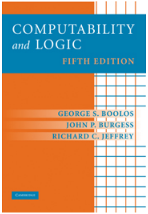
Mechanising Turing Machines and Computability Theory in Isabelle

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

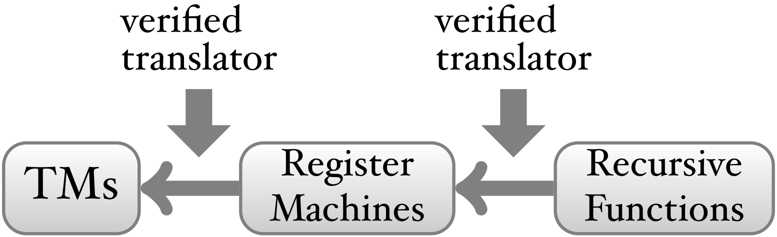
- Norrish formalised computability theory in HOL starting from the lambda-calculus
  - for technical reasons we could not follow him
  - some proofs use TMs (Wang tilings)

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

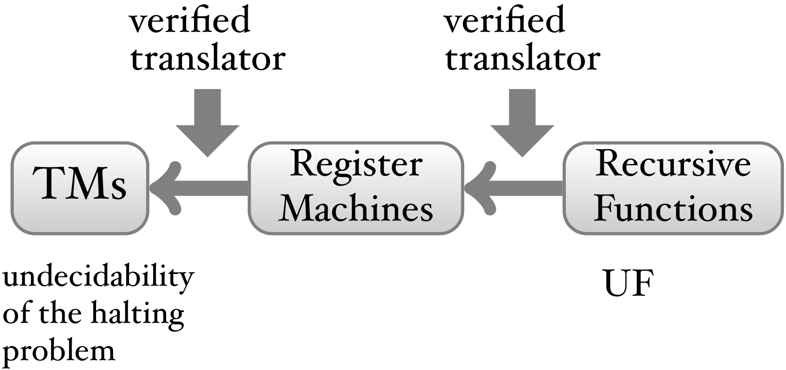
"In particular, the fact that the universal machine operates with a different alphabet with respect to the machines it simulates is annoying." [Asperti and Ricciotti]

The Big Picture

The Big Picture



The Big Picture



The Big Picture

verified translator

verified translator

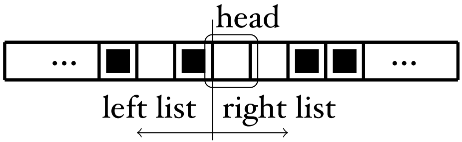
undecidability of the halting problem

UF

correct UTM by translation

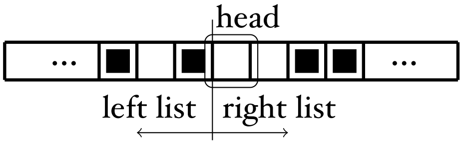
Turing Machines

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



Turing Machines

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

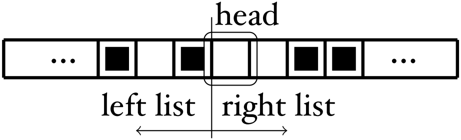


- *steps* function:

  What does the TM calculate 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 calculate 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

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

Register Machines

- programs are lists of 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

Spaghetti Code!

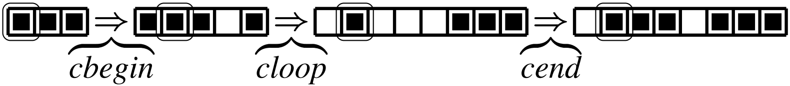
Recursive Functions

$$
\begin{array}{lcll}
rec & ::= & Z & \text{zero-function} \\
 & \mid & S & \text{successor-function} \\
 & \mid & Id^n_m & \text{projection} \\
 & \mid & Cn^n\, f\, gs & \text{composition} \\
 & \mid & Pr^n\, f\, g & \text{primitive recursion} \\
 & \mid & Mn^n\, f & \text{minimisation}
\end{array}
$$

- $eval :: rec \Rightarrow nat\ list \Rightarrow nat$
  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

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

$\quad \forall\ tp.$

$\quad\quad$ if $P\ tp$ holds then

$\quad\quad \exists\ n.$ such that

$\quad\quad$ *is_final (steps (1, tp) p n)* $\wedge$

$\quad\quad$ *Q holds_for (steps (1, tp) p n)*

Hoare Logic for TMs

- Hoare-triples and Hoare-pairs:

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

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

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

$\forall\ tp.$
  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

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

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

Hoare Reasoning

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

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

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

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

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

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

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

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

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

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

Midway Conclusion

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

	sizes:
UF	140843 constructors
URM	*2* Mio instructions
UTM	*38* Mio states

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

Midway Conclusion

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

	sizes:
UF	140843 constructors
URM	*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

Inspiration from other Works

Inspiration from other Works

Stealing

~~Inspiration~~ Stealing from other Works

- Jensen, Benton, Kennedy (**2013**), *High-Level Separation Logic for Low-Level Code*
- Myreen (**2008**), *Formal Verification of Machine-Code Programs*, PhD thesis
- Klein, Kolanski, Boyton (**2012**), *Mechanised Separation Algebra*

Better Composability

- an idea from Jensen, Benton, Kennedy who looked at X86 assembly programs and macros

- assembly for TMs:

$$\begin{array}{l} \mathit{move\_one\_left} \stackrel{\mathit{def}}{=} \\ \quad \Lambda\ \mathit{exit}. \\ \quad\quad \mathit{Inst}\ (L, \mathit{exit})\ (L, \mathit{exit})\ ; \\ \quad\quad \mathit{Label}\ \mathit{exit} \end{array}$$

$\Rightarrow$ represent "state" labels as functions (with bound variables $\Rightarrow$ locality)

Better Composability

$$
\begin{aligned}
\mathit{move\_left\_until\_zero} \overset{def}{=}& \\
& \Lambda\ \mathit{start}\ \mathit{exit}. \\
& \quad \mathit{Label}\ \mathit{start}\ ; \\
& \quad \mathit{if\_zero}\ \mathit{exit}\ ; \\
& \quad \mathit{move\_left}\ ; \\
& \quad \mathit{jmp}\ \mathit{start}\ ; \\
& \quad \mathit{Label}\ \mathit{exit}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{if\_zero}\ e &\overset{def}{=} \Lambda\ \mathit{exit}.\ \mathit{Inst}\ (W_0, e), (W_1, \mathit{exit});\ \mathit{Label}\ \mathit{exit} \\
\mathit{jmp}\ e &\overset{def}{=} \mathit{Inst}\ (W_0, e), (W_1, e)
\end{aligned}
$$

The Trouble With Hoare-Triples

- Whenever we wanted to prove

$$\{P\}\ p\ \{Q\}$$

(1) we had to find a termination order proving that p terminates (not easy)

(2) we had to find invariants for each state (not easy either)

The Trouble With Hoare-Triples

- Whenever we wanted to prove

$$\{P\}\ p\ \{Q\}$$

(1) we had to find a termination order proving that p terminates (not easy)

(2) we had to find invariants for each state (not easy either)

very little opportunity for automation

Separation Algebra

- use some infrastructure introduced by Klein et al in Isabelle/HOL
- and an idea by Myreen

$$\{\!|p|\!\}\ c\ \{\!|q|\!\}$$

p, c, q will be assertions in a separation logic

Separation Algebra

- use some infrastructure introduced by Klein et al in Isabelle/HOL
- and an idea by Myreen

$$\{\!|p|\!\}\ c\ \{\!|q|\!\}$$

$p,\ c,\ q$ will be assertions in a separation logic

e.g. $\{\!|st\ i \star hd\ n \star ones\ u\ v \star zero\ (v + 1)|\!\}$

Separation Triples

$$
\{\!|p|\!\}\ c\ \{\!|q|\!\} \stackrel{def}{=} \\
\quad \forall\ cf\ r. \\
\quad\quad (p \star c \star r)\ cf \text{ implies} \\
\quad\quad \exists\ k.\ (q \star c \star r)\ (run\ k\ cf)
$$

c can be *i:[move_left_until_zero]:j*

Automation

- we introduced some tactics for handling sequential programs

$$\{\!|p|\!\}\ i{:}[c_1\ ;\ \ldots\ ;\ c_n]{:}j\ \{\!|q|\!\}$$

- for loops we often only have to do inductions on the length of the input (e.g. how many *1*s are on the tape)

Automation

- we introduced some tactics for handling sequential programs

$$\{\!|p|\!\}\ i{:}[c_1\ ;\ \ldots\ ;\ c_n]{:}j\ \{\!|q|\!\}$$

- for loops we often only have to do inductions on the length of the input (e.g. how many *1*s are on the tape)
- these macros allow us to completely get rid of register machines

Conclusion

- What started out as a student project, turned out to be much more fun than first thought.

- Where can you claim that you proved the correctness of a *38* Mio instruction program. (ca. *7000* is the soa)

- We learned a lot about current verification technology for low-level code (we had no infrastructure: CPU model).