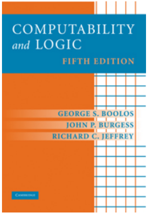
Mechanising Turing Machines and Computability Theory in Isabelle

Jian Xu Xingyuan Zhang
PLA University of Science and Technology

Christian Urban
King's College London

Why Turing Machines?

- At the beginning, it was just a student project about computability.



Computability and Logic (5th. ed)
Boolos, Burgess and Jeffrey

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

Some Previous Works

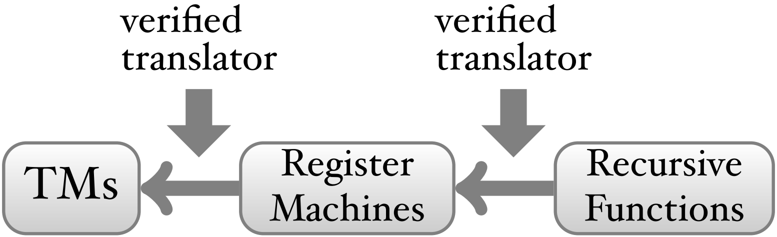
- Norrish formalised computability theory in HOL starting from the lambda-calculus
  - for technical reasons we could not follow him
  - some proofs use TMs (Wang tilings)

- Asperti and Ricciotti formalised TMs in Matita
  - no undecidability $\Rightarrow$ interest in complexity
  - their UTM operates on a different alphabet than the TMs it simulates.

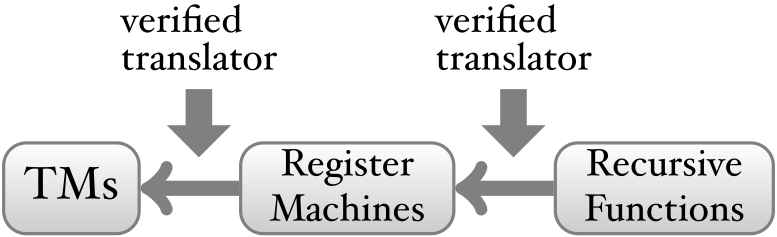
"In particular, the fact that the universal machine operates with a different alphabet with respect to the machines it simulates is annoying." [Asperti and Ricciotti]

The Big Picture

The Big Picture



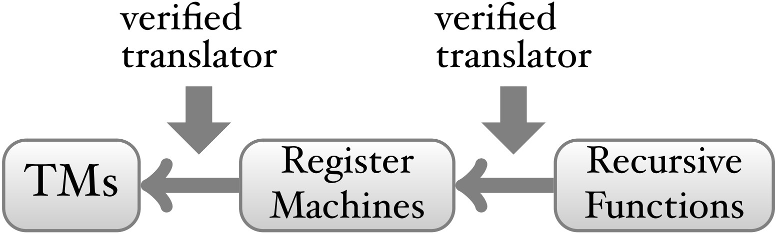
The Big Picture



undecidabilty of the halting problem

UF

The Big Picture



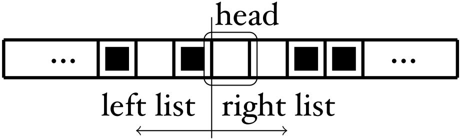
undecidabilty of the halting problem

UF

correct UTM by translation

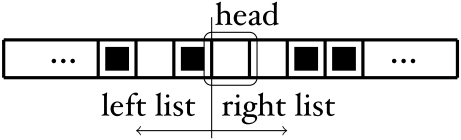
Turing Machines

- tapes are lists and contain *0*s or *1*s only



Turing Machines

- tapes are lists and contain *0*s or *1*s only

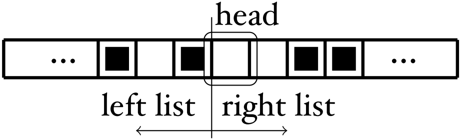


- *steps* function:

  What does the TM claclulate after it has executed n steps?

Turing Machines

- tapes are lists and contain *0*s or *1*s only



- *steps* function:

  What does the TM claclulate after it has executed *n* steps?

- designate the *0*-state as "halting state" and remain there forever, i.e. have a *Nop*-action

Register Machines

- programs are lists of instructions

$$
\begin{array}{rcll}
I & ::= & \mathit{Goto}\ L & \text{jump to instruction } L \\
 & \mid & \mathit{Inc}\ R & \text{increment register } R \text{ by one} \\
 & \mid & \mathit{Dec}\ R\ L & \text{if content of } R \text{ is non-zero,} \\
 & & & \text{then decrement it by one} \\
 & & & \text{otherwise jump to instruction } L
\end{array}
$$

Register Machines

Spaghetti Code!

f instructions

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

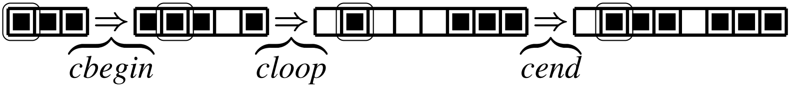
Recursive Functions

rec	$::=$	Z	zero-function
	$\mid$	S	successor-function
	$\mid$	Id^n_m	projection
	$\mid$	$Cn^n\ f\ gs$	composition
	$\mid$	$Pr^n\ f\ g$	primitive recursion
	$\mid$	$Mn^n\ f$	minimisation

- $eval :: rec \Rightarrow nat\ list \Rightarrow nat$
  can be defined by simple recursion
  (HOL has *Least*)
- you define
  - addition, multiplication, logical operations, quantifiers...
  - coding of numbers (Cantor encoding), UTM

Copy Turing Machine

- TM that copies a number on the input tape



$$copy \stackrel{def}{=} cbegin \; ; \; cloop \; ; \; cend$$

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

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

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

Hoare Logic for TMs

- Hoare-triples

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

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

Hoare Logic for TMs

- Hoare-triples and Hoare-pairs:

$\boxed{\{P\}\ p\ \{Q\}} \stackrel{def}{=}$
$\forall\ tp.$
if $P\ tp$ holds then
$\exists\ n.$ such that
$is\_final\ (steps\ (1,\ tp)\ p\ n)\ \wedge$
$Q\ holds\_for\ (steps\ (1,\ tp)\ p\ n)$

$\boxed{\{P\}\ p \uparrow} \stackrel{def}{=}$
$\forall\ tp.$
if $P\ tp$ holds then
$\forall\ n.\ \neg\ is\_final\ (steps\ (1,\ tp)\ p\ n)$

Some Derived Rules

$$\frac{P' \mapsto P \qquad \{P\}\, p\, \{Q\} \qquad Q \mapsto Q'}{\{P'\}\, p\, \{Q'\}}$$

$$\frac{\{P\}\, p_1\, \{Q\} \qquad \{Q\}\, p_2\, \{R\}}{\{P\}\, p_1\, ;\, p_2\, \{R\}} \qquad \frac{\{P\}\, p_1\, \{Q\} \qquad \{Q\}\, p_2 \uparrow}{\{P\}\, p_1\, ;\, p_2 \uparrow}$$

Undecidability

$$contra \stackrel{def}{=} copy \,;\, H \,;\, dither$$

Undecidability

$$contra \stackrel{def}{=} copy \;;\; H \;;\; dither$$

- Suppose H decides *contra* called with code of *contra* halts, then

$$P_1 \stackrel{def}{=} \lambda tp.\ tp = ([\,],\ \langle code\ contra\rangle)$$
$$P_2 \stackrel{def}{=} \lambda tp.\ tp = ([0],\ \langle(code\ contra,\ code\ contra)\rangle)$$
$$P_3 \stackrel{def}{=} \lambda tp.\ \exists\, k.\ tp = (0^k,\ \langle 0\rangle)$$

$$\dfrac{\dfrac{\{P_1\}\ copy\ \{P_2\} \quad \{P_2\}\ H\ \{P_3\}}{\{P_1\}\ copy\ ;\ H\ \{P_3\}} \qquad \{P_3\}\ dither\uparrow}{\{P_1\}\ contra\uparrow}$$

Undecidability

$$contra \stackrel{def}{=} copy \; ; \; H \; ; \; dither$$

- Suppose H decides *contra* called with code of *contra* does *not* halt, then

$$Q_1 \stackrel{def}{=} \lambda tp.\; tp = ([\,], \langle code\ contra \rangle)$$
$$Q_2 \stackrel{def}{=} \lambda tp.\; tp = ([0], \langle (code\ contra,\ code\ contra) \rangle)$$
$$Q_3 \stackrel{def}{=} \lambda tp.\; \exists\, k.\; tp = (0^k, \langle 1 \rangle)$$

$$\dfrac{\dfrac{\{Q_1\}\ copy\ \{Q_2\} \qquad \{Q_2\}\ H\ \{Q_3\}}{\{Q_1\}\ copy\ ;\ H\ \{Q_3\}} \qquad \{Q_3\}\ dither\ \{Q_3\}}{\{Q_1\}\ contra\ \{Q_3\}}$$

Hoare Reasoning

- reasoning is still quite demanding;
  the invariants of the copy-machine:

$$I_1\ n\ (l, r) \stackrel{def}{=} (l, r) = ([], 1^n) \qquad \text{(starting state)}$$

$$I_2\ n\ (l, r) \stackrel{def}{=} \exists\, i\, j.\ 0 < i \wedge i + j = n \wedge (l, r) = (1^i, 1^j)$$

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

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

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

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

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

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

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

Midway Conclusion

- feels awfully like reasoning about machine code
- compositional constructions / reasoning not frictionless
- sizes

	sizes:
UF	140843 constructors
Reg. Mach.	*2* Mio instructions
UTM	*38* Mio states

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

Midway Conclusion

- feels awfully like reasoning about machine code
- compositional constructions / reasoning not frictionless
- sizes

	sizes:
UF	140843 constructors
Reg. Mach.	*2* Mio instructions
UTM	*38* Mio states

- an observation: our treatment of recursive functions is a mini-version of the work by Myreen & Owens about deeply embedding HOL

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