Advanced Semantics of Programming Languages

Pierre CLAIRAMBAULT & Colin RIBA

LIP - ENS de Lyon

Course 01
09/11

A Naive Introduction

(based on simple examples)

The First Example

Consider the programs

```
foo(f:int->int):
  return f(5) + f(5)
```

and

```
bar(f:int->int):
  a = f(5)
  return a + a
```

The First Example

Consider the programs

```
foo(f:int->int):
  return f(5) + f(5)
```

and

```
bar(f:int->int):
  a = f(5)
  return a + a
```

Are these two programs equivalent ?

The First Example

Consider the programs

```
foo(f:int->int):
  return f(5) + f(5)
```

and

```
bar(f:int->int):
  a = f(5)
  return a + a
```

Are these two programs equivalent ?

- They are not equivalent if **f** can access a global reference.

The First Example

Consider the programs

```
foo(f:int->int):
  return f(5) + f(5)
```

and

```
bar(f:int->int):
  a = f(5)
  return a + a
```

Are these two programs equivalent ?

- They are not equivalent if **f** can access a global reference.
- They are equivalent if **f** behaves as a **function**, say

$$[\![\mathbf{f}]\!] \quad : \quad [\![\mathbf{int}]\!] \longrightarrow [\![\mathbf{int}]\!]$$

where $[\![\mathbf{int}]\!]$ is a set representing the type **int**.

The First Example

Consider the programs

```
foo(f:int->int):
  return f(5) + f(5)
```

and

```
bar(f:int->int):
  a = f(5)
  return a + a
```

Are these two programs equivalent ?

- They are not equivalent if `f` can access a global reference.
- They are equivalent if `f` behaves as a **function**, say

$$[\![\mathtt{f}]\!] \;:\; [\![\mathtt{int}]\!] \longrightarrow [\![\mathtt{int}]\!]$$

  where $[\![\mathtt{int}]\!]$ is a set representing the type `int`.

Objectives of the course.

- Mathematical models of programming languages (denotational semantics, category theory, type systems).

The First Example

Consider the programs

```
foo(f:int->int):
  return f(5) + f(5)
```

and

```
bar(f:int->int):
  a = f(5)
  return a + a
```

Are these two programs equivalent ?

- They are not equivalent if **f** can access a global reference.
- They are equivalent if **f** behaves as a **function**, say

$$[\![\mathtt{f}]\!] \ : \ [\![\mathtt{int}]\!] \longrightarrow [\![\mathtt{int}]\!]$$

  where $[\![\mathtt{int}]\!]$ is a set representing the type **int**.

Objectives of the course.

- Mathematical models of programming languages (denotational semantics, category theory, type systems).

Methodology of the course.

- Begin with simple approaches.
- Then progressively model more complex behaviours.

The First Example

Consider the programs

```
foo(f:int->int):
  return f(5) + f(5)
```

and

```
bar(f:int->int):
  a = f(5)
  return a + a
```

Are these two programs equivalent ?

- They are not equivalent if `f` can access a global reference.
- They are equivalent if `f` behaves as a **function**, say

$$[\![\mathtt{f}]\!] \quad : \quad [\![\mathtt{int}]\!] \longrightarrow [\![\mathtt{int}]\!]$$

  where $[\![\mathtt{int}]\!]$ is a set representing the type `int`.

Objectives of the course.

- Mathematical models of programming languages (denotational semantics, category theory, type systems).

Methodology of the course.

- Begin with simple approaches.
- Then progressively model more complex behaviours.

Now:

- A naive introduction to some basic ideas.

Types as Sets

- Assume types, say **`int`**, **`bool`**, are to be interpreted as sets $[\![\texttt{int}]\!]$, $[\![\texttt{bool}]\!]$.

Types as Sets

- Assume types, say $\mathtt{int}$, $\mathtt{bool}$, are to be interpreted as sets $[\![\mathtt{int}]\!]$, $[\![\mathtt{bool}]\!]$.

Question.

- Can we assume

$$[\![\mathtt{bool}]\!] \quad := \quad \{\mathtt{true}, \mathtt{false}\} \tag{1}$$

Types as Sets

- Assume types, say **int**, **bool**, are to be interpreted as sets $[\![\texttt{int}]\!]$, $[\![\texttt{bool}]\!]$.

Question.

- Can we assume

$$[\![\texttt{bool}]\!] \quad := \quad \{\texttt{true}, \texttt{false}\} \tag{1}$$

Answer.

- Consider the non-terminating program

```
loop (b:bool):
  while true:
    skip
  return true
```

Types as Sets

- Assume types, say $\mathtt{int}$, $\mathtt{bool}$, are to be interpreted as sets $[\![\mathtt{int}]\!]$, $[\![\mathtt{bool}]\!]$.

Question.

- Can we assume

$$[\![\mathtt{bool}]\!] \quad := \quad \{\mathtt{true}, \mathtt{false}\} \tag{1}$$

Answer.

- Consider the non-terminating program

```
loop (b:bool):
  while true:
    skip
  return true
```

- If $[\![\mathtt{bool}]\!]$ is as in (1), then we can not have

$$[\![\mathtt{loop}]\!] \quad : \quad [\![\mathtt{bool}]\!] \quad \longrightarrow \quad [\![\mathtt{bool}]\!]$$

Types as Sets

- Assume types, say **`int`**, **`bool`**, are to be interpreted as sets $[\![\mathtt{int}]\!]$, $[\![\mathtt{bool}]\!]$.

Question.

- Can we assume

$$[\![\mathtt{bool}]\!] \quad := \quad \{\mathtt{true}, \mathtt{false}\} \tag{1}$$

Answer.

- Consider the non-terminating program

```
loop (b:bool):
  while true:
    skip
  return true
```

- If $[\![\mathtt{bool}]\!]$ is as in (1), then we can not have

$$[\![\mathtt{loop}]\!] \quad : \quad [\![\mathtt{bool}]\!] \quad \longrightarrow \quad [\![\mathtt{bool}]\!]$$

- We shall therefore represent divergence and assume

$$[\![\mathtt{bool}]\!] \quad := \quad \{\bot, \mathtt{true}, \mathtt{false}\} \qquad \text{($\bot$ "$=$" divergence)}$$

  We can then have, as expected:

$$[\![\mathtt{loop}]\!](a) \quad = \quad \bot \qquad \text{(for all $a \in [\![\mathtt{bool}]\!]$)}$$

A Taste of Finitary PCF

Motivation.

- A simple language to discuss

$$[\![\mathbf{bool}]\!] \quad := \quad \{\bot, \mathbf{true}, \mathbf{false}\}$$

A Taste of Finitary PCF

Motivation.

- A simple language to discuss

$$[\![\mathtt{bool}]\!] \quad := \quad \{\bot, \mathtt{true}, \mathtt{false}\}$$

The Language of Finitary PCF.

$$\begin{array}{rcl} \tau, \sigma & ::= & \mathtt{bool} \quad | \quad \sigma \to \tau \\ t, u & ::= & t\,u \quad | \quad \mathtt{fun}\,x \to t \quad | \quad \mathtt{true} \quad | \quad \mathtt{false} \quad | \quad \mathtt{if}\,t\,\mathtt{then}\,u\,\mathtt{else}\,v \quad | \quad \Omega \end{array}$$

A Taste of Finitary PCF

Motivation.

- A simple language to discuss

$$[\![\mathtt{bool}]\!] \quad := \quad \{\bot, \mathtt{true}, \mathtt{false}\}$$

The Language of Finitary PCF.

$$\begin{array}{lcl} \tau, \sigma & ::= & \mathtt{bool} \quad | \quad \sigma \to \tau \\ t, u & ::= & t\,u \quad | \quad \mathtt{fun}\,x \to t \quad | \quad \mathtt{true} \quad | \quad \mathtt{false} \quad | \quad \mathtt{if}\,t\,\mathtt{then}\,u\,\mathtt{else}\,v \quad | \quad \Omega \end{array}$$

- Purely functional language with Booleans and divergence (Ω).

A Taste of Finitary PCF

Motivation.

- A simple language to discuss

$$[\![\mathbf{bool}]\!] \quad := \quad \{\bot, \mathbf{true}, \mathbf{false}\}$$

The Language of Finitary PCF.

$$\begin{array}{rcl} \tau, \sigma & ::= & \mathbf{bool} \mid \sigma \to \tau \\ t, u & ::= & t\,u \mid \mathbf{fun}\,x \to t \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{if}\,t\,\mathbf{then}\,u\,\mathbf{else}\,v \mid \Omega \end{array}$$

- Purely functional language with Booleans and divergence (Ω).

We assume *call-by-name* evaluation:

$$\begin{array}{rcl} (\mathbf{fun}\,x \to t)u & = & t[u/x] \\ \mathbf{if}\,\mathbf{true}\,\mathbf{then}\,t\,\mathbf{else}\,u & = & t \\ \mathbf{if}\,\mathbf{false}\,\mathbf{then}\,t\,\mathbf{else}\,u & = & u \end{array}$$

Example.

Consider the two following `or` programs:

```
or_l := fun a, b ->
  if a then a else b
```

vs

```
or_r := fun a, b ->
  if b then b else a
```

Example.

Consider the two following `or` programs:

```
or_l := fun a, b ->
  if a then a else b
```

vs

```
or_r := fun a, b ->
  if b then b else a
```

Questions.

- What are the functions $[\![$`or_l`$]\!]$, $[\![$`or_r`$]\!]$?
- Are the programs `or_l` and `or_r` equivalent ?

Example.
Consider the two following **or** programs:

```
or_l := fun a, b ->
  if a then a else b
```

vs

```
or_r := fun a, b ->
  if b then b else a
```

Questions.

- What are the functions $[\![$**or_l**$]\!]$, $[\![$**or_r**$]\!]$?
- Are the programs **or_l** and **or_r** equivalent ?

Example.
Consider, for **b** $\in$ {**true**, **false**}, the program

```
taste_b := fun f ->
  if f(true, Ω) and
     f(Ω, true) and
     not(f(false, false))
  then b
  else true
```

Example.
Consider the two following **or** programs:

```
or_l := fun a, b ->
  if a then a else b
```

vs

```
or_r := fun a, b ->
  if b then b else a
```

Questions.

- What are the functions $[\![$**or_l**$]\!]$, $[\![$**or_r**$]\!]$?
- Are the programs **or_l** and **or_r** equivalent ?

Example.
Consider, for **b** $\in \{$**true**, **false**$\}$, the program

```
taste_b := fun f ->
  if f(true, Ω) and
     f(Ω, true) and
     not(f(false, false))
  then b
  else true
```

Questions.

- Do we have $[\![$**taste_true**$]\!] = [\![$**taste_false**$]\!]$?
- Are **taste_true** and **taste_false** equivalent ?

A Taste of PCF

Motivation.

- Extend Finitary PCF with general recursion.
- Mathematically cleaner if an infinite type is assumed (say the natural numbers).

A Taste of PCF

Motivation.

- Extend Finitary PCF with general recursion.
- Mathematically cleaner if an infinite type is assumed (say the natural numbers).

The Language of PCF.

$$\begin{array}{lcll} \tau, \sigma & ::= & \dots & \mid \ \mathbf{nat} \\ t, u & ::= & \dots & \mid \ t\mathtt{+1} \ \mid \ t\mathtt{-1} \ \mid \ \mathtt{z?} \ \mid \ Y \ \mid \ \underline{n} \qquad \text{(for each } n \in \mathbb{N}\text{)} \end{array}$$

- Y is a *fixpoint* combinator:

$$Y\,t \quad = \quad t\,(Y\,t)$$

A Taste of PCF

Motivation.

- Extend Finitary PCF with general recursion.
- Mathematically cleaner if an infinite type is assumed (say the natural numbers).

The Language of PCF.

$$\begin{array}{rcllllllllllll} \tau, \sigma & ::= & \ldots & \mid & \mathbf{nat} \\ t, u & ::= & \ldots & \mid & t\mathtt{+1} & \mid & t\mathtt{-1} & \mid & \mathtt{z?} & \mid & Y & \mid & \underline{n} & \qquad \text{(for each } n \in \mathbb{N}\text{)} \end{array}$$

- Y is a *fixpoint* combinator:

$$Y\,t \quad = \quad t\,(Y\,t)$$

Examples.

- We could have defined

$$\Omega \quad := \quad Y(\mathtt{fun}\,x \to x)$$

A Taste of PCF

Motivation.

- Extend Finitary PCF with general recursion.
- Mathematically cleaner if an infinite type is assumed (say the natural numbers).

The Language of PCF.

$$\begin{array}{rclllllllllll} \tau, \sigma & ::= & \dots & \mid & \mathbf{nat} & & & & & & & & \\ t, u & ::= & \dots & \mid & t\texttt{+1} & \mid & t\texttt{-1} & \mid & \texttt{z?} & \mid & Y & \mid & \underline{n} \end{array} \qquad \text{(for each } n \in \mathbb{N}\text{)}$$

- Y is a *fixpoint* combinator:

$$Y\,t \quad = \quad t\,(Y\,t)$$

Examples.

- We could have defined

$$\Omega \quad := \quad Y(\mathbf{fun}\,x \to x)$$

- Addition

$$\begin{array}{lcl} \mathbf{add}\ \underline{0}\ u & = & u \\ \mathbf{add}\ t\texttt{+1}\ u & = & (\mathbf{add}\ t\ u)\texttt{+1} \end{array}$$

A Taste of PCF

Motivation.

- Extend Finitary PCF with general recursion.
- Mathematically cleaner if an infinite type is assumed (say the natural numbers).

The Language of PCF.

$$\begin{array}{lcllllllllll} \tau, \sigma & ::= & \dots & \mid & \mathbf{nat} \\ t, u & ::= & \dots & \mid & t\mathtt{+1} & \mid & t\mathtt{-1} & \mid & \mathtt{z?} & \mid & Y & \mid & \underline{n} \end{array} \qquad \text{(for each } n \in \mathbb{N}\text{)}$$

- Y is a *fixpoint* combinator:

$$Y\,t \quad = \quad t\,(Y\,t)$$

Examples.

- We could have defined

$$\Omega \quad := \quad Y(\mathtt{fun}\, x \to x)$$

- Addition

$$\begin{array}{lcl} \mathtt{add}\ \underline{0}\ u & = & u \\ \mathtt{add}\ t\mathtt{+1}\ u & = & (\mathtt{add}\ t\ u)\mathtt{+1} \end{array}$$

can be defined as

$$\mathtt{add} \quad := \quad Y\,\mathtt{add\_rec}$$

where

```
add_rec := fun f, x, y ->
  if (z? x) then y else (f x-1 y)+1
```

A Denotational Semantics for PCF ?

A Denotational Semantics for PCF ?

We Would Like

- each type τ to be interpreted as a set $[\![\tau]\!]$,

A Denotational Semantics for PCF ?

We Would Like

- each type τ to be interpreted as a set $[\![\tau]\!]$,
- a program t of type say $\sigma \to \tau$ to be interpreted as a function

$$[\![t]\!] \quad : \quad [\![\sigma]\!] \quad \longrightarrow \quad [\![\tau]\!]$$

A Denotational Semantics for PCF ?

We Would Like

- each type τ to be interpreted as a set $[\![\tau]\!]$,
- a program t of type say $\sigma \to \tau$ to be interpreted as a function

$$[\![t]\!] \quad : \quad [\![\sigma]\!] \quad \longrightarrow \quad [\![\tau]\!]$$

Difficulty.

- Equation

$$Y\,t \quad = \quad t\,(Y\,t)$$

  imposes

$$[\![Y]\!] \quad : \quad (S \to S) \quad \longrightarrow \quad S$$

A Denotational Semantics for PCF ?

We Would Like

- each type τ to be interpreted as a set $[\![\tau]\!]$,
- a program t of type say $\sigma \to \tau$ to be interpreted as a function

$$[\![t]\!] \quad : \quad [\![\sigma]\!] \quad \longrightarrow \quad [\![\tau]\!]$$

Difficulty.

- Equation

$$Y\,t \quad = \quad t\,(Y\,t)$$

  imposes

$$[\![Y]\!] \quad : \quad (S \to S) \quad \longrightarrow \quad S$$

Traditional Solution.

- Restrict $S \to S$ to the continuous functions for a suitable topology (cpos, Scott domains, etc).

A Denotational Semantics for PCF ?

We Would Like

- each type τ to be interpreted as a set $[\![\tau]\!]$,
- a program t of type say $\sigma \to \tau$ to be interpreted as a function

$$[\![t]\!] \quad : \quad [\![\sigma]\!] \quad \longrightarrow \quad [\![\tau]\!]$$

Difficulty.

- Equation

$$Y\,t \quad = \quad t\,(Y\,t)$$

imposes

$$[\![Y]\!] \quad : \quad (S \to S) \quad \longrightarrow \quad S$$

Traditional Solution.

- Restrict $S \to S$ to the continuous functions for a suitable topology (cpos, Scott domains, etc).

Gödel's System T.

- Restrict Y to recursion over $\mathbb{N}$:

$$\begin{array}{lcl} \mathtt{rec}\ u\ v\ \underline{0} & = & u \\ \mathtt{rec}\ u\ v\ \underline{t\mathtt{+1}} & = & v\ t\ (\mathtt{rec}\ u\ v\ t) \end{array}$$

A Denotational Semantics for PCF ?

We Would Like

- each type τ to be interpreted as a set $[\![\tau]\!]$,
- a program t of type say $\sigma \to \tau$ to be interpreted as a function

$$[\![t]\!] \quad : \quad [\![\sigma]\!] \quad \longrightarrow \quad [\![\tau]\!]$$

Difficulty.

- Equation

$$Y\, t \quad = \quad t\,(Y\, t)$$

imposes

$$[\![Y]\!] \quad : \quad (S \to S) \quad \longrightarrow \quad S$$

Traditional Solution.

- Restrict $S \to S$ to the continuous functions for a suitable topology (cpos, Scott domains, etc).

Gödel's System T.

- Restrict Y to recursion over $\mathbb{N}$:

$$\begin{array}{lcl} \mathbf{rec}\; u\; v\; \underline{0} & = & u \\ \mathbf{rec}\; u\; v\; t\mathbf{+1} & = & v\; t\; (\mathbf{rec}\; u\; v\; t) \end{array}$$

- Allows to see important basic techniques in a simple setting.

Rough Outline

Indicative Outline

Courses 1–6: (C. RIBA)

- Set-theoretic semantics of System T.
- Denotational semantics of PCF (cpos, logical relations).
- Further topics among:
  - Polymorphism (Girard-Reynolds System F).
  - Recursive types.
  - Intersection types.
  - Scott domains and PCF definability.
  - Results on the set-theoretic semantics of the simply-typed λ-calculus.

Indicative Outline

Courses 1–6: (C. Riba)

- Set-theoretic semantics of System T.
- Denotational semantics of PCF (cpos, logical relations).
- Further topics among:
  - Polymorphism (Girard-Reynolds System F).
  - Recursive types.
  - Intersection types.
  - Scott domains and PCF definability.
  - Results on the set-theoretic semantics of the simply-typed λ-calculus.

Courses 7–12: (P. Clairambault)

- Categories, functors and natural transformations.
- Cartesian closed categories and the λ-calculus.
- Monads.
- Further topics among:
  - Categorical models of linear logic.
  - Game semantics.

Indicative Outline

Courses 1–6: (C. RIBA)

- Set-theoretic semantics of System T.
- Denotational semantics of PCF (cpos, logical relations).
- Further topics among:
  - Polymorphism (Girard-Reynolds System F).
  - Recursive types.
  - Intersection types.
  - Scott domains and PCF definability.
  - Results on the set-theoretic semantics of the simply-typed λ-calculus.

Courses 7–12: (P. CLAIRAMBAULT)

- Categories, functors and natural transformations.
- Cartesian closed categories and the λ-calculus.
- Monads.
- Further topics among:
  - Categorical models of linear logic.
  - Game semantics.

Courses 13–: Survey of Some Active Research Topics.