

G54FOP: Lecture 15

Denotational Semantics and Domain Theory I

Henrik Nilsson

University of Nottingham, UK

This Lecture

- Introduction to Denotational Semantics
- The relation between Operational and Denotational Semantics

Denotational Semantics (1)

Operational Semantics (review):

- The meaning of a term is the term it (ultimately) reduces to, if any:
 - Stuck terms
 - Infinite reduction sequences
- No **inherent** meaning (structure) beyond syntax of terms.
- Focus on behaviour; important aspects of semantics (such as non-termination) **emerges** from the behaviour.

Denotational Semantics (2)

Denotational Semantics:

- Idea: **Semantic function** maps (abstract) syntax directly to meaning in a **semantic domain**.
- Domains consist of appropriate semantic objects (Booleans, numbers, functions, ...) and have structure; in particular, an **information ordering**.
- The semantic functions are **total**; in particular, even a **diverging** computation is mapped to an element in the semantic domain.

Denotational Semantics (3)

Example:

$$\llbracket (1 + 2) * 3 \rrbracket = 9$$

Denotational Semantics (3)

Example:

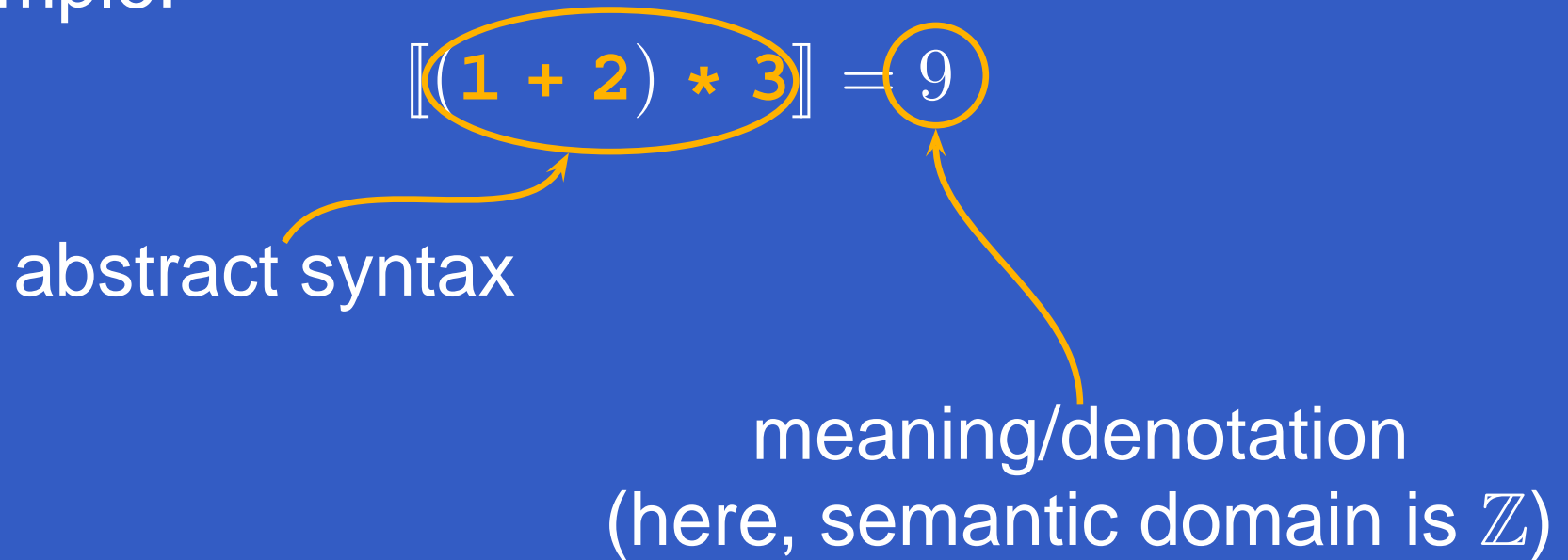
$$\llbracket (1 + 2) * 3 \rrbracket = 9$$

abstract syntax



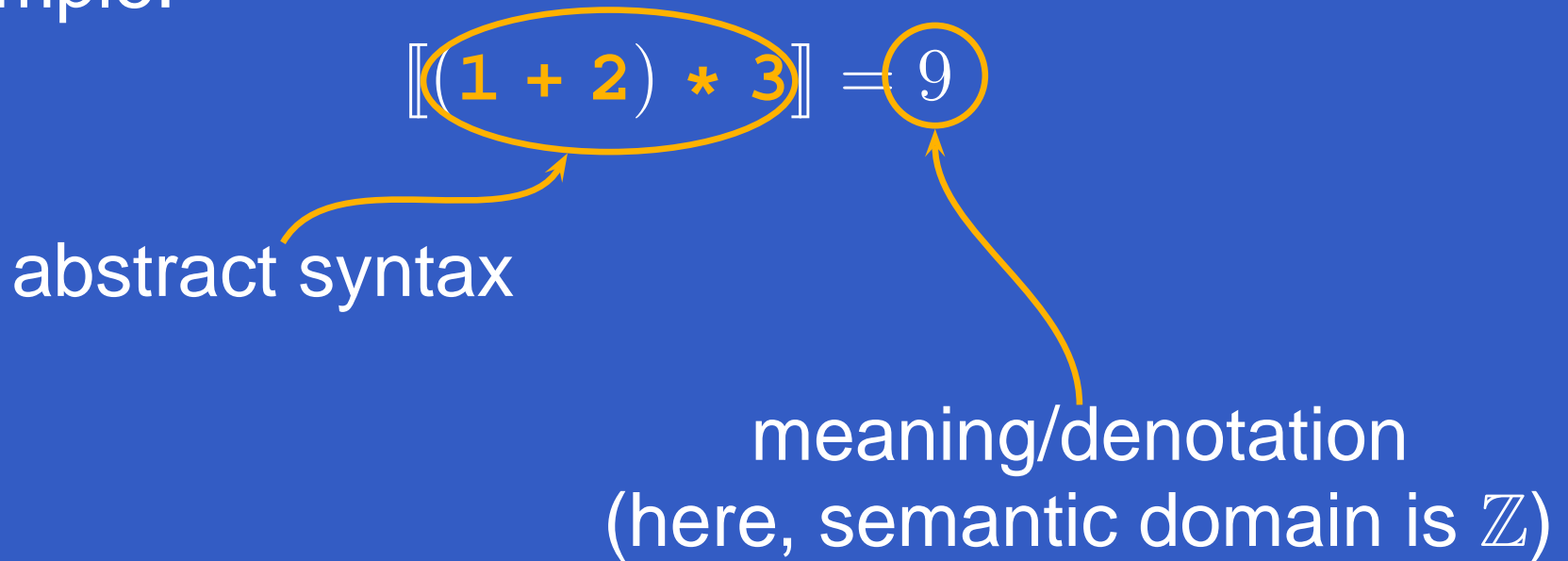
Denotational Semantics (3)

Example:



Denotational Semantics (3)

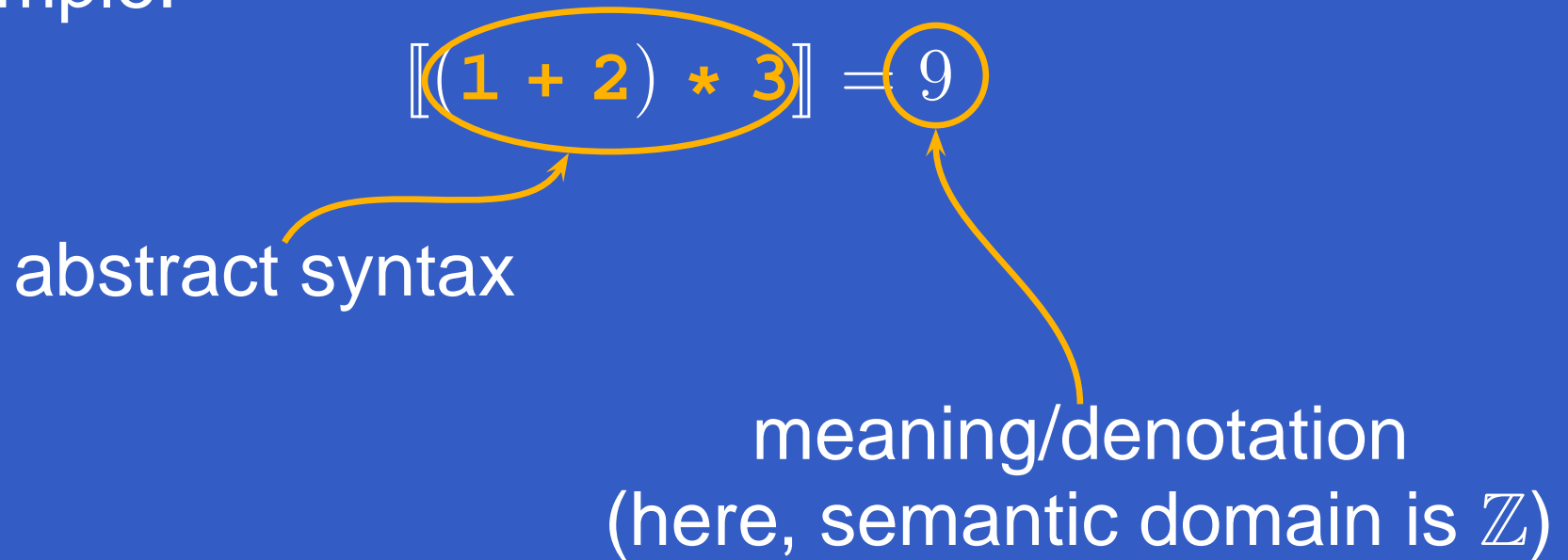
Example:



$\llbracket \cdot \rrbracket$, or variations like $E[\cdot]$, $C[\cdot]$: **semantic functions**

Denotational Semantics (3)

Example:



$\llbracket \cdot \rrbracket$, or variations like $E[\cdot]$, $C[\cdot]$: **semantic functions**

\llbracket and \rrbracket : **Scott brackets** or **semantic brackets**.

Compositionality (1)

- It is usually required that a denotational semantics is **compositional**: that the meaning of a program fragment is given in terms of the meaning of its **parts**.

Compositionality (1)

- It is usually required that a denotational semantics is **compositional**: that the meaning of a program fragment is given in terms of the meaning of its **parts**.
- Compositionality ensures that

Compositionality (1)

- It is usually required that a denotational semantics is **compositional**: that the meaning of a program fragment is given in terms of the meaning of its **parts**.
- Compositionality ensures that
 - the semantics is well-defined

Compositionality (1)

- It is usually required that a denotational semantics is **compositional**: that the meaning of a program fragment is given in terms of the meaning of its **parts**.
- Compositionality ensures that
 - the semantics is well-defined
 - important meta-theoretical properties hold

Compositionality (2)

Example:

$$\llbracket (1 + 2) * 3 \rrbracket = \llbracket 1 + 2 \rrbracket \times \llbracket 3 \rrbracket$$

Compositionality (2)

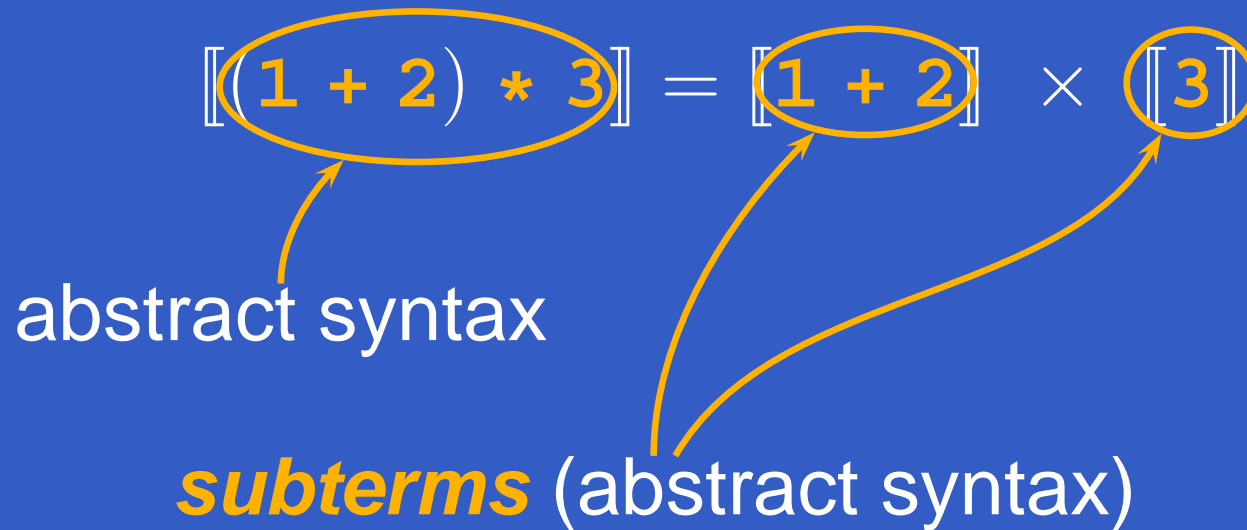
Example:

$$\llbracket (1 + 2) * 3 \rrbracket = \llbracket 1 + 2 \rrbracket \times \llbracket 3 \rrbracket$$

abstract syntax

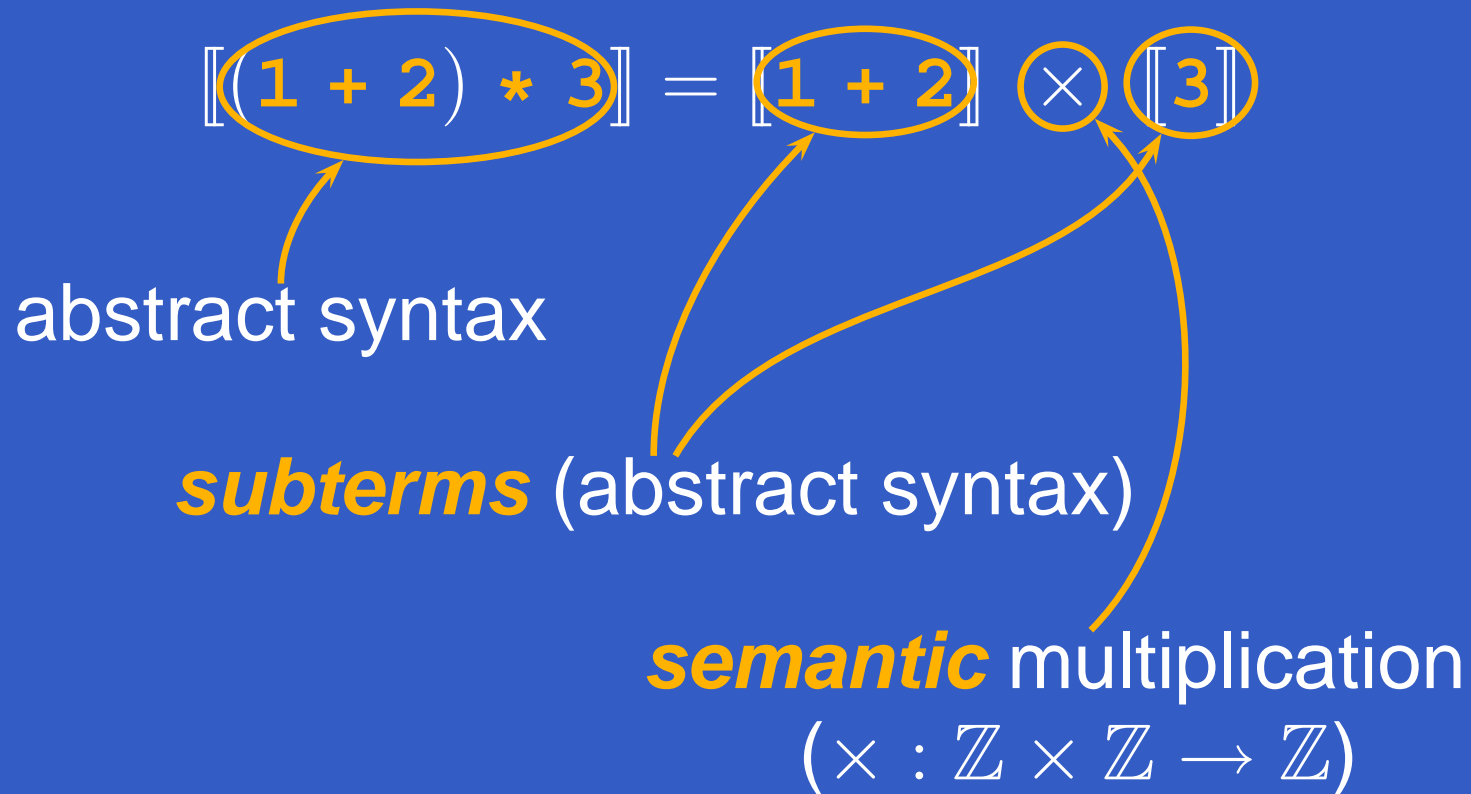
Compositionality (2)

Example:



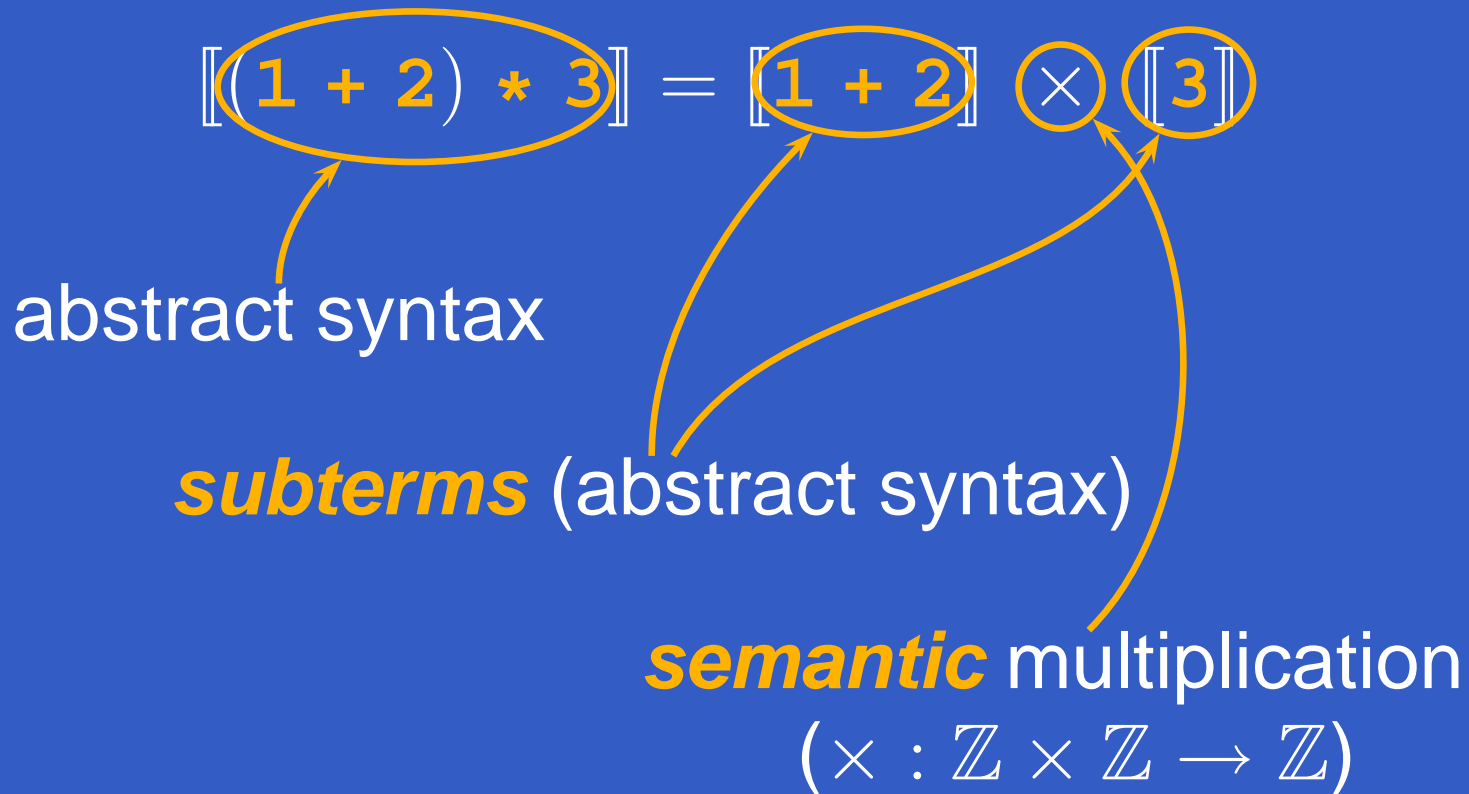
Compositionality (2)

Example:



Compositionality (2)

Example:



The meaning of the whole is given by composing the meaning of the parts.

Definition

Formally, a **denotational semantics** for a language L is given by a pair

$$\langle D, [[\cdot]] \rangle$$

where

- D is the **semantic domain**
- $[[\cdot]] : L \rightarrow D$ is the **valuation function** or **semantic function**.

Definition

Formally, a **denotational semantics** for a language L is given by a pair

$$\langle D, [[\cdot]] \rangle$$

where

- D is the **semantic domain**
- $[[\cdot]] : L \rightarrow D$ is the **valuation function** or **semantic function**.

In simple cases D may be a set, but usually more structure is required leading to **domains** as defined in **domain theory**.

Example: Simple Expr. Language (1)

$e \rightarrow$		<i>expressions:</i>
	true	<i>constant true</i>
	false	<i>constant false</i>
	if e then e else e	<i>conditional</i>
	0	<i>constant zero</i>
	succ e	<i>successor</i>
	pred e	<i>predecessor</i>
	iszero e	<i>zero test</i>

Example: Simple Expr. Language (2)

Develop a denotational semantics $\langle D, \llbracket \cdot \rrbracket \rangle$ for E , picking \mathbb{N} as the semantic domain for simplicity:

$$D = \mathbb{N}$$

$$\llbracket \cdot \rrbracket : e \rightarrow \mathbb{N}$$

Example: Simple Expr. Language (2)

Develop a denotational semantics $\langle D, \llbracket \cdot \rrbracket \rangle$ for E , picking \mathbb{N} as the semantic domain for simplicity:

$$\begin{aligned} D &= \mathbb{N} \\ \llbracket \cdot \rrbracket &: e \rightarrow \mathbb{N} \end{aligned}$$

However, as there are both Booleans and natural numbers in the **object** language, a more refined choice for the semantics at the **meta** level would have been $\mathbb{N} \uplus \mathbb{B}$, the **disjoint union** of natural numbers and Booleans.

Example: Simple Expr. Language (2)

Develop a denotational semantics $\langle D, \llbracket \cdot \rrbracket \rangle$ for E , picking \mathbb{N} as the semantic domain for simplicity:

$$\begin{aligned} D &= \mathbb{N} \\ \llbracket \cdot \rrbracket &: e \rightarrow \mathbb{N} \end{aligned}$$

However, as there are both Booleans and natural numbers in the **object** language, a more refined choice for the semantics at the **meta** level would have been $\mathbb{N} \uplus \mathbb{B}$, the **disjoint union** of natural numbers and Booleans.

(On whiteboard)

Exercises (1)

1. Find the denotation of

if (iszero (succ 0)) then true else false

Exercises (2)

2. Consider the following language extension:

e	\rightarrow	<i>expressions:</i>
...		
	not e	<i>logical negation</i>
	e && e	<i>logical conjunction</i>
	e + e	<i>addition</i>
	e - e	<i>subtraction</i>
	e * e	<i>multiplication</i>

Extend the denotational semantics accordingly.

Operational and Denotational Sem. (1)

Given a language L , suppose we have:

- a big-step operational semantics

$$\Downarrow \subseteq L \times V$$

where $V \subseteq L$ is the set of values

- a denotational semantics

$$\langle D, \llbracket \cdot \rrbracket \rangle$$

where $\llbracket \cdot \rrbracket : L \rightarrow D$

How should these be related?

Operational and Denotational Sem. (2)

Closed terms $t_1, t_2 \in L$ are **semantically** or **denotationally equivalent** iff

$$\llbracket t_1 \rrbracket = \llbracket t_2 \rrbracket$$

Assume D is **ground** (no functions; i.e., our closed terms are **programs** that output something “printable”). We adopt a function

$$\cdot : D \rightarrow V$$

that maps a semantic value $d \in D$ to its **term representation** $v \in V$.

Operational and Denotational Sem. (3)

Assuming termination:

- **Correctness** of operational semantics w.r.t. denotational semantics:

$$t \Downarrow v \Rightarrow \llbracket t \rrbracket = \llbracket v \rrbracket$$

- **Completeness** of operational semantics w.r.t. denotational semantics:

$$\llbracket t \rrbracket = d \Rightarrow t \Downarrow \underline{d}$$

Operational and Denotational Sem. (4)

Assuming termination:

- **Computational adequacy** of operational semantics w.r.t. denotational semantics (or vice versa, depending on point of view):

$$t \Downarrow v \Leftrightarrow \llbracket t \rrbracket = \llbracket v \rrbracket$$