

# CAP — Catch up course

Today: browse through several topics that will be useful in the second part of the CAP course

1. IMP and Hoare logic
2. Caml
3. Partial orders and fixpoints
4. Operational semantics for IMP and FUN  
more inference rules

## *1. IMP, a small imperative programming language*

## A first example

everybody should be able to read the following program, and understand what each command does:

```
Q := 0;  
R := X;  
while R >= Y do (  
    Q := Q+1;  
    R := R-Y;  
)
```

# The grammar of IMP

an infinite set  $\mathcal{V}$  of *variable identifiers*  $X, Y, Z, \dots$

arithmetical expressions

$a ::= X \mid a_1 + a_2 \mid a_1 * a_2 \mid -a \mid 1, 2, 3, \dots$

programs

$p ::= X := a \mid p_1; p_2 \mid \text{if } a \geq 0 \text{ then } p_1 \text{ else } p_2$   
 $\quad \mid \text{while } a \geq 0 \text{ do } p \mid \text{skip}$

- ▶ **skip**: program that does nothing (maybe it is  $X := X$ )
- ▶ we could have **boolean expressions**, and programs of the form  
 $\text{if } b \text{ then } p_1 \text{ else } p_2, \text{ while } b \text{ do } p$

$b ::= a \geq 0 \mid \neg b \mid b_1 \wedge b_2$

# Reasoning about the execution of IMP programs: Hoare logic

$$\{A\} p \{B\}$$

*if the initial state satisfies **assertion**  $A$ , and if the execution of **program**  $p$  terminates, then the final state satisfies **assertion**  $B$*

example:

```
 $\{X \geq 0 \wedge Y > 0\}$   
Q := 0;  
R := X;  
while R >= Y do (  
  Q := Q+1;  
  R := R-Y;  
)  
 $\{X = Y * Q + R \wedge R < Y\}$ 
```

# The rules of Hoare logic, and how to read them

inference rules

$$\overline{\{A[a/X]\} X := a \{A\}}$$

$$\overline{\{A\} \text{skip} \{A\}}$$

$$\frac{\{A_1\} p_1 \{A_2\} \quad \{A_2\} p_2 \{A_3\}}{\{A_1\} p_1; p_2 \{A_3\}}$$

$$\frac{\{A \wedge a \geq 0\} p_1 \{B\} \quad \{A \wedge \neg(a \geq 0)\} p_2 \{B\}}{\{A\} \text{if } a \geq 0 \text{ then } p_1 \text{ else } p_2 \{B\}}$$

$$\frac{\{A_I \wedge a \geq 0\} p \{A_I\}}{\{A_I\} \text{while } a \geq 0 \text{ do } p \{A_I \wedge \neg(a \geq 0)\}}$$

$$\frac{A_1 \Rightarrow A_2 \quad \{A_2\} p \{B_2\} \quad B_2 \Rightarrow B_1}{\{A_1\} p \{B_1\}}$$

# Building a derivation in Hoare logic

```
    { $X \geq 0 \wedge Y > 0$ }  
Q := 0;  
R := X;  
while R >= Y do (  
    Q := Q+1;  
    R := R-Y;  
)  
    { $X = Y * Q + R \wedge R < Y$ }
```

on the board

## Other examples

```
{true} while true do skip {false}
```

```
    { $X = N \wedge N > 0$ }  
Y := 1;  
while X>0 do (  
    Y := Y * X;  
    X := X - 1;  
);  
    { $Y = N!$ }
```

## *2. A real functional programming language: OCaml*

DEMO

file `camlcatchup.ml`

*(will be made available from the www page of the course)*

### 3. *Fixpoints*

## Partial orders

- ▶ a **partially ordered set** (poset) is given by  $(S, \sqsubseteq)$ , where relation  $\sqsubseteq$  is **reflexive, transitive, antisymmetric**
  - ▶ examples:  $(\mathbb{N}, \leq)$   $(\mathcal{P}(S), \subseteq)$
  - ▶ Hasse diagrams

# Partial orders

- ▶ a **partially ordered set** (poset) is given by  $(S, \sqsubseteq)$ , where relation  $\sqsubseteq$  is **reflexive, transitive, antisymmetric**
  - ▶ examples:  $(\mathbb{N}, \leq)$   $(\mathcal{P}(S), \subseteq)$
  - ▶ Hasse diagrams
- ▶ let  $(S, \sqsubseteq)$  be a poset, and consider  $E \subseteq S$  ( $E$  is a subset of  $S$ )
  - ▶  $u \in S$  is an **upper bound** of  $E$  if  $\forall x \in E, x \sqsubseteq u$
  - ▶  $u \in S$  is a **least upper bound (lub)** of  $E$  if all upper bounds of  $E$  are above  $u$ , that is,  $\forall u' \in S, (\forall x \in E, x \sqsubseteq u') \Rightarrow u \sqsubseteq u'$   
a least upper bound of  $E$  is written  $\cup E$
  - ▶ symmetrically, **lower bound** and **greatest lower bound (glb)**, written  $\cap E$
- ▶ NB: glbs and lubs do not always exist.  
two remarkable such elements, when they exist, are
  - ▶  $\perp = \cap S$ , the least element of  $S$ , and
  - ▶  $\top = \cup S$ , the greatest element of  $S$

## Particular kinds of partial orders

- ▶ a **complete lattice** is a poset  $(S, \sqsubseteq)$  with “everything”:
  - ▶ for any  $E \subseteq S$ ,  $\cap E$  and  $\cup E$  exist
  - ▶ so in particular,  $\perp$  and  $\top$  also exist

examples:

- ▶  $(\mathcal{P}(S), \subseteq, \emptyset, S, \cup, \cap)$
- ▶  $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq, -\infty, \infty, \max, \min)$

# Particular kinds of partial orders

- ▶ a **complete lattice** is a poset  $(S, \sqsubseteq)$  with “everything”:
  - ▶ for any  $E \subseteq S$ ,  $\cap E$  and  $\cup E$  exist
  - ▶ so in particular,  $\perp$  and  $\top$  also exist

examples:

- ▶  $(\mathcal{P}(S), \subseteq, \emptyset, S, \cup, \cap)$
  - ▶  $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq, -\infty, \infty, \max, \min)$
- ▶ a **complete partial order (cpo)** is a poset  $(S, \sqsubseteq)$  such that for every *chain*  $C \subseteq S$ ,  $\cup C$  exists
  - ▶ a chain is some  $\{x_1, x_2, \dots\} \subseteq S$  such that  $\forall i, x_i \sqsubseteq x_{i+1}$
  - ▶ a cpo has a least element, which is  $\cup \emptyset$

## Two theorems about fixpoints

- ▶ **Knaster-Tarski**: the set of fixpoints of a monotone function  $f : L \rightarrow L$ , where  $L$  is a complete lattice, forms a complete lattice.
- ▶ **Kleene**: if  $f$  is a continuous function on a complete partial order, then  $\bigcap \{\perp, f(\perp), f(f(\perp)), \dots\}$  is the least fixpoint of  $f$   
*continuous:  $f(\cup D) = \cup f(D)$  for  $D$  directed (i.e.  $\forall x, y \in D$ ,  $x$  and  $y$  have an upper bound in  $D$ )*

- ▶ these two theorems exist, and their proofs are rather elementary
- ▶ frequently used tools when reasoning mathematically about programs and their runs

*Operational semantics of IMP and FUN*

# The operational semantics of IMP programs

- ▶ exists in several flavours,  
we define here the **big step operational semantics**

$$\sigma, p \Downarrow \sigma'$$

The execution of program  $p$  in initial state  $\sigma$  terminates and yields final state  $\sigma'$ .

what's a “state”?

- ▶ a memory state, an **environment**
- ▶ a map from variables to integers  $\sigma : \mathcal{V} \rightarrow \mathbb{Z}$   
given some program  $p$ ,  $\sigma$  is a partial mapping from a *finite set of variables* to  $\mathbb{Z}$
- ▶  $\sigma, p \Downarrow \sigma'$  is called a *judgment*  
it is defined by inference rules

on the board

# A small functional programming language: FUN

grammar

an infinite set of FUN variables  $x, y, z, \dots$

FUN programs (are expressions)

$e ::= e_1 \ e_2 \mid \text{fun } x \rightarrow e \mid \text{let } x = e_1 \text{ in } e_2 \mid x \mid 1, 2, 3, \dots$

big step operational semantics

$e \Downarrow v$

- ▶  $v$  is a *value*  $v ::= \text{fun } x \rightarrow e \mid 1, 2, 3, \dots$
- ▶ no environment