

Advanced Topics in Theoretical Computer Science

Part 2: Register machines (3)

24.11.2021

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

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 $x_1, x_2, x_3 \dots, x_n$; 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

LOOP	=	Set of all LOOP computable functions
WHILE	=	Set of all total WHILE computable functions
WHILE ^{part}	=	Set of all WHILE computable functions (including the partial ones)
GOTO	=	Set of all total GOTO computable functions
GOTO ^{part}	=	Set of all GOTO computable functions (including the partial ones)
TM	=	Set of all total TM computable functions
TM ^{part}	=	Set of all TM computable functions (including the partial ones)

Relationships between LOOP, WHILE, GOTO

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

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.

WHILE and GOTO

Theorem.

- (1) WHILE = GOTO
- (2) WHILE^{part} = GOTO^{part}

WHILE and GOTO

Theorem.

(1) WHILE = GOTO

(2) WHILE^{part} = GOTO^{part}

Proof:

To show:

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

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

WHILE and GOTO

Theorem.

- (1) WHILE = GOTO
- (2) WHILE^{part} = GOTO^{part}

Proof:

I. WHILE \subseteq GOTO and WHILE^{part} \subseteq GOTO^{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 \bar{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 ;

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

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. WHILE = GOTO; WHILE^{part} = GOTO^{part}

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

Proof:

II. GOTO \subseteq WHILE and GOTO^{part} \subseteq WHILE^{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^{part} = GOTO^{part}

Proof:

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

WHILE and GOTO

Proof (ctd.)

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

```
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.)

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

```
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 $1 \leq n < k$:

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

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

else $x_{\text{index}} := j_{n+1}$ **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 $\text{WHILE}^{\text{part}} = \text{TM}^{\text{part}}$

GOTO \subseteq TM

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

GOTO \subseteq TM

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

Proof (idea)

It is sufficient to prove that for every GOTO program

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

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

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

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

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 equivalent 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 $f : \mathbb{N} \rightarrow \mathbb{N}$ 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 $f : \mathbb{N} \rightarrow \mathbb{N}$ which is not LOOP computable then we showed that LOOP \neq TM.

Idea of the proof:

For every unary LOOP-computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ 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, \dots 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, \dots 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, \dots 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: $\Psi \in \text{TM}$

Claim 2: $\Psi \notin \text{LOOP}$

LOOP \neq TM

Claim 1: $\Psi \in \text{TM}$

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: $\Psi \notin \text{LOOP}$

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

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

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

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

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

Still to show:

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

Summary

We showed that:

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

Still to show:

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

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