Formalizing Turing Machines

Andrea Asperti & Wilmer Ricciotti

Department of Computer Science, University of Bologna
Mura Anteo Zamboni 7, 40127, Bologna, ITALY
asperti@cs.unibo.it

Wollic 2012

Buenos Aires, Argentina, September 2012

Abstract

We discuss the formalization, in the Matita Theorem Prover, of a few, basic results on Turing Machines, up to the existence of a (certified) Universal Machine.

The work is a first step towards the creation of a formal repository in Complexity Theory, and a piece of a long term work of logical revisitation of the foundations of Complexity.

Aim of the talk

Provide evidence that formalizing and checking (elements of) Computablity/Complexity Theory is an effort that

- can be done
- is worth to be done
- will eventually be done

Content

Outline

About Matita

Matita [7] (pencil) is an implementation of the Calculus of (Co-)Inductive Constructions alternative to Coq.

Distinctive features

- light
- completely functional
- native open terms [9]
- bidirectional type inference [8]
- small step execution of structured tactics (tinycals) [18]
- well documented

A good environment for learning the practice of formal development and the internals of interactive provers.

Some Matita developments

- Number theory: Properties of Möbius μ, Euler φ and Chebyshev Θ functions; Bertrand's postulate [5]
- Constructive analysis: Lebesgue's dominated convergence theorem [16]
- Formal topology: elements of pointless topology [17]
- Programming languages metatheory: solution to the POPLmark challenge [6]
- Compilers verification: EU Project CerCo (Certified Complexity) for the verification of a formally certified complexity preserving compiler for the C programming language [2].

Outline

Formalization

Formal encoding in a format suitable for automatic verification.

Major achievement in different areas of Computer Science:

- hardware verification
- formal languages and compilers
- protocols and security
- metatheory of programming languages
- ...

Very little work in Computability and Complexity Theory (Norrish [12]).

Turing Machines

(Too) many variants

- deterministic/ non deterministic
- number of tapes/pushdowns stores
- alphabet
- on-line/off-line (strong on-line)
- memory models: tape/pushdown/stack (oblivious tapes)

Ming Li [11]

It is essential to understand the precise relationship among those computing models, e.g., with or without nondeterminism and/or some more tapes (or pushdown stores).

Some results (deterministic case)

Upper bounds:

- 1 tape simulation of k tapes in $O(t^2)$ (Hartmanis & Stearns [10])
- 2 tape simulation of k tapes in $O(tlogt)$ (Hennie & Stearns [20])

Lower bounds:

- 2 tapes are better than 1 (Rabin [15])
- k tapes are better than $k-1$ (Aanderaa [1], Paul, Seiferas & Simon [14])
- simulating k tapes by $k-1$ takes $\Omega(nlog^{1/k}n)$ time for strong on-line machines (Paul [13])
- simulating one queue or two pushdown stores by one tape takes $\Omega(n^{1.618})$ time (Vitanyi [22])
- ...

Motivations

Small variations in the memory model have sensible implications on complexity.

A mechanical check would be welcome.

Motivations internal to ITP

New domains present new problems and induce innovative techniques:

- Higher order languages& Type systems
  $\rightarrow$ binding problems and (re)naming of variables
  $\rightarrow$ nominal techniques
- Semantics of programming languages
  $\rightarrow$ local memory modifications
  $\rightarrow$ separation logics
- Computability & Complexity Theory
  $\rightarrow$???
  $\rightarrow$???

Main motivation

We are interested in formalizing Turing Machines . . .

precisely because we are not really interested in them.

We need to find the right level of abstraction, for reasoning about complexity in a machine independent way.

Interactive provers can really help in this study.

Main motivation

We are interested in formalizing Turing Machines . . .

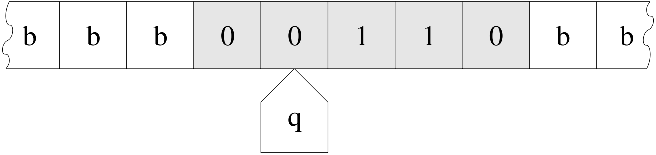
precisely because we are not really interested in them.

We need to find the right level of abstraction, for reasoning about complexity in a machine independent way.

Interactive provers can really help in this study.

Outline

Turing Machines



We shall work with single tape Turing Machines.

The machine

```
record TM (sig:FinSet): Type :=
  { states  : FinSet;
    trans  :  states  × (option  sig ) →
                  states  ×  (option  (sig  ×  move));
    start :  states ;
    halt  :  states  → bool}.
```

Since *trans* works on finite sets, its graph is a finite set too, and we have library functions to pass between the two representations.

Computations

```
record config (sig, states:FinSet): Type :=
  {cstate : states; ctape: tape sig}.

definition step :=λsig.λM:TM sig.λc:config sig (states sig M).
  let current_char :=current ? (ctape ?? c) in
  let ⟨news,mv⟩ :=trans sig M ⟨cstate ?? c, current_char⟩ in
  mk_config ?? news (tape_move sig (ctape ?? c) mv).

let rec loop (A:Type) n (f:A→A) p a on n :=
  match n with
  [ O ⇒ None ?
  | S m ⇒ if p a then (Some ? a) else loop A m f p (f a) ].

definition loopM :=λsig,M,i,inc.
  loop ? i (step sig M) (λc.halt sig M (cstate ?? c)) inc.
```

Semantics

We express semantics in terms of relations between tapes (not configurations!) realized by the machine:

```
definition initc :=λsig.λM:TM sig.λt.
  mk_config sig (states sig M) (start sig M) t.

definition Realize :=λsig.λM:TM sig.λR:relation (tape sig).
∀t.∃i.∃outc.
  loopM sig M i (initc sig M t) = Some ? outc ∧ R t (ctape ?? outc).
```

notation: $M \models R$

Remark We work with tapes for compositionality reasons: Turing machine may work with a common notion tape but have different internal states.

Variants (w.r.t. termination)

Realizability implies termination; we may define a weaker notion

```
definition WRealize :=λsig.λM:TM sig.λR:relation (tape sig ).
∀t, i ,outc.
  loopM sig M i ( initc  sig  M t) = Some ? outc → R t (ctape ?? outc).
```

notation: $M \Vdash R$

Weak realizability + termination implies realizablity.

Variants (w.r.t. final state)

Conditional realizability:

```
definition accRealize sig (M:TM sig) (q:states sig M) Rtrue Rfalse.
∀t.∃i.∃outc.
  loopM sig M i (initc sig M t) = Some ? outc ∧
    (cstate ?? outc = q → Rtrue t (ctape ?? outc)) ∧
    (cstate ?? outc ≠ q → Rfalse t (ctape ?? outc)).
```

notation: $M \models_q [Rtrue, Rfalse]$

Outline

Sequential composition

```
definition seq_trans :=λsig. λM1,M2 : TM sig.
λp. let ⟨s,a⟩ :=p in
  match s with
  [ inl s1 ⇒
      if halt sig M1 s1 then ⟨inr … (start sig M2), None ?⟩
      else let ⟨news1,m⟩ :=trans sig M1 ⟨s1,a⟩ in ⟨ inl … news1,m⟩
  | inr s2 ⇒
      let ⟨news2,m⟩ :=trans sig M2 ⟨s2,a⟩ in ⟨ inr … news2,m⟩
  ].

definition seq :=λsig. λM1,M2 : TM sig.
  mk_TM sig
    (FinSum (states sig M1) (states sig M2))
    (seq_trans sig M1 M2)
    (inl … (start sig M1))
    (λs.match s with [inl _ ⇒ false | inr s2 ⇒ halt sig M2 s2]).
```

Semantics of Sequential composition

$$\text{if } M_1 \models R_1 \text{ and } M_2 \models R_2 \text{ then}$$

$$M_1 \cdot M_2 \models R_1 \circ R_2$$

The proof is less trivial than expected: M_1 and M_2 work with their own internal states, and we should "lift" their computation to the states of the sequential machine.

Semantics of Sequential composition

$$\text{if } M_1 \models R_1 \text{ and } M_2 \models R_2 \text{ then}$$

$$M_1 \cdot M_2 \models R_1 \circ R_2$$

The proof is less trivial than expected: M_1 and M_2 work with their own internal states, and we should "lift" their computation to the states of the sequential machine.

If then else

```
definition if_trans :=λsig. λM1,M2,M3:TM sig. λq:states sig M1.λp.
let ⟨s,a⟩ :=p in
  match s with
  [ inl s1 ⇒
      if halt sig M1 s1 then
        if s1==q then ⟨inr …(inl …(start sig M2)), None ?⟩
        else ⟨ inr …(inr …(start sig M3)), None ?⟩
      else let ⟨news1,m⟩ :=trans sig M1 ⟨s1,a⟩ in
       ⟨ inl …news1,m⟩
  | inr s' ⇒
      match s' with
      [ inl s2 ⇒ let ⟨news2,m⟩ :=trans sig M2 ⟨s2,a⟩ in
         ⟨ inr …(inl …news2),m⟩
      | inr s3 ⇒ let ⟨news3,m⟩ :=trans sig M3 ⟨s3,a⟩ in
         ⟨ inr …(inr …news3),m⟩ ] ].
```

Semantics of if_then_else

if $M_1 \models_{acc} [Rtrue, Rfalse]$, $M_2 \models R_2$ and $M_3 \models R_3$

then

$$ifTM\ sig\ M_1\ M_2\ M_3\ acc \models (Rtrue \circ R_2) \cup (Rfalse \circ R_3)$$

While

```
definition  while_trans  :=λsig. λM : TM sig. λq:states  sig  M. λp.
  let ⟨s,a⟩ :=p in
  if  s  == q then ⟨start ? M, None ?⟩
  else  trans  ? M p.

definition  whileTM :=λsig. λM : TM sig. λq: states  ? M.
  mk_TM sig
    (states  ? M)
    (while_trans  sig  M q)
    (start  sig  M)
    (λs. halt  sig  M s ∧ ¬ s==q).
```

Semantics of while

$$\text{if } M \models_q [Rtrue, Rfalse]$$
$$\text{then}$$
$$whileTM\ sig\ M\ q \Vdash Rtrue^* \circ Rfalse$$

where $\Vdash$ denotes weak realizability.

We can reduce the termination of whileTM to the well foundedness of $Rtrue^{-1}$.

Basic Machines

write c	write the character *c* on the tape at the current head position
move_r	move the head one step to the right
move_l	move the head one step to the left
test_char f	perform a boolean test *f* on the current character and enter state *tc_true* or *tc_false* according to the result of the test
swap_r	swap the current character with its right neighbour
swap_l	swap the current character with its left neighbour

Outline

Normal Machines

A normal Turing machine is an ordinary machine where:

1. the tape alphabet is $\{0, 1\}$;
2. the finite states are supposed to be an initial interval of the natural numbers.

By convention, we assume the starting state is 0.

```
record normalTM : Type :=
{ no_states : nat;
  pos_no_states : (0 < no_states);
  ntrans : (initN no_states) × Option bool
              → (initN no_states) × Option (bool × Move);
  nhalt : initN no_states → bool}.
```

The universal machine

- Every TM can be transformed into a Normal Machine with a linear slow-down
- The Universal Machine simulates Normal Machines but is not itself a Normal Machine; it works on a richer alphabet comprising a few separators; moreover, each character can be "marked" with a boolean, for copying purposes.

The structure of the tape

The efficient way to simulate a machine with a single tape is to keep the program (as well as the current state) close to the head.

The tape has the following structure (q is a string of booleans!)

$$\alpha\#\langle q, c\rangle\# tuples \#\beta$$

where $\alpha c \beta$ is (morally) the tape of the emulated machine.

An emulation step consists in

- search among the tuples one matching $\langle q, c\rangle$;
- update the state-character pair
- execute the tape move

Library functions

We need a good library of functions for copying and comparing strings. Both rely on the use of (pairs of) marks to identify source and target positions:

- **mark** mark the current cell
- **clear_mark** clear the mark (if any) from the current cell
- **adv_mark_r** shift the mark one position to the right
- **adv_mark_l** shift the mark one position to the left
- **adv_both_marks** shift the marks at the right and left of the head one position to the right
- **match_and_advance f** if the current character satisfies the boolean test f then advance both marks and otherwise remove them
- **adv_to_mark_r** move the head to the next mark on the right
- **adv_to_mark_l** move the head to the next mark on the left

The main theorem

Every relation over tapes can be reflected into a corresponding relation on the low-level tape used by the Universal Machine.

theorem sem_universal2: ∀M:normalTM. ∀R.
M ||= R → universalTM ||= (low_R M (start ? M) R).

Moreover, if *M* terminate, then the simulation terminates too.

theorem terminate_UTM: ∀M:normalTM.∀t.
M ↓ t → universalTM ↓ (low_config M (mk_config ?? (start ? M) t)).

Proofs are long but not particularly complex.

Outline

Size and cost

name	dimension	content
mono.ma	475 lines	mono-tape Turing machines
if_machine.ma	335 lines	conditional composition
while_machine	166 lines	while composition
basic_machines.ma	282 lines	basic atomic machines
move_char.ma	310 lines	character copying
alphabet.ma	110 lines	alphabet of the universal machine
marks.ma	901 lines	operations exploiting marks
copy.ma	579 lines	string copy
normalTM.ma	319 lines	normal Turing machines
tuples.ma	276 lines	encoding of tuples
match_machines.ma	727 lines	machines implementing matching
move_tape.ma	778 lines	machines for moving the simulated tape
uni_step.ma	585 lines	emulation of a high-level step
universal.ma	394 lines	the universal machine
total	6237 lines	

Outline

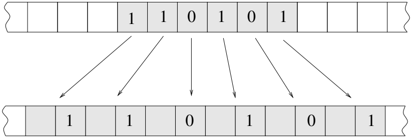
The cost of interpreting

Let us say that an interpeter is fair [3] if it simulates a program preserving (the order of) its complexity.

Is the previous interpreter fair?

Not so clear: booleans on the simulated tape are part of larger alphabet, and require a richer encoding. Sticking to a boolean alphabet, this means that each boolean must be "padded" into a small string of booleans.

This transformation may require a quadratic time on a single tape machine:



Rephrasing the problem

Is it possible to define a notion of pairing on single tape turing machines (in a categorical sense), in such a way that the diagonal function has linear complexity?

In general, is there a truly finitistic computational model admitting a *fair* interpereter?

Bibliography (1)

- S.O.Aandreaa. *On k-tape versus (k-1)-tape real time computation.* In R. Karp, editor, Complexity of Computation, pages 75–96. 1974.
- R. Amadio, A.Asperti, N.Ayache, B. Campbell, D. Mulligan, R. Pollack, Y.Régis-Gianas, C. Sacerdoti Coen, and I. Stark. *Certified complexity*. Procedia CS, 7:175–177, 2011.
- A.Asperti. *The intensional content of rice's theorem*. POPL08, San Francisco, California, USA, pp 113–119. ACM, 2008.
- A.Asperti, A.Ciabattoni. *Effective applicative structures*. CTCS '95, Cambridge, UK, LNCS 953, pages 81–95, 1995.
- A.Asperti, W.Ricciotti. *About the formalization of some results by Chebyshev in number theory.* TYPES'08, LNCS 5497, pp 19–31, 2009.
- A.Asperti, W.Ricciotti, C.Sacerdoti Coen, E.Tassi. *Formal metatheory of programming languages in the Matita interactive theorem prover*. Journal of Automated Reasoning: Special Issue on the Poplmark Challenge. Published online, May 2011.
- A.Asperti, W.Ricciotti, C.Sacerdoti Coen, E.Tassi. *The Matita interactive theorem prover*. CADE-2011, Wroclaw, Poland, LNCS 6803, 2011.

Bibliography (2)

- A.Asperti, W.Ricciotti, C.Sacerdoti Coen, E.Tassi. *A bi-directional refinement algorithm for the calculus of (co)inductive constructions*. LMCS 8(1), 2012.
- A.Asperti, W.Ricciotti, C.Sacerdoti Coen, E.Tassi. *A compact kernel for the Calculus of Inductive Constructions*. Sadhana 34(1):71–144, 2009.
- J. Hartmanis and R. E. Stearns. *On the computational complexity of algorithms.* Transaction of the AMS, 117:285–306, 1965.
- M. Li. *Simulating Two Pushdown Stores by One Tape in* $O(n^{1.5}\sqrt{logn})$.
- M.Norrish. *Mechanised computability theory*. Interactive Theorem Proving (ITP 2011), Berg en Dal, The Netherlands, August 22-25, 2011. LNCS 6898, pages 297–311, 2011.
- W.J.Paul. On-line simulation of k+1 tapes by k tapes requires nonlinear time. FOCS'82, Chicago, Illinois, USA, pages 53–56. IEEE Computer Society, 1982.
- W.J.Paul, J.I.Seiferas, J.Simon. *An information-theoretic approach to time bounds for on-line computation*. ACM Symposium on Theory of Computing, Los Angeles, California, USA, pp 357–367, 1980.
- M. O. Rabin. *Real time computation*. Israel Journal of Mathematics, 1:203–211, 1963.

Bibliography (3)

- C.Sacerdoti Coen, E.Tassi. *A constructive and formal proof of Lebesgue's dominated convergence theorem in the interactive theorem prover Matita.* Journal of Formalized Reasoning, 1:51–89, 2008.
- C.Sacerdoti Coen, E.Tassi. *Formalizing Overlap Algebras in Matita*. MSCS 21:1–31, 2011.
- C.Sacerdoti Coen, E.Tassi, S.Zacchiroli. *Tinycals: step by step tacticals*. UITP 2006, ENTCS 174, pp.125–142, 2006.
- M. Sipser.*Introduction to the Theory of Computation*. PWS, 1996.
- R. E. Stearns F. C. Hennie. *Two-tape simulation of multi tape turing machines*. Journal of ACM, 13(4):533–546, 1966.
- A. M. Turing. *On computable numbers, with an application to the entscheidungsproblem.* Proc. of the London Math. Society, 2(42):230–265, 1936.
- P.M.B.Vitányi. *An $n^{1.618}$ lower bound on the time to simulate one queue or two pushdown stores by one tape.* Inf. Process. Lett., 21(3):147–152, 1985.