Part 2: Register machines (2)

17.11.2021

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: 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, λ-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
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 $x_1, x_2, x_3, \ldots, x_n$; each register contains a natural number.

• A LOOP-, WHILE- or GOTO-program.
Definition (State of a register machine)
The state $s$ of a register machine is a map: $s : \{ x_i \mid i \in \mathbb{N} \} \to \mathbb{N}$
which associates with every register a natural number as value.

Definition (Initial state; Input)
Let $m_1, \ldots, m_k \in \mathbb{N}$ 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 $m_1, \ldots, m_k \in \mathbb{N}$ halts in a state $s_{\text{term}}$
then: $s_{\text{term}}(x_{k+1})$ is the output of the machine.
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 $s_2$. 
Definition (Computed function)
A register machine $P$ computes a function

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

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

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

- $P$ terminates if and only if $f(m_1, \ldots, m_k)$ is defined
- If $P$ terminates, then the output of $P$ is $f(m_1, \ldots, m_k)$
- Additional condition (next page)
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 $x_1, \ldots, x_k$ 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 $x_1, \ldots, x_k$ 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.
Example:

The program:

\[
P := \text{loop } x_2 \text{ do } x_2 := x_2 - 1 \text{ end}; \quad x_2 := x_2 + 1;
\]

\[
\text{loop } x_1 \text{ do } x_1 := x_1 - 1 \text{ end}
\]

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 $f$
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

\[
\begin{align*}
\text{LOOP} &= \text{Set of all LOOP computable functions} \\
\text{WHILE} &= \text{Set of all WHILE computable functions} \\
\text{GOTO} &= \text{Set of all GOTO computable functions} \\
\text{TM} &= \text{Set of all TM computable functions}
\end{align*}
\]
Register Machines: Computable function

\[
\begin{align*}
\text{LOOP} & = \text{Set of all LOOP computable functions} \\
\text{WHILE} & = \text{Set of all WHILE computable functions} \\
\text{GOTO} & = \text{Set of all GOTO computable functions} \\
\text{TM} & = \text{Set of all TM computable functions}
\end{align*}
\]

Still not precise:

WHILE/GOTO/TM computable functions can also be partial
Register Machines: Computable function

\[
\begin{align*}
\text{LOOP} & \quad = \quad \text{Set of all total LOOP computable functions} \\
\text{WHILE} & \quad = \quad \text{Set of all total WHILE computable functions} \\
\text{GOTO} & \quad = \quad \text{Set of all total GOTO computable functions} \\
\text{TM} & \quad = \quad \text{Set of all total TM computable functions} \\
\text{WHILE}^{\text{part}} & \quad = \quad \text{Set of all total or partial WHILE computable functions} \\
\text{GOTO}^{\text{part}} & \quad = \quad \text{Set of all total or partial GOTO computable functions} \\
\text{TM}^{\text{part}} & \quad = \quad \text{Set of all total or partial TM computable functions}
\end{align*}
\]
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).
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 $j \neq i$
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 $j \neq i$
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:
  - $\Delta(P_1)(s_1, s')$
  - $\Delta(P_2)(s', s_2)$
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 $s'_0, s'_1, \ldots, s'_n$ with:
  - $s_1(x_i) = n$
  - $s_1 = s'_0$
  - $s_2 = s'_n$
  - $\Delta(P)(s'_k, s'_{k+1})$ for $0 \leq k < n$
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 $s'_0, s'_1, \ldots, s'_n$ 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.
Program end: If there is no next program line, then the program execution terminates.

We say that a LOOP program terminates on an input $n_1, \ldots, n_k$ 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.
**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.
**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.
LOOP Programs

Additional instructions

- \( x_i := 0 \)
  
  \[
  \text{loop } x_i \text{ do } x_i := x_i - 1 \text{ end}
  \]

- \( x_i := c \) for \( c \in \mathbb{N} \)
  
  \[
  x_i := 0; \\
  x_i := x_i + 1; \\
  \ldots \ \\
  \left\{ \begin{array}{l}
  x_i := x_i + 1 \\
  \end{array} \right\} \quad \text{c times}
  \]

- \( x_i := x_j \)
  
  \[
  x_i := 0; \\
  \text{loop } x_j \text{ do } x_i := x_i + 1 \text{ end}
  \]
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
Additional instructions

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

- \( x_i := e_1 + e_2 \) \((e_1, e_2\) arithmetical expressions\)
  \[
  \begin{align*}
  &x_i := e_1; \\
  &x_n := e_2; \\
  &\text{loop } x_n \text{ do } x_i := x_i + 1 \text{ end; } x_n := 0
  \end{align*}
  \]

- \( x_i := e_1 - e_2 \) \((e_1, e_2\) arithmetical expressions\)
  \[
  \begin{align*}
  &x_i := e_1; \\
  &x_n := e_2; \\
  &\text{loop } x_n \text{ do } x_i := x_i - 1 \text{ end; } x_n := 0
  \end{align*}
  \]

- \( x_i := e_1 \ast e_2 \) \((e_1, e_2\) arithmetical expressions\)
  \[
  \begin{align*}
  &x_i := 0; \\
  &x_n := e_1; \\
  &\text{loop } x_n \text{ do } x_i := x_i + e_2 \text{ end; } x_n := 0
  \end{align*}
  \]
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 which contains the atomic programs and is closed under sequential composition and “while constructions”.

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:
  - $\Delta(P_1)(s_1, s')$
  - $\Delta(P_2)(s', s_2)$
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 $s'_0, s'_1, \ldots, s'_n$ 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 $0 \leq k < n$
  - $s'_n(x_i) = 0$
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 $s'_0, s'_1, \ldots, s'_n$ 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.
**Theorem.** LOOP ⫋ 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 \( P_{\text{while}} \):

\[
\begin{align*}
\text{while } x_i \neq 0 \text{ do} & \quad \text{** simulate } x_n := x_i \text{ **} \\
& \begin{align*}
\phantom{\text{while } x_i \neq 0 \text{ do}}
& x_n := x_n + 1; x_{n+1} := x_{n+1} + 1; x_i := x_i - 1 \\
& \text{end;}
\end{align*}
\end{align*}
\]

\[
\begin{align*}
\text{while } x_{n+1} \neq 0 \text{ do} & \quad \text{** restore } x_i \text{ **} \\
& \begin{align*}
\phantom{\text{while } x_{n+1} \neq 0 \text{ do}}
& x_i := x_i + 1; x_{n+1} := x_{n+1} - 1 \\
& \text{end;}
\end{align*}
\end{align*}
\]

\[
\begin{align*}
\text{while } x_n \neq 0 \text{ do} & \quad \text{** simulate the loop instruction **} \\
& \begin{align*}
\phantom{\text{while } x_n \neq 0 \text{ do}}
& P; x_n := x_n - 1 \\
& \text{end}
\end{align*}
\end{align*}
\]

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

**Theorem.** LOOP ⊆ 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).

$\Delta(P')(s_1, s_2)$ iff there exists $s$ with $\Delta(P'_1)(s_1, s)$ and $\Delta(P'_2)(s, s_2)$

iff there exists $s$ with $\Delta(P_1)(s_1, s)$ and $\Delta(P_2)(s, s_2)$ iff $\Delta(P)(s_1, s_2)$
WHILE and LOOP

Theorem. LOOP ⊆ 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).

$$\Delta(P')(s_1, s_2) \iff \text{there exists } s \text{ with } \Delta(P_1')(s_1, s) \text{ and } \Delta(P_2')(s, s_2)$$

$$\Delta(P')(s_1, s_2) \iff \text{there exists } s \text{ with } \Delta(P_1)(s_1, s) \text{ and } \Delta(P_2)(s, s_2) \iff \Delta(P)(s_1, s_2)$$

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:

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

$$\text{while } x_{n+1} \neq 0 \text{ do } x_i := x_i + 1; x_{n+1} := x_{n+1} - 1 \text{ end;} \text{ while } x_n \neq 0 \text{ do } P_1'; x_n := x_n - 1 \text{ end}$$

$$\Delta(P')(s_1, s_2) = \Delta(P)(s_1, s_2) \text{ (show that } P \text{ and } P' \text{ change values of registers in the same way).}$$
Consequences of the proof:

**Corollary**

The instructions defined in the context of LOOP programs:

\[
\begin{align*}
    x_i & := c \\
    x_i & := x_j \\
    x_i & := x_j + c \\
    x_i & := x_j + x_k \\
    x_i & := x_j \times x_k, \\
\end{align*}
\]

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

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

Question:
Are all total WHILE computable functions LOOP computable or LOOP $\subseteq$ WHILE?
Partial WHILE computable functions

Notation

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

Question:
Are all total WHILE computable functions LOOP computable or $\text{LOOP} \subset \text{WHILE}$?

Later we will show that:

- one can construct a total TM computable function which cannot be computed with a LOOP program
- $\text{WHILE computable} = \text{TM computable}$
Overview

- Register machines (Random access machines)
- LOOP programs
- WHILE programs
- GOTO programs
- Relationships between LOOP, WHILE, GOTO
- Relationships between register machines and Turing machines
**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 $I_1, \ldots, I_k$ are GOTO instructions and $j_1, \ldots, j_k$ are indices then
  $j_1 : I_1; \ldots; j_k : I_k$ 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:

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

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 there exists $n \geq 0$ and there exist:

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

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

$$P = j_1 : I_1; \ j_2 : I_2; \ \ldots; \ j_k : I_k$$

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 there exists $n \geq 0$ and there exist:

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

such that the following hold:

1. For $0 \leq p \leq n$, if $j_s : I_s$ is the line in $P$ with $j_s = z_p$ (and the current state is $s'_p$):
   
   (2a) if $I_s$ is $x_i := x_i + 1$ then: $s'_{p+1}(x_i) = s'_p(x_i) + 1$
   
   $s'_{p+1}(x_j) = s'_p(x_j)$ for $j \neq i$

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

and ....  (continuation on next page)
Let $P$ be a GOTO program of the form:

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

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

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

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

such that the following hold:

(2) For $0 \leq p \leq n$, if $j_s : l_s$ is the line in $P$ with $j_s = z_p$ (and the current state is $s'_p$):

(2b) if $l_s$ is $x_i := x_i - 1$ then:

$$s'_{p+1}(x_i) = \begin{cases} 
  s'_p(x_i) - 1 & \text{if } s'_p(x_i) > 0 \\
  0 & \text{if } s'_p(x_i) = 0
\end{cases}$$

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

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

and .... (continuation on next page)
Let $P$ be a GOTO program of the form:

$$ P = j_1 : I_1; \ j_2 : I_2; \ \ldots; \ j_k : I_k $$

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 there exists \( n \geq 0 \) and there exist:

- states \( s'_0, \ldots, s'_n \)
- indices \( z_0, \ldots, z_n \)

such that the following hold:

1. For \( 0 \leq p \leq n \), if \( j_s : I_s \) is the line in $P$ with \( j_s = z_p \) (and the current state is \( s'_p \)):
   - if \( I_s \) is if \( x_i = 0 \) goto \( j_{\text{goto}} \) then:
     $$ s'_{p+1} = s'_p $$
     $$ z_{p+1} = \begin{cases} 
     j_{\text{goto}} & \text{if } x_i = 0 \\
     j_{s+1} & \text{otherwise}
     \end{cases} $$
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

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

(1) WHILE $\equiv$ GOTO

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

Theorem.
(1) WHILE = GOTO
(2) WHILE\text{part} = GOTO\text{part}

Proof (next time)

To show:

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

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