

# CS520 Lecture 3

## Simple Imperative Language

December 1, 2019

### 3.1 Motivation or objective

1. Most real-world PLs support computation by state update and that by function application. The former is often referred as imperative computation, and the latter as functional or applicative computation. Our goal is to study core of PL concepts and ideas for imperative computation.
2. Actually, it is more appropriate to say that our aim is a formal & mathematical analysis of the core PL concepts for imperative computation. We will study or learn mathematical tools for this. Also, we will show how to express and analyze big design decisions of such an imperative PLs.
3. We will look at 1) some basic concepts and results of domain theory, 2) variable declaration and binding, 3) syntactic sugar, error handling and 4) the notions of soundness and full abstraction. (Well, I just listed all the key items in the chapter 2 of the book)
4. A good way to learn the material of this chapter is to ask yourself : What should you do in order to design an imperative programming language and build a foundation of the designed language?  
Think about this a little bit, and compare your answer with what I'll explain.

### 3.2 Syntax

1. Variables, and read and update of them ... key concepts or operations for imperative computation.

2. Syntax supports these concepts and operations:

$\langle \text{intexp} \rangle ::= 0|1|\dots|\langle \text{var} \rangle^1|\dots$  – same as before

$\langle \text{boolexp} \rangle ::= \text{true}|\text{false}|\dots$  –

almost the same as that of  $\langle \text{assert} \rangle$ .

The exception is that  $\langle \text{boolexp} \rangle$  doesn't include quantifiers. Q. why?

$\langle \text{comm} \rangle ::= \langle \text{var} \rangle := \langle \text{intexp} \rangle^2|\text{skip}|\langle \text{comm} \rangle; \langle \text{comm} \rangle^3$

$|\text{if } \langle \text{boolexp} \rangle \text{ then } \langle \text{comm} \rangle \text{ else } \langle \text{comm} \rangle$

$|\text{while } \langle \text{boolexp} \rangle \text{ do } \langle \text{comm} \rangle$

As in the case of predicate logic, you can understand  $\langle \text{comm} \rangle$  as the set of all finite derivation trees, or as an (multi-sorted) initial algebra for the signature determined by the grammar.

3. It is language for expressing a sequence of variable reads and variable updates.

### 3.3 Basic domain theory

1. Giving a denotational semantics to our simple imperative language is not as straightforward as doing so with predicate logic, because of loop.

2.  $\llbracket - \rrbracket : \langle \text{comm} \rangle \rightarrow \dots$  We want the following equation for loop holding to hold:

$$\begin{aligned}\llbracket \text{while } b \text{ do } c \rrbracket &= \llbracket \text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else } \text{skip} \rrbracket \\ &= \dots \llbracket \text{while } b \text{ do } c \rrbracket \dots \\ &= F(\llbracket \text{while } b \text{ do } c \rrbracket) \text{ for some } F\end{aligned}$$

But a function  $F$  on a set may or may not have such fixed point.

3. Then, why should  $F$  in the above have a fixed point? Be-

cause it is something that can be implemented by a program.

$$F \stackrel{\text{informally}}{=} \llbracket \text{if } b \text{ then } (c; -) \text{ else skip} \rrbracket$$

One objective of domain theory is to formalise good properties enjoyed by such program-implementable functions without going the low-level details of computability theory.

#### 4. High-level meta heuristic behind domain theory

1. Consider a set together with some structure.
2. Use functions between such sets that respect the structures.
3. Why 1) and 2)? Because if done well, functions in 2) will always have fixed points.

#### 5. Key definitions

##### **Definition 1**

*A binary relation  $\sqsubseteq$  on a set  $S$  is a partial order if*

1.  *$x \sqsubseteq x$  for all  $x \in S$  (reflexivity)*
2. *for all  $x, y, z \in S$ , if  $x \sqsubseteq y$  and  $y \sqsubseteq z$ , then  $x \sqsubseteq z$  (transitivity)*
3. *for all  $x, y \in S$ , if  $x \sqsubseteq y$  and  $y \sqsubseteq x$ , then  $x = y$ .*

*A set  $S$  with a partial order  $\sqsubseteq$  is called partially ordered set or poset.*

##### **Definition 2**

*A chain in a poset  $(S, \sqsubseteq)$  is a (countably) infinite sequence*

$$x_0, x_1, \dots, x_n, \dots$$

*of elements in  $S$  s.t.  $x_n \sqsubseteq x_{n+1}$  for all  $n \geq 0$ .*

**Definition 3**

A pre-domain is a poset  $(S, \sqsubseteq)$  s.t. all chains have least upper bounds.

meaning : For every chain  $\{x_n\}_{n \leq 0} \in S$ , there exists  $y = \sqcup_{n=0}^{\infty} x_n \in S$  s.t.

1)  $x_n \sqsubseteq y$  for all  $n \leq 0$

2) for any  $z$ , if  $x_n \sqsubseteq z$  for every  $n$ , then  $y \sqsubseteq z$ .

**Definition 4**

A domain is a pre-domain  $(S, \sqsubseteq)$  that has the least element, often denoted  $\perp$ .

maening : for all  $x \in S$ ,  $\perp \sqsubseteq x$ .

**Definition 5**

Let  $(S_1, \sqsubseteq_1)$  and  $(S_2, \sqsubseteq_2)$  be pre-domains.

A function  $f \in [S_1 \rightarrow S_2]$  is continuous if

for every chain  $\{x_n\}_{n \leq 0}$  in  $S_1$ ,  $f(\sqcup_{n=0}^{\infty} x_n)$  is the least upper bound of  $\{f(x_n)\}_{n \leq 0}$

A function  $f \in [S_1 \rightarrow S_2]$  is monotone if for all  $x, y \in S_1$ ,  $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$ .

When  $S_1$  and  $S_2$  are domains with least elements  $\perp_1$  and  $\perp_2$ , we say that a function  $f \in [S_1 \rightarrow S_2]$  is strict if  $f(\perp_1) = \perp_2$ .

Exercise : Show that if  $f$  is continuous, it is monotone.

6. What's going on here? What are the intuitions behind these definitions?

1.  $x \sqsubseteq y$  - intuitively means that  $y$  has more information than  $x$  or  $x$  and  $y$  have the same amount of info.

(a)  $\mathbb{Z}^{*,\omega} = \mathbb{Z}^* \cup \mathbb{Z}^\omega$  - finite or infinite sequence of integers.

$x \sqsubseteq y$  iff  $x$  is an initial subsequence (or prefix) of  $y$ .

$$\langle 3, 1, 4 \rangle \sqsubseteq \langle 3, 1, 4, 1, 5 \rangle$$

$$\langle 3, 1, 4 \rangle \not\sqsubseteq \langle 2, 1, 4, 1 \rangle$$

(b)  $\mathbb{Z}_\perp = \mathbb{Z} \cup \{\perp\}$  - lifted  $\mathbb{Z}$

$$\forall x, y \in \mathbb{Z}_\perp \quad x \sqsubseteq y \text{ iff } x = y \text{ or } x = \perp$$

(c)  $(2^{\mathbb{Z}}, \subset)$

$$\infty =: \top$$

(d) vertical domain of natural numbers -

$$\vdots$$

$$1$$

$$\perp = 0$$

2. The monotonicity means the preservation of the approximation relation.

One intuition behind continuity of  $f$  is that in order to produce finite amount of information in its output,  $f$  consumes only finite amount information in its input.

(a)  $set \in [\mathbb{Z}^{*,\omega} \rightarrow 2^{\mathbb{Z}}]$

$$set(\langle x_1, \dots, x_n \rangle) = \{x_1, \dots, x_n\}$$

$$set(\langle x_1, \dots, x_n, \dots \rangle) = \{x_1, \dots, x_n, \dots\}$$

If  $A \subset set(s)$  and  $A$  is finite, then there is a finite prefix  $s_0$  of  $s$  (i.e.,  $s_0 \sqsubseteq s$ ) s.t.  $A \subset set(s_0)$

Exercise : Show that if  $f \in [\mathbb{Z}^{*,\omega} \rightarrow 2^{\mathbb{Z}}]$  is continuous, it satisfies above property. Also show that if  $f \in [\mathbb{Z}^{*,\omega} \rightarrow 2^{\mathbb{Z}}]$  satisfies above property, then  $f$  is continuous.

(b)  $f \in [2^{\mathbb{Z}} \rightarrow \mathbb{N}^{\top}]$

$$f(A) = \begin{cases} |A| & \text{if } A \text{ is finite} \\ \top & \text{if } A \text{ is infinite} \end{cases}$$

Exercise : A function from  $2^{\mathbb{Z}}$  to a predomain  $P$  is finitely generated if for all  $A \in 2^{\mathbb{Z}}$ ,  $f(A)$  is the least upper bound of  $\{f(A_0) : A_0 \overset{\subset}{f}in A\}$ .  
Show that  $f$  is continuous iff it is finitely generated.

(c) John Reynolds' phrase in page 108:

“Instead, it is based on the physical limitations of communication: one cannot predict the future of input, nor receive an infinite amount of information in a finite amount of time, nor produce output except at finite times.”<sup>4</sup>

7. One important reason of doing domain theory is to have the following “least fixed-point theorem”:

**Proposition 1** (Least Fixed-Point Theorem)

*If  $D$  is a domain and  $f$  is a continuous function from  $D$  to  $D$ ,*

$$x = \sqcup_{n=0}^{\infty} f^n \perp$$

*is the least fixed-point of  $f$ . (That is,  $f(x) = x$  and whenever  $f(y) = y$ , we have  $x \sqsubseteq y$ )*

*Proof.* By the exercise before,  $f$  is monotone. Using induction and  $\perp$  is being the least element, we can show that  $\{f^n \perp\}_{n \leq 0}$  is a chain in  $D$ . Since  $D$  is a domain, the least upper bound  $x := \sqcup_{n=0}^{\infty} f^n \perp$  exists. Furthermore, by the continuity of  $f$ ,

$$f(x) = f(\sqcup_{n=0}^{\infty} f^n \perp) = \sqcup_{n=0}^{\infty} f^{n+1}(\perp) = \sqcup_{n=0}^{\infty} f^n(\perp) = x$$

$\therefore x$  is a fixed point of  $f$ .

To show that  $x$  is a least such, consider a fixed point  $y$  of  $f$ . Then, by induction, we can show that  $y$  is an upper bound of the chain  $\{f^n \perp\}_{n \leq 0}$ .  $\therefore x \sqsubseteq y$ .  $\square$

<sup>4</sup>Domain theory attempty to capture this aspect of computation.

8. When  $P, P'$  are predomains, we write  $[P \rightarrow_c P']$  for the set of continuous functions. When  $[P \rightarrow_c P']^5$  is given pointwise order  $\sqsubseteq$

$$f \sqsubseteq g \text{ iff } f(x) \sqsubseteq_{P'} g(x) \text{ for all } x \in P, f, g \in [P \rightarrow_c P']$$

it becomes a predomain, where the limit of a chain  $\{f_n\}_{n \leq 0}$  is defined pointwise  $x \mapsto \sqcup_{n=0}^{\infty} f_n(x)$ . Furthermore, if  $P'$  is a domain with the least element  $\perp'$ , then  $[P \rightarrow_c P']$  is also a domain with  $x \mapsto \perp'$  as its least element.

9.  $D$  is a domain with the least element  $\perp$ .

Define  $Y_D$  to be the following function from  $[D \rightarrow_c D]$  to  $D$ :

$$Y_D(f) = \sqcup_{n=0}^{\infty} f^n \perp$$

### Lemma 1

$Y_D$  is continuous.<sup>6</sup>

*Proof.* Exercise. □

10. There are a lot of interesting results in domain theory, some of which we will cover later in the course. Before finishing this mini review of domain theory, I want to explain the lifting construction.

1.  $P_{\perp} := P \cup \{\perp\}$  for a predomain  $P$ .

$$x \sqsubseteq_{P_{\perp}} y \text{ iff } x = \perp \text{ or } x, y \in P \text{ and } x \sqsubseteq_P y \text{ for } x, y \in P_{\perp}$$

Intuitively, we are adding the least element to  $P$  and converting  $P$  to a domain.

2.  $i_{\uparrow} \in [P \rightarrow_c P_{\perp}]$ , sometimes called unit,  $i_{\uparrow}(x) = x$

---

<sup>5</sup>Although we rush here, this function space construction is very important. It lets domain theory be applicable to functional languages

<sup>6</sup>This means that the very act of computing a fixed point of a given function is continuous.



3. for each  $f \in [P \rightarrow_c P'_\perp]$ ,

$$\begin{aligned} f_\perp &\in [P_\perp \rightarrow_c P'_\perp] \\ f_\perp(\perp) &:= \perp \\ f_\perp(x) &:= f(x) \end{aligned}$$

sometimes called kleisli extension.

4. Why should we care about 2. and 3. ? Because they allow us to compose continuous function from  $P$  to  $P'_\perp$

$$\begin{aligned} f &\in [P \rightarrow_c P'_\perp] \text{ and } g \in [P' \rightarrow_c P''_\perp] \\ \Rightarrow (g_\perp \circ f) &\in [P \rightarrow_c P''_\perp]^7 \end{aligned}$$

### 3.4 Denotational Semantics of the simple imperative language

1. Recall that  $\Sigma = \langle \text{var} \rangle \rightarrow \mathbb{Z}$ .  $\Sigma$  is a predomain when given the discrete order  $\sqsubseteq$ ,  $x \sqsubseteq y$  iff  $x = y$

$$\begin{aligned} \llbracket - \rrbracket_{\text{intexp}} &\in \langle \text{intexp} \rangle \rightarrow [\Sigma \rightarrow \mathbb{Z}] \\ \llbracket - \rrbracket_{\text{boolexp}} &\in \langle \text{boolexp} \rangle \rightarrow [\Sigma \rightarrow \mathbb{B}] \\ \llbracket - \rrbracket_{\text{comm}} &\in \langle \text{comm} \rangle \rightarrow [\Sigma \rightarrow_c \Sigma_\perp] \end{aligned}$$

$$\llbracket x := e \rrbracket \sigma = [\sigma | x : \llbracket e \rrbracket \sigma]$$

$$\llbracket \text{skip} \rrbracket \sigma = \sigma$$

$$\llbracket c_1 ; c_2 \rrbracket \sigma = \llbracket c_2 \rrbracket_\perp (\llbracket c_1 \rrbracket \sigma)$$

$$\llbracket \text{if } b \text{ then } c_1 \text{ else } c_2 \rrbracket \sigma = \text{if } \llbracket b \rrbracket \sigma \text{ then } \llbracket c_1 \rrbracket \sigma \text{ else } \llbracket c_2 \rrbracket \sigma$$

$$\llbracket \text{while } b \text{ do } c \rrbracket \sigma = Y_{\Sigma_\perp}(F)$$

$$\text{where } F \in [\Sigma \rightarrow_c \Sigma_\perp] \rightarrow_c [\Sigma \rightarrow_c \Sigma_\perp]$$

$$F f x \sigma := \text{if } \llbracket b \rrbracket \sigma \text{ then } (f_\perp \circ \llbracket c \rrbracket) \sigma \text{ else } \sigma$$

---

<sup>7</sup>We can view  $(-)_{\perp} \circ (c)$  as a new composition operator  $\circ'$