

# Advanced Topics in Theoretical Computer Science

## Part 2: Register machines (2)

15.11.2017

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

## 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

# Last time: 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.

## 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

# Last time: 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.

# Last time: 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.

## 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.

# Last time: 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  $s_2$ .

# Last time: 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**

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.



# Last time: Computed 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  $x_2$ .

# Last time: 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  $f$

## Last time: Computable function

---

- LOOP = Set of all **total** LOOP computable functions
  - WHILE = Set of all **total** WHILE computable functions
  - GOTO = Set of all **total** GOTO computable functions
  - TM = Set of all **total** TM computable functions
- 
- WHILE<sup>part</sup> = Set of all **total or partial** WHILE computable functions
  - GOTO<sup>part</sup> = Set of all **total or partial** GOTO computable functions
  - TM<sup>part</sup> = Set of all **total or partial** 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

# Last time: LOOP Programs - Syntax

---

## Definition

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

The set of all LOOP programs is the smallest set with the properties (1),(2),(3).

# Last time: 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  $j \neq i$

# Last time: 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:
  - $\Delta(P_1)(s_1, s')$
  - $\Delta(P_2)(s', s_2)$

### (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.

# Last time: 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.

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

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



# LOOP Programs

---

## Additional instructions

- $x_i := 0$

loop  $x_i$  do  $x_i := x_i - 1$  end

- $x_i := c$  for  $c \in \mathbb{N}$

$x_i := 0;$   
 $x_i := x_i + 1;$   
...  
 $x_i := x_i + 1$  } c times

- $x_i := x_j$

$x_i := 0;$   
loop  $x_j$  do  $x_i := x_i + 1$  end

# LOOP Programs

---

## Additional instructions

- $x_i := x_j + x_k$

$x_i := x_j;$

loop  $x_k$  do  $x_i := x_i + 1$  end

- $x_i := x_j - x_k$

$x_i := x_j;$

loop  $x_k$  do  $x_i := x_i - 1$  end

- $x_i := x_j * x_k$

$x_i := 0;$

loop  $x_k$  do  $x_i := x_i + x_j$  end

# LOOP Programs

---

## Additional instructions

In what follows,  $x_n, x_{n+1}, \dots$  denote new registers (not used before).

- $x_j := e_1 + e_2$  ( $e_1, e_2$  arithmetical expressions)  
 $x_j := e_1;$   
 $x_n := e_2;$   
loop  $x_n$  do  $x_j := x_j + 1$  end;  $x_n := 0$
- $x_j := e_1 - e_2$  ( $e_1, e_2$  arithmetical expressions)  
 $x_j := e_1;$   
 $x_n := e_2;$   
loop  $x_n$  do  $x_j := x_j - 1$  end;  $x_n := 0$
- $x_j := e_1 * e_2$  ( $e_1, e_2$  arithmetical expressions)  
 $x_j := 0;$   
 $x_n := e_1;$   
loop  $x_n$  do  $x_j := x_j + e_2$  end;  $x_n := 0$

# LOOP Programs

---

## Additional instructions

- if  $x_i = 0$  then  $P_1$  else  $P_2$  end  
   $x_n := 1 - x_i$ ;  
   $x_{n+1} := 1 - x_n$ ;  
  loop  $x_n$  do  $P_1$  end;  
  loop  $x_{n+1}$  do  $P_2$  end;  
   $x_n := 0; x_{n+1} := 0$
- if  $x_i \leq x_j$  then  $P_1$  else  $P_2$  end  
   $x_n := x_i - x_j$ ;  
  if  $x_n = 0$  then  $P_1$  else  $P_2$  end  
   $x_n := 0$

# 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

# 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 **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** instruction and a **WHILE** program.

The family of all WHILE programs is the smallest set with properties (1),(2),(3)

# WHILE Programs: Semantics

---

## Definition (Semantics of WHILE programs)

Let  $P$  be a WHILE 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  $j \neq i$

# WHILE Programs: Semantics

---

## Definition (Semantics of WHILE programs)

Let  $P$  be a WHILE 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:
  - $\Delta(P_1)(s_1, s')$
  - $\Delta(P_2)(s', s_2)$



# WHILE Programs: Semantics

## Definition (Semantics of WHILE programs ctd.)

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

### (3) While programs

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

# WHILE Programs: Semantics

## Definition (Semantics of WHILE programs ctd.)

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

### (3) While programs

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

**Remark:** The number of loop iterations is not fixed at the beginning. The contents of  $P$  may influence the number of iterations. Infinite loop are possible.

# WHILE and LOOP

**Theorem.**  $\text{LOOP} \subseteq \text{WHILE}$

i.e., every LOOP computable function is also WHILE computable

**Proof (Idea)** We first show that the LOOP instruction “loop  $x_i$  do  $P$  end” can be simulated by the following WHILE program  $P_{\text{while}}$ :

```
while  $x_i \neq 0$  do                                     ** simulate  $x_n := x_i$  **
   $x_n := x_n + 1; x_{n+1} := x_{n+1} + 1; x_i := x_i - 1$ 
end;

while  $x_{n+1} \neq 0$  do                                  ** restore  $x_i$  **
   $x_i := x_i + 1; x_{n+1} := x_{n+1} - 1$ 
end;

while  $x_n \neq 0$  do                                     ** simulate the loop instruction **
   $P; x_n := x_n - 1$ 
end
```

Here  $x_n, x_{n+1}$  are new registers (in which at the beginning 0 is stored; not used in  $P$ ).

# WHILE and LOOP

---

It is easy to see that the new WHILE program  $P_{\text{while}}$  “simulates”  
`loop  $x_i$  do  $P$  end` , i.e.

$$(s, s') \in \Delta(\text{loop } x_i \text{ do } P \text{ end}) \text{ iff } (s, s') \in \Delta(P_{\text{while}})$$

Using this, it can be proved (by structural induction) that every LOOP program can be simulated by a WHILE program.

# WHILE and LOOP

---

**Theorem.**  $\text{LOOP} \subseteq \text{WHILE}$  (every LOOP computable function is WHILE computable)

Proof: Structural induction

**Induction basis:** We show that the property is true for all atomic LOOP programs, i.e. for programs of the form  $x_i := x_i + 1$  and of the form  $x_i := x_i - 1$ .  
(Obviously true, because these programs are also WHILE programs).

# WHILE and LOOP

---

**Theorem.**  $\text{LOOP} \subseteq \text{WHILE}$  (every LOOP computable function is WHILE computable)

**Proof:** Structural induction

**Induction basis:** We show that the property is true for all atomic LOOP programs, i.e. for programs of the form  $x_i := x_i + 1$  and of the form  $x_i := x_i - 1$ . (Obviously true, because these programs are also WHILE programs).

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

**Induction hypothesis:** We assume that the property holds for all “subprograms” of  $P$ .

**Induction step:** We show that then it also holds for  $P$ . Proof depends on form of  $P$ .

# WHILE and LOOP

**Theorem.**  $\text{LOOP} \subseteq \text{WHILE}$  (every LOOP computable function is WHILE computable)

**Proof:** Structural induction

**Induction basis:** We show that the property is true for all atomic LOOP programs, i.e. for programs of the form  $x_i := x_i + 1$  and of the form  $x_i := x_i - 1$ . (Obviously true, because these programs are also WHILE programs).

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

**Induction hypothesis:** We assume that the property holds for all “subprograms” of  $P$ .

**Induction step:** We show that then it also holds for  $P$ . Proof depends on form of  $P$ .

**Case 1:**  $P = P_1; P_2$ . By the induction hypothesis, there exist WHILE programs  $P'_1, P'_2$  with  $\Delta(P_i) = \Delta(P'_i)$ . Let  $P' = P'_1; P'_2$  (a WHILE program).

$$\begin{aligned} \Delta(P')(s_1, s_2) & \text{ iff } \text{there exists } s \text{ with } \Delta(P'_1)(s_1, s) \text{ and } \Delta(P'_2)(s, s_2) \\ & \text{ iff } \text{there exists } s \text{ with } \Delta(P_1)(s_1, s) \text{ and } \Delta(P_2)(s, s_2) \quad \text{iff} \quad \Delta(P)(s_1, s_2) \end{aligned}$$

# WHILE and LOOP

**Theorem.**  $\text{LOOP} \subseteq \text{WHILE}$  (every LOOP computable function is WHILE computable)

**Proof:** Structural induction

**Induction basis:** We show that the property is true for all atomic LOOP programs, i.e. for programs of the form  $x_i := x_i + 1$  and of the form  $x_i := x_i - 1$ . (Obviously true, because these programs are also WHILE programs).

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

**Induction hypothesis:** We assume that the property holds for all “subprograms” of  $P$ .

**Induction step:** We show that then it also holds for  $P$ . Proof depends on form of  $P$ .

**Case 1:**  $P = P_1; P_2$ . By the induction hypothesis, there exist WHILE programs  $P'_1, P'_2$  with  $\Delta(P_i) = \Delta(P'_i)$ . Let  $P' = P'_1; P'_2$  (a WHILE program).

$$\begin{aligned} \Delta(P')(s_1, s_2) & \text{ iff } \text{there exists } s \text{ with } \Delta(P'_1)(s_1, s) \text{ and } \Delta(P'_2)(s, s_2) \\ & \text{ iff } \text{there exists } s \text{ with } \Delta(P_1)(s_1, s) \text{ and } \Delta(P_2)(s, s_2) \quad \text{iff } \Delta(P)(s_1, s_2) \end{aligned}$$

**Case 2:**  $P = \text{loop } x_i \text{ do } P_1$ . By the induction hypothesis, there exists a WHILE program  $P'_1$  with  $\Delta(P_1) = \Delta(P'_1)$ . Let  $P'$  be the following WHILE program:

$P' =$  while  $x_i \neq 0$  do  $x_n := x_n + 1; x_{n+1} := x_{n+1} + 1; x_i := x_i - 1$  end;  
while  $x_{n+1} \neq 0$  do  $x_i := x_i + 1; x_{n+1} := x_{n+1} - 1$  end; while  $x_n \neq 0$  do  $P'_1; x_n := x_n - 1$  end

$\Delta(P')(s_1, s_2) = \Delta(P)(s_1, s_2)$  (show that  $P$  and  $P'$  change values of registers in the same way).



# LOOP $\subseteq$ WHILE

---

## Consequences of the proof:

### Corollary

The instructions defined in the context of LOOP programs:

$x_i := c$        $x_i := x_j$        $x_i := x_j + c$        $x_i := x_j + x_k$        $x_i = x_j * x_k,$   
if  $x_i = 0$  then  $P_i$  else  $P_j$       if  $x_i \leq x_j$  then  $P_i$  else  $P_j$

can also be used in WHILE programs.

# Partial WHILE computable functions

---

## Non-termination

WHILE programs can contain infinite loops. Therefore:

- WHILE programs do not always terminate
- WHILE computable functions can be undefined for some inputs (are partial functions)

# Partial WHILE computable functions

---

## Non-termination

WHILE programs can contain infinite loops. Therefore:

- WHILE programs do not always terminate
- WHILE computable functions can be undefined for some inputs (are partial functions)

**Example:**  $P := \text{while } x_1 \neq 0 \text{ do } x_1 := x_1 + 1 \text{ end}$

computes  $f : \mathbb{N} \rightarrow \mathbb{N}$  with:

$$f(n) := \begin{cases} 0 & \text{if } n = 0 \\ \text{undefined} & \text{if } n \neq 0 \end{cases}$$

# Partial WHILE computable functions

---

## Non-termination

WHILE programs can contain infinite loops. Therefore:

- WHILE programs do not always terminate
- WHILE computable functions can be undefined for some inputs (are partial functions)

## Notation

- $\text{WHILE}$  = The set of all **total** WHILE computable functions
- $\text{WHILE}^{\text{part}}$  = The set of **all** WHILE computable functions (including the partial ones)

# Partial WHILE computable functions

---

## Notation

- $WHILE$  = The set of all **total** WHILE computable functions
- $WHILE^{part}$  = The set of **all** WHILE computable functions  
(including the partial ones)

## Question:

Are all **total** WHILE computable functions LOOP computable  
or  $LOOP \subset WHILE$ ?

# Partial WHILE computable functions

---

## Notation

- $WHILE$  = The set of all **total** WHILE computable functions
- $WHILE^{part}$  = The set of **all** WHILE computable functions  
(including the partial ones)

## Question:

Are all **total** WHILE computable functions LOOP computable  
or  $LOOP \subset WHILE$ ?

Later we will show that:

- one can construct a total TM computable function which cannot be computed with a LOOP program
- $WHILE$  computable = TM computable

# Overview

---

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

# GOTO Programs: Syntax

---

**Definition:** An index (line number) is a natural number  $j \geq 0$ .



# GOTO Programs: Syntax

---

**Definition:** An index (line number) is a natural number  $j \geq 0$ .

## 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**

# Differences between WHILE and GOTO

---

Different structure:

- **WHILE programs** contain **WHILE programs**  
**Recursive** definition of syntax and semantics.
- **GOTO programs** are a list of **GOTO instructions**  
**Non recursive** definition of syntax and semantics.

# GOTO Programs: Semantics

---

Let  $P$  be a GOTO program of the form:

$$P = j_1 : l_1; j_2 : l_2; \dots; j_k : l_k$$

Let  $j_{k+1}$  be an index which does not occur in  $P$  (program end).

**Definition.**  $\Delta(P)(s_1, s_2)$  holds if and only if for every  $n \geq 0$  there exist:

- states  $s'_0, \dots, s'_n$
- indices  $z_0, \dots, z_n$

such that the following hold:

(1a)  $s'_0 = s_1$

(1b)  $s'_n = s_2$

(1c)  $z_0 = j_1$

(1d)  $z_n = j_{k+1}$

and ....

(continuation on next page)

# GOTO Programs: Semantics

---

Let  $P$  be a GOTO program of the form:

$$P = j_1 : l_1; j_2 : l_2; \dots; j_k : l_k$$

Let  $j_{k+1}$  be an index which does not occur in  $P$  (program end).

**Definition (ctd.).**  $\Delta(P)(s_1, s_2)$  holds if and only if for every  $n \geq 0$  there exist:

- states  $s'_0, \dots, s'_n$
- indices  $z_0, \dots, z_n$

such that the following hold:

(2) For  $0 \leq l \leq n$ , if  $j_s : l_s$  is the line in  $P$  with  $j_s = z_l$ :

(2a) if  $l_s$  is  $x_i := x_i + 1$  then:  $s'_{i+1}(x_i) = s'_i(x_i) + 1$

$s'_{i+1}(x_j) = s'_i(x_j)$  for  $j \neq i$

$z_{i+1} = j_{s+1}$

and ....

(continuation on next page)

# GOTO Programs: Semantics

Let  $P$  be a GOTO program of the form:

$$P = j_1 : l_1; j_2 : l_2; \dots; j_k : l_k$$

Let  $j_{k+1}$  be an index which does not occur in  $P$  (program end).

**Definition (ctd.).**  $\Delta(P)(s_1, s_2)$  holds if and only if for every  $n \geq 0$  there exist:

- states  $s'_0, \dots, s'_n$
- indices  $z_0, \dots, z_n$

such that the following hold:

(2) For  $0 \leq l \leq n$ , if  $j_s : l_s$  is the line in  $P$  with  $j_s = z_l$ :

$$(2b) \text{ if } l_s \text{ is } x_i := x_i - 1 \text{ then: } s'_{i+1}(x_i) = \begin{cases} s'_i(x_i) - 1 & \text{if } s'_i(x_i) > 0 \\ 0 & \text{if } s'_i(x_i) = 0 \end{cases}$$

$$s'_{i+1}(x_j) = s'_i(x_j) \text{ for } j \neq i$$

$$z_{i+1} = j_{s+1}$$

and ....

(continuation on next page)

# GOTO Programs: Semantics

---

Let  $P$  be a GOTO program of the form:

$$P = j_1 : l_1; j_2 : l_2; \dots; j_k : l_k$$

Let  $j_{k+1}$  be an index which does not occur in  $P$  (program end).

**Definition (ctd.).**  $\Delta(P)(s_1, s_2)$  holds if and only if for every  $n \geq 0$  there exist:

- states  $s'_0, \dots, s'_n$
- indices  $z_0, \dots, z_n$

such that the following hold:

(2) For  $0 \leq l \leq n$ , if  $j_s : l_s$  is the line in  $P$  with  $j_s = z_l$ :

$$(2c) \quad \text{if } l_s \text{ is if } x_i = 0 \text{ goto } j_{\text{goto}} \text{ then:} \quad \begin{aligned} s'_{i+1} &= s'_i \\ z_{i+1} &= \begin{cases} j_{\text{goto}} & \text{if } x_i = 0 \\ j_{s+1} & \text{otherwise} \end{cases} \end{aligned}$$

# GOTO Programs: Semantics

---

## Remark

The number of line changes (iterations) is not fixed at the beginning.  
Infinite loops are possible.

# GOTO Programs: Semantics

---

## Remark

The number of line changes (iterations) is not fixed at the beginning. Infinite loops are possible.

## Notation

- $\text{GOTO}$  = The set of all **total** GOTO computable functions
- $\text{GOTO}^{\text{part}}$  = The set of **all** GOTO computable functions  
(including the partial ones)



# WHILE and GOTO

---

## Theorem.

- (1) WHILE = GOTO
- (2) WHILE<sup>part</sup> = GOTO<sup>part</sup>

# WHILE and GOTO

---

## Theorem.

(1)  $\text{WHILE} = \text{GOTO}$

(2)  $\text{WHILE}^{\text{part}} = \text{GOTO}^{\text{part}}$

Proof:

To show:

I.  $\text{WHILE} \subseteq \text{GOTO}$  and  $\text{WHILE}^{\text{part}} \subseteq \text{GOTO}^{\text{part}}$

II.  $\text{GOTO} \subseteq \text{WHILE}$  and  $\text{GOTO}^{\text{part}} \subseteq \text{WHILE}^{\text{part}}$

# WHILE and GOTO

---

## Theorem.

- (1) WHILE = GOTO
- (2) WHILE<sup>part</sup> = GOTO<sup>part</sup>

Proof:

**I. WHILE  $\subseteq$  GOTO and WHILE<sup>part</sup>  $\subseteq$  GOTO<sup>part</sup>**

It is sufficient to prove that `while  $x_i \neq 0$  do  $P$  end` can be simulated with GOTO instructions.

We can assume without loss of generality that  $P$  does not contain any `while` (we can replace the occurrences of “while” from inside out).

# WHILE and GOTO

---

Proof (ctd.)

while  $x_i \neq 0$  do  $P$  end

is replaced by:

$j_1$  : if  $x_i = 0$  goto  $j_3$ ;

$P'$ ;

$j_2$  : if  $x_n = 0$  goto  $j_1$ ;

\*\* Since  $x_n = 0$  unconditional jump \*\*

$j_3$  :  $x_n := x_n - 1$

where:

- $x_n$  is a new register, which was not used before.
- $P'$  is obtained from  $P$  by possibly renaming the indices.

# WHILE and GOTO

---

Proof (ctd.)

while  $x_i \neq 0$  do  $P$  end

is replaced by:

$j_1$  : if  $x_i = 0$  goto  $j_3$ ;

$P'$ ;

$j_2$  : if  $x_n = 0$  goto  $j_1$ ;

$j_3$  :  $x_n := x_n - 1$

\*\* Since  $x_n = 0$  unconditional jump \*\*

where:

- $x_n$  is a new register, which was not used before.
- $P'$  is obtained from  $P$  by possibly renaming the indices.

**Remark:** Totality is preserved by this transformation. Semantics is the same.

# WHILE and GOTO

---

Proof (ctd.)

Using the fact that `while  $x_i \neq 0$  do  $P$  end` can be simulated by a GOTO program we can show (by structural induction) that every WHILE program can be simulated by a GOTO program.

# Relationships between LOOP, WHILE, GOTO

---

**Theorem.**  $\text{WHILE} = \text{GOTO}$ ;  $\text{WHILE}^{\text{part}} = \text{GOTO}^{\text{part}}$

**Proof:** I.  $\text{WHILE} \subseteq \text{GOTO}$ ;  $\text{WHILE}^{\text{part}} \subseteq \text{GOTO}^{\text{part}}$  (WHILE programs expressible as GOTO programs). Proof by structural induction.

# Relationships between LOOP, WHILE, GOTO

---

**Theorem.**  $\text{WHILE} = \text{GOTO}$ ;  $\text{WHILE}^{\text{part}} = \text{GOTO}^{\text{part}}$

**Proof:** I.  $\text{WHILE} \subseteq \text{GOTO}$ ;  $\text{WHILE}^{\text{part}} \subseteq \text{GOTO}^{\text{part}}$  (WHILE programs expressible as GOTO programs). Proof by structural induction.

**Induction basis:** We show that the property is true for all atomic WHILE programs, i.e. for programs of the form  $x_j := x_j \pm 1$  (expressible as  $j : x_j := x_j \pm 1$ ).



# Relationships between LOOP, WHILE, GOTO

---

**Theorem.**  $\text{WHILE} = \text{GOTO}$ ;  $\text{WHILE}^{\text{part}} = \text{GOTO}^{\text{part}}$

**Proof:** I.  $\text{WHILE} \subseteq \text{GOTO}$ ;  $\text{WHILE}^{\text{part}} \subseteq \text{GOTO}^{\text{part}}$  (WHILE programs expressible as GOTO programs). Proof by structural induction.

**Induction basis:** We show that the property is true for all atomic WHILE programs, i.e. for programs of the form  $x_j := x_j \pm 1$  (expressible as  $j : x_j := x_j \pm 1$ ).

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

**Induction hypothesis:** We assume that the property holds for all “subprograms” of  $P$ .

**Induction step:** We show that then it also holds for  $P$ . Proof depends on form of  $P$ .

# Relationships between LOOP, WHILE, GOTO

---

**Theorem.**  $\text{WHILE} = \text{GOTO}; \text{WHILE}^{\text{part}} = \text{GOTO}^{\text{part}}$

**Proof:** I.  $\text{WHILE} \subseteq \text{GOTO}; \text{WHILE}^{\text{part}} \subseteq \text{GOTO}^{\text{part}}$  (WHILE programs expressible as GOTO programs). Proof by structural induction.

**Induction basis:** We show that the property is true for all atomic WHILE programs, i.e. for programs of the form  $x_j := x_j \pm 1$  (expressible as  $j : x_j := x_j \pm 1$ ).

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

**Induction hypothesis:** We assume that the property holds for all “subprograms” of  $P$ .

**Induction step:** We show that then it also holds for  $P$ . Proof depends on form of  $P$ .

**Case 1:**  $P = P_1; P_2$ . By the induction hypothesis, there exist GOTO programs  $P'_1, P'_2$  with  $\Delta(P_i) = \Delta(P'_i)$ . We can assume w.l.o.g. that the indices used for labelling the instructions are disjoint. Let  $P' = P'_1; P'_2$  (a GOTO program). We can show that  $\Delta(P')(s_1, s_2)$  iff  $\Delta(P)(s_1, s_2)$  as before.

# Relationships between LOOP, WHILE, GOTO

**Theorem.**  $\text{WHILE} = \text{GOTO}$ ;  $\text{WHILE}^{\text{part}} = \text{GOTO}^{\text{part}}$

**Proof:** I.  $\text{WHILE} \subseteq \text{GOTO}$ ;  $\text{WHILE}^{\text{part}} \subseteq \text{GOTO}^{\text{part}}$  (WHILE programs expressible as GOTO programs). Proof by structural induction.

**Induction basis:** We show that the property is true for all atomic WHILE programs, i.e. for programs of the form  $x_i := x_i \pm 1$  (expressible as  $j : x_i := x_i \pm 1$ ).

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

**Induction hypothesis:** We assume that the property holds for all “subprograms” of  $P$ .

**Induction step:** We show that then it also holds for  $P$ . Proof depends on form of  $P$ .

**Case 1:**  $P = P_1; P_2$ . By the induction hypothesis, there exist GOTO programs  $P'_1, P'_2$  with  $\Delta(P_i) = \Delta(P'_i)$ . We can assume w.l.o.g. that the indices used for labelling the instructions are disjoint. Let  $P' = P'_1; P'_2$  (a GOTO program). We can show that  $\Delta(P')(s_1, s_2)$  iff  $\Delta(P)(s_1, s_2)$  as before.

**Case 2:**  $P = \text{while } x_i \neq 0 \text{ do } P_1 \text{ end}$ . By the induction hypothesis, there exists a GOTO program  $P'_1$  such that  $\Delta(P_1) = \Delta(P'_1)$ . Let  $P'$  be the following GOTO program:  $j_1 : \text{if } x_i = 0 \text{ goto } j_3; P'_1; j_2 : \text{if } x_i = 0 \text{ goto } j_1; j_3 : x_i := x_i - 1$

It can be checked that  $\Delta(P')(s_1, s_2)$  iff  $\Delta(P)(s_1, s_2)$ .

# WHILE and GOTO

---

## Theorem.

- (1) WHILE = GOTO
- (2) WHILE<sup>part</sup> = GOTO<sup>part</sup>

Proof:

**II. GOTO  $\subseteq$  WHILE and GOTO<sup>part</sup>  $\subseteq$  WHILE<sup>part</sup>**

It is sufficient to prove that every GOTO program can be simulated with WHILE instructions.

# WHILE and GOTO

---

Proof (ctd.)

$j_1 : l_1; j_2 : l_2; \dots; j_k : l_k$

is replaced by the following while program:

```
xindex := j1;  
while xindex ≠ 0 do  
  if xindex = j1 then l'1 end;  
  if xindex = j2 then l'2 end;  
  ...  
  if xindex = jk then l'k end  
end
```

# WHILE and GOTO

---

Proof (ctd.)

$j_1 : l_1; j_2 : l_2; \dots; j_k : l_k$

is replaced by the following while program:

```
xindex := j1;  
while xindex ≠ 0 do  
  if xindex = j1 then l'1 end;  
  if xindex = j2 then l'2 end;  
  ...  
  if xindex = jk then l'k end  
end
```

For  $1 \leq i < k$ :

If  $l_i$  is  $x_i := x_i \pm 1$ :

$l'_i$  is  $x_i := x_i \pm 1; x_{\text{index}} := j_{i+1}$

If  $l_i$  is **if**  $x_i = 0$  **goto**  $j_{\text{goto}}$ :

$l'_i$  is **if**  $x_i = 0$  **then**  $x_{\text{index}} := j_{\text{goto}}$

**else**  $x_{\text{index}} := j_{i+1}$  **end**

In addition,  $j_{k+1} = 0$

# GOTO and WHILE are equally powerful

---

## Consequences of the proof:

### Corollary 1

The instructions defined in the context of LOOP programs:

$x_i := c$        $x_i := x_j$        $x_i := x_j + c$        $x_i := x_j + x_k$        $x_i = x_j * x_k,$   
if  $x_i = 0$  then  $P_i$  else  $P_j$       if  $x_i \leq x_j$  then  $P_i$  else  $P_j$

can also be used in GOTO programs.

# GOTO and WHILE are equally powerful

---

## Consequences of the proof:

### Corollary 2

Every WHILE computable function can be computed by a **WHILE+IF** program with **one while loop only**.



# GOTO and WHILE are equally powerful

---

## Consequences of the proof:

### Corollary 2

Every WHILE computable function can be computed by a **WHILE+IF** program with **one while loop only**.

Proof: We showed that:

- (i) every WHILE program can be simulated by a GOTO program
- (ii) every GOTO program can be simulated by a WHILE program with only one loop, containing also some if instructions (WHILE-IF program).

Let  $P$  be a WHILE program.  $P$  can be simulated by a GOTO program  $P'$ .  $P'$  can be simulated by a WHILE-IF program with one WHILE loop only.

# GOTO and WHILE are equally powerful

---

## Consequence of the proof:

Every WHILE computable function can be computed by a **WHILE+IF** program with **one while loop only**.

## Other consequences

- GOTO programming is not more powerful than WHILE programming

# GOTO and WHILE are equally powerful

---

## Consequence of the proof:

Every WHILE computable function can be computed by a **WHILE+IF** program with **one while loop only**.

## Other consequences

- GOTO programming is not more powerful than WHILE programming
- “Spaghetti-Code” (GOTO) is not more powerful than “structured code” (WHILE)

# 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

# Relationships

---

Already shown:

$$\text{LOOP} \subseteq \text{WHILE} = \text{GOTO} \subsetneq \text{WHILE}^{\text{part}} = \text{GOTO}^{\text{part}}$$

# Relationships

---

Already shown:

$$\text{LOOP} \subseteq \text{WHILE} = \text{GOTO} \subsetneq \text{WHILE}^{\text{part}} = \text{GOTO}^{\text{part}}$$

To be proved:

- $\text{LOOP} \neq \text{WHILE}$
- $\text{WHILE} = \text{TM}$  and  $\text{WHILE}^{\text{part}} = \text{TM}^{\text{part}}$