

#### **Hoare logic**

Hoare logic (Hoare, 1969) is a logic to check properties of sequential, state-transforming programs. ("Sequential" means that there is no parallelism or concurrency during the execution.)

### **Hoare triples**

The logic is based on Hoare triples

 $\llbracket\phi\rrbracket C\llbracket\psi\rrbracket$ 

where C is a program (also called "command") and  $\phi$  and  $\psi$  are logical formulæ.

•  $\phi$  is called the **precondition** and  $\psi$  is called the **postcondition** of *C*.

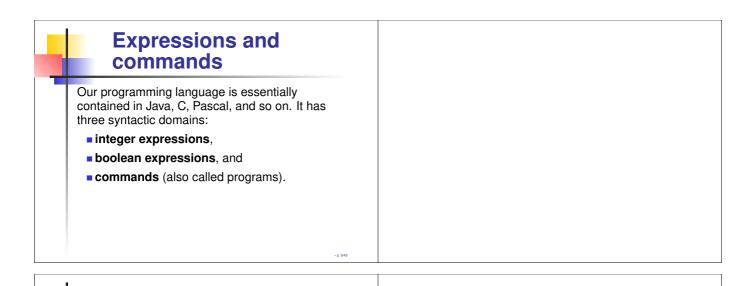
Remark: while studying Hoare logic, we shall use Greek letters  $\phi, \psi, \ldots$  to denote formulæ, to be compatible with Huth/Ryan.

# An idealized prog. language

We shall focus on an **idealized** imperative sequential programming language, which is

- sufficiently big to show that our study is realistic, and
- sufficiently small to allow an easy treatment.

Using idealized programming languages is a key technique in programming-language theory.



#### **Integer expressions**

Integer expressions are built in the familiar way from variables x, y, z, ..., integers, and basic operations like + and \*, e.g.

7 - 5 x + (y - 3)

# Grammar of integer expressions

The grammar (in Backus-Naur form) of integer expressions is

E ::= n |x| (-E) | (E + E) | (E - E) | (E \* E),

where  $n \in \{\ldots, -2, -1, 0, 1, 2, \ldots\}$  and x is any variable.

### **Boolean expressions**

The grammar of boolean expressions is

$$\begin{split} B ::= & \texttt{true} \mid \texttt{false} \mid (!B) \mid (B \& B) \mid (B ||B) \mid (E < E) \\ & |(E == E) \mid (E! = E) \end{split}$$

where ! stands for negation, & for conjunction, || of disjunction, == for equality, and != for inequality.

#### Commands

The commands we consider are given as follows:

Semantics of commands

The intuitive meaning of the programming constructs is described on the following slides.

## Assignment

The atomic command

x = E

is the usual assignment statement; it evaluates the integer expression E in the current state of the store and then overwrites the current value stored in x with the result of the evaluation.

## Sequential composition

The compound command

 $C_1; C_2$ 

is the sequential composition of the commands  $C_1$  and  $C_2$ . It begins by execution  $C_1$ . If that execution terminates, then it executes  $C_2$  in the state resulting from the execution of  $C_1$ . If the execution of  $C_1$  does not terminate, then neither does  $C_1$ ;  $C_2$ .

#### **If-statements**

#### The statement

if B then  $\{C_1\}$  else  $\{C_2\}$ 

first evaluates the boolean expression B in the current state; if the result is true, then  $C_1$  is executed; it the result is false, then  $C_2$  is executed.

#### While-loops

The construct

while  $B\{C\}$ 

means that:

- 1. the boolean expression *B* is evaluated in the current state;
- 2. if *B* evaluates to false, then the while-loop terminates;
- 3. if *B* evaluates to true, then *C* will be executed. If the execution of *C* terminates, we go back to Step (1).

# Example of semantics: factorial

The factorial n! of a natural number n is defined inductively by

$$0! = 1$$

$$(n+1)! = (n+1) \cdot n!.$$

For example,

 $4! = 4 \cdot 3! = \ldots = 4 \cdot 3 \cdot 2 \cdot 1 \cdot 0! = 24.$ 

The next slide contains a program for computing the factorial of x.

## **Example: factorial**

The program Fac1 below is intended to compute the factorial of x and to store the result in y. We shall prove later—using Hoare logic—that Fac1 really does this.

```
y = 1;
z = 0;
while (z != x) {
    z = z + 1;
    y = y * z;
}
```

## Partial correctness vs. total correctness

There are two readings for a Hoare triple  $(\phi)C(\psi)$ :

**Partial correctness:** if the initial state satisfies  $\phi$  and *C* is executed **and** terminates, then the resulting state satisfies  $\psi$ . We write

 $\models_{par} \llbracket \phi \rrbracket C \llbracket \psi \rrbracket.$ 

**Total correctness**: if the initial state satisfies  $\phi$ , then *C* terminates and the resulting state satisfies  $\psi$ . We write

 $\models_{tot} \llbracket \phi \rrbracket C \llbracket \psi \rrbracket.$ 

## On the meaning of $\models$

- Note that  $\models_{par}$  and  $\models_{tot}$  are not exactly in the same spirit as  $\models$  in propositional logic or predicate logic.
- Hoare logic is the only logic where we deviate from our usual use of ⊨, to be compabtible with old literature.
- There is a more modern version, Hennessy-Milner logic that introduces the "right" notion of ⊨.
- I'll explain this briefly after we've seen Hoare logic.

## Partial correctness vs. total correctness

- Note that total correctness implies partial correctness.
- However, it often happens that partial correctness is proved first, and total correctness in a second step.

#### Two versions of Hoare logic

We shall present two versions of Hoare logic:

 First, we shall present Hoare logic for partial correctness. If a triple is derivable in that logic, we shall write

#### $\vdash_{par} [\![\phi]\!] C [\![\psi]\!].$

 Then we shall modify it to obtain a Hoare logic for total correctness. If a triple is derivable in that logic, we shall write

 $\vdash_{tot} \llbracket \phi \rrbracket C \llbracket \psi \rrbracket.$ 

## Soundness and completeness

Soundness for partial correctness means

 $\vdash_{par} (\phi) C(\psi) \text{ implies } \models_{par} (\phi) C(\psi).$ 

Completeness for partial correctness means

 $\models_{par} (\![\phi]\!] C(\![\psi]\!] \quad \text{implies} \quad \vdash_{par} (\![\phi]\!] C(\![\psi]\!].$ 

Similarly for total correctness.

## Soundness and completeness

- We shall prove soundness (in a slighty informal way) as we go along.
- Completeness holds, but the proof is beyond the scope of this course.

# The shape of formulæ

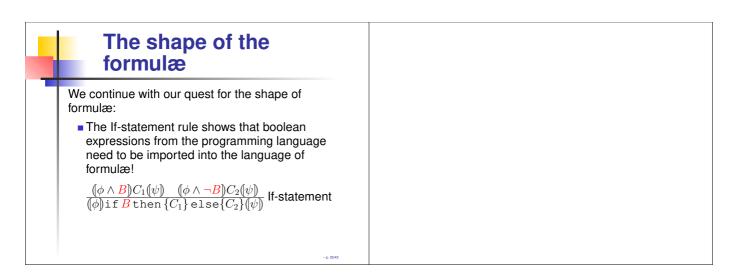
Recall Hoare triples are of the form

 $\llbracket \phi \rrbracket C \llbracket \psi \rrbracket.$ 

- Which shape have the formulæ  $\phi$ ,  $\psi$ ?
- To answer this question, it is useful to take a peek at an inference rule of Hoare logic.

# The rule for if-statements

 $\frac{(\!(\phi \land B)\!)C_1(\!(\psi)\!) \quad (\!(\phi \land \neg B)\!)C_2(\!(\psi)\!)}{(\!\phi)\!) \text{if }B \text{ then } \{C_1\} \text{ else} \{C_2\}(\!(\psi)\!)} \text{ If-statement}$ 



## The shape of the formulæ

 Boolean expressions in turn involve integer expressions:

 $B ::= true | \dots | (B\&B) | (B||B)$ | (E < E) | (E == E) | (E! = E)

## The shape of the formulæ

So the formulæ used in Hoare logic look like formulæ of predicate logic, over a signature whose

- function symbols are the operations +, \*,
   -,... of the programming language (so terms in the sense of predicate logic are the same as the integer expressions of the programming language), and whose
- relation symbols <, =, ≠, ... are only different notations for the operations <, ==, ! =, ... of the programming language.

# The shape of the formulæ

- So the atomic formulæ are essentially the same as boolean expressions of the programming language).
- The boolean connectives ∧, ∨, ¬, ... are only different notations for the operations &, ||, !, ... of the programming language.

## Rules for partial correctness

$\frac{(\!(\phi)\!)C_1(\!(\eta)\!)  (\!(\eta)\!)C_2(\!(\psi)\!)}{(\!(\phi)\!)C_1; C_2(\!(\psi)\!)} \operatorname{Composition} \\ \frac{(\!(\psi)\!E/x\!]_2 = E(\!(\psi)\!)}{(\!(\psi)\!E/x\!]_2 = E(\!(\psi)\!)} \operatorname{Assignment}$
$\frac{(\phi \land B)C_1(\psi)}{(\phi) \text{ if } B \text{ then } \{C_1\} \text{ else}\{C_2\}(\psi)} \text{ If-statement}$
$\frac{(\![\psi \land B]\!]C(\![\psi]\!]}{(\![\psi]\!]while B\{C\}(\![\psi \land \neg B]\!]} \text{ Partial-while }$
$\frac{\vdash \phi' \to \phi  (\phi)C(\psi)  \psi \to \psi'}{(\phi')C(\psi')} \text{ Implied}$

## The rule for composition

The inference rule for composition looks as follows:

 $\frac{(\!(\phi)\!)C_1(\!(\eta)\!) \quad (\!(\eta)\!)C_2(\!(\psi)\!)}{(\!(\phi)\!)C_1;C_2(\!(\psi)\!)} \text{ Composition.}$ 

Thus, if  $C_1$  takes  $\phi$ -states to  $\eta$ -states, and  $C_2$  takes  $\eta$ -states to  $\psi$ -states, then running  $C_1$  followed by  $C_2$  takes  $\phi$ -states to  $\psi$ -states.

## The rule for assignment

Inference rule for assignment:

$$\overline{(\psi[E/x])x = E(\psi)}$$
 Assignment

Rationale behind this rule:

• If the initial state is *s*, then the state *s'* after the assignment is like *s*, except that the variable *x* has now value *E*. Let us write

 $s' = s[x \mapsto E].$ 

If s' is to satisfy  $\psi$ , then which formula  $\phi$  must s satisfy? Answer:  $\phi = \psi[E/x]$ .

# Partial correctness of while-statements

The rule for the partial correctness of while-loops looks as follows:

 $\frac{(\!(\psi \land B)\!)C(\!(\psi)\!)}{(\!(\psi)\!) \texttt{while} B\left\{C\right\}(\!(\psi \land \neg B)\!)} \text{ Partial-while}$ 

The idea is that we have to find some **invariant**  $\psi$ , i.e. some formula that does not change during the execution of *C* (even if the state changes).

#### The rule "Implied"

$$\frac{\vdash \phi' \to \phi \quad (\![\phi]\!] C(\![\psi]\!] \quad \vdash \psi \to \psi'}{(\![\phi']\!] C(\![\psi']\!]} \text{ Implied}$$

- This rule allows the precondition to be strengthened (i.e. to assume more than necessary) and the postcondition to be weakened (i.e. to conclude less then possible).
- It allows to import proofs from predicate logic into the proofs of program logic.

#### . . . .

#### A space issue

Unfortunately, even proofs for the partial correctness of small programs do not fit on one page. For example, an attempt to verify Fac1 yields the proof below:

Proofs get too wide, and a lot of information is copied from one line to the next.

#### **Tableaux**

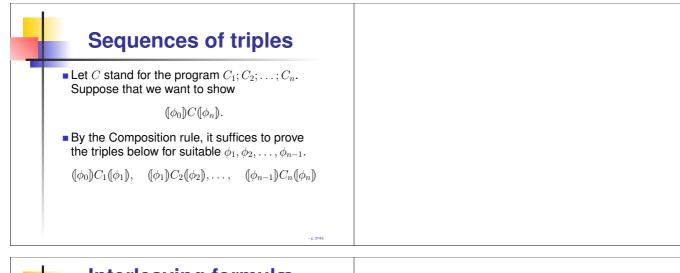
To make Hoare logic easier to use, we introduce a different presentation called **tableaux**. We think of a program as a sequence

 $C_1;$   $C_2;$   $\vdots$   $C_n$ 

where each of the  $C_i$  is either an assignment, an if-statement, or a while-statement.

### Tableaux in logic

- Tableaux are ways of presenting of proofs that are optimized for ease of use.
- There are different of tableaux methods for different logics.
- For example, tableaux for predicate logic (as seen in second-semester course) are very different from the ones for Hoare logic we are now studying.



# Interleaving formulæ with code

This suggest that we should design a calculus which presents a proof of  $(\phi_0)C(\phi_n)$  by interleaving formulas with code as in

## **Midconditions**

- The formulæ φ<sub>1</sub>,..., φ<sub>n-1</sub> are called midconditions.
- Each of the steps

 $\begin{array}{c} \left[\!\left[\phi_i\right]\!\right] \\ C_{i+1}; \\ \left[\!\left[\phi_{i+1}\right]\!\right] \end{array}$ 

will appeal to the lf-statement rule, or the Partial-while rule, or the Assignment rule, depending on  $C_{i+1}$ .

## Weakest preconditions

- Because the Assignment rule works upward, it is most convenient to start with the final condition  $\phi_n$  and work upwards, using  $C_n$  to obtain  $\phi_{n-1}$  and so on.
- Getting  $\phi_i$  from  $\phi_{i+1}$  and  $C_{i+1}$  is mechanical for assignments and if-statements; the  $\phi_i$  so obtained is the **weakest precondition** for  $C_{i+1}$  with postcondition  $\phi_{i+1}$ .
- That is,  $\phi_i$  is the logically weakest formula whose truth at the beginning of the execution of  $C_{i+1}$  is enough to guarantee  $\phi_{i+1}$ .

### **Using tableaux**

- The tableau for  $(\![\phi]\!]C_1; \ldots; C_n(\![\psi]\!]$  is typically constructed by starting at with the precondition  $\psi$  and pushing it upwards through  $C_n$ , then  $C_{n-1}, \ldots$ , until a formula  $\phi'$  emerges at the top.
- $\phi'$  is a precondition which guarantees that the postcondition  $\psi$  will hold if the program terminates.
- Finally, we check if φ' follows from the given precondition φ by using the "Implied" rule.

# Using the "Implied" rule in tableaux

- The "Implied" rule allows us to write one formula  $\phi_2$  directly underneath another formula  $\phi_1$  (with no code in between), if  $\phi_1$  implies  $\phi_2$  in the sense of predicate logic.
- When using the "Implied" rule, we shall not write out the predicate-logic proof of  $\phi_1 \vdash \phi_2$ , because we focus on the program logic.

### **Example tableau**

- See blackboard.
- We create the proof bottom-up.
- For checking the proof, it also makes sense to proceed top-down.

– p. 43