \bot )</td>
<td>( \bot )</td>
</tr>
</tbody>
</table>
### Parallel vs Sequential OR

<table>
<thead>
<tr>
<th></th>
<th>true</th>
<th>false</th>
<th>⊥</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td>true</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>true</th>
<th>false</th>
<th>⊥</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>⊥</td>
</tr>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>
**Parallel vs Sequential OR**

<table>
<thead>
<tr>
<th></th>
<th>true</th>
<th>false</th>
<th>(\perp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>(\perp)</td>
</tr>
<tr>
<td>(\perp)</td>
<td>true</td>
<td>(\perp)</td>
<td>(\perp)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>true</th>
<th>false</th>
<th>(\perp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>(\perp)</td>
</tr>
<tr>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
</tr>
</tbody>
</table>

*or* is sequential, but *por* is not.
There is no closed PCF term

\[ t : \text{bool} \rightarrow \text{bool} \rightarrow \text{bool} \]

satisfying

\[ [t] = \text{por} : \mathbb{B}_\bot \rightarrow \mathbb{B}_\bot \rightarrow \mathbb{B}_\bot . \]
The denotational model of PCF in domains and continuous functions is not fully abstract.
The denotational model of PCF in domains and continuous functions is not fully abstract.

For well-chosen $T_{\text{true}}$ and $T_{\text{false}}$,

$$T_{\text{true}} \equiv_{\text{ctx}} T_{\text{false}} : (\text{bool} \to \text{bool} \to \text{bool}) \to \text{bool}$$

$$[T_{\text{true}}] \neq [T_{\text{false}}] \in (\mathbb{B} \to \mathbb{B} \to \mathbb{B}) \to \mathbb{B}$$
The denotational model of PCF in domains and continuous functions is not fully abstract.

For well-chosen $T_{\text{true}}$ and $T_{\text{false}}$,

$$T_{\text{true}} \equiv_{\text{ctx}} T_{\text{false}} : (\text{bool} \to \text{bool} \to \text{bool}) \to \text{bool}$$

$$[[T_{\text{true}}]] \neq [[T_{\text{false}}]] \in (\mathbb{B} \to \mathbb{B} \to \mathbb{B}) \to \mathbb{B}$$

Idea:

- for all $f \in PCF_{\text{bool} \to \text{bool} \to \text{bool}}$, ensure $T_b f \uparrow_{\text{bool}}$...
- but $[[T_b]] (\text{por}) = [[b]]$. 
Example of full abstraction failure

\[ T_b \stackrel{\text{def}}{=} \text{fun}\ f : \text{bool} \to (\text{bool} \to \text{bool}). \]
\[ \quad \text{if}(f \true \Omega_{\text{bool}}) \text{ then} \]
\[ \quad \quad \text{if} (f \Omega_{\text{bool}} \true) \text{ then} \]
\[ \quad \quad \quad \text{if} (f \false \false) \text{ then} \Omega_{\text{bool}} \text{ else } b \]
\[ \quad \text{else} \Omega_{\text{bool}} \]
\[ \text{else} \Omega_{\text{bool}} \]
1) To $J$ hold for all $f$ a PCF

\[ J \text{ true } \rightarrow \text{ true } \]

\[ J \text{ false } \rightarrow \text{ false } \]

\[ J \text{ hold } \rightarrow \text{ if } \]

\[ \text{ if } \text{ true } \rightarrow \text{ true } \]

\[ \text{ if } \text{ false } \rightarrow \text{ false } \]

\[ \text{ if } \text{ satisfies } \]

\[ \text{ then } J \text{ (true, } \perp \text{B) = true} \]

\[ \text{ then } J \text{ (false, true) = true} \]

\[ \text{ then } J \text{ (false, false) = false} \]
<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

\[(e) = \left( \bigvee \right) = \text{max} \text{ true, false} \]

1. cannot exist

So for every \( f \in \text{PROP} \), if \( \text{bool} \to \text{bool} \to \text{bool} \to \text{bool} \)
FULL ABSTRACTION
BEYOND FULL ABSTRACTION FAILURE
• PCF is not expressive enough to present the model?
• The model does not adequately capture PCF?
• Contexts are too weak: they do not distinguish enough programs?
\[ \Gamma \vdash t : \tau \]

\[ \Gamma \vdash t_1 : \tau \quad \Gamma \vdash t_2 : \tau \]

\[ \Gamma \vdash \text{por}(t_1, t_2) : \tau \]

\[ t \Downarrow^{\tau} \nu \]

\[ \text{PORL} \quad \frac{t_1 \Downarrow_{\text{bool}} \text{true} \quad \text{PORR} \quad \frac{t_2 \Downarrow_{\text{bool}} \text{true}}{\text{por}(t_1, t_2) \Downarrow_{\text{bool}} \text{true}} \quad \text{PORF} \quad \frac{t_1 \Downarrow_{\text{bool}} \text{false} \quad t_2 \Downarrow_{\text{bool}} \text{false}}{\text{por}(t_1, t_2) \Downarrow_{\text{bool}} \text{false}} \]
If we extend the semantics of PCF to PCF+\texttt{por} with
\[\llbracket \texttt{por} \rrbracket = \texttt{por}\]
the resulting denotational semantics is fully abstract.
If we extend the semantics of PCF to PCF+$\texttt{por}$ with

\[[\texttt{por}] = \texttt{por}\]

the resulting denotational semantics is fully abstract...

but is PCF+$\texttt{por}$ still a reasonable model of programming language?
## Fully abstract semantics for PCF

- first step: dI-domains & stable functions $\rightarrow$ no **por** any more, but still not fully abstract...
- only proper answers in the late 90s (!): logical relations and game semantics
Fully abstract semantics for PCF

- first step: dI-domains & stable functions → no por any more, but still not fully abstract...
- only proper answers in the late 90s (!): logical relations and game semantics

Real languages have effects

- If you add effects (references, control flow...) to a language, contexts become much more expressive.
- Full abstraction becomes different: somewhat easier... but is contextual equivalence still a reasonable idea?
WHERE TO GO FROM HERE?
Source of a very rich literature:

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

Separate

• the structure needed to interpret a language (generic)
• how to construct this structure in particular examples (specific)
CATEGORICAL SEMANTICS

Separate

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

Interpret:

• a type $\tau$ as an object in a category;
• a term $\Gamma \vdash t : \tau$ as a morphism/arrow $[t] : [\Gamma] \to [\tau]$. 
CATEGORICAL SEMANTICS

Separate

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

Interpret:

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

Example: $\lambda$-calculus $\rightarrow$ cartesian closed categories
OCaml’s ADT:

```ocaml
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...
An important aspect of programming languages!
BEYOND PURE LANGUAGES

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

Modelled as a monad $T$ (example: $T(A) \overset{\text{def}}{=} (A \times \text{State})^{\text{State}}$)
Effects: control flow (errors), mutability/state, input-output...
An important aspect of programming languages!

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

\[
\Gamma \rightarrow (\Gamma \rightarrow (\Gamma \rightarrow \text{State}))^{\text{State}}
\]

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

\[
[\Gamma] \rightarrow \Gamma \rightarrow (\Gamma \rightarrow \text{State}) \rightarrow \Gamma \rightarrow \Gamma \rightarrow \text{State}
\]
Easter: axiomatic semantic (Hoare Logic and Model Checking)
Easter: *axiomatic semantic* (Hoare Logic and Model Checking)

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