

# Mathematical Structures in Programming

Zhenjiang Hu

Summer Term, 2002

# プログラム構造論

- 講義内容
  - 構成的手法に基づくプログラミング方法論を扱う。
- 工学系研究科の講義名: 計算機言語論 (特論)
- 担当教官
  - 名前: 胡 振江 (HU, Zhenjiang)
  - 居場所: 工学部 9 号館 133 室・工学部 6 号館 350 室
  - URL: <http://www.ipl.t.u-tokyo.ac.jp/~hu>
  - Email: [hu@ipl.t.u-tokyo.ac.jp](mailto:hu@ipl.t.u-tokyo.ac.jp)
- 成績
  - 出席: 40%
  - レポート (中間と期末の 2 回): 30% + 30%
- 講義 URL: <http://www.ipl.t.u-tokyo.ac.jp/~hu/pub/teach/msp02>

# Introduction

## What is Programming?

- Programming is the art of designing efficient algorithms that meet their specification.
- Two factors by which algorithms may be judged:
  - Correctness: do they solve the right problems
  - Performance: how fast do they run and how much space do they use
- Classical way of judging the quality of an algorithm is
  - by tracing execution patterns,
  - by providing test inputs, or
  - by supplying formal proofs.

## Verification of Algorithms

- Verification is the process of proving the correctness of an algorithm after it has been designed.
- Verification of algorithms is important:
  - bank systems
  - flight scheduling systems

But it is often regarded as a waste of time and were largely rejected or neglected by the software community.

- Verification of algorithms is difficult, including
  - development of specification languages
  - tools supporting program verification

*Could a program and its verification be constructed hand in hand, while making a posteriori program verification superfluous?*

## Calculational Style of Programming

- Developed by E.W. Dijkstra and others during 1970s.
- Programs are derived from their specification by formula manipulation.
  - The calculation that leads to the algorithm are carried out in small steps;
  - Each individual step is easily verified.
- In this way the design decision is manifest.
  - Program derivation is not mechanical; it is challenging activity and it requires creativity.

*this calculational way of programming shows where creativity comes in. It is this method that will be explained and exemplified in this class!*

## Two Views of Programming

- A Common View: a program is a recipe, which
  - explains what steps have to be performed to achieve a certain goal.

```
first do this;  
then apply that;  
perform the following N times;
```

...

- Another View: a program together with its specification is a theorem, which
    - expresses that the program satisfy the specification.
- ⇒ all programs require proofs (as theorems do).

*We shall derive programs according to their specification in a constructive way, such that program development and correctness proof go hand in hand.*

# An Example of Specification and Program

- A Specification:

```

||
var  $x, y$  : int      state space  $\mathcal{Z} \times \mathcal{Z}$ 
 $\{x = A \wedge y = B\}$     precondition
maximum             program under construct
 $\{x = AmaxB\}$         postcondition
||

```

- A Program:

```

||
var  $x, y$  : int
 $\{x = A \wedge y = B\}$ 
if  $x < y \rightarrow x := y$  ||  $x \geq y \rightarrow \text{skip}$  fi
 $\{x = AmaxB\}$ 
||

```



## The Textbook and References

- The Textbook
  - A. Kaldewaij, *Programming: The Derivation of Algorithms*, Pretence-Hall, 1990.
- References
  - First book on science of programming:
    - \* E.W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, 1976.
  - Many notations and exercises follow the two books:
    - \* D. Gries, *The Science of Programming*, Springer Verlag, 1981.
    - \* E.W. Dijkstra and W.H.J. Feijen, *A Method of Programming*, Addison Wesley, 1988.
  - Other good books:
    - \* Spivey, *Programming from Specification*, Prentice-Hall, 1990.
    - \* R. Bird and O. de Moor, *Algebras of Programming*, Prentice-Hall, 1996.

# Chapter 1: Predicate Calculus

## Predicates on State Space

- A predicate is a function on a state space  $\mathcal{X}$  (set of program variables):

$$P : \mathcal{X} \rightarrow \{\text{false}, \text{true}\}$$

which actually defines a subset of  $\mathcal{X}$ .

- An Example
  - Suppose we have two program variables

$x, y$  : integer

then the state space  $\mathcal{X}$  is

$$\mathcal{X} = \mathbb{Z} \times \mathbb{Z}.$$

A predicate on  $\mathcal{X}$  is

$$x \geq 2 \wedge y = 3.$$

## Predicate Logic

- Predicates
  - Constants: true, false
  - Negation:  $\neg P$
  - Conjunction:  $P \wedge Q$
  - Disjunction:  $P \vee Q$
  - Implication:  $P \Rightarrow Q$
  - Equivalence:  $P \equiv Q$
- Note: Negation has the highest priority while equivalence has the lowest.

$$P \Rightarrow Q \equiv \neg P \vee Q$$

should be read as

$$(P \Rightarrow Q) \equiv ((\neg P) \vee Q)$$

## Validity

- $P$  is true for all states is denoted by  $[P]$ .
- Examples
  - $[\text{true}]$
  - $[Q \equiv Q]$
  - $[\neg(P \wedge Q) \equiv \neg P \vee \neg Q]$ , many and many
  - $[(x + 1)^2 = x^2 + 2x + 1]$
  - $[x \geq 1 \Rightarrow x \geq 0]$
- If  $[P \Rightarrow Q]$ , then
  - $P$  is stronger than  $Q$ , or
  - $Q$  is weaker than  $P$ .

Q: What is the weakest predicate and what is the strongest predicate?

## Substitution

- Substitution of expression  $E$  for variable  $x$  in expression  $Q$  is denoted by

$$Q(x := E)$$

- Examples

- $[(x^2 + 2 * x)(x := x + 1) = (x + 1)^2 + 2 * (x + 1)]$
- $[(x + 2 * y = z)(x, y := y, x) \equiv y + 2 * x = z]$
- $[(x = E)(x := E) \equiv E = E(x := E)]$
- $[(P(x := y))(x := y) \equiv P(x := y)]$
- $[(P(x := y))(y := x) \equiv P(y := x)]$
- $[(P \wedge Q)(x := E) \equiv P(x := E) \wedge Q(x := E)]$

## Quantification

- Existential quantification:

$$(\exists i : R : P)$$

- $i$ : a bound variable, or a dummy (of type  $\mathcal{Z}$ )
- $R$ : a range
- $P$ : a term

Examples:

- $(\exists i : i \in \mathcal{Z} \wedge i \geq 0 : x = 2i)$
- $[(\exists i : \text{false} : P) \equiv \text{false}]$

- Universal quantification:

$$(\forall i : R : P)$$

## Chapter 2: The Guarded Command Language



## Hoare's Triples

$$\{P\}S\{Q\}$$

each execution of  $S$  in a state satisfying  $P$  terminates in a state satisfying  $Q$ .

- $P$ : precondition or assertion
- $S$ : statement (program)
- $Q$ : postcondition or assertion

## General Rules of Programs

- Trivial ( $S$  will not be executed.)

$$\{\text{false}\}S\{P\}$$

- Termination

$$\{P\}S\{\text{true}\}$$

- Excluded Miracles

$$\{P\}S\{\text{false}\} \text{ is equivalent to } [P \equiv \text{false}]$$

- Strengthening of Precondition

$$\{P\}S\{Q\} \text{ and } [P_0 \Rightarrow P] \text{ implies } \{P_0\}S\{Q\}$$

- Weakening of Postcondition

$$\{P\}S\{Q\} \text{ and } [Q \Rightarrow Q_0] \text{ implies } \{P\}S\{Q_0\}$$

- Conjunction

$$\{P\}S\{Q\} \text{ and } \{P\}S\{R\} \text{ is equivalent to } \{P\}S\{Q \wedge R\}$$

- Disjunction

$$\{P\}S\{Q\} \text{ and } \{R\}S\{Q\} \text{ is equivalent to } \{P \vee R\}S\{Q\}$$

## Predicate Transformer

- $wp.S.Q$  denotes the weakest precondition of  $S$  with respect to  $Q$ .

$\{P\}S\{Q\}$  is equivalent to  $[P \Rightarrow wp.S.Q]$

- Examples:
  - $[wp.S.false \equiv false]$
  - $[wp.S.Q \wedge wp.S.R \equiv wp.S.(Q \wedge R)]$
  - $[wp.S.Q \vee wp.S.R \Rightarrow wp.S.(Q \vee R)]$

## Exercises

### Problem 1

now that

$$\{P_0\}S\{Q_0\} \text{ and } \{P_1\}S\{Q_1\}$$

applies

$$\{P_0 \wedge P_1\}S\{Q_0 \wedge Q_1\} \text{ and } \{P_0 \vee P_1\}S\{Q_0 \vee Q_1\}.$$