Advanced Topics in Theoretical Computer Science

Part 2: Register machines (3)

21.11.2013

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: sofronie@uni-koblenz.de

Wintersemester 2013-2014

Contents

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

Until now

- Register machines (definition; state; input/output; semantics)

  Computed function

  Computable functions (LOOP, WHILE, GOTO, TM)

- LOOP Programs (syntax, semantics)

  Every LOOP program terminates for every input

  All LOOP computable functions are total

  Additional instructions

- WHILE Programs (syntax, semantics)

  WHILE programs do not always terminate

  WHILE computable functions can be undefined for some inputs

- GOTO Programs (syntax, semantics)

  GOTO programs do not always terminate

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)

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: $\Delta(x_i := x_i \pm 1)(s_1, s_2)$ iff:

- $s_2(x_i) = s_1(x_i) \pm 1$
- $s_2(x_j) = s_1(x_j)$ for all $j \neq i$

(2) Sequential composition: $\Delta(P_1; P_2)(s_1, s_2)$ iff there exists s' s.t.:

- $\Delta(P_1)(s_1, s')$
- $\Delta(P_2)(s', s_2)$

(3) Loop programs: $\Delta(\text{loop } x_i \text{ do } P \text{ end})(s_1, s_2)$ iff 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 $0 \leq k < n$

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)

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: $\Delta(x_i := x_i \pm 1)(s_1, s_2)$ iff:

- $s_2(x_i) = s_1(x_i) \pm 1$
- $s_2(x_j) = s_1(x_j)$ for all $j \neq i$

(2) Sequential composition: $\Delta(P_1; P_2)(s_1, s_2)$ iff there exists s' s.t.:

- $\Delta(P_1)(s_1, s')$
- $\Delta(P_2)(s', s_2)$

(3) While programs: $\Delta(\text{while } x_i \neq 0 \text{ do } P \text{ end})(s_1, s_2)$ iff 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 $0 \leq k < n$
- $s'_n(x_i) = 0$

GOTO Programs: Syntax

Indices (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 $I_1, \dots, I_k$ are GOTO instructions and $j_1, \dots, j_k$ are indices then
  - $j_1 : I_1; \dots; j_k : I_k$ is a GOTO program

GOTO Programs: Semantics

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 $\Delta(P)(s_1, s_2)$ holds iff for every $n \geq 0$ there exist states $s'_0, \ldots, s'_n$ and indices $z_0, \ldots, z_n$ s.t.:

- $s'_0 = s_1, s'_n = s_2;\ z_0 = j_1, z_n = j_{k+1}$.
- For $0 \leq l \leq n$, if $j_s : I_s$ is the line in P with $j_s = z_l$:

  if $I_s = x_i := x_i \pm 1$ then:
  $$s'_{i+1}(x_i) = s'_i(x_i) \pm 1$$
  $$s'_{i+1}(x_j) = s'_i(x_j) \text{ for } j \neq i$$
  $$z_{i+1} = j_{s+1}$$

  if $I_s =$ if $x_i = 0$ goto j_{goto} then:
  $$s'_{i+1} = s'_i$$
  $$z_{i+1} = \begin{cases} j_{goto} & \text{if } x_i = 0 \\ j_{s+1} & \text{otherwise} \end{cases}$$

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.

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 computableif there exists a Turing machine which computes f

Computable functions

$$
\begin{aligned}
\mathrm{LOOP} &= \text{Set of all LOOP computable functions}\\
\mathrm{WHILE} &= \text{Set of all total WHILE computable functions}\\
\mathrm{WHILE}^{\mathrm{part}} &= \text{Set of all WHILE computable functions}\\
& \text{(including the partial ones)}\\
\mathrm{GOTO} &= \text{Set of all total GOTO computable functions}\\
\mathrm{GOTO}^{\mathrm{part}} &= \text{Set of all GOTO computable functions}\\
& \text{(including the partial ones)}\\
\mathrm{TM} &= \text{Set of all total TM computable functions}\\
\mathrm{TM}^{\mathrm{part}} &= \text{Set of all TM computable functions}\\
& \text{(including the partial ones)}
\end{aligned}
$$

Relationships between LOOP, WHILE, GOTO

Theorem. LOOP $\subseteq$ WHILE (every LOOP computable function is WHILE computable)

WHILE and GOTO

Theorem.

(1) WHILE $=$ GOTO

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

WHILE and GOTO

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

Proof:

To show:

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

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

WHILE and GOTO

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

Proof:

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

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:

$$
\begin{array}{ll}
j_1: & \text{if } x_i = 0 \text{ goto } j_3; \\
P'; & \\
j_2: & \text{if } x_n = 0 \text{ goto } j_1; \\
j_3: & x_n := x_n - 1
\end{array}
$$

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

where:

- x_n is a new register, which was not used before.
- P' is obtained from P by assigning to all instructions without an index an arbitrary new index.

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 assigning to all instructions without an index an arbitrary new index.

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

Proof: I. WHILE $\subseteq$ 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: $P =$ **while** $x_i \neq 0$ **do** P_1 **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 : if $x_i = 0$ goto j_3; P'; j_2 : if $x_n = 0$ goto j_1; j_3 : $x_n := x_n - 1$
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) $\text{WHILE}^{\text{part}} = \text{GOTO}^{\text{part}}$

Proof:

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

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

WHILE and GOTO

Proof (ctd.)

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

is replaced by the following while program:

$x_{\text{index}} := j_1;$
while $x_{\text{index}} \neq 0$ do
    if $x_{\text{index}} = j_1$ then I_1' end;
    if $x_{\text{index}} = j_2$ then I_2' end;
    $\ldots$
    if $x_{\text{index}} = j_k$ then I_k' end;
end

WHILE and GOTO

Proof (ctd.)

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

is replaced by the following while program:

$x_{\text{index}} := j_1;$
while $x_{\text{index}} \neq 0$ do
    if $x_{\text{index}} = j_1$ then I'_1 end;
    if $x_{\text{index}} = j_2$ then I'_2 end;
    $\ldots$
    if $x_{\text{index}} = j_k$ then I'_k end;
end

For $1 \leq i < k$:

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

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

If I_i is if $x_i = 0$ goto j_{goto}:

I'_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:

$x_i := c \qquad x_i := x_j \qquad x_i := x_j * x_k \qquad x_i = x_j * x_k,$

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) ist 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:

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

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 $\text{WHILE}^{\text{part}} = \text{TM}^{\text{part}}$

GOTO $\subseteq$ TM

Theorem GOTO $\subseteq$ TM and GOTO$^{\text{part}}$ $\subseteq$ TM$^{\text{part}}$

GOTO $\subseteq$ TM

Theorem. GOTO $\subseteq$ TM and GOTO$^{\text{part}}$ $\subseteq$ TM$^{\text{part}}$

Proof (idea)

It is sufficient to prove that for every GOTO program

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

we can construct an equivalent Turing machine.

GOTO $\subseteq$ TM

Proof (continued)

Let r be the number of registers used in P.

We construct a Turing machine M with r half tapes over the alphabet $\Sigma = \{\#, |\}$.

- Tape i contains as many $|$'s as the value of x_i is.
- There is a state s_n of M for every instruction $j_n : I_n$.
- When M is in state s_n, it does what corresponds to instruction I_n:
  - Increment or decrement the register
  - Evaluate jump condition
  - Change its state to the corresponding next state.

GOTO $\subseteq$ TM

Proof (continued)

Let r be the number of registers used in P.

We construct a Turing machine M with r half tapes over the alphabet $\Sigma = \{\#, |\}$.

- Tape i contains as many $|$'s as the value of x_i is.
- There is a state s_n of M for every instruction $j_n : I_n$.
- When M is in state s_n, it does what corresponds to instruction I_n:
  - Increment or decrement the register
  - Evaluate jump condition
  - Change its state to the corresponding next state.

It is clear that we can construct a TM which does everything above.

GOTO $\subseteq$ TM

Proof (continued)

- Tape i contains as many $|$'s as the value of x_i is.
- There is a state s_n of M for every program $P_n = j_n : I_n$.
- When M is in state s_n, it does what corresponds to instruction I_n:
  - Increment or decrement the register
  - Evaluate jump condition
  - Change its state to the corresponding next state.

I_n	M_n
$x_i := x_i + 1$	$> \|^{(i)} R^{(i)}$
$x_i := x_i - 1$	$> L^{(i)} \stackrel{\#^{(i)}}{\rightarrow} R^{(i)}$ $\downarrow^{\|(i)}$ $\#^{(i)}$

GOTO $\subseteq$ TM

Proof (continued)

- Tape i contains as many $|$'s as the value of x_i is.
- There is a state s_n of M for every program $P_n = j_n : I_n$.
- When M is in state s_n, it does what corresponds to instruction I_n:
  - Increment or decrement the register
  - Evaluate jump condition
  - Change its state to the corresponding next state.

I_n	M_n
$x_i := x_i + 1$	$> \vert^{(i)} R^{(i)}$
$x_i := x_i - 1$	$> L^{(i)} \stackrel{\#^{(i)}}{\rightarrow} R^{(i)}$ $\downarrow^{\vert(i)}$ $\#^{(i)}$

P_n	M_n
$P_{n_1} ; P_{n_2}$	$> M_{n_1} M_{n_2}$
if $x_i = 0$ goto j	$> L^{(i)} \stackrel{\#^{(i)}}{\rightarrow} R^{(i)} \rightarrow M_j$ $\downarrow^{\vert(i)}$ $R^{(i)} \rightarrow M_{n+1}$

GOTO $\subseteq$ TM

Proof (continued)

In "Theoretische Informatik I" it was proved:

For every *TM* with several tapes there exists an equivalent standard TM with only one tape.

GOTO $\subseteq$ TM

Proof (continued)

In “Theoretische Informatik I” it was proved:

For every *TM* with several tapes there exists an equovalent Standard TM with only one tape.

Therefore there exists a Standard TM which simulates program *P*

GOTO $\subseteq$ TM

Proof (continued)

In "Theoretische Informatik I" it was proved:

For every *TM* with several tapes there exists an equivalent standard TM with only one tape.

Therefore there exists a standard TM which simulates program P

Remark: We will prove later that
TM $\subseteq$ GOTO and therefore TM $=$ GOTO $=$ WHILE.

LOOP $\neq$ TM

In what follows we consider only LOOP programs which have only one input.

LOOP $\neq$ TM

In what follows we consider only LOOP programs which have only one input.

If there exists a total TM-computable function $f : \mathbb{N} \to \mathbb{N}$ which is not LOOP computable then we showed that LOOP $\neq$ TM

LOOP $\neq$ TM

In what follows we consider only LOOP programs which have only one input.

If there exists a total TM-computable function $f : \mathbb{N} \to \mathbb{N}$ which is not LOOP computable then we showed that LOOP $\neq$ TM

Idea of the proof:

For every unary LOOP-computable function $f : \mathbb{N} \to \mathbb{N}$ there exists a LOOP program P_f which computes it.

We show that:

- The set of all unary LOOP programs is recursively enumerable
- There exists a Turing machine M_{LOOP} such that if $P_1, P_2, P_3, \dots$ is an enumeration of all (unary) LOOP programs then if P_i computes from input m output o then M_{LOOP} computes from input (i, m) the output o.
- We construct a TM-computable function which is not LOOP computable using a "diagonalisation" argument.

LOOP $\neq$ TM

Lemma. The set of all LOOP programs is recursively enumerable.

LOOP $\neq$ TM

Lemma. The set of all LOOP programs is recursively enumerable.

Proof (Idea) Regard any LOOP program as a word over the alphabet:

$$\Sigma_{LOOP} = \{;\,,\ x,\ :=,\ +,\ -,\ 1,\ \text{loop}, \text{do}, \text{end}\}$$

x_i is encoded as x^i.

We can easily construct a grammar which generates all LOOP programs.

Proposition (TI 1): The recursively enumerable languages are exactly the languages generated by arbitrary grammars (i.e. languages of type 0).

Remark: The same holds also for WHILE programs, GOTO programs and Turing machines

LOOP $\neq$ TM

Lemma.
There exists a Turing machine M_{LOOP} which simulates all LOOP programs

More precisely:
Let $P_1, P_2, P_3, \ldots$ be an enumeration of all LOOP programs.
If P_i computes from input m output o then M_{LOOP} computes from input (i, m) the output o.

LOOP $\neq$ TM

Lemma.
There exists a Turing machine M_{LOOP} which simulates all LOOP programs

More precisely:
Let $P_1, P_2, P_3, \ldots$ be an enumeration of all LOOP programs.
If P_i computes from input m output o then M_{LOOP} computes from input (i, m) the output o.

Proof: similar to the proof that there exists an universal TM, which simulates all Turing machines.

LOOP $\neq$ TM

Lemma.
There exists a Turing machine M_{LOOP} which simulates all LOOP programs.

More precisely:
Let $P_1, P_2, P_3, \ldots$ be an enumeration of all LOOP programs.
If P_i computes from input m output o then M_{LOOP} computes from input (i, m) the output o.

Proof: similar to the proof that there exists an universal TM, which simulates all Turing machines.

Remark: The same holds also for WHILE programs, GOTO programs and Turing machines

LOOP $\neq$ TM

Theorem: LOOP $\neq$ TM

Proof: Let $\Psi : \mathbb{N} \rightarrow \mathbb{N}$ be defined by:

$\Psi(i) = P_i(i) + 1$ — Output of the i-th LOOP program P_i on input i to which 1 is added.

Ψ is clearly total. We will show that the following hold:

Claim 1: $\Psi \in$ TM

Claim 2: $\Psi \notin$ LOOP

LOOP $\neq$ TM

Claim 1: $\Psi \in$ TM

Proof: We have shown that:

- the set of all LOOP programs is r.e., i.e. there is a Turing machine M_0 which enumerates $P_1, \dots, P_n, \dots$ (as Gödel numbers)
- there exists a Turing machine M_{LOOP} which simulates all LOOP programs

In order to construct a Turing machine which computes Ψ we proceed as follows:

- We use M_0 to compute from i the LOOP program P_i
- We use M_{LOOP} to compute $P_i(i)$
- We add 1 to the result.

LOOP $\neq$ TM

Claim 2: $\Psi \notin$ LOOP

Proof: We assume, in order to derive a contradiction, that $\Psi \in LOOP$, i.e. there exists a LOOP program P_{i_0} which computes Ψ.

Then:

- The output of P_{i_0} on input i_0 is $P_{i_0}(i_0)$.
- $\Psi(i_0) = P_{i_0}(i_0) + 1 \neq P_{i_0}(i_0)$

Contradiction!

LOOP $\neq$ TM

Claim 2: $\Psi \notin$ LOOP

Proof: We assume, in order to derive a contradiction, that $\Psi \in LOOP$, i.e. there exists a LOOP program P_{i_0} which computes Ψ.

Then:

- The output of P_{i_0} on input i_0 is $P_{i_0}(i_0)$.
- $\Psi(i_0) = P_{i_0}(i_0) + 1 \neq P_{i_0}(i_0)$

Contradiction!

Remark: This does not hold for WHILE programs, GOTO programs and Turing machines.

LOOP $\neq$ TM

Claim 2: $\Psi \notin$ LOOP

Proof: We assume, in order to derive a contradiction, that $\Psi \in LOOP$, i.e. there exists a LOOP program P_{i_0} which computes Ψ.

Then:

- The output of P_{i_0} on input i_0 is $P_{i_0}(i_0)$.
- $\Psi(i_0) = P_{i_0}(i_0) + 1 \neq P_{i_0}(i_0)$

Contradiction!

Remark: This does not hold for WHILE programs, GOTO programs and Turing machines.

Why?

LOOP $\neq$ TM

Claim 2: $\Psi \not\in$ LOOP

Proof: We assume, in order to derive a contradiction, that $\Psi \in LOOP$, i.e. there exists a LOOP program P_{i_0} which computes Ψ.

Then:

- The output of P_{i_0} on input i_0 is $P_{i_0}(i_0)$.
- $\Psi(i_0) = P_{i_0}(i_0) + 1 \neq P_{i_0}(i_0)$

Contradiction!

Remark: This does not hold for WHILE programs, GOTO programs and Turing machines.

The proof relies on the fact that Ψ is total (otherwise $P_{i_0}(i_0) + 1$ could be undefined).

Summary

We showed that:

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

Summary

We showed that:

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

Still to show:

- TM $\subseteq$ WHILE
- TM$^{\text{part}}$ $\subseteq$ WHILE$^{\text{part}}$

Summary

We showed that:

- $\mathrm{LOOP} \subsetneq \mathrm{WHILE} = \mathrm{GOTO} \subseteq \mathrm{TM}$
- $\mathrm{WHILE} = \mathrm{GOTO} \subsetneq \mathrm{WHILE}^{\mathrm{part}} = \mathrm{GOTO}^{\mathrm{part}} \subseteq \mathrm{TM}^{\mathrm{part}}$
- $\mathrm{LOOP} \neq \mathrm{TM}$

Still to show:

- $\mathrm{TM} \subseteq \mathrm{WHILE}$
- $\mathrm{TM}^{\mathrm{part}} \subseteq \mathrm{WHILE}^{\mathrm{part}}$

For proving this, another model of computation will be used:
recursive functions