

# Advanced Topics in Theoretical Computer Science

## Part 2: Register machines

26.4.2016

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: [sofronie@uni-koblenz.de](mailto:sofronie@uni-koblenz.de)

# Contents

---

- Recapitulation: Turing machines and Turing computability
- Register machines (LOOP, WHILE, GOTO)
- Recursive functions
- The Church-Turing Thesis
- Computability and (Un-)decidability
- Complexity
- Other computation models: e.g. Büchi Automata,  $\lambda$ -calculus

# Contents

---

- Recapitulation: Turing machines and Turing computability
- Register machines (LOOP, WHILE, GOTO)
- Recursive functions
- The Church-Turing Thesis
- Computability and (Un-)decidability
- Complexity
- Other computation models: e.g. Büchi Automata,  $\lambda$ -calculus

## 2. Register Machines

---

- Register machines (Random access machines)
- LOOP Programs
- WHILE Programs
- GOTO Programs
- Relationships between LOOP, WHILE, GOTO
- Relationships between register machines and Turing machines

## 2. Register Machines

---

- Register machines (Random access machines)
- LOOP Programs
- WHILE Programs
- GOTO Programs
- Relationships between LOOP, WHILE, GOTO
- Relationships between register machines and Turing machines

# Register Machines

---

The register machine gets its name from its one or more “registers”:

In place of a Turing machine’s tape and head (or tapes and heads) the model uses multiple, uniquely-addressed registers, each of which holds a single positive integer.

# Register Machines

---

In comparison to Turing machines:

- equally powerful fundament for computability theory
- **Advantage:** Programs are easier to understand

# Register Machines

---

In comparison to Turing machines:

- equally powerful fundament for computability theory
- **Advantage:** Programs are easier to understand

similar to ...

the imperative kernel of programming languages

pseudo-code



# Register Machines

---

Computation of  $a \bmod b$  (pseudocode)

$r := a;$

while  $r \geq b$  do

$r := r - b$

end;

return  $r$

# Register Machines

---

## Definition: Questions

Which instructions (if, while, goto?)

# Register Machines

---

## Definition: Questions

Which instructions (if, while, goto?)

Which data types? (integers? strings?)

# Register Machines

---

## Definition: Questions

Which instructions (if, while, goto?)

Which data types? (integers? strings?)

Which data structures? (arrays?)

# Register Machines

---

## Definition: Questions

Which instructions (if, while, goto?)

Which data types? (integers? strings?)

Which data structures? (arrays?)

Which atomic instructions?

# Register Machines

---

## Definition: Questions

Which instructions (if, while, goto?)

Which data types? (integers? strings?)

Which data structures? (arrays?)

Which atomic instructions?

Which Input/Output?

# Register Machines

---

## Settings (Informally)

- **Instruction set:**
  - Various variants:  
loop or while or if + goto

# Register Machines

---

## Settings (Informally)

- **Instruction set:**
  - Various variants:  
loop or while or if + goto
- **Data types:**
  - The natural numbers.  
This is the only difference to normal computers



# Register Machines

---

## Settings (Informally)

- **Instruction set:**
  - Various variants:  
loop or while or if + goto
- **Data types:**
  - The natural numbers.  
This is the only difference to normal computers
- **Data structures**
  - Unbounded but finite number of registers denoted  $x_1, x_2, x_3 \dots, x_n$ ;  
each register contains a natural number  
(no arrays, objects, ...)

# Register Machines

---

## Settings (Informally)

- **Atomic instructions:**
  - Increment/Decrement a register

# Register Machines

---

## Settings (Informally)

- **Atomic instructions:**
  - Increment/Decrement a register
- **Input/Output**
  - **Input:**  $n$  input values in the first  $n$  registers  
All the other registers are 0 at the beginning.
  - **Output:** In register  $n + 1$ .

# Example: LOOP Programs

---

## Syntax

### Definition

- **Atomic programs:** For each register  $x_i$ :
  - $x_i := x_i + 1$
  - $x_i := x_i - 1$are LOOP instructions and also LOOP programs.

# Example: LOOP Programs

---

## Syntax

### Definition

- **Atomic programs:** For each register  $x_i$ :
  - $x_i := x_i + 1$
  - $x_i := x_i - 1$are LOOP instructions and also LOOP programs.
- If  $P_1, P_2$  are LOOP programs then
  - $P_1; P_2$  is a LOOP program

# Example: LOOP Programs

---

## Syntax

### Definition

- **Atomic programs:** For each register  $x_i$ :
  - $x_i := x_i + 1$
  - $x_i := x_i - 1$are LOOP instructions and also LOOP programs.
- If  $P_1, P_2$  are LOOP programs then
  - $P_1; P_2$  is a LOOP program
- If  $P$  is a LOOP program then
  - `loop  $x_i$  do  $P$  end` is a LOOP instruction and a LOOP program.

# Example: LOOP Programs

---

## Syntax

### Definition

- **Atomic programs:** For each register  $x_i$ :
  - $x_i := x_i + 1$
  - $x_i := x_i - 1$are **LOOP** instructions and also **LOOP** programs.
- If  $P_1, P_2$  are **LOOP** programs then
  - $P_1; P_2$  is a **LOOP** program
- If  $P$  is a **LOOP** program then
  - **loop**  $x_i$  **do**  $P$  **end** is a **LOOP** program (and a **LOOP** instruction)

# Example: **WHILE** Programs

---

## Syntax

### Definition

- **Atomic programs:** For each register  $x_i$ :
  - $x_i := x_i + 1$
  - $x_i := x_i - 1$are **WHILE** instructions and also **WHILE** programs.
- If  $P_1, P_2$  are **WHILE** programs then
  - $P_1; P_2$  is a **WHILE** program
- If  $P$  is a **WHILE** program then
  - **while**  $x_i \neq 0$  **do**  $P$  **end** is a **WHILE** program (and a **WHILE** instruction)



# Example: GOTO Programs

---

**Syntax** Indexes (numbers for the lines in the program)  $j \geq 0$

## Definition

- **Atomic programs:**

- $x_i := x_i + 1$

- $x_i := x_i - 1$

are **GOTO** instructions for each register  $x_i$ .

- If  $x_i$  is a register and  $j$  is an index then

- if  $x_i = 0$  goto  $j$  is a **GOTO** instruction.

- If  $l_1, \dots, l_k$  are GOTO instructions and  $j_1, \dots, j_k$  are indices then

- $j_1 : l_1; \dots; j_k : l_k$  is a **GOTO program**

# Register Machines

---

## Definition

A register machine is a machine consisting of the following elements:

- A finite (but unbounded) number of registers  $x_1, x_2, x_3 \dots, x_n$ ; each register contains a natural number.
- A LOOP-, WHILE- or GOTO-program.

# Register Machines: State

---

## Definition (State of a register machine)

The state  $s$  of a register machine is a map:

$$s : \{x_i \mid i \in \mathbb{N}\} \rightarrow \mathbb{N}$$

which associates with every register a natural number as value.

# Register Machines: State

---

## Definition (Initial state; Input)

Let  $m_1, \dots, m_k \in \mathbb{N}$  be given as input to a register machine.

In the input state  $s_0$  we have

- $s_0(x_i) = m_i$  for all  $1 \leq i \leq k$
- $s_0(x_i) = 0$  for all  $i > k$

# Register Machines: State

---

## Definition (Initial state; Input)

Let  $m_1, \dots, m_k \in \mathbb{N}$  be given as input to a register machine.

In the input state  $s_0$  we have

- $s_0(x_i) = m_i$  for all  $1 \leq i \leq k$
- $s_0(x_i) = 0$  for all  $i > k$

## Definition (Output)

If a register machine started with the input  $m_1, \dots, m_k \in \mathbb{N}$

halts in a state  $s_{\text{term}}$  then:

$$s_{\text{term}}(x_{k+1})$$

is the output of the machine.

# Register Machines: Semantics

---

## Definition (The semantics of a register machine)

The semantics  $\Delta(P)$  of a register machine  $P$  is a (binary) relation

$$\Delta(P) \subseteq S \times S$$

on the set  $S$  of all states of the machine.

$(s_1, s_2) \in \Delta(P)$  means that if  $P$  is executed in state  $s_1$  then it halts in state  $s_2$ .

# Register Machines: Computed function

---

## Definition (Computed function)

A register machine  $P$  computes a function

$$f : \mathbb{N}^k \rightarrow \mathbb{N}$$

if and only if for all  $m_1, \dots, m_k \in \mathbb{N}$  the following holds:

If we start  $P$  with initial state with the input  $m_1, \dots, m_k$  then:

- $P$  terminates if and only if  $f(m_1, \dots, m_k)$  is defined
- If  $P$  terminates, then the output of  $P$  is  $f(m_1, \dots, m_k)$
- **Additional condition** (next page)

# Register Machines: Computed function

---

## Definition (Computed function) (ctd.)

### **Additional condition**

We additionally require that when a register machine halts, all the registers (with the exception of the output register) contain again the values they had in the initial state.

- Input registers  $x_1, \dots, x_k$  contain the initial values
- The registers  $x_i$  with  $i > k + 1$  contain value 0



# Register Machines: Computed function

---

## Definition (Computed function) (ctd)

### Additional condition

We additionally require that when a register machine halts, all the registers (with the exception of the output register) contain again the values they had in the initial state.

- Input registers  $x_1, \dots, x_k$  contain the initial values
- The registers  $x_i$  with  $i > k + 1$  contain value 0

**Consequence:** A machine which does not fulfill the additional condition (even only for some inputs) does not compute a function at all.

# Register Machines: Computable function

---

## Example:

The program:

```
 $P := \text{loop } x_2 \text{ do } x_2 := x_2 - 1 \text{ end; } x_2 := x_2 + 1;$   
     $\text{loop } x_1 \text{ do } x_1 := x_1 - 1 \text{ end}$ 
```

does not compute a function: At the end,  $P$  has value 0 in  $x_1$  and 1 in  $x_2$ .

# Register Machines: Computable function

---

**Definition.** A function  $f$  is

- **LOOP computable** if there exists a register machine with a LOOP program, which computes  $f$

# Register Machines: Computable function

---

**Definition.** A function  $f$  is

- **LOOP computable** if there exists a register machine with a LOOP program, which computes  $f$
- **WHILE computable** if there exists a register machine with a WHILE program, which computes  $f$

# Register Machines: Computable function

---

**Definition.** A function  $f$  is

- **LOOP computable** if there exists a register machine with a LOOP program, which computes  $f$
- **WHILE computable** if there exists a register machine with a WHILE program, which computes  $f$
- **GOTO computable** if there exists a register machine with a GOTO program, which computes  $f$

# Register Machines: Computable function

---

**Definition.** A function  $f$  is

- **LOOP computable** if there exists a register machine with a LOOP program, which computes  $f$
- **WHILE computable** if there exists a register machine with a WHILE program, which computes  $f$
- **GOTO computable** if there exists a register machine with a GOTO program, which computes  $f$
- **TM computable** if there exists a Turing machine which computes  $f$

# Register Machines: Computable function

---

**Definition.** A function  $f$  is

- **LOOP computable** if there exists a register machine with a LOOP program, which computes  $f$
- **WHILE computable** if there exists a register machine with a WHILE program, which computes  $f$
- **GOTO computable** if there exists a register machine with a GOTO program, which computes  $f$
- **TM computable** if there exists a Turing machine which computes  $f$

LOOP = Set of all LOOP computable functions

WHILE = Set of all WHILE computable functions

GOTO = Set of all GOTO computable functions

TM = Set of all TM computable functions

# Register Machines: Overview

---

- Register machines (Random access machines)
- LOOP Programs
- WHILE Programs
- GOTO Programs
- Relationships between LOOP, WHILE, GOTO
- Relationships between register machines and Turing machines



# LOOP Programs: Syntax

---

## Definition

- **Atomic programs:** For each register  $x_i$ :
  - $x_i := x_i + 1$
  - $x_i := x_i - 1$are LOOP instructions and also LOOP programs.
- If  $P_1, P_2$  are LOOP programs then
  - $P_1; P_2$  is a LOOP program
- If  $P$  is a LOOP program then
  - `loop  $x_i$  do  $P$  end` is a LOOP instruction and a LOOP program.

# LOOP Programs: Semantics

---

## Definition (Semantics of LOOP programs)

Let  $P$  be a LOOP program.  $\Delta(P)$  is inductively defined as follows:

(1) On atomic programs:

- $\Delta(x_i := x_i + 1)(s_1, s_2)$  if and only if:
  - $s_2(x_i) = s_1(x_i) + 1$
  - $s_2(x_j) = s_1(x_j)$  for all  $j \neq i$

# LOOP Programs: Semantics

---

## Definition (Semantics of LOOP programs)

Let  $P$  be a LOOP program.  $\Delta(P)$  is inductively defined as follows:

(1) On atomic programs:

- $\Delta(x_i := x_i + 1)(s_1, s_2)$  if and only if:
  - $s_2(x_i) = s_1(x_i) + 1$
  - $s_2(x_j) = s_1(x_j)$  for all  $j \neq i$
- $\Delta(x_i := x_i - 1)(s_1, s_2)$  if and only if:
  - $s_2(x_i) = \begin{cases} s_1(x_i) - 1 & \text{if } s_1(x_i) > 0 \\ 0 & \text{if } s_1(x_i) = 0 \end{cases}$
  - $s_2(x_j) = s_1(x_j)$  for all  $j \neq i$

# LOOP Programs: Semantics

---

## Definition (Semantics of LOOP programs)

Let  $P$  be a LOOP program.  $\Delta(P)$  is inductively defined as follows:

(2) Sequential composition:

- $\Delta(P_1; P_2)(s_1, s_2)$  if and only if there exists  $s'$  such that:
  - $\Delta(P_1)(s_1, s')$
  - $\Delta(P_2)(s', s_2)$

# LOOP Programs: Semantics

---

## Definition (Semantics of LOOP programs ctd.)

Let  $P$  be a LOOP program.  $\Delta(P)$  is inductively defined as follows:

### (3) Loop programs

- $\Delta(\text{loop } x_i \text{ do } P \text{ end})(s_1, s_2)$  if and only if there exist states  $s'_0, s'_1, \dots, s'_n$  with:
  - $s_1(x_i) = n$
  - $s_1 = s'_0$
  - $s_2 = s'_n$
  - $\Delta(P)(s'_k, s'_{k+1})$  for  $0 \leq k < n$

# LOOP Programs: Semantics

## Definition (Semantics of LOOP programs ctd.)

Let  $P$  be a LOOP program.  $\Delta(P)$  is inductively defined as follows:

### (3) Loop programs

- $\Delta(\text{loop } x_i \text{ do } P \text{ end})(s_1, s_2)$  if and only if there exist states  $s'_0, s'_1, \dots, s'_n$  with:
  - $s_1(x_i) = n$
  - $s_1 = s'_0$
  - $s_2 = s'_n$
  - $\Delta(P)(s'_k, s'_{k+1})$  for  $0 \leq k < n$

### Remark:

The number of steps in the loop is the value of  $x_i$  at the beginning of the loop. Changes to  $x_i$  during the loop are not relevant.

# LOOP programs: Semantics

---

**Program end:** If there is no next program line, then the program execution terminates.

We say that a LOOP program terminates on an input  $n_1, \dots, n_k$  if its execution on this input terminates (in the sense above) after a finite number of steps.

# LOOP computable functions

---

**Theorem.** Every LOOP program terminates for every input.



# LOOP computable functions

---

**Theorem.** Every LOOP program terminates for every input.

**Proof (Idea):** We prove by induction on the structure of a LOOP program that all LOOP programs terminate:

**Induction basis:** Show that all atomic programs terminate (simple)

Let  $P$  be a non-atomic LOOP program.

**Induction hypothesis:** We assume that all subprograms of  $P$  terminate on all inputs.

**Induction step:** We prove that then  $P$  terminates on every input as well.

**Case 1:**  $P = P_1; P_2$  (Proof: Ind. hypothesis:  $P_1$  and  $P_2$  terminate, so  $P$  terminates)

**Case 2:**  $P = \text{loop } x_i \text{ do } P_1 \text{ end}$

Proof: By the Induction hypothesis,  $P_1$  terminates. Since the number of steps in the loop (the initial value of  $x_i$ ) is fixed, no infinite loop is possible.

# LOOP computable functions

---

**Theorem.** Every LOOP program terminates for every input.

**Proof (Idea):** We prove by induction on the structure of a LOOP program that all LOOP programs terminate:

**Induction basis:** Show that all atomic programs terminate (simple)

Let  $P$  be a non-atomic LOOP program.

**Induction hypothesis:** We assume that all subprograms of  $P$  terminate on all inputs.

**Induction step:** We prove that then  $P$  terminates on every input as well.

**Case 1:**  $P = P_1; P_2$  (Proof: Ind. hypothesis:  $P_1$  and  $P_2$  terminate, so  $P$  terminates)

**Case 2:**  $P = \text{loop } x_i \text{ do } P_1 \text{ end}$

Proof: By the Induction hypothesis,  $P_1$  terminates. Since the number of steps in the loop (the initial value of  $x_i$ ) is fixed, no infinite loop is possible.

**Consequence:** All LOOP computable functions are total.