# Formal Techniques for Software Engineering: Denotational Semantics

Rocco De Nicola

IMT Institute for Advanced Studies, Lucca
rocco.denicola@imtlucca.it

May 2013

Lesson 4

# Key motivations

Two main contributions of denotational semantics for programming languages:

1. Compositionality
2. Characterization of recursion

### Core ingredients



$$X = fX$$

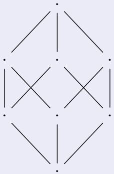
CPOs — fixpoints of functions

# Key motivations

Two main contributions of denotational semantics for programming languages:

1. Compositionality
2. Characterization of recursion

## Core ingredients



$$X = fX$$

CPOs    fixpoints of functions

# Functions Representation

**Big Questions**

- What is a function?
- How do we define a function?


**The $\lambda$-calculus**

We need a formal language for defining

- functions
- functions composition
- functions evaluation

$\lambda$-calculus was introduced in the thirties and permits describing *all computable functions*

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\,e\,] \rightsquigarrow \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

$succ \equiv \lambda x.\ x+1$ $\qquad succ\,3 \rightsquigarrow 3+1$ $\qquad succ\,succ \rightsquigarrow (\lambda x.\ x+1)+1$

$poly \equiv \lambda x.\ x^2 - 3{\cdot}x+1$ $\qquad poly\,5 \rightsquigarrow 5^2 - 3{\cdot}5+1$

$g \equiv \lambda x.\lambda y.\ x-y$ $\qquad g\,3\,4 \rightsquigarrow^2 3-4$ $\qquad g\,3 \rightsquigarrow \lambda y.\ 3-y$

$h \equiv \lambda y.\lambda x.\ x-y$ $\qquad h\,3\,4 \rightsquigarrow^2 4-3$ $\qquad h\,x \rightsquigarrow \lambda z.\ z-x$

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\ e\,] \rightsquigarrow \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

$succ \equiv \lambda x.\ x+1$ $\qquad succ\,3 \rightsquigarrow 3+1$ $\qquad succ\,succ \rightsquigarrow (\lambda x.\ x+1)+1$

$poly \equiv \lambda x.\ x^2 - 3\cdot x + 1$ $\qquad poly\,5 \rightsquigarrow 5^2 - 3\cdot 5 + 1$

$g \equiv \lambda x.\lambda y.\ x - y$ $\qquad g\,3\,4 \rightsquigarrow^2 3 - 4$ $\qquad g\,3 \rightsquigarrow \lambda y.\ 3 - y$

$h \equiv \lambda y.\lambda x.\ x - y$ $\qquad h\,3\,4 \rightsquigarrow^2 4 - 3$ $\qquad h\,x \rightsquigarrow \lambda z.\ z - x$

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow \ \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\ e\,] \ \rightsquigarrow\ \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

| | | |
|---|---|---|
| $succ \equiv \lambda x.\ x+1$ | $succ\,3 \ \rightsquigarrow\ 3+1$ | $succ\,succ \ \rightsquigarrow\ (\lambda x.\ x+1)+1$ |
| $poly \equiv \lambda x.\ x^2 - 3\cdot x + 1$ | $poly\,5 \ \rightsquigarrow\ 5^2 - 3\cdot 5 + 1$ | |
| $g \equiv \lambda x.\lambda y.\ x - y$ | $g\,3\,4 \ \rightsquigarrow^2\ 3 - 4$ | $g\,3 \ \rightsquigarrow\ \lambda y.\ 3 - y$ |
| $h \equiv \lambda y.\lambda x.\ x - y$ | $h\,3\,4 \ \rightsquigarrow^2\ 4 - 3$ | $h\,x \ \rightsquigarrow\ \lambda z.\ z - x$ |

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\,e\,] \rightsquigarrow \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

| | | |
|---|---|---|
| $succ \equiv \lambda x.\ x+1$ | $succ\,3 \rightsquigarrow 3+1$ | $succ\,succ \rightsquigarrow (\lambda x.\ x+1)+1$ |
| $poly \equiv \lambda x.\ x^2 - 3\cdot x + 1$ | $poly\,5 \rightsquigarrow 5^2 - 3\cdot 5 + 1$ | |
| $g \equiv \lambda x.\lambda y.\ x - y$ | $g\,3\,4 \rightsquigarrow^2 3 - 4$ | $g\,3 \rightsquigarrow \lambda y.\ 3 - y$ |
| $h \equiv \lambda y.\lambda x.\ x - y$ | $h\,3\,4 \rightsquigarrow^2 4 - 3$ | $h\,x \rightsquigarrow \lambda z.\ z - x$ |

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\ e\,] \ \rightsquigarrow\ \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

| | | |
|---|---|---|
| $succ \equiv \lambda x.\ x+1$ | $succ\,3 \ \rightsquigarrow\ 3+1$ | $succ\ succ \ \rightsquigarrow\ (\lambda x.\ x+1)+1$ |
| $poly \equiv \lambda x.\ x^2 - 3 \cdot x + 1$ | $poly\,5 \ \rightsquigarrow\ 5^2 - 3 \cdot 5 + 1$ | |
| $g \equiv \lambda x.\lambda y.\ x - y$ | $g\,3\,4 \ \rightsquigarrow^2\ 3 - 4$ | $g\,3 \ \rightsquigarrow\ \lambda y.\ 3 - y$ |
| $h \equiv \lambda y.\lambda x.\ x - y$ | $h\,3\,4 \ \rightsquigarrow^2\ 4 - 3$ | $h\,x \ \rightsquigarrow\ \lambda z.\ z - x$ |

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\,e\,] \rightsquigarrow \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

$succ \equiv \lambda x.\ x+1$ $\quad succ\,3 \rightsquigarrow 3+1$ $\quad succ\,succ \rightsquigarrow (\lambda x.\ x+1)+1$

$poly \equiv \lambda x.\ x^2 - 3\cdot x + 1$ $\quad poly\,5 \rightsquigarrow 5^2 - 3\cdot 5 + 1$

$g \equiv \lambda x.\lambda y.\ x - y$ $\quad g\,3\,4 \rightsquigarrow^2 3 - 4$ $\quad g\,3 \rightsquigarrow \lambda y.\ 3 - y$

$h \equiv \lambda y.\lambda x.\ x - y$ $\quad h\,3\,4 \rightsquigarrow^2 4 - 3$ $\quad h\,x \rightsquigarrow \lambda z.\ z - x$

$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow \ldots$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\,e\,] \ \rightsquigarrow\ \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

| | | |
|---|---|---|
| $succ \equiv \lambda x.\ x+1$ | $succ\,3 \ \rightsquigarrow\ 3+1$ | $succ\,succ \ \rightsquigarrow\ (\lambda x.\ x+1)+1$ |
| $poly \equiv \lambda x.\ x^2 - 3{\cdot}x + 1$ | $poly\,5 \ \rightsquigarrow\ 5^2 - 3{\cdot}5 + 1$ | |
| $g \equiv \lambda x.\lambda y.\ x - y$ | $g\,3\,4 \ \rightsquigarrow^2\ 3 - 4$ | $g\,3 \ \rightsquigarrow\ \lambda y.\ 3 - y$ |
| $h \equiv \lambda y.\lambda x.\ x - y$ | $h\,3\,4 \ \rightsquigarrow^2\ 4 - 3$ | $h\,x \ \rightsquigarrow\ \lambda z.\ z - x$ |

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\,e\,] \ \leadsto\ \mathcal{C}[\,d\{e \mapsto x\}\,]$ | ($e$ not a composition) |

E.g. :

| | | |
|---|---|---|
| $succ \equiv \lambda x.\ x+1$ | $succ\,3 \ \leadsto\ 3+1$ | $succ\,succ \ \leadsto\ (\lambda x.\ x+1)+1$ |
| $poly \equiv \lambda x.\ x^2 - 3{\cdot}x+1$ | $poly\,5 \ \leadsto\ 5^2 - 3{\cdot}5+1$ | |
| $g \equiv \lambda x.\lambda y.\ x - y$ | $g\,3\,4 \ \leadsto^2\ 3-4$ | $g\,3 \ \leadsto\ \lambda y.\ 3-y$ |
| $h \equiv \lambda y.\lambda x.\ x - y$ | $h\,3\,4 \ \leadsto^2\ 4-3$ | $h\,x \ \leadsto\ \lambda z.\ z-x$ |

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \leadsto\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \leadsto\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \leadsto\ \ldots$$

non $\beta$-reducible expressions are **canonical**

## $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\,e\,] \ \rightsquigarrow\ \mathcal{C}[\,d\{e \mapsto x\}\,]$ | ($e$ not a composition) |

E.g. :

| | | |
|---|---|---|
| $succ \equiv \lambda x.\ x+1$ | $succ\,3 \ \rightsquigarrow\ 3+1$ | $succ\,succ \ \rightsquigarrow\ (\lambda x.\ x+1)+1$ |
| $poly \equiv \lambda x.\ x^2 - 3\cdot x + 1$ | $poly\,5 \ \rightsquigarrow\ 5^2 - 3\cdot 5 + 1$ | |
| $g \equiv \lambda x.\lambda y.\ x - y$ | $g\,3\,4 \ \rightsquigarrow^2\ 3 - 4$ | $g\,3 \ \rightsquigarrow\ \lambda y.\ 3 - y$ |
| $h \equiv \lambda y.\lambda x.\ x - y$ | $h\,3\,4 \ \rightsquigarrow^2\ 4 - 3$ | $h\,x \ \rightsquigarrow\ \lambda z.\ z - x$ |

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\, e\,] \leadsto \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

$succ \equiv \lambda x.\ x+1$ $\quad succ\,3 \leadsto 3+1$ $\quad succ\,succ \leadsto (\lambda x.\ x+1)+1$

$poly \equiv \lambda x.\ x^2 - 3\cdot x + 1$ $\quad poly\,5 \leadsto 5^2 - 3\cdot 5 + 1$

$g \equiv \lambda x.\lambda y.\ x - y$ $\quad g\,3\,4 \leadsto^2 3 - 4$ $\quad g\,3 \leadsto \lambda y.\ 3 - y$

$h \equiv \lambda y.\lambda x.\ x - y$ $\quad h\,3\,4 \leadsto^2 4 - 3$ $\quad h\,x \leadsto \lambda z.\ z - x$

$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \leadsto (\lambda x.\,x\,x)(\lambda x.\,x\,x) \leadsto (\lambda x.\,x\,x)(\lambda x.\,x\,x) \leadsto \ldots$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\,e\,] \rightsquigarrow \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

| | | |
|---|---|---|
| $succ \equiv \lambda x.\ x+1$ | $succ\,3 \rightsquigarrow 3+1$ | $succ\,succ \rightsquigarrow (\lambda x.\ x+1)+1$ |
| $poly \equiv \lambda x.\ x^2 - 3\cdot x + 1$ | $poly\,5 \rightsquigarrow 5^2 - 3\cdot 5 + 1$ | |
| $g \equiv \lambda x.\lambda y.\ x - y$ | $g\,3\,4 \rightsquigarrow^2 3 - 4$ | $g\,3 \rightsquigarrow \lambda y.\ 3 - y$ |
| $h \equiv \lambda y.\lambda x.\ x - y$ | $h\,3\,4 \rightsquigarrow^2 4 - 3$ | $h\,x \rightsquigarrow \lambda z.\ z - x$ |

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow (\lambda x.\,x\,x)(\lambda x.\,x\,x) \rightsquigarrow \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\ e\,] \ \rightsquigarrow\ \mathcal{C}[\,d\{e \mapsto x\}\,]$ | ($e$ not a composition) |

E.g. :

| | | |
|---|---|---|
| $succ \equiv \lambda x.\ x+1$ | $succ\,3 \ \rightsquigarrow\ 3+1$ | $succ\ succ \ \rightsquigarrow\ (\lambda x.\ x+1)+1$ |
| $poly \equiv \lambda x.\ x^2 - 3\cdot x + 1$ | $poly\,5 \ \rightsquigarrow\ 5^2 - 3\cdot 5 + 1$ | |
| $g \equiv \lambda x.\lambda y.\ x - y$ | $g\,3\,4 \ \rightsquigarrow^2\ 3 - 4$ | $g\,3 \ \rightsquigarrow\ \lambda y.\ 3 - y$ |
| $h \equiv \lambda y.\lambda x.\ x - y$ | $h\,3\,4 \ \rightsquigarrow^2\ 4 - 3$ | $h\,x \ \rightsquigarrow\ \lambda z.\ z - x$ |

$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ \ldots$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\, g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\, e\,] \ \rightsquigarrow\ \mathcal{C}[\,d\{e \mapsto x\}\,]$ | *($e$ not a composition)* |

E.g. :

$succ \equiv \lambda x.\ x+1 \qquad succ\,3 \ \rightsquigarrow\ 3+1 \qquad succ\,succ \ \rightsquigarrow\ (\lambda x.\ x+1)+1$

$poly \equiv \lambda x.\ x^2 - 3 \cdot x + 1 \qquad poly\,5 \ \rightsquigarrow\ 5^2 - 3 \cdot 5 + 1$

$g \equiv \lambda x.\lambda y.\ x - y \qquad g\,3\,4 \ \rightsquigarrow^2\ 3-4 \qquad g\,3 \ \rightsquigarrow\ \lambda y.\ 3-y$

$h \equiv \lambda y.\lambda x.\ x - y \qquad h\,3\,4 \ \rightsquigarrow^2\ 4-3 \qquad h\,x \ \rightsquigarrow\ \lambda z.\ z-x$

$$(\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ (\lambda x.\,x\,x)(\lambda x.\,x\,x) \ \rightsquigarrow\ \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus (a quick and dirty intro to relevant notation)

A calculus of function composition

| | | |
|---|---|---|
| $\lambda$-abstraction | $\lambda x.\ e$ | $x$ is a variable and $e$ an expression |
| composition | $f \circ g$ | often denoted as $f\,g$ |
| $\beta$-reduction | $\mathcal{C}[\,(\lambda x.\ d)\ e\,] \ \leadsto\ \mathcal{C}[\,d\{e \mapsto x\}\,]$ | ($e$ not a composition) |

E.g. :

| | | |
|---|---|---|
| $succ \equiv \lambda x.\ x+1$ | $succ\,3 \ \leadsto\ 3+1$ | $succ\ succ \ \leadsto\ (\lambda x.\ x+1)+1$ |
| $poly \equiv \lambda x.\ x^2 - 3 \cdot x + 1$ | $poly\,5 \ \leadsto\ 5^2 - 3 \cdot 5 + 1$ | |
| $g \equiv \lambda x. \lambda y.\ x - y$ | $g\,3\,4 \ \leadsto^2\ 3 - 4$ | $g\,3 \ \leadsto\ \lambda y.\ 3 - y$ |
| $h \equiv \lambda y. \lambda x.\ x - y$ | $h\,3\,4 \ \leadsto^2\ 4 - 3$ | $h\,x \ \leadsto\ \lambda z.\ z - x$ |

$$(\lambda x.\, x\,x)(\lambda x.\, x\,x) \ \leadsto\ (\lambda x.\, x\,x)(\lambda x.\, x\,x) \ \leadsto\ (\lambda x.\, x\,x)(\lambda x.\, x\,x) \ \leadsto\ \ldots$$

non $\beta$-reducible expressions are **canonical**

# $\lambda$-calculus

- $0 \equiv \lambda f.\lambda x.\ x$
- $1 \equiv \lambda f.\lambda x.\ f\,x$
- $2 \equiv \lambda f.\lambda x.\ f\,f\,x$
- $3 \equiv \lambda f.\lambda x.\ f\,f\,f\,x$
- $\ldots$
- $succ \ \equiv\ \lambda n.\lambda f.\lambda x.\ f\,(n\,f\,x)$
- $plus \ \equiv\ \lambda m.\lambda n.\lambda f.\lambda x.\ m\,f\,(n\,f\,x) \ \equiv\ \lambda m.\lambda n.\ m\ succ\ n$

- *true* $\equiv \lambda x.\lambda y.\ x$
- *false* $\equiv \lambda x.\lambda y.\ y$
- *and* $\equiv \lambda p.\lambda q.\ p\ q\ p$
- *or* $\equiv \lambda p.\lambda q.\ p\ p\ q$
- *not* $\equiv \lambda p.\lambda a.\lambda b.\ p\ b\ a$
- *cond* $\equiv \lambda p.\lambda a.\lambda b.\ p\ a\ b$

# An Example Computation

## Conditional Statement

$$cond(x,y,z) = \begin{cases} y & \text{if } x = \mathbf{tt} \\ z & \text{if } x = \mathbf{ff} \end{cases}$$

## Conditional Statement in $\lambda$-calculus

- $\mathbf{tt} \equiv \lambda m.\, \lambda n.\, m$
- $\mathbf{ff} \equiv \lambda m.\, \lambda n.\, n$ *same definition of 0*
- $cond \equiv \lambda a.\, \lambda b.\, \lambda c.\, a\ b\ c$

## Computing $cond(e,y,z)$

- Assume $e \leadsto^* \mathbf{tt}$

$$cond(e,y,z) \leadsto^* cond(\mathbf{tt},y,z) \equiv (\lambda a.\lambda b.\lambda c.\, a\ b\ c)\ (\lambda m.\, \lambda n.\, m)\ y\ z$$
$$\leadsto^* (\lambda m.\lambda n.\, m)\ y\ z \leadsto^* y$$

- Check the case $e \leadsto^* \mathbf{ff}$

# $\lambda$-calculus

**Every $\lambda$-object is a function**

- numbers, boolean constants, states
- arithmetic functions ($+$,$*$,...)
- boolean predicates ($\leq$, $\wedge$,...).
- ...

**Main Constructors**

All computable functions can be

- defined by means of (more elementary)functions composition
- evaluated by means of $\beta$-reductions.

**Recursion**

Recursively defined functions (functions using their own definition) are dealt by means of so called fixed point theory.

# $\lambda$-calculus

## Every $\lambda$-object is a function

- numbers, boolean constants, states
- arithmetic functions ($+$,$*$,$\ldots$)
- boolean predicates ($\leq$, $\wedge$,$\ldots$).
- $\ldots$

## Main Constructors

All computable functions can be

- defined by means of (more elementary)functions composition
- evaluated by means of $\beta$-reductions.

## Recursion

Recursively defined functions (functions using their own definition) are dealt by means of so called fixed point theory.

# $\lambda$-calculus

### Every $\lambda$-object is a function

- numbers, boolean constants, states
- arithmetic functions ($+$,$*$,...)
- boolean predicates ($\leq$, $\wedge$,...).
- ...

### Main Constructors

All computable functions can be

- defined by means of (more elementary)functions composition
- evaluated by means of $\beta$-reductions.

### Recursion

Recursively defined functions (functions using their own definition) are dealt by means of so called fixed point theory.

# Recap of basic assumptions

## Syntactic categories

- $\mathbf{Num}$, numerals
- $\mathbf{Var}$, variables
- $\mathbf{Aexp}$, arithmetic expressions
- $\mathbf{Bexp}$, boolean expressions
- $\mathbf{Stm}$, statements

## Semantic Functions

We are assume availability of some ($\lambda$-defined) semantic functions:

- $\mathcal{N}$ : $\mathbf{Num} \to \mathbf{Z}$
- $+$ : $\mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$ (same for $-$, $*$,...)
- $\leq$ : $\mathbf{Z} \times \mathbf{Z} \to \{\mathbf{tt}, \mathbf{ff}\}$ (same for $=$,$\neq$,$<$,$\geq$,...)

## State Function

- $s$ : $\mathbf{Var} \to \mathbf{Z}$

# Recap of basic assumptions

## Syntactic categories

- **Num**, numerals
- **Var**, variables
- **Aexp**, arithmetic expressions
- **Bexp**, boolean expressions
- **Stm**, statements

## Semantic Functions

We are assume availability of some ($\lambda$-defined) semantic functions:

- $\mathcal{N}\ : \mathbf{Num} \rightarrow \mathbf{Z}$
- $+\ : \mathbf{Z} \times \mathbf{Z} \rightarrow \mathbf{Z}$ (same for $-$, $*$,...)
- $\leq : \mathbf{Z} \times \mathbf{Z} \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ (same for $=$,$\neq$,$<$,$\geq$,...)

## State Function

- $s\ :\ \mathbf{Var} \rightarrow \mathbf{Z}$

# Recap of basic assumptions

## Syntactic categories

- **Num**, numerals
- **Var**, variables
- **Aexp**, arithmetic expressions
- **Bexp**, boolean expressions
- **Stm**, statements

## Semantic Functions

We are assume availability of some ($\lambda$-defined) semantic functions:

- $\mathcal{N}\ : \mathbf{Num} \rightarrow \mathbf{Z}$
- $+\ : \mathbf{Z} \times \mathbf{Z} \rightarrow \mathbf{Z}$ (same for $-$, $*$,...)
- $\leq : \mathbf{Z} \times \mathbf{Z} \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$ (same for $=$,$\neq$,$<$,$\geq$,...)

## State Function

- $s\ :\ \mathbf{Var}\ \rightarrow\ \mathbf{Z}$

# Compositionality - Example 1

Given a function

$$[\![B]\!] : \mathbf{State} \to \{\mathbf{tt}, \mathbf{ff}\}$$

and two partial functions

$$[\![C]\!], [\![D]\!] : \mathbf{State} \hookrightarrow \mathbf{State}$$

we define the function

$$[\![\texttt{if}\, B\, \texttt{then}\, C\, \texttt{else}\, D]\!] : \mathbf{State} \hookrightarrow \mathbf{State}$$

$$[\![\texttt{if}\, B\, \texttt{then}\, C\, \texttt{else}\, D]\!] = \lambda s.\, cond([\![B]\!]\, s, [\![C]\!]\, s, [\![D]\!]\, s) \qquad (s \in \mathbf{State})$$

where

$$cond(x,y,z) = \begin{cases} y & \text{if } x = \mathbf{tt} \\ z & \text{if } x = \mathbf{ff} \end{cases}$$ is a function composing the semantics of the sub-commands

# Compositionality - Example 1

Given a function

$$[\![B]\!] : \mathbf{State} \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$$

and two partial functions

$$[\![C]\!], [\![D]\!] : \mathbf{State} \hookrightarrow \mathbf{State}$$

we define the function

$$[\![\texttt{if}\, B\, \texttt{then}\, C\, \texttt{else}\, D]\!] : \mathbf{State} \hookrightarrow \mathbf{State}$$

$$[\![\texttt{if}\, B\, \texttt{then}\, C\, \texttt{else}\, D]\!] \quad = \quad \lambda s.\, cond([\![B]\!]\, s, [\![C]\!]\, s, [\![D]\!]\, s) \qquad (s \in \mathbf{State})$$

where

$$cond(x, y, z) = \begin{cases} y & \text{if } x = \mathbf{tt} \\ z & \text{if } x = \mathbf{ff} \end{cases}$$ is a function composing the semantics of the sub-commands

# Compositionality - Example 2

## Sequential composition

Given partial functions $[\![C]\!], [\![D]\!] : \textbf{State} \hookrightarrow \textbf{State}$

$$[\![C\,;D]\!] \;=\; [\![D]\!] \circ [\![C]\!] \;=\; \lambda s.\ [\![D]\!]\,[\![C]\!]\,s \qquad (s \in \textbf{State})$$

Comparing the denotational with the operational approach

## Structural semantics

$$[\mathrm{comp}^1_{\mathrm{sos}}] \quad \frac{\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_1';S_2,\ s'\rangle}$$

$$[\mathrm{comp}^2_{\mathrm{sos}}] \quad \frac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle}$$

## Natural semantics

$$[\mathrm{comp}_{\mathrm{ns}}] \quad \frac{\langle S_1,\ s\rangle \rightarrow s',\ \langle S_2,\ s'\rangle \rightarrow s''}{\langle S_1;S_2,\ s\rangle \rightarrow s''}$$

# Compositionality - Example 2

## Sequential composition

Given partial functions $[\![C]\!], [\![D]\!] : \ \mathbf{State} \hookrightarrow \mathbf{State}$

$$[\![C\,;D]\!] \;=\; [\![D]\!] \circ [\![C]\!] \;=\; \lambda s.\ [\![D]\!]\,[\![C]\!]\,s \qquad (s \in \mathbf{State})$$

Comparing the denotational with the operational approach

## Structural semantics

$$[\text{comp}^1_{\text{sos}}] \qquad \frac{\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_1';S_2,\ s'\rangle}$$

$$[\text{comp}^2_{\text{sos}}] \qquad \frac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1;S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle}$$

## Natural semantics

$$[\text{comp}_{\text{ns}}] \qquad \frac{\langle S_1,\ s\rangle \to s',\ \langle S_2,\ s'\rangle \to s''}{\langle S_1;S_2,\ s\rangle \to s''}$$

# Semantics of arithmetic expressions

$$\mathcal{A} : \textbf{Aexp} \rightarrow (\ \textbf{State} \rightarrow \textbf{Z}\ )$$

By structural induction on the syntax of arithmetic expressions (equivalently, by case analysis on the outermost operator).

$$\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \cdot \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{aligned}$$

We could have used the $\lambda$-notation:

- $\mathcal{A}[\![n]\!] = \lambda s.\ \mathcal{N}[\![n]\!]$
- $\mathcal{A}[\![x]\!] = \lambda s.\ s(x)$
- ...

# Semantics of arithmetic expressions

$$\mathcal{A} : \mathbf{Aexp} \rightarrow (\mathbf{State} \rightarrow \mathbf{Z})$$

By structural induction on the syntax of arithmetic expressions (equivalently, by case analysis on the outermost operator).

$$\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \cdot \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{aligned}$$

We could have used the $\lambda$-notation:

- $\mathcal{A}[\![n]\!] = \lambda s.\ \mathcal{N}[\![n]\!]$
- $\mathcal{A}[\![x]\!] = \lambda s.\ s(x)$
- ...

# Semantics of boolean expressions

$$\mathcal{B} : \mathbf{Bexp} \to (\ \mathbf{State} \ \to \{\mathbf{tt}, \mathbf{ff}\}\ )$$

By structural induction on the syntax of boolean expressions.

$$\begin{array}{rcl}
\mathcal{B}[\![\texttt{false}]\!]s & = & \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s \neq \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s \leq \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg\ b]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]s = \mathbf{ff} \end{cases}
\end{array}$$

# Denotational Semantics of While

$$\mathcal{S}_{ds} : \quad Prog \quad \rightarrow \quad (\textbf{State} \hookrightarrow \textbf{State})$$

$$\mathcal{S}_{\text{ds}}[\![x := a]\!]s = s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\mathcal{S}_{\text{ds}}[\![\texttt{skip}]\!] = \text{id}$$

$$\mathcal{S}_{\text{ds}}[\![S_1 ; S_2]\!] = \mathcal{S}_{\text{ds}}[\![S_2]\!] \circ \mathcal{S}_{\text{ds}}[\![S_1]\!]$$

$$\mathcal{S}_{\text{ds}}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!] = \text{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\text{ds}}[\![S_1]\!], \mathcal{S}_{\text{ds}}[\![S_2]\!])$$

$$\mathcal{S}_{\text{ds}}[\![\texttt{while } b \texttt{ do } S]\!] = \text{FIX } F$$

$$\text{where } F\ g = \text{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{\text{ds}}[\![S]\!], \text{id})$$

$$id \equiv \lambda s.\ s$$

# Semantics of while-do: details

Given:

- a function $\mathcal{A}[\![b]\!] : \mathbf{State} \to \{\mathbf{tt}, \mathbf{ff}\}$
- a partial function $\mathcal{S}_{ds}[\![C]\!] : \mathbf{State} \hookrightarrow \mathbf{State}$

we let: $\mathcal{S}_{ds}[\![\, \texttt{while}\ b\ \texttt{do}\ C \,]\!] \ = \ \mathit{fix}\ F_{b,C}$

where: $F_{b,C} = \lambda w.\, \lambda s.\, \mathit{cond}(\, [\![b]\!]\, s,\ w\, [\![C]\!]\, s,\ s\,)$

$\mathit{fix}\ F_{b,C}$
denotes a partial function $[\![W]\!] : \mathbf{State} \hookrightarrow \mathbf{State}$ such that:

$$[\![W]\!] \ = \ F_{b,C}\, [\![W]\!]$$

Questions:

- Does this equation have solutions?
- How many solutions does it have?
- Which one should we take?

# Semantics of while-do: details

Given:

- a function $\mathcal{A}[\![b]\!] : \textbf{State} \to \{\textbf{tt}, \textbf{ff}\}$
- a partial function $\mathcal{S}_{ds}[\![C]\!] : \textbf{State} \hookrightarrow \textbf{State}$

we let: $\mathcal{S}_{ds}[\![\, \texttt{while}\ b\ \texttt{do}\ C \,]\!] = \mathit{fix}\ F_{b,C}$

where: $F_{b,C} = \lambda w.\, \lambda s.\, \mathit{cond}(\, [\![b]\!]\, s,\ w\, [\![C]\!]\, s,\ s\,)$

$\mathit{fix}\ F_{b,C}$
denotes a partial function $[\![W]\!] : \textbf{State} \hookrightarrow \textbf{State}$ such that:

$$[\![W]\!] = F_{b,C}\, [\![W]\!]$$

Questions:

- Does this equation have solutions?
- How many solutions does it have?
- Which one should we take?

# Semantics of while-do: an example

$$\mathcal{S}_{ds}[\![\, \texttt{while}\ x > 0\ \texttt{do}\ (\, y := x * y;\, x := x - 1\,)\,]\!] \;=\; \mathit{fix}\ F_{b,C}$$

we take as given

$\mathbf{State} = \{x, y\} \to \mathbf{Z}$ *the content of the memory at locations $x$ and $y$*

we look for a solution $X$ to the equation

$$X \;=\; \lambda s.\, cond(\,[\![\, x{>}0 \,]\!]\, s,\; X\, [\![\, y := x{*}y;\, x := x - 1 \,]\!]\, s,\; s\,)$$

Note that $X$ is a function in $\mathbf{State} \hookrightarrow \mathbf{State}$

How do we calculate the fixed point $X$?

# Semantics of while-do: an example

$$\mathcal{S}_{ds}[\![\,\texttt{while}\; x > 0\; \texttt{do}\; (\, y := x * y ; x := x - 1\,)\,]\!] \;=\; \mathit{fix}\; F_{b,C}$$

we take as given

$$\mathbf{State} = \{x, y\} \to \mathbf{Z}$$ *the content of the memory at locations $x$ and $y$*

we look for a solution $X$ to the equation

$$X \;=\; \lambda s.\, cond(\,[\![ x{>}0 ]\!]\, s,\; X\,[\![ y{:=}x{*}y ; x{:=}x{-}1 ]\!]\, s,\; s\,)$$

Note that $X$ is a function in $\mathbf{State} \hookrightarrow \mathbf{State}$

How do we calculate the fixed point $X$?

# Semantics of while-do: an example

$$\mathcal{S}_{ds}[\![\texttt{while}\ x > 0\ \texttt{do}\ (\,y := x * y; x := x - 1\,)\,]\!] \;=\; \mathit{fix}\ F_{b,C}$$

we take as given

$$\mathbf{State} = \{x, y\} \rightarrow \mathbf{Z}$$ *the content of the memory at locations $x$ and $y$*

we look for a solution $X$ to the equation

$$X \;=\; \lambda s.\, cond(\,[\![x{>}0]\!]\,s,\; X\,[\![y := x{*}y; x := x - 1]\!]\,s,\; s\,)$$

Note that $X$ is a function in $\mathbf{State} \hookrightarrow \mathbf{State}$

How do we calculate the fixed point $X$?

# Semantics of while-do: an example

$$\mathcal{S}_{ds}[\![ \texttt{while}\ x > 0\ \texttt{do}\ (y := x * y ; x := x - 1)]\!] = \mathit{fix}\ F_{b,C}$$

we take as given

$$\mathbf{State} = \{x, y\} \to \mathbf{Z}$$ *the content of the memory at locations $x$ and $y$*

we look for a solution $X$ to the equation

$$X = \lambda s.\ \mathit{cond}([\![ x>0 ]\!]\, s,\ X [\![ y := x*y ; x := x-1 ]\!]\, s,\ s)$$

Note that $X$ is a function in $\mathbf{State} \hookrightarrow \mathbf{State}$

How do we calculate the fixed point $X$?

# Semantics of while-do: an example

$$\mathcal{S}_{ds}[\![\texttt{while}\ x > 0\ \texttt{do}\ (\,y := x * y; x := x - 1\,)\,]\!] \;=\; \mathit{fix}\ F_{b,C}$$

we take as given

$$\mathbf{State} = \{x, y\} \to \mathbf{Z}$$ *the content of the memory at locations $x$ and $y$*

we look for a solution $X$ to the equation

$$X \;=\; \lambda s.\, cond(\,[\![\,x{>}0\,]\!]\,s,\ X\,[\![\,y := x{*}y; x := x - 1\,]\!]\,s,\ s\,)$$

Note that $X$ is a function in $\mathbf{State} \hookrightarrow \mathbf{State}$

How do we calculate the fixed point $X$?

# Fixed Points

## A recursive function

$f = \lambda x.(x = 0) \to 1, f(x+1)$

- $f$ indicates any function that yields 1 on argument 0, but its value for the other arguments is no specified.
- to compute $f$ on $x$, check whether $x = 0$, if so put $f(x) = 1$; otherwise evaluate $f$ on $x + 1$.

## A possible solution

If $\perp$ denotes absence of information then $g \equiv \lambda x.\ (x = 0) \to 1, \perp$ could be the solution of the equation defining f

$$
\begin{aligned}
&\lambda x.(x = 0) \to 1, \underbrace{(\lambda x.(x = 0) \to 1, \perp)}_{g}(x+1) \\
&\quad = \lambda x.(x = 0) \to 1, (x + 1 = 0) \to 1, \perp \\
&\quad = \lambda x.(x = 0) \to 1, \perp \qquad \textit{for no } x \geq 0 \textit{ we have } x + 1 = 0 \\
&\quad \equiv g.
\end{aligned}
$$

# Fixed Points

However, ... each function of the form

$$g_k \equiv \lambda x.(x = 0) \rightarrow 1, k$$

is a solution of the equation for $f$, whichever $k \in \mathbb{N}$ we take.

- Among all solutions, $g$ corresponds to the results obtained from the computational interpretation of $f$.
- To evaluate $f$ for a generic $x > 0$, espand the body of $f$ to discover that it is necessary to evaluate it on $x + 1$, then on $x + 2$, and so on ....
- $g$ is less defined than each $g_k$; indeed $g$ is defined only on 0 and for this value all $g_k$ take the same value:

$$\forall k. \, g(0) = g_k(0).$$

# Fixed Points

Given a function $f : D \to D$, the *fixed point* of $f$ is any element $d \in D$ such that $fd = d$.

## Examples

- The solution of $x = 2x + 1$ is $-1$, i.e, the fixed point of the function: $f \equiv \lambda x.2x + 1$
- Function $\tau : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$ takes a function as arguments and yields another function:

$$\tau \equiv \lambda f.(\lambda x.(x = 0) \to 1, f(x + 1).$$

- Consider

$$\tau_{fact} = \lambda f.\lambda x.(x = 0) \to 1, x * f(x - 1)$$

  If we let

$$fact = \lambda x.(x = 0) \to 1, x * fact(x - 1)$$

  we have

$$\tau_{fact}(fact) = fact$$

# Fixed point

## Ordering Functions

let $f, g : D \to D'$ be two functions.

$$f \sqsubseteq g \quad \Leftrightarrow \quad \forall x.\ \text{if } f(x) \text{ is defined then } f(x) = g(x).$$

We then say that $f$ approximates $g$ or $f$ is less defined than $g$

## Minimal fixed points

The fixed point that is less defined than all the others (minimal fixed point) is the one that corresponds to the operational intuition behind functions specifications.

## Finding fixed points

- $\Omega \equiv \lambda x.\ \bot$, is the function undefined everywhere and represents the worst approximation of every function.
- To calculate fixed points we make use of $\Omega$:

$$\tau\ \Omega = \lambda x.(x = 0) \to 1, \Omega(x + 1) = g.$$

# Fixed point

## Ordering Functions

Solution $g$ is obtained by applying $\tau$ to its approximands!
Consider $\tau_{fact}$, then $Fact_1 \equiv \tau_{fact}\ \Omega$ is

- $Fact_1 \equiv \tau_{fact}\ \Omega = \lambda x.(x = 0) \rightarrow 1, x * \Omega(x-1) = \lambda x.(x = 0) \rightarrow 1, \perp$
  $Fact_1$ is not the fixed point of $\tau_{fact}$ but a function more defined than $\Omega$. It is a better approximation than $\Omega$ of factorial function.
- The sequence
  $$Fact_2 \equiv \tau_{fact}\ Fact_1,\ \ Fact_3 \equiv \tau_{fact}\ Fact_2, \ldots$$
  is a chain of better and better approximations of factorial whose limit is the factorial function.
- The minimum fixed point of the specification of factorial function is obtained by a sequence of approximation steps.
- Each approximation is finitely represented and is, obviously, non recursive.

# Semantics of while-do: an example - continued

$$f = \lambda Z.\, \lambda s.\, cond(\, [\![ x{>}0 ]\!]\, s,\; Z\, [\![ y{:=}x{*}y; x{:=}x-1 ]\!]\, s,\; s\,)$$

We look for $X$ such that $X = f\,X$ and we start from $\Omega$

$$f^1 = f\Omega = \begin{cases} (x,y) & \text{if } x \leq 0 \\ \bot & \text{if } x \geq 1 \end{cases}$$

$$f^2 = \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, 1 * y) & \text{if } x = 1 \\ \bot & \text{if } x \geq 2 \end{cases}$$

$$f^3 = \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, 1 * y) & \text{if } x = 1 \\ (0, 2 * y) & \text{if } x = 2 \\ \bot & \text{if } x \geq 3 \end{cases}$$

$$f^4 = \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, 1 * y) & \text{if } x = 1 \\ (0, 2 * y) & \text{if } x = 2 \\ (0, 6 * y) & \text{if } x = 3 \\ \bot & \text{if } x \geq 4 \end{cases}$$

$$\vdots$$

$$f^{n+1} = \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, x! * y) & \text{if } 1 \leq x \leq n \\ \bot & \text{if } x \geq n+1 \end{cases}$$

$$\Omega \sqsubseteq f^1 \sqsubseteq f^2 \sqsubseteq f^3 \sqsubseteq \ldots \sqsubseteq f^n \sqsubseteq \ldots$$

# Semantics of while-do: an example - continued

$$f \;=\; \lambda Z.\, \lambda s.\, cond(\, [\![\, x{>}0 \,]\!]\, s,\; Z\, [\![\, y{:=}x{*}y;\, x{:=}x-1 \,]\!]\, s,\; s\,)$$

We look for $X$ such that $X = f\,X$ and we start from $\Omega$

$$f^1 = f\Omega = \begin{cases} (x,y) & \text{if } x \le 0 \\ \bot & \text{if } x \ge 1 \end{cases}$$

$$f^2 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1 * y) & \text{if } x = 1 \\ \bot & \text{if } x \ge 2 \end{cases}$$

$$f^3 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1 * y) & \text{if } x = 1 \\ (0, 2 * y) & \text{if } x = 2 \\ \bot & \text{if } x \ge 3 \end{cases}$$

$$f^4 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1 * y) & \text{if } x = 1 \\ (0, 2 * y) & \text{if } x = 2 \\ (0, 6 * y) & \text{if } x = 3 \\ \bot & \text{if } x \ge 4 \end{cases}$$

$$\vdots$$

$$f^{n+1} = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, x! * y) & \text{if } 1 \le x \le n \\ \bot & \text{if } x \ge n+1 \end{cases}$$

$$\Omega \sqsubseteq f^1 \sqsubseteq f^2 \sqsubseteq f^3 \sqsubseteq \ldots \sqsubseteq f^n \sqsubseteq \ldots$$

# Semantics of while-do: an example - continued

$$f \;=\; \lambda Z.\, \lambda s.\, cond(\, [\![ x>0 ]\!]\, s,\; Z\, [\![ y:=x*y; x:=x-1 ]\!]\, s,\; s\,)$$

We look for $X$ such that $X = f\,X$ and we start from $\Omega$

$$f^1 \;=\; f\Omega \;=\; \begin{cases} (x,y) & \text{if } x \leq 0 \\ \bot & \text{if } x \geq 1 \end{cases}$$

$$f^2 \;=\; \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, 1*y) & \text{if } x = 1 \\ \bot & \text{if } x \geq 2 \end{cases}$$

$$f^3 \;=\; \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ \bot & \text{if } x \geq 3 \end{cases}$$

$$f^4 \;=\; \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ (0, 6*y) & \text{if } x = 3 \\ \bot & \text{if } x \geq 4 \end{cases}$$

$\vdots$

$$f^{n+1} \;=\; \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, x!*y) & \text{if } 1 \leq x \leq n \\ \bot & \text{if } x \geq n+1 \end{cases}$$

$$\Omega \sqsubseteq f^1 \sqsubseteq f^2 \sqsubseteq f^3 \sqsubseteq \ldots \sqsubseteq f^n \sqsubseteq \ldots$$

# Semantics of while-do: an example - continued

$$f \;=\; \lambda Z.\, \lambda s.\, cond(\, [\![ x>0 ]\!]\, s,\; Z\, [\![ y:=x*y; x:=x-1 ]\!]\, s,\; s\,)$$

We look for $X$ such that $X = f\,X$ and we start from $\Omega$

$$f^1 = f\Omega = \begin{cases} (x,y) & \text{if } x \le 0 \\ \bot & \text{if } x \ge 1 \end{cases}$$

$$f^2 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ \bot & \text{if } x \ge 2 \end{cases}$$

$$f^3 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ \bot & \text{if } x \ge 3 \end{cases}$$

$$f^4 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ (0, 6*y) & \text{if } x = 3 \\ \bot & \text{if } x \ge 4 \end{cases}$$

$$\vdots$$

$$f^{n+1} = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, x!*y) & \text{if } 1 \le x \le n \\ \bot & \text{if } x \ge n+1 \end{cases}$$

$$\Omega \sqsubseteq f^1 \sqsubseteq f^2 \sqsubseteq f^3 \sqsubseteq \ldots \sqsubseteq f^n \sqsubseteq \ldots$$

# Semantics of while-do: an example - continued

$$f = \lambda Z.\, \lambda s.\, cond(\,[\![\, x{>}0 \,]\!]\, s,\; Z\,[\![\, y{:=}x{*}y\,;\, x{:=}x{-}1 \,]\!]\, s,\; s\,)$$

We look for $X$ such that $X = f\,X$ and we start from $\Omega$

$$f^1 = f\Omega = \begin{cases} (x,y) & \text{if } x \le 0 \\ \bot & \text{if } x \ge 1 \end{cases}$$

$$f^2 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ \bot & \text{if } x \ge 2 \end{cases}$$

$$f^3 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ \bot & \text{if } x \ge 3 \end{cases}$$

$$f^4 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ (0, 6*y) & \text{if } x = 3 \\ \bot & \text{if } x \ge 4 \end{cases}$$

$\vdots$

$$f^{n+1} = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, x!*y) & \text{if } 1 \le x \le n \\ \bot & \text{if } x \ge n+1 \end{cases}$$

$$\Omega \sqsubseteq f^1 \sqsubseteq f^2 \sqsubseteq f^3 \sqsubseteq \ldots \sqsubseteq f^n \sqsubseteq \ldots$$

# Semantics of while-do: an example - continued

$$f \;=\; \lambda Z.\, \lambda s.\, cond(\,[\![\, x{>}0 \,]\!]\, s,\; Z\,[\![\, y{:=}x{*}y\,;\, x{:=}x-1 \,]\!]\, s,\; s\,)$$

We look for $X$ such that $X = f\,X$ and we start from $\Omega$

$$f^1 = f\Omega = \begin{cases} (x,y) & \text{if } x \leq 0 \\ \bot & \text{if } x \geq 1 \end{cases}$$

$$f^2 = \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, 1*y) & \text{if } x = 1 \\ \bot & \text{if } x \geq 2 \end{cases}$$

$$f^3 = \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ \bot & \text{if } x \geq 3 \end{cases}$$

$$f^4 = \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ (0, 6*y) & \text{if } x = 3 \\ \bot & \text{if } x \geq 4 \end{cases}$$

$$\vdots$$

$$f^{n+1} = \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, x!*y) & \text{if } 1 \leq x \leq n \\ \bot & \text{if } x \geq n+1 \end{cases}$$

$$\Omega \sqsubseteq f^1 \sqsubseteq f^2 \sqsubseteq f^3 \sqsubseteq \ldots \sqsubseteq f^n \sqsubseteq \ldots$$

# Semantics of while-do: an example - continued

$$f = \lambda Z.\,\lambda s.\,cond(\,[\![x{>}0]\!]\,s,\ Z\,[\![y{:=}x{*}y;x{:=}x-1]\!]\,s,\ s\,)$$

We look for $X$ such that $X = f\,X$ and we start from $\Omega$

$$f^1 = f\Omega = \begin{cases} (x,y) & \text{if } x \le 0 \\ \bot & \text{if } x \ge 1 \end{cases}$$

$$f^2 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ \bot & \text{if } x \ge 2 \end{cases}$$

$$f^3 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ \bot & \text{if } x \ge 3 \end{cases}$$

$$f^4 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ (0, 6*y) & \text{if } x = 3 \\ \bot & \text{if } x \ge 4 \end{cases}$$

$$\vdots$$

$$f^{n+1} = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, x!*y) & \text{if } 1 \le x \le n \\ \bot & \text{if } x \ge n+1 \end{cases}$$

$$\Omega \sqsubseteq f^1 \sqsubseteq f^2 \sqsubseteq f^3 \sqsubseteq \ldots \sqsubseteq f^n \sqsubseteq \ldots$$

# Semantics of while-do: an example - continued

$$f \;=\; \lambda Z.\,\lambda s.\,cond(\,[\![\,x{>}0\,]\!]\,s,\; Z\,[\![\,y{:=}x{*}y;x{:=}x-1\,]\!]\,s,\; s\,)$$

We look for $X$ such that $X = f\,X$ and we start from $\Omega$

$$f^1 = f\Omega = \begin{cases} (x,y) & \text{if } x \le 0 \\ \bot & \text{if } x \ge 1 \end{cases}$$

$$f^2 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ \bot & \text{if } x \ge 2 \end{cases}$$

$$f^3 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ \bot & \text{if } x \ge 3 \end{cases}$$

$$f^4 = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, 1*y) & \text{if } x = 1 \\ (0, 2*y) & \text{if } x = 2 \\ (0, 6*y) & \text{if } x = 3 \\ \bot & \text{if } x \ge 4 \end{cases}$$

$$\vdots$$

$$f^{n+1} = \begin{cases} (x,y) & \text{if } x \le 0 \\ (0, x!*y) & \text{if } 1 \le x \le n \\ \bot & \text{if } x \ge n+1 \end{cases}$$

$$\boxed{\Omega \sqsubseteq f^1 \sqsubseteq f^2 \sqsubseteq f^3 \sqsubseteq \ldots \sqsubseteq f^n \sqsubseteq \ldots}$$

# Semantics of while-do: an example - continued

$$f \;=\; \lambda Z.\,\lambda s.\,cond(\,[\![\,x{>}0\,]\!]\,s,\; Z\,[\![\,y{:=}x{*}y;\,x{:=}x-1\,]\!]\,s,\; s\,)$$

We look for $X$ such that $X = f X$ and we start from $\Omega$

$$f^{\omega} \;=\; \bigsqcup_{i\in\mathbb{N}} f^{i} \;=\; \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, x!*y) & \text{if } x \geq 1 \end{cases}$$

We have that

$$f^{\omega} \;=\; f f^{\omega}$$

and, wrt every other fixpoint $w$ :

$$w \;=\; f w \quad\Rightarrow\quad f^{\omega} \sqsubseteq w$$

# Semantics of while-do: an example - continued

$$f \;=\; \lambda Z.\, \lambda s.\, cond(\, [\![ x>0 ]\!]\, s,\; Z\, [\![ y := x*y; x := x-1 ]\!]\, s,\; s\,)$$

We look for $X$ such that $X = f\,X$ and we start from $\Omega$

$$f^{\omega} \;=\; \bigsqcup_{i \in \mathbb{N}} f^{i} \;=\; \begin{cases} (x, y) & \text{if } x \leq 0 \\ (0, x\,! * y) & \text{if } x \geq 1 \end{cases}$$

We have that

$$f^{\omega} \;=\; f f^{\omega}$$

and, wrt every other fixpoint $w$ :

$$w \;=\; f w \quad \Rightarrow \quad f^{\omega} \sqsubseteq w$$

# Semantics of while-do: an example - continued

$$f \;=\; \lambda Z.\,\lambda s.\,cond(\,[\![\,x{>}0\,]\!]\,s,\; Z\,[\![\,y{:=}x{*}y; x{:=}x-1\,]\!]\,s,\; s\,)$$

We look for $X$ such that $X = fX$ and we start from $\Omega$

$$f^{\omega} \;=\; \bigsqcup_{i\in\mathbb{N}} f^{i} \;=\; \begin{cases} (x,y) & \text{if } x \leq 0 \\ (0, x!*y) & \text{if } x \geq 1 \end{cases}$$

We have that

$$f^{\omega} \;=\; ff^{\omega}$$

and, wrt every other fixpoint $w$ :

$$w \;=\; fw \quad\Rightarrow\quad f^{\omega} \sqsubseteq w$$

# Semantics of while-do: an example - continued

$$f \;=\; \lambda Z.\, \lambda s.\, cond(\, [\![ x{>}0 ]\!]\, s,\; Z\, [\![ y{:=}x{*}y; x{:=}x-1 ]\!]\, s,\; s\,)$$

We look for $X$ such that $X = f X$ and we start from $\Omega$

$$f^{\omega} \;=\; \bigsqcup_{i \in \mathbb{N}} f^{i} \;=\; \begin{cases} (x, y) & \text{if } x \leq 0 \\ (0, x! * y) & \text{if } x \geq 1 \end{cases}$$

We have that

$$f^{\omega} \;=\; f f^{\omega}$$

and, wrt every other fixpoint $w$ :

$$w \;=\; f w \quad \Rightarrow \quad f^{\omega} \sqsubseteq w$$

# Defining the Semantics of Fixpoints

$\langle D, \preccurlyeq \rangle$ is a po-set if $\preccurlyeq$ is a partial order *(refl + antisym + tr)*

e.g. $\langle \mathbf{State} \hookrightarrow \mathbf{State}, \sqsubseteq \rangle$ is a poset

A chain $C = d_0 \preccurlyeq d_1 \preccurlyeq d_2 \preccurlyeq \ldots$ is a totally ordered subset of $D$

$lub(C) = \bigsqcup_{i \geq 0} d_i$ satisfies:

- $\forall i \geq 0 \;\; d_i \preccurlyeq lub(C)$
- $\forall i \geq 0 \;\; d_i \preccurlyeq U$ implies $lub(C) \preccurlyeq U$




$\langle 2^{\{a,b,c\}}, \subseteq \rangle$

$\langle D, \preccurlyeq \rangle$ is a complete po-set (CPO) if

- $\bot \in D$ and $\bot \preccurlyeq d$ for every $d \in D$
- $lub(C) \in D$ for every chain $C$

# Defining the Semantics of Fixpoints

$\langle D, \preccurlyeq \rangle$ is a po-set if $\preccurlyeq$ is a partial order *(refl + antisym + tr)*

e.g. $\langle \textbf{State} \hookrightarrow \textbf{State}, \sqsubseteq \rangle$ is a poset

A chain $C = d_0 \preccurlyeq d_1 \preccurlyeq d_2 \preccurlyeq \ldots$ is a totally ordered subset of $D$

$lub(C) = \bigsqcup_{i \geq 0} d_i$ satisfies:

- $\forall i \geq 0 \;\; d_i \preccurlyeq lub(C)$
- $\forall i \geq 0 \;\; d_i \preccurlyeq U$ implies $lub(C) \preccurlyeq U$




$\langle 2^{\{a,b,c\}}, \subseteq \rangle$

$\langle D, \preccurlyeq \rangle$ is a complete po-set (CPO) if

- $\bot \in D$ and $\bot \preccurlyeq d$ for every $d \in D$
- $lub(C) \in D$ for every chain $C$

# Defining the Semantics of Fixpoints

$\langle D, \preccurlyeq \rangle$ is a po-set if $\preccurlyeq$ is a partial order *(refl + antisym + tr)*

e.g. $\langle \mathbf{State} \hookrightarrow \mathbf{State}, \sqsubseteq \rangle$ is a poset

A chain $C = d_0 \preccurlyeq d_1 \preccurlyeq d_2 \preccurlyeq \ldots$ is a totally ordered subset of $D$

$lub(C) = \bigsqcup_{i \geq 0} d_i$ satisfies:

- $\forall i \geq 0 \;\; d_i \preccurlyeq lub(C)$
- $\forall i \geq 0 \;\; d_i \preccurlyeq U$ implies $lub(C) \preccurlyeq U$

$\langle D, \preccurlyeq \rangle$ is a complete po-set (CPO) if

- $\bot \in D$ and $\bot \preccurlyeq d$ for every $d \in D$
- $lub(C) \in D$ for every chain $C$




$\langle 2^{\{a,b,c\}}, \subseteq \rangle$

# Defining the Semantics of Fixpoints

$\langle D, \preccurlyeq \rangle$ is a po-set if $\preccurlyeq$ is a partial order *(refl + antisym + tr)*

e.g. $\langle \mathbf{State} \hookrightarrow \mathbf{State}, \sqsubseteq \rangle$ is a poset

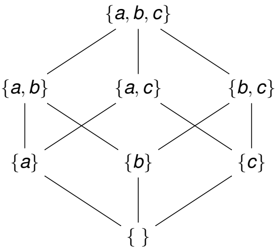
A chain $C = d_0 \preccurlyeq d_1 \preccurlyeq d_2 \preccurlyeq \ldots$ is a totally ordered subset of $D$

$lub(C) = \bigsqcup_{i \geq 0} d_i$ satisfies:

- $\forall i \geq 0 \;\; d_i \preccurlyeq lub(C)$
- $\forall i \geq 0 \;\; d_i \preccurlyeq U$ implies $lub(C) \preccurlyeq U$

$\langle D, \preccurlyeq \rangle$ is a complete po-set (CPO) if

- $\bot \in D$ and $\bot \preccurlyeq d$ for every $d \in D$
- $lub(C) \in D$ for every chain $C$




$\langle 2^{\{a,b,c\}}, \subseteq \rangle$

# Computations as Chains

We restrict to (possibly infinite) chains, and their lub *(ref.lattices)*

| | |
|---|---|
| $\langle D, \preccurlyeq \rangle$ | a CPO as an information domain (with refinement) |
| $f \;:\; D \to D$ | an information transformer |

$f$ monotone : $\quad a \preccurlyeq b \quad \Rightarrow \quad f(a) \preccurlyeq f(b)$ *(information preserving)*

$f$ continuous : $\quad$ (1) $lub(f\,C) \in D$ $\quad$ for every chain $C$

(2) $f(lub(C)) \;=\; lub(f\,C)$ $\quad$ for every chain $C$

*(limit preserving)*

continuous $\subsetneq$ monotone

$f, g$ continuous $\Rightarrow$ $f \circ g$ continuous

# Computations as Chains

We restrict to (possibly infinite) chains, and their lub *(ref.lattices)*

| | |
|---|---|
| $\langle D, \preccurlyeq \rangle$ | a CPO as an information domain (with refinement) |
| $f : D \rightarrow D$ | an information transformer |

$f$ monotone : $a \preccurlyeq b \;\Rightarrow\; f(a) \preccurlyeq f(b)$ *(information preserving)*

$f$ continuous : (1) $lub(f\,C) \in D$ for every chain $C$

(2) $f(lub(C)) = lub(f\,C)$ for every chain $C$

*(limit preserving)*

continuous $\subsetneq$ monotone

$f, g$ continuous $\Rightarrow$ $f \circ g$ continuous

# Computations as Chains

We restrict to (possibly infinite) chains, and their lub *(ref.lattices)*

| | |
|---|---|
| $\langle D, \preccurlyeq \rangle$ | a CPO as an information domain (with refinement) |
| $f \,:\, D \to D$ | an information transformer |

$f$ monotone : $a \preccurlyeq b \;\Rightarrow\; f(a) \preccurlyeq f(b)$ *(information preserving)*

$f$ continuous : (1) $lub(f\,C) \in D$ for every chain $C$

(2) $f(lub(C)) = lub(f\,C)$ for every chain $C$

*(limit preserving)*

continuous $\subsetneq$ monotone

$f, g$ continuous $\Rightarrow$ $f \circ g$ continuous

# Computations as Chains

We restrict to (possibly infinite) chains, and their lub *(ref.lattices)*

| | |
|---|---|
| $\langle D, \preccurlyeq \rangle$ | a CPO as an information domain (with refinement) |
| $f \;:\; D \rightarrow D$ | an information transformer |

$f$ monotone : $a \preccurlyeq b \quad \Rightarrow \quad f(a) \preccurlyeq f(b)$ *(information preserving)*

$f$ continuous : (1) $lub(f\,C) \in D$ for every chain $C$

(2) $f(lub(C)) \;=\; lub(f\,C)$ for every chain $C$

*(limit preserving)*

continuous $\subsetneq$ monotone

$$\boxed{f, g \text{ continuous} \quad \Rightarrow \quad f \circ g \text{ continuous}}$$

# Fixpoints as Limits of Chains

**Theorem (Tarski Fixpoint Theorem)**

*$\langle D, \preccurlyeq \rangle$ CPO*
*A continuous function $f : D \to D$*

1. *has a fixpoint*
2. *has a minimal fixpoint (denoted $\mathit{fix}\, f$)*
3. $\mathit{fix}\, f \;=\; lub\left\{f^i \perp \mid i \in \mathbb{N}\right\}$ *where*

$$f^0 x = x$$
$$f^{i+1} x = f(f^i x)$$

A constructive result of fixpoint existence

As a consequence, we can denote $\mathit{fix}\, f$ also as $\bigsqcup_{i \in \mathbb{N}} f^i$

# Fixpoints as Limits of Chains

**Theorem (Tarski Fixpoint Theorem)**

*$\langle D, \preccurlyeq \rangle$ CPO*
*A continuous function $f : D \to D$*

1. *has a fixpoint*
2. *has a minimal fixpoint (denoted $\mathit{fix}\, f$)*
3. $\mathit{fix}\, f \;=\; lub\,\{f^i \perp \mid i \in \mathbb{N}\}$ *where*

$$f^0 x = x$$
$$f^{i+1} x = f(f^i x)$$

A constructive result of fixpoint existence

As a consequence, we can denote $\mathit{fix}\, f$ also as $\bigsqcup_{i \in \mathbb{N}} f^i$

# Fixpoints as Limits of Chains

**Theorem (Tarski Fixpoint Theorem)**

*$\langle D, \preccurlyeq \rangle$ CPO*
*A continuous function $f : D \to D$*

1. *has a fixpoint*
2. *has a minimal fixpoint (denoted $\mathit{fix}\, f$)*
3. $\mathit{fix}\, f \;=\; lub\, \{f^i \perp \mid i \in \mathbb{N}\}$ *where*

$$f^0 x = x$$
$$f^{i+1} x = f(f^i x)$$

A constructive result of fixpoint existence

As a consequence, we can denote $\mathit{fix}\, f$ also as $\bigsqcup_{i \in \mathbb{N}} f^i$

# Well-Definedness of Denotational Semantics

$$\mathcal{S}_{\rm ds}[\![x := a]\!]s = s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\mathcal{S}_{\rm ds}[\![\texttt{skip}]\!] = \text{id}$$

$$\mathcal{S}_{\rm ds}[\![S_1 \,;\, S_2]\!] = \mathcal{S}_{\rm ds}[\![S_2]\!] \circ \mathcal{S}_{\rm ds}[\![S_1]\!]$$

$$\mathcal{S}_{\rm ds}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!] = \text{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\rm ds}[\![S_1]\!], \mathcal{S}_{\rm ds}[\![S_2]\!])$$

$$\mathcal{S}_{\rm ds}[\![\texttt{while } b \texttt{ do } S]\!] = \text{FIX } F$$

$$\text{where } F\ g = \text{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{\rm ds}[\![S]\!], \text{id})$$

1. $\langle \textbf{State} \hookrightarrow \textbf{State}, \sqsubseteq \rangle$ is a CPO
2. *cond* is continuous
3. $\circ$ preserves continuity
4. *fix* is always applied to continuous functions

$\mathcal{S}_{ds}[\![P]\!]$ exists for every $P \in \textbf{Stm}$

# Well-Definedness of Denotational Semantics

$$\mathcal{S}_{\mathrm{ds}}[\![x := a]\!]s = s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\mathcal{S}_{\mathrm{ds}}[\![\mathtt{skip}]\!] = \mathrm{id}$$

$$\mathcal{S}_{\mathrm{ds}}[\![S_1 \,;\, S_2]\!] = \mathcal{S}_{\mathrm{ds}}[\![S_2]\!] \circ \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]$$

$$\mathcal{S}_{\mathrm{ds}}[\![\mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2]\!] = \mathrm{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\mathrm{ds}}[\![S_1]\!], \mathcal{S}_{\mathrm{ds}}[\![S_2]\!])$$

$$\mathcal{S}_{\mathrm{ds}}[\![\mathtt{while}\ b\ \mathtt{do}\ S]\!] = \mathrm{FIX}\ F$$

$$\text{where } F\ g = \mathrm{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!], \mathrm{id})$$

1. $\langle \mathbf{State} \hookrightarrow \mathbf{State}, \sqsubseteq \rangle$ is a CPO
2. *cond* is continuous
3. $\circ$ preserves continuity
4. *fix* is always applied to continuous functions

$\mathcal{S}_{ds}[\![P]\!]$ exists for every $P \in \mathbf{Stm}$

# Well-Definedness of Denotational Semantics

$$\mathcal{S}_{ds}[\![x := a]\!]s = s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\mathcal{S}_{ds}[\![\texttt{skip}]\!] = \text{id}$$

$$\mathcal{S}_{ds}[\![S_1 ; S_2]\!] = \mathcal{S}_{ds}[\![S_2]\!] \circ \mathcal{S}_{ds}[\![S_1]\!]$$

$$\mathcal{S}_{ds}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!] = \text{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{ds}[\![S_1]\!], \mathcal{S}_{ds}[\![S_2]\!])$$

$$\mathcal{S}_{ds}[\![\texttt{while } b \texttt{ do } S]\!] = \text{FIX } F$$
$$\text{where } F\ g = \text{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{ds}[\![S]\!], \text{id})$$

1. $\langle \textbf{State} \hookrightarrow \textbf{State}, \sqsubseteq \rangle$ is a CPO
2. *cond* is continuous
3. $\circ$ preserves continuity
4. *fix* is always applied to continuous functions

$\mathcal{S}_{ds}[\![P]\!]$ exists for every $P \in \textbf{Stm}$

# Well-Definedness of Denotational Semantics

$$\mathcal{S}_{ds}[\![x := a]\!]s = s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\mathcal{S}_{ds}[\![\texttt{skip}]\!] = \text{id}$$

$$\mathcal{S}_{ds}[\![S_1 \; ; \; S_2]\!] = \mathcal{S}_{ds}[\![S_2]\!] \circ \mathcal{S}_{ds}[\![S_1]\!]$$

$$\mathcal{S}_{ds}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!] = \text{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{ds}[\![S_1]\!], \mathcal{S}_{ds}[\![S_2]\!])$$

$$\mathcal{S}_{ds}[\![\texttt{while } b \texttt{ do } S]\!] = \text{FIX } F$$
$$\text{where } F \; g = \text{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{ds}[\![S]\!], \text{id})$$

1. $\langle \textbf{State} \hookrightarrow \textbf{State}, \sqsubseteq \rangle$ is a CPO
2. *cond* is continuous
3. $\circ$ preserves continuity
4. *fix* is always applied to continuous functions

$\mathcal{S}_{ds}[\![P]\!]$ exists for every $P \in \textbf{Stm}$

# Well-Definedness of Denotational Semantics

$$\mathcal{S}_{\rm ds}[\![x := a]\!]s = s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\mathcal{S}_{\rm ds}[\![\texttt{skip}]\!] = \text{id}$$

$$\mathcal{S}_{\rm ds}[\![S_1 \; ; \; S_2]\!] = \mathcal{S}_{\rm ds}[\![S_2]\!] \circ \mathcal{S}_{\rm ds}[\![S_1]\!]$$

$$\mathcal{S}_{\rm ds}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!] = \text{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\rm ds}[\![S_1]\!], \mathcal{S}_{\rm ds}[\![S_2]\!])$$

$$\mathcal{S}_{\rm ds}[\![\texttt{while } b \texttt{ do } S]\!] = \text{FIX } F$$
$$\text{where } F\ g = \text{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{\rm ds}[\![S]\!], \text{id})$$

1. $\langle \mathbf{State} \hookrightarrow \mathbf{State}, \sqsubseteq \rangle$ is a CPO
2. *cond* is continuous
3. $\circ$ preserves continuity
4. *fix* is always applied to continuous functions

$\mathcal{S}_{ds}[\![P]\!]$ exists for every $P \in \mathbf{Stm}$

# Well-Definedness of Denotational Semantics

$$\mathcal{S}_{\text{ds}}[\![x := a]\!]s = s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\mathcal{S}_{\text{ds}}[\![\texttt{skip}]\!] = \text{id}$$

$$\mathcal{S}_{\text{ds}}[\![S_1 \,;\, S_2]\!] = \mathcal{S}_{\text{ds}}[\![S_2]\!] \circ \mathcal{S}_{\text{ds}}[\![S_1]\!]$$

$$\mathcal{S}_{\text{ds}}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!] = \text{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\text{ds}}[\![S_1]\!], \mathcal{S}_{\text{ds}}[\![S_2]\!])$$

$$\mathcal{S}_{\text{ds}}[\![\texttt{while } b \texttt{ do } S]\!] = \text{FIX } F$$

$$\text{where } F\ g = \text{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{\text{ds}}[\![S]\!], \text{id})$$

1. $\langle \textbf{State} \hookrightarrow \textbf{State}, \sqsubseteq \rangle$ is a CPO
2. *cond* is continuous
3. $\circ$ preserves continuity
4. *fix* is always applied to continuous functions

$\mathcal{S}_{ds}[\![P]\!]$ exists for every $P \in \textbf{Stm}$

# Equivalence of Semantics

Recap:

$$\mathcal{S}_{\mathrm{sos}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S,\ s\rangle \Rightarrow^* s' \\ \underline{\mathrm{undef}} & \text{otherwise} \end{cases}$$

To show: $\mathcal{S}_{sos}[\![P]\!] = \mathcal{S}_{ds}[\![P]\!]$

$\langle P, s\rangle \Rightarrow^* s'$ implies $\mathcal{S}_{ds}[\![P]\!]s = s'$ (check other dir.)

A stronger invariant: (a form of $\sqsubseteq$)

$\langle P, s\rangle \Rightarrow s'$ implies $\mathcal{S}_{ds}[\![P]\!]s = s'$

$\langle P, s\rangle \Rightarrow \langle P', s'\rangle$ implies $\mathcal{S}_{ds}[\![P]\!]s = \mathcal{S}_{ds}[\![P']\!]s'$

By structural induction, then by induction on the length of $\Rightarrow^*$

# Equivalence of Semantics

Recap:

$$\mathcal{S}_{\text{sos}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S,\ s\rangle \Rightarrow^* s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

To show: $\mathcal{S}_{sos}[\![P]\!] \;=\; \mathcal{S}_{ds}[\![P]\!]$

$\langle P, s\rangle \Rightarrow^* s'$ implies $\mathcal{S}_{ds}[\![P]\!]s = s'$ (check other dir.)

A stronger invariant: (a form of $\sqsubseteq$)

$\langle P, s\rangle \Rightarrow s'$ implies $\mathcal{S}_{ds}[\![P]\!]s = s'$

$\langle P, s\rangle \Rightarrow \langle P', s'\rangle$ implies $\mathcal{S}_{ds}[\![P]\!]s = \mathcal{S}_{ds}[\![P']\!]s'$

By structural induction, then by induction on the length of $\Rightarrow^*$

# Equivalence of Semantics

Recap:

$$\mathcal{S}_{\text{sos}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S,\ s\rangle \Rightarrow^* s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

To show: $\mathcal{S}_{sos}[\![P]\!] \;=\; \mathcal{S}_{ds}[\![P]\!]$

$\langle P, s\rangle \Rightarrow^* s'$ implies $\mathcal{S}_{ds}[\![P]\!]s = s'$ (check other dir.)

A stronger invariant: (a form of $\sqsubseteq$)

$\langle P, s\rangle \Rightarrow s'$ implies $\mathcal{S}_{ds}[\![P]\!]s = s'$

$\langle P, s\rangle \Rightarrow \langle P', s'\rangle$ implies $\mathcal{S}_{ds}[\![P]\!]s = \mathcal{S}_{ds}[\![P']\!]s'$

By structural induction, then by induction on the length of $\Rightarrow^*$

# Equivalence of Semantics

Recap:

$$\mathcal{S}_{\text{sos}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S,\ s\rangle \Rightarrow^* \ s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

To show: $\mathcal{S}_{sos}[\![P]\!] \ = \ \mathcal{S}_{ds}[\![P]\!]$

$\langle P, s\rangle \Rightarrow^* s'$ implies $\mathcal{S}_{ds}[\![P]\!]s = s'$ (check other dir.)

A stronger invariant: (a form of $\sqsubseteq$)

$\langle P, s\rangle \Rightarrow s'$ implies $\mathcal{S}_{ds}[\![P]\!]s = s'$

$\langle P, s\rangle \Rightarrow \langle P', s'\rangle$ implies $\mathcal{S}_{ds}[\![P]\!]s = \mathcal{S}_{ds}[\![P']\!]s'$

By structural induction, then by induction on the length of $\Rightarrow^*$