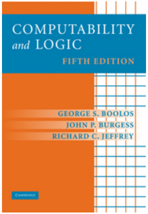
Turing Machines and Separation Logic
(Work in Progress)


Jian Xu Xingyuan Zhang
PLA University of Science and Technology

Christian Urban
King's College London

Why Turing Machines?

- we wanted to formalise computability theory
- at the beginning, it was just a student project



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

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

Why Turing Machines?

- we wanted to formalise computability theory
  - TMs are a fantastic model of low-level code
  - completely unstructured

  - good testbed for verification techniques
  - Can we verify a program with 38 Mio instructions?
  - we can delay implementing a concrete machine model (for OS/low-level code verification)
- found an inconsistency in the definition of halting computations (Chap. 3 vs Chap. 8)

Why Turing Machines?

- we wanted to formalise computability theory
  - TMs are a fantastic model of low-level code
  - completely unstructured Spaghetti Code
  - good testbed for verification techniques
  - Can we verify a program with 38 Mio instructions?
  - we can delay implementing a concrete machine model (for OS/low-level code verification)
- found an inconsistency in the definition of halting computations (Chap. 3 vs Chap. 8)

Some Previous Works

(but not interested in low-level code)

- Norrish formalised computability theory in HOL starting from the lambda-calculus
  - for technical reasons we could not follow his work
  - 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

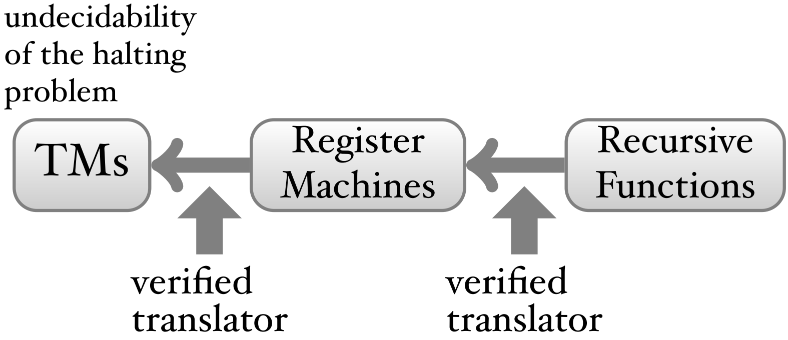
undecidability
of the halting
problem

TMs

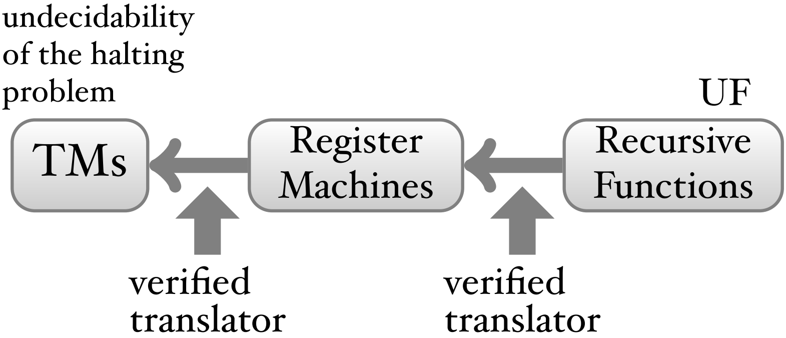
Register Machines

Recursive Functions

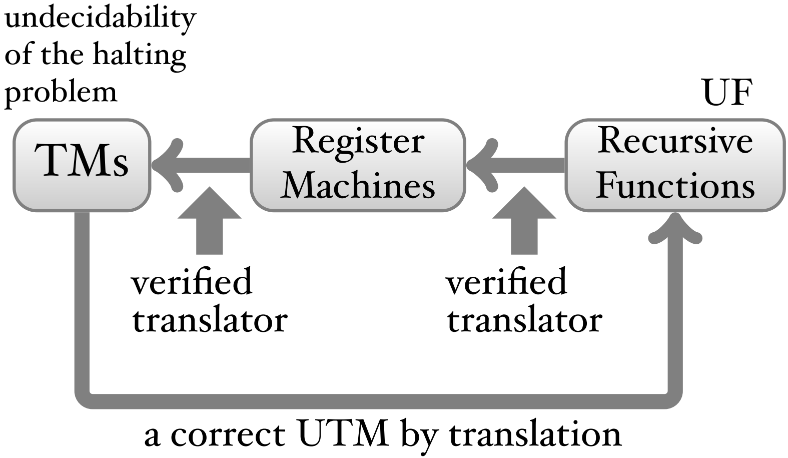
The Big Picture



The Big Picture

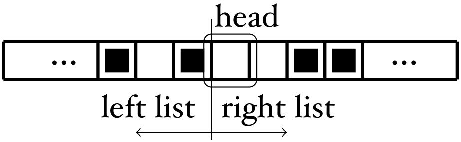


The Big Picture



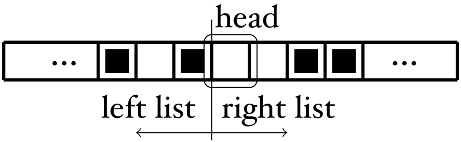
Turing Machines

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



Turing Machines

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

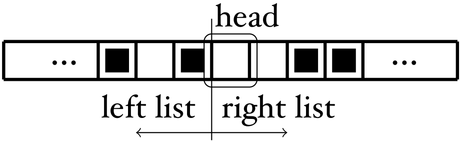


- *steps* function:

  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 the 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 the content of R is non-zero, then decrement it by one otherwise jump to instruction L

Spaghetti Code!

Recursive Functions

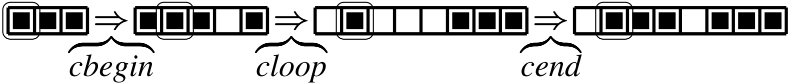
$$
\begin{array}{llll}
rec & ::= & Z & \text{zero-function} \\
 & \mid & S & \text{successor-function} \\
 & \mid & Id^n_m & \text{projection} \\
 & \mid & Cn^n\, f\, gs & \text{composition} \\
 & \mid & Pr^n\, f\, g & \text{primitive recursion} \\
 & \mid & Mn^n\, f & \text{minimisation}
\end{array}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

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

Copy Turing Machine

- TM that copies a number on the input tape

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



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

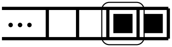
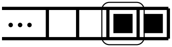
$cloop \stackrel{def}{=}$
$[(R, 0), (R, 2), (R, 3),$
$(W_0, 2), (R, 3), (R, 4),$
$(W_1, 5), (R, 4), (L, 6),$
$(L, 5), (L, 6), (L, 1)]$

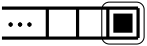
$cend \stackrel{def}{=}$
$[(L, 0), (R, 2), (W_1, 3),$
$(L, 4), (R, 2), (R, 2),$
$(L, 5), (W_0, 4), (R, 0),$
$(L, 5)]$

Dither Machine

- TM that is the identity with *1* and loops with *0*

start tape

halting case:  $\Rightarrow$ 

non-halting case:  $\Rightarrow$ loops

$$dither \stackrel{def}{=} [(W_0, 1), (R, 2), (L, 1), (L, 0)]$$

Hoare Logic for TMs

- Hoare-triples

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

$\forall\, tp.$
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:

$\{P\}\ p\ \{Q\} \stackrel{def}{=}$
$\forall\, tp.$
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 whether *contra* called with the code of *contra* halts, then

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

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

Undecidability

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

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

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

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

Hoare Reasoning

- reasoning is quite demanding, e.g. the invariants of the copy-machine:

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

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

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

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

$$\begin{aligned} I_0\ n\ (l, r) \stackrel{def}{=}\ & 1 < n \land (l, r) = (1^{n-2}, [1, 1, 0, 1]) \lor \qquad \text{(halting state)} \\ & n = 1 \land (l, r) = ([], [0, 1, 0, 1]) \end{aligned}$$

$$\begin{aligned} J_1\ n\ (l, r) \stackrel{def}{=}\ & \exists\, i\, j.\ i + j + 1 = n \land (l, r) = (1^i, 1{::}1{::}0^j @ 1^j) \land 0 < j \lor \\ & 0 < n \land (l, r) = ([], 0{::}1{::}0^n @ 1^n) \qquad \text{(starting state)} \end{aligned}$$

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

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

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

Midway Conclusion

- feels awfully like reasoning about machine code
- compositional constructions / reasoning is 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)

The Trouble With Hoare-Triples

- Whenever we wanted to prove

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

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

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

The Trouble With Hoare-Triples

- Whenever we wanted to prove

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

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

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

very little opportunity for automation

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 assembler programs and macros

- assembler for TMs:

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

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

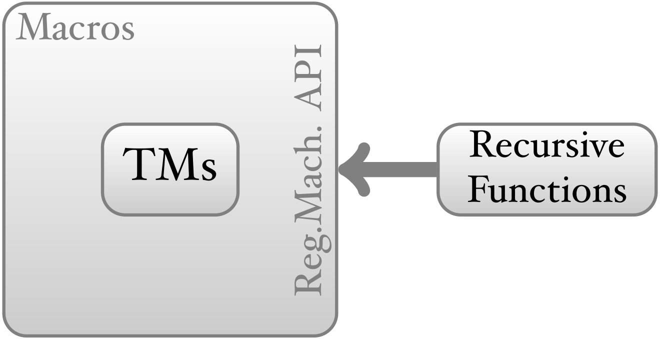
Better Composability

$$\begin{array}{l} move\_left\_until\_zero \stackrel{def}{=} \\ \qquad \Lambda\ start\ exit. \\ \qquad\quad Label\ start\ ; \\ \qquad\quad if\_zero\ exit\ ; \\ \qquad\quad move\_left\ ; \\ \qquad\quad jmp\ start\ ; \\ \qquad\quad Label\ exit \end{array}$$

$$if\_zero\ e \stackrel{def}{=} \Lambda\ exit.\ Inst\ (W_0, e), (W_1, exit); Label\ exit$$

$$jmp\ e \stackrel{def}{=} Inst\ (W_0, e), (W_1, e)$$

An RM-API with TMs



- Suppose the first four registers of an RM contain 1,2,0 and 3, then the encoding is

Inc a

$$
\begin{aligned}
\mathit{Inc}\ a \stackrel{\mathit{def}}{=}\ & \\
& \mathit{locate}\ a\ ;\\
& \mathit{right\_until\_zero}\ ;\\
& \mathit{move\_right}\ ;\\
& \mathit{shift\_right}\ ;\\
& \mathit{move\_left}\ ;\\
& \mathit{left\_until\_double\_zero}\ ;\\
& \mathit{write\_one}\ ;\\
& \mathit{left\_until\_double\_zero}\ ;\\
& \mathit{move\_right}\ ;\\
& \mathit{move\_right}
\end{aligned}
$$

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

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

$$\{\!|\ st\ i \star hd\ v \star zero\ u \star ones\ (u + 1)\ v\ |\!\}$$
$$i{:}[left\_until\_zero]{:}j$$
$$\{\!|\ st\ j \star hd\ u \star zero\ u \star ones\ (u + 1)\ v\ |\!\}$$

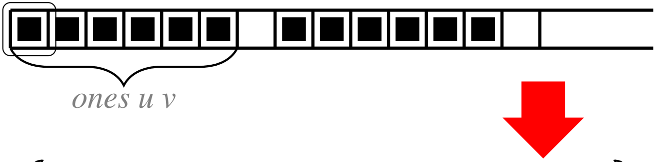
Inductions over *ones*

- What most simplifies the work is that we can do inductions over the "input" (inductively defined assertions)

Inductions over *ones*

- What most simplifies the work is that we can do inductions over the "input" (inductively defined assertions)

- Suppose *right_until_zero*:



$$\{\!|\ st\ i \star hd\ u \star zero\ (v + 1) \star ones\ u\ v\ |\!\}$$
$$i{:}[right\_until\_zero]{:}j$$
$$\{\!|\ st\ j \star hd\ (v + 1) \star zero\ (v + 1) \star ones\ u\ v\ |\!\}$$

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 *0*s/*1*s are on the tape)
- no termination measures are needed

Register Machines

- We could also use Jensen's et al work to give a more appropriate view on register machines

$$\{\!|p|\!\}\ i{:}[rm\_c]{:}j\ \{\!|q|\!\}$$

- Rule for *Inc*

$$\begin{array}{ll} RM. & \{\!|\ pc\ i \star m\ a\ v\ |\!\} \\ & i{:}[\ Inc\ a\ ]{:}j \\ & \{\!|\ pc\ j \star m\ a\ (Suc\ v)|\!\} \end{array}$$

- Rules for *Dec*

$$\begin{array}{ll} RM. & \{\!|(pc\ i \star m\ a\ (Suc\ v))|\!\} \\ & i\ {:}[\ Dec\ a\ e\ ]{:}\ j \\ & \{\!|pc\ j \star m\ a\ v|\!\} \end{array} \qquad \begin{array}{ll} RM. & \{\!|\ pc\ i \star m\ a\ 0\ |\!\} \\ & i{:}[\ Dec\ a\ e\ ]{:}j \\ & \{\!|\ pc\ e \star m\ a\ 0\ |\!\} \end{array}$$

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).
- The existing literature on TMs & RMs leave out quite a bit of the story (not to mention contains bugs).