Advanced Topics in Theoretical Computer Science

Part 2: Register machines (2)

15.11.2012

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: sofronie@uni-koblenz.de

Until now

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

Today

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

Today

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

GOTO Programs: Syntax

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

GOTO Programs: Syntax

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

Definition

- **Atomic programs:**

  $x_i := x_i + 1$

  $x_i := x_i - 1$

  are GOTO instructions for each register x_i.

- If x_i is a register and j is an index then

  if $x_i = 0$ goto j is a GOTO instruction.

- If $I_1, \ldots, I_k$ 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:

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

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

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

GOTO Programs: Semantics

Let P be a GOTO program of the form:

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

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 for every $n \geq 0$ there exist:

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

such that the following hold:

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

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

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

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

and ....

(continuation on next page)

GOTO Programs: Semantics

Let P be a GOTO program of the form:

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

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 for every $n \geq 0$ there exist:

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

such that the following hold:

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

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

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

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

and .... (continuation on next page)

GOTO Programs: Semantics

Let P be a GOTO program of the form:

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

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 for every $n \geq 0$ there exist:

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

such that the following hold:

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

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

GOTO Programs: Semantics

Remark

The number of line changes (iterations) is not fixed at the beginning.

Infinite loops are possible.

GOTO Programs: Semantics

Remark

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

Notation

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

WHILE and GOTO

Theorem.

(1) WHILE $=$ GOTO

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

WHILE and GOTO

Theorem.

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

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

Proof:

To show:

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

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

WHILE and GOTO

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

Proof:

I. WHILE $\subseteq$ 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:

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.

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.

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

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:

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

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 $\mathsf{GOTO} \subseteq \mathsf{TM}$ and $\mathsf{GOTO}^{\mathsf{part}} \subseteq \mathsf{TM}^{\mathsf{part}}$

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
$x_i := x_i + 1$	$> \|^{(i)} R^{(i)}$
$x_i := x_i - 1$	$\begin{array}{l} > L^{(i)} \xrightarrow{\#^{(i)}} R^{(i)} \\ \quad\ \downarrow^{\|(i)} \\ \quad\ \#^{(i)} \end{array}$

GOTO $\subseteq$ TM

Proof (continued)

- Tape i contains as many $\vert$'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
$x_i := x_i + 1$	$> \vert^{(i)} R^{(i)}$
$x_i := x_i - 1$	$> L^{(i)} \xrightarrow{\#^{(i)}} R^{(i)}$ $\downarrow^{\vert(i)}$ $\#^{(i)}$

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

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

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

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: $\Psi \not\in$ 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!

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:

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

Summary

We showed that:

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

Still to show:

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

Summary

We showed that:

- LOOP $\subsetneq$ 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$^{\text{part}}$

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