Advanced Topics in Theoretical Computer Science

Part 2: Register machines: wrapping up

28.11.2013

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: sofronie@uni-koblenz.de

Contents

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

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 $s'_0, s'_1, \dots, s'_n$ 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:

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

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 $s'_0, \dots, s'_n$ and indices $z_0, \dots, z_n$ 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 $j_s = z_l$:

$$\begin{array}{ll}
\text{if } I_s = x_i := x_i \pm 1 \text{ then:} & s'_{i+1}(x_i) = s'_i(x_i) \pm 1 \\
 & s'_{i+1}(x_j) = s'_i(x_j) \text{ for } j \neq i \\
 & z_{i+1} = j_{s+1} \\
\text{if } I_s = \text{if } x_i = 0 \text{ goto } j_{\text{goto}} \text{ then:} & s'_{i+1} = s'_i \\
 & z_{i+1} = \begin{cases} j_{\text{goto}} & \text{if } x_i = 0 \\ j_{s+1} & \text{otherwise} \end{cases}
\end{array}$$

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

Theorem. Every LOOP program terminates for every input.

Consequence: All LOOP computable functions are total.

WHILE and GOTO programs can contain infinite loops. Therefore:

- WHILE programs do not always terminate
- WHILE computable functions can be undefined for some inputs (are partial functions)

Computable functions

LOOP = Set of all LOOP computable functions

WHILE = Set of all total WHILE computable functions

$\text{WHILE}^{\text{part}}$ = Set of all WHILE computable functions (including the partial ones)

GOTO = Set of all total GOTO computable functions

$\text{GOTO}^{\text{part}}$ = 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. LOOP $\subseteq$ WHILE (every LOOP computable function is WHILE computable)

Proof: 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.

Proof: II. $\text{WHILE} \supseteq \text{GOTO}$ and $\text{WHILE}^{\text{part}} \supseteq \text{GOTO}^{\text{part}}$

We proved that every GOTO program can be simulated with WHILE instructions.

Corollary
Every WHILE computable function can be computed by a **WHILE+IF** program with **one while loop only**.

Relationships between LOOP, WHILE, GOTO

Theorem: 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.

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:

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