Reasoning about Turing Machines and Low-Level Code

Christian Urban

in cooperation with Jian Xu and Xingyuan Zhang

A Trend in Verification

- in the past:

  model a problem mathematically and proof properties about the model

- needs elegance, is still very hard

A Trend in Verification

- in the past:

  model a problem mathematically and proof properties about the model

- needs elegance, is still very hard

- does not help with ensuring the correctness of running programs

A Trend in Verification

- make the specification executable (e.g. Compcert)

A Trend in Verification

- make the specification executable (e.g. Compcert)

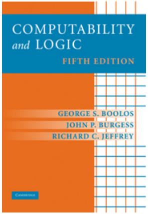
  you would expect the trend would be to for example model C, implement your programs in C and verify the programs written in C (e.g. seL4)

A Trend in Verification

- but actually people start to verify machine code directly (e.g. bignum arithmetic implemented in x86-64 – 700 instructions)
- CPU models exists, but the strategy is to use a small subset which you use in your programs

Why Turing Machines

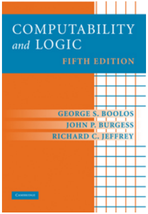
- at the beginning it was just a nice student project about computability



- found an inconsistency in the definition of halting computations (Chap. 3 vs Chap. 8)

Why Turing Machines

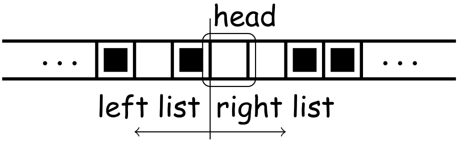
- at the beginning it was just a nice student project about computability



- found an inconsistency in the definition of halting computations (Chap. 3 vs Chap. 8)
- Norrish formalised computability via lambda-calculus (and nominal); Asperti and Riccioti formalised TMs but didn't get proper UTM

Turing Machines

- tapes contain 0 or 1 only

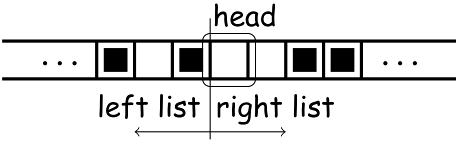


- steps function

  What does the tape look like after the TM has executed n steps?

Turing Machines

- tapes contain 0 or 1 only



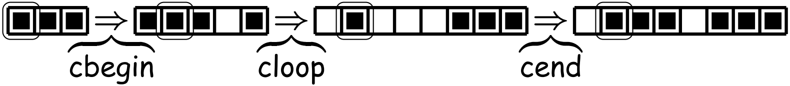
- steps function

  What does the tape look like after the TM has executed n steps?

designate the 0-state as halting state and remain there forever

Copy Turing Machines

- TM that copies a number on the input tape



$$\mathrm{cbegin} \stackrel{\mathrm{def}}{=} [(W_0, 0), (R, 2), (R, 3), (R, 2), (W_1, 3), (L, 4), (L, 4), (L, 0)]$$

$$\mathrm{cloop} \stackrel{\mathrm{def}}{=} [(R, 0), (R, 2), (R, 3), (W_0, 2), (R, 3), (R, 4), (W_1, 5), (R, 4), (L, 6), (L, 5), (L, 6), (L, 1)]$$

$$\mathrm{cend} \stackrel{\mathrm{def}}{=} [(L, 0), (R, 2), (W_1, 3), (L, 4), (R, 2), (R, 2), (L, 5), (W_0, 4), (R, 0), (L, 5)]$$

Hoare Logic for TMs

- Hoare-triples and Hoare-pairs:

$\{P\}\ p\ \{Q\} \stackrel{\text{def}}{=}$

$\forall$ tp.
if P tp holds then
$\exists$ n. such that
is_final (steps (1, tp) p n) $\wedge$
Q holds_for (steps (1, tp) p n)

$\{P\}\ p \uparrow \stackrel{\text{def}}{=}$

$\forall$ tp.
if P tp holds then
$\forall$ n. $\neg$ is_final (steps (1, tp) p n)

Hoare Reasoning

- reasoning is still quite difficult—invariants

$$I_1\ n\ (l, r) \stackrel{def}{=} (l, r) = ([], 0^n) \qquad \text{(starting state)}$$

$$I_2\ n\ (l, r) \stackrel{def}{=} \exists\, i\ j.\ 0 < i \wedge i + j = n \wedge (l, r) = (0^i, 0^j)$$

$$I_3\ n\ (l, r) \stackrel{def}{=} 0 < n \wedge (l, tl\ r) = (1{::}0^n, [])$$

$$I_4\ n\ (l, r) \stackrel{def}{=} 0 < n \wedge (l, r) = (0^n, [1, 0]) \vee (l, r) = (0^{n - 1}, [0, 1, 0])$$

$$I_0\ n\ (l, r) \stackrel{def}{=} 1 < n \wedge (l, r) = (0^{n - 2}, [0, 0, 1, 0]) \vee \qquad \text{(halting state)}$$
$$n = 1 \wedge (l, r) = ([], [1, 0, 1, 0])$$

$$J_1\ n\ (l, r) \stackrel{def}{=} \exists\, i\ j.\ i + j + 1 = n \wedge (l, r) = (0^i, 0{::}0{::}1^j\ @\ 0^j) \wedge 0 < j \vee$$
$$0 < n \wedge (l, r) = ([], 1{::}0{::}1^n\ @\ 0^n) \qquad \text{(starting state)}$$

$$J_0\ n\ (l, r) \stackrel{def}{=} 0 < n \wedge (l, r) = ([1], 0{::}1^n\ @\ 0^n) \qquad \text{(halting state)}$$

$$K_1\ n\ (l, r) \stackrel{def}{=} 0 < n \wedge (l, r) = ([1], 0{::}1^n\ @\ 0^n) \qquad \text{(starting state)}$$

$$K_0\ n\ (l, r) \stackrel{def}{=} 0 < n \wedge (l, r) = ([1], 0^n\ @\ (1{::}0^n)) \qquad \text{(halting state)}$$

Register Machines

- instructions

I	::=	Inc R	increment register R by one
	|	Dec R L	if content of R is non-zero, then decrement it by one otherwise jump to instruction L
	|	Goto L	jump to instruction L

Recursive Functions

- addition, multiplication, ...
- logical operations, quantifiers...
- coding of numbers (Cantor encoding)
- UF

Recursive Functions

- addition, multiplication, ...
- logical operations, quantifiers...
- coding of numbers (Cantor encoding)
- UF

- Recursive Functions $\Rightarrow$ Register Machines
- Register Machines $\Rightarrow$ Turing Machines

Sizes

- UF (size: 140843)
- Register Machine (size: 2 Mio instructions)
- UTM (size: 38 Mio states)

old version: RM (12 Mio) UTM (112 Mio)

Separation Algebra

- introduced a separation algebra framework for register machines and TMs
- we can semi-automate the reasoning for our small TMs
- we can assemble bigger programs out of smaller components

- looks awfully like “real” assembly code

Separation Algebra

- introduced a separation algebra framework for register machines and TMs
- we can semi-automate the reasoning for our small TMs
- we can assemble bigger programs out of smaller components

- looks awfully like "real" assembly code
- Conclusion: we have a playing ground for reasoning about low-level code; we work on automation