Advanced Topics in Theoretical Computer Science

Part 2: Register machines (3)

23.11.2022

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

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

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 computable if there exists a Turing machine which computes f

Computable functions

$$
\begin{array}{rcl}
\text{LOOP} & = & \text{Set of all LOOP computable functions} \\[1em]
\text{WHILE} & = & \text{Set of all total WHILE computable functions} \\[1em]
\text{WHILE}^{\text{part}} & = & \text{Set of all WHILE computable functions} \\
& & \text{(including the partial ones)} \\[1em]
\text{GOTO} & = & \text{Set of all total GOTO computable functions} \\[1em]
\text{GOTO}^{\text{part}} & = & \text{Set of all GOTO computable functions} \\
& & \text{(including the partial ones)} \\[1em]
\text{TM} & = & \text{Set of all total TM computable functions} \\[1em]
\text{TM}^{\text{part}} & = & \text{Set of all TM computable functions} \\
& & \text{(including the partial ones)}
\end{array}
$$

Relationships between LOOP, WHILE, GOTO

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

Corollary

The instructions defined in the context of LOOP programs:

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

can also be used in WHILE programs.

WHILE and GOTO

> **Theorem.**
> (1) WHILE $=$ GOTO
> (2) WHILE$^{\mathsf{part}}$ $=$ GOTO$^{\mathsf{part}}$

WHILE and GOTO

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

Proof:

To show:

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

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

WHILE and GOTO

> **Theorem.**
> (1) WHILE $=$ GOTO
> (2) WHILE$^{\mathsf{part}}$ $=$ GOTO$^{\mathsf{part}}$

Proof:

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

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

We assume that P can be simulated with a GOTO program $\overline{P}$ (i.e. that we construct equivalent GOTO programs "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;
   $\overline{P}'$;
   $j_2 :$ 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.

- $\overline{P}'$ is obtained from $\overline{P}$ by possibly renaming the indices.

WHILE and GOTO

Proof (ctd.)

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

is replaced by:

$\qquad$
$$
\begin{aligned}
&j_1: && \text{if } x_i = 0 \text{ goto } j_3; \\
&\overline{P}'; \\
&j_2: && \text{if } x_n = 0 \text{ goto } j_1; && \text{** Since } x_n = 0 \text{ unconditional jump **} \\
&j_3: && x_n := x_n - 1
\end{aligned}
$$

where:

- x_n is a new register, which was not used before.

- $\overline{P}'$ is obtained from $\overline{P}$ 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.** WHILE $=$ GOTO; WHILE$^{\text{part}} =$ GOTO$^{\text{part}}$

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

Relationships between LOOP, WHILE, GOTO

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

Proof: I. WHILE $\subseteq$ GOTO; WHILE$^{\text{part}}$ $\subseteq$ 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 $j : x_i := x_i \pm 1$).

Relationships between LOOP, WHILE, GOTO

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

Proof: I. WHILE $\subseteq$ GOTO; WHILE$^{\text{part}}$ $\subseteq$ 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.

Relationships between LOOP, WHILE, GOTO

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

Proof: I. WHILE $\subseteq$ GOTO; WHILE$^{\text{part}}$ $\subseteq$ 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.

Relationships between LOOP, WHILE, GOTO

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

Proof: I. WHILE $\subseteq$ GOTO; WHILE$^{\text{part}}$ $\subseteq$ 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 $\overline{P}_1$ such that $\Delta(P_1) = \Delta(\overline{P}_1)$. Let P' be the following GOTO program: $j_1 :$ if $x_i = 0$ goto j_3; $\overline{P}_1'$; $j_2 :$ if $x_n = 0$ goto j_1; $j_3 :$ $x_n := x_n - 1$

(where $\overline{P}_1'$ is obtained from $\overline{P}_1$ by possibly renaming some indices).
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$^{\text{part}}$ = GOTO$^{\text{part}}$

Proof:

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

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

WHILE and GOTO

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

Proof:

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

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

We make the following assumptions (w.l.o.g):

1) All indices occurring in the program are ≥ 1

2) All indices used for goto instructions occur as labels of instructions

Proof (ctd.)

$j_1 : l_1 ; j_2 : l_2 ; \ldots ; j_k : l_k$ (w.l.o.g. we can assume that $j_i \geq 1$ for all $1 \leq i \leq k$)

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 l_1' end;
    if $x_{\text{index}} = j_2$ then l_2' end;
    $\ldots$
    if $x_{\text{index}} = j_k$ then l_k' end
end

Proof (ctd.)

$$j_1 : l_1 ; j_2 : l_2 ; \ldots ; j_k : l_k \qquad \text{(w.l.o.g. we can assume that } j_i \geq 1 \text{ for all } 1 \leq i \leq k\text{)}$$

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 l_1' end;
    if $x_{\text{index}} = j_2$ then l_2' end;
    $\ldots$
    if $x_{\text{index}} = j_k$ then l_k' end
end

For $1 \leq n < k$:
If l_n is $x_i := x_i \pm 1$:

$$l_n' \text{ is } x_i := x_i \pm 1; x_{\text{index}} := j_{n+1}$$

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

$$l_n' \text{ is } \quad \text{if } x_i = 0 \text{ then } x_{\text{index}} := j_{\text{goto}}$$
$$\text{else } x_{\text{index}} := j_{n+1} \text{ end}$$

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

Consequences of the proof:

Corollary 1

The instructions defined in the context of LOOP programs:

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

if $x_i = 0$ then P_i else P_j $\qquad$ 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:

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

To be proved:

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

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
$x_i := x_i + 1$	$> \mid^{(i)} R^{(i)}$
$x_i := x_i - 1$	$> L^{(i)} \overset{\#^{(i)}}{\to} R^{(i)}$ $\quad \downarrow \mid^{(i)}$ $\quad \#^{(i)}$

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
$x_i := x_i + 1$	$> \mid^{(i)} R^{(i)}$
$x_i := x_i - 1$	$> L^{(i)} \overset{\#^{(i)}}{\to} R^{(i)}$ $\downarrow \mid^{(i)}$ $\#^{(i)}$

P_n	M_n
$P_{n_1} ; P_{n_2}$	$> M_{n_1} M_{n_2}$
j_n : if $x_i = 0$ goto j_k	$> L^{(i)} \overset{\#^{(i)}}{\to} R^{(i)} \to M_k$ $\downarrow \mid^{(i)}$ $R^{(i)} \to M_{n+1}$

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 equivalent standard TM with only one tape.

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

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 $P_1, P_2, P_3, \ldots$ 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 a 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:

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

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 $P_1, \ldots, P_n, \ldots$ (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 $\Psi \in \textit{LOOP}$, 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: $\Psi \notin \mathsf{LOOP}$

Proof: We assume, in order to derive a contradiction, that $\Psi \in \mathit{LOOP}$, 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:

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

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

Still to show:

- TM $\subseteq$ WHILE

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

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