Formalising Multi-Tape Turing Machines In Coq

Second Bachelor Seminar Talk

Maxi Wuttke

Saarland University

Programming Systems Lab

April 20, 2018

Advisor: Yannick Forster

Supervisor: Prof. Dr. Gert Smolka

Small Recap: Multi-Tape Turing Machines in Coq

- T_Σ: type of tapes
- TM_Σ^n: n-tape Turing machine with alphabet Σ
- $M : PTM_\Sigma^n(F)$: partitioned machines ($f : Q \to F$ for finite F)
- $M \models R$: Realisation
- ($M \downarrow S$: Termination)

[Asperti and Ricciotti (2015)]

Realisation

Definition (Realisation, $M \models R$)

Let $R \subseteq \mathsf{T}^n \times F \times \mathsf{T}^n$.

$$M \models R := \forall t\ t'\ q\ k.\ M(t) \triangleright^k (q, t') \rightarrow R\ t\ \left(f(q), t'\right)$$

Lemma (Monoticity of $\models$)

If $M \models R_1$ and $R_1 \subseteq R_2$, then $M \models R_2$.

Combinators for Parametrised Machines

Combinators for partitioned machines:

- If $M_1\ M_2\ M_3$
- $M_1;\ M_2$
- While M

Combinators for Parametrised Machines

Combinators for partitioned machines:

- If $M_1\ M_2\ M_3$
- $M_1;\ M_2$
- While M

Lemma (Correctness of While)

Let $R \subseteq T^n \times (\mathbb{B} \times F) \times T^n$.
If $M \models R$, *then*

$$\mathit{While}\ M \models \Big(\bigcup_{a:F} R|_{\mathbf{true};a}\Big)^* \circ R|_{\mathbf{false}}.$$

Machine Lifts

(Admissible) typing rule for sequential composition:

$$\frac{M_1 : \mathrm{PTM}^n_\Sigma(F_1) \qquad M_2 : \mathrm{PTM}^n_\Sigma(F_2)}{M_1;\ M_2 : \mathrm{PTM}^n_\Sigma(F_2)}$$

Problem:

When combining machines, the numbers of tapes and the alphabet have to match!

Machine Lifts

(Admissible) typing rule for sequential composition:

$$\frac{M_1 : \mathrm{PTM}^n_\Sigma(F_1) \qquad M_2 : \mathrm{PTM}^n_\Sigma(F_2)}{M_1;\ M_2 : \mathrm{PTM}^n_\Sigma(F_2)}$$

Problem:

When combining machines, the numbers of tapes and the alphabet have to match!

Solution: Two operations on machines:

- n-Lift: add/rearrange tapes
- Σ-Lift: add/translate symbols

n-Lift

$$\begin{aligned} i &: \mathsf{Fin}_m \hookrightarrow \mathsf{Fin}_n \\ M &: \mathsf{PTM}^m_\Sigma(F) \\ R &\subseteq \mathsf{T}^m \times F \times \mathsf{T}^m \end{aligned}$$

$$\begin{aligned} \Uparrow_i M &: \mathsf{PTM}^n_\Sigma(F) \\ \Uparrow_i R &\subseteq \mathsf{T}^n \times F \times \mathsf{T}^n \end{aligned}$$

$$\Uparrow_i R := \lambda t\ (a, t').R\ (i^{-1}\ t)\ (a, i^{-1}\ t')\ \wedge$$
$$\forall k \notin \mathsf{ran}\ i.\ t[k] = t'[k]$$

Lemma (Correctness of the n-Lift)

If $M \models R$, then $\Uparrow_i M \models \Uparrow_i R$.

Encoding Values on Tapes

- Type class for *encodable* types X and alphabets Σ:

```
Class encodable (X : Type) (sig : finType) := {
  encode : X -> list sig;
}.
```

Encoding Values on Tapes

- Type class for *encodable* types X and alphabets Σ:

```
Class encodable (X : Type) (sig : finType) := {
  encode : X -> list sig;
}.
```

- Define class of predicates for *value-containing* of tapes:

Definition (value-containing, $t \simeq x$)

Let X be encodable over Σ, and $t : \mathsf{T}_{\mathsf{Unit}+\Sigma}$.

$$t \simeq x := \exists r1.\ t = r_1\ (\mathsf{inl}\,())\ (\underset{\uparrow}{\mathsf{inr}\,y})\ (\mathsf{map}\ \ \mathsf{inr}\ \ ys)$$

for $\texttt{encode}(x) = y :: ys$.

A Relation for (Unary) Function Computation

Convention:

- Input tape(s): "callee-saved"
- Output tape: is initially right; grows to left
- Internal tapes: right before and after execution

A Relation for (Unary) Function Computation

Convention:

- Input tape(s): "callee-saved"
- Output tape: is initially right; grows to left
- Internal tapes: right before and after execution

Let X, Y be encodable over Σ.

Computes $(f : X \to Y) :=$

$\lambda t\ (\_, t').\forall (x : X).$

$t[0] \simeq x \to$

isRight $t[1] \to$

$(\forall i : \mathsf{Fin}_n.\ \mathsf{isRight}\ t[2 + i]) \to$

$t'[0] \simeq x \wedge$

$t'[1] \simeq f\ x \wedge$

$(\forall i : \mathsf{Fin}_n.\ \mathsf{isRight}\ t'[2 + i])$

A Relation for (Unary) Function Computation

Convention:

- Input tape(s): "callee-saved"
- Output tape: is initially right; grows to left
- Internal tapes: right before and after execution

Let X, Y be encodable over Σ.

$$\begin{aligned}
&\text{Computes } (f : X \to Y) := \\
&\quad \lambda t\ (_, t').\forall (x : X). \\
&\qquad t[0] \simeq x \to \\
&\qquad \text{isRight } t[1] \to \\
&\qquad (\forall i : \text{Fin}_n.\ \text{isRight } t[2+i]) \to \\
&\qquad t'[0] \simeq x \wedge \\
&\qquad t'[1] \simeq f\ x \wedge \\
&\qquad (\forall i : \text{Fin}_n.\ \text{isRight } t'[2+i])
\end{aligned}$$

$$\begin{aligned}
&\text{Computes2 } (f : X \to Y \to Z) := \\
&\quad \lambda t\ (_, t').\forall (x : X)\ (y : Y). \\
&\qquad t[0] \simeq x \to \\
&\qquad t[1] \simeq y \to \\
&\qquad \text{isRight } t[2] \to \\
&\qquad (\forall i : \text{Fin}_n.\ \text{isRight } t[3+i]) \to \\
&\qquad t'[0] \simeq x \wedge \\
&\qquad t'[1] \simeq y \wedge \\
&\qquad t'[1] \simeq f\ x\ y \wedge \\
&\qquad (\forall i : \text{Fin}_n.\ \text{isRight } t'[3+i])
\end{aligned}$$

Value-Manipulating Machines

- Match
- Constructor
- CopyValue
- MoveRight

Match $\mathbb{N}$

```
if (n--) {
  // ...
} else {
  // ...
}
```

If $(\Uparrow_{[i]} \text{MatchNat})\ M_1\ M_2$
(Where n is stored on tape i)

Match $\mathbb{N}$

```
if (n--) {
  // ...
} else {
  // ...
}
```

If $(\Uparrow_{[i]} \text{MatchNat})\ M_1\ M_2$
(Where n is stored on tape i)

$$
\begin{aligned}
&\text{Match\_Nat\_Rel} := \\
&\quad \lambda t\ (a, t').\forall (n : \mathbb{N}), \\
&\qquad t[0] \simeq n \rightarrow \\
&\qquad \text{match } n \text{ with} \\
&\qquad \mid 0 \Rightarrow t' = t\ \wedge\ a = \textbf{false} \\
&\qquad \mid S\ n' \Rightarrow t'[0] \simeq n'\ \wedge\ a = \textbf{true} \\
&\qquad \text{end}
\end{aligned}
$$

General Design of A Turing Machine

1. Write the program in pseudo code
2. Partition the program into smaller programs
3. For each partition: map variables to tapes
4. For each partition: write a machine using value-manipulating machines and combinators
5. Compose machines
6. For each machine: define a correctness relation and prove correctness

Case Study: Designing A Multiplication Machine

```
c := 0
while (m--) {
  c += n
}
return c
```

Case Study: Designing A Multiplication Machine

```
m' := m
c := 0
while (m'--) {
  Add(n, c, c')
  Reset(c)
  c := c'
  Reset(c')
}
Reset(m')
return c
```

Case Study: Designing A Multiplication Machine

```
m' := m
c := 0
while (true) {
  if (m'--) {
    Add(n, c, c')
    Reset(c)
    c := c'
    Reset(c')
    continue
  } else {
    break
  }
}
Reset(m')
return c
```

Case Study: Designing A Multiplication Machine

```
m' := m
c := 0
while (true) {
  if (m'--) {
    Add(n, c, c')
    Reset(c)
    c := c'
    Reset(c')
    continue
  } else {
    break
  }
}
Reset(m')
return c
```

Init

Step

Loop

Reset

Multiplication: Step Machine

We assume a machine Add $\models$ Computes2 *add*.

```
if (m'--) {
  Add(n, c, c')
  Reset(c)
  c := c'
  Reset(c')
  continue
} else {
  break
}
```

```
t0: m' (t0 for MatchNat)
t1: n  (t0 for Add)
t2: c  (t1 for Add) (t1 for CopyValue)
t3: c' (t2 for Add) (t0 for CopyValue)
t4: ?  (t3 for Add)
```

Multiplication: Step Machine

We assume a machine Add $\models$ Computes2 *add*.

```
if (m'--) {
  Add(n, c, c')
  Reset(c)
  c := c'
  Reset(c')
  continue
} else {
  break
}
```

Mult_Step :=
  If ($\Uparrow_{[0]}$ MatchNat)
  (Return(
    $\Uparrow_{[1;2;3;4]}$ Add;
    $\Uparrow_{[2]}$ MoveToRight;
    $\Uparrow_{[3;2]}$ CopyValue;
    $\Uparrow_{[3]}$ MoveToRight
) (**true**, ()))
  (Nop (**false**, ()))

```
t0: m' (t0 for MatchNat)
t1: n  (t0 for Add)
t2: c  (t1 for Add) (t1 for CopyValue)
t3: c' (t2 for Add) (t0 for CopyValue)
t4: ?  (t3 for Add)
```

Multiplication: Verification of Mult_Loop

Mult_Step_Rel :=

$\lambda t\ (a, t').\forall (c\ m'\ n : \mathbb{N}).$

$t[0] \simeq m' \rightarrow$

$t[1] \simeq n \rightarrow t[2] \simeq c \rightarrow$

isRight $t[3] \rightarrow$ isRight $t[4] \rightarrow$

match m' with

$\mid O \Rightarrow t' = t \ \wedge \ a = (\textbf{false},\ ())$

$\mid S\ m'' \Rightarrow$

$t'[0] \simeq m'' \wedge$

$t'[1] \simeq n \ \wedge \ t'[2] \simeq n + c \wedge$

isRight $t'[3] \ \wedge$ isRight $t'[4] \ \wedge \ a = (\textbf{true}, ())$

end

Mult_Loop_Rel :=

$\lambda t\ ((), t').\forall (c\ m'\ n : \mathbb{N}).$

$t[0] \simeq m' \rightarrow$

$t[1] \simeq n \rightarrow$

$t[2] \simeq c \rightarrow$

isRight $t[3] \rightarrow$

isRight $t[4] \rightarrow$

$t'[0] \simeq 0 \wedge$

$t'[1] \simeq n \wedge$

$t'[2] \simeq m' \cdot n + c \wedge$

isRight $t'[3] \wedge$

isRight $t'[4]$

Multiplication: Verification of Mult_Loop

Mult_Step_Rel :=

$\lambda t\ (a, t').\forall (c\ m'\ n : \mathbb{N}).$

$t[0] \simeq m' \rightarrow$

$t[1] \simeq n \rightarrow t[2] \simeq c \rightarrow$

isRight $t[3] \rightarrow$ isRight $t[4] \rightarrow$

match m' with

$\mid O \Rightarrow t' = t \ \wedge \ a = (\textbf{false},\ ())$

$\mid S\ m'' \Rightarrow$

$t'[0] \simeq m'' \wedge$

$t'[1] \simeq n \ \wedge \ t'[2] \simeq n + c \wedge$

isRight $t'[3] \ \wedge$ isRight $t'[4] \ \wedge \ a = (\textbf{true}, ())$

end

Lemma 1: Mult_Step $\models$ Mult_Step_Rel.

Mult_Loop_Rel :=

$\lambda t\ ((), t').\forall (c\ m'\ n : \mathbb{N}).$

$t[0] \simeq m' \rightarrow$

$t[1] \simeq n \rightarrow$

$t[2] \simeq c \rightarrow$

isRight $t[3] \rightarrow$

isRight $t[4] \rightarrow$

$t'[0] \simeq 0 \wedge$

$t'[1] \simeq n \wedge$

$t'[2] \simeq m' \cdot n + c \wedge$

isRight $t'[3] \wedge$

isRight $t'[4]$

Multiplication: Verification of Mult_Loop

Mult_Step_Rel :=

$\lambda t\ (a, t').\forall(c\ m'\ n : \mathbb{N}).$

$t[0] \simeq m' \rightarrow$

$t[1] \simeq n \rightarrow t[2] \simeq c \rightarrow$

isRight $t[3] \rightarrow$ isRight $t[4] \rightarrow$

match m' with

$\mid O \Rightarrow t' = t \ \wedge\ a = (\textbf{false},\ ())$

$\mid S\ m'' \Rightarrow$

$t'[0] \simeq m'' \wedge$

$t'[1] \simeq n \ \wedge\ t'[2] \simeq n + c \wedge$

isRight $t'[3] \ \wedge$ isRight $t'[4] \ \wedge\ a = (\textbf{true}, ())$

end

Mult_Loop_Rel :=

$\lambda t\ ((), t').\forall(c\ m'\ n : \mathbb{N}).$

$t[0] \simeq m' \rightarrow$

$t[1] \simeq n \rightarrow$

$t[2] \simeq c \rightarrow$

isRight $t[3] \rightarrow$

isRight $t[4] \rightarrow$

$t'[0] \simeq 0 \wedge$

$t'[1] \simeq n \wedge$

$t'[2] \simeq m' \cdot n + c \wedge$

isRight $t'[3] \wedge$

isRight $t'[4]$

Lemma 1: Mult_Step $\models$ Mult_Step_Rel.

Lemma 2: While Mult_Step $\models$ Mult_Loop_Rel.

Proof: It is enough to show $\left(\bigcup_{a:\mathsf{Unit}} \text{Mult\_Step\_Rel}|_{\textbf{true};a}\right)^* \circ \text{Mult\_Step\_Rel}|_{\textbf{false}} \subseteq \text{Mult\_Loop\_Rel}$, which can be shown by star induction.

1. Assume Mult_Step_Rel t $((\textbf{false}, ()), t')$; show Mult_Loop_Rel t $((), t')$.
2. Assume Mult_Step_Rel t $((\textbf{true}, ()), t')$ and Mult_Loop_Rel t' $((), t'')$; show Mult_Loop_Rel t $((), t'')$.

Multiplication: Putting Machines Together

$$
\begin{aligned}
\text{Mult\_Loop} &:= \text{While Mult\_Step} \\
\text{Mult} &:= \Uparrow_{[0;5]}\ \text{CopyValue;} \\
&\quad\ \Uparrow_{[2]}\ \text{O;} \\
&\quad\ \Uparrow_{[5;1;2;3;4]}\ \text{Mult\_Loop;} \\
&\quad\ \Uparrow_{[5]}\ \text{MoveToRight}
\end{aligned}
$$

Lemma (Correctness of Mult)

Mult $\models$ *Computes2 mult.*

Possible Future Work

- Implement UNIV
- Implement an interpreter for L

Possible Future Work

- Implement UNIV
- Implement an interpreter for L

Thank you!

Project home page:
`https://www.ps.uni-saarland.de/~wuttke/bachelor.php`

Related Work

Andrea Asperti, Wilmer Ricciotti

Formalizing Turing Machines

WoLLIC 2012

Xu, Jian and Zhang, Xingyuan and Urban, Christian

Mechanising Turing Machines and Computability Theory in Isabelle/HOL

ITP 2013

Andrea Asperti and Wilmer Ricciotti

A formalization of multi-tape Turing machines

Theoretical Computer Science, 2015

Alberto Ciaffaglione

Towards Turing computability via coinduction

Science of Computer Programming, 2016

Termination

Definition (Termination, $M \downarrow T$)

Let $T \subseteq \mathsf{T}^n \times \mathbb{N}$.

$$\mathbf{M} \downarrow \mathbf{T} := \forall t\ k.\ T\ t\ k \to \exists c_{out},\ M(t) \triangleright^k c_{out}$$

Lemma (Monoticity of $\downarrow$)

If $M \downarrow T_1$ and $T_2 \subseteq T_1$, then $M \downarrow T_2$.

Σ-Lift

Let $i : \Sigma \hookrightarrow \Gamma$ be an injection.

Let **default** : Σ^n.

Let $M : \mathrm{PTM}^n_\Sigma$ and $R \subseteq \mathrm{T}^n_\Sigma \times F \times \mathrm{T}^n_\Sigma$.

Then $\Uparrow_{i,s} M : \mathrm{MTM}^n_\Gamma$ and $\Uparrow_{i,s} R \subseteq \mathrm{T}^n_\Gamma \times F \times \mathrm{T}^n_\Gamma$ with

$$\Uparrow_{i;s} R := \lambda t\ (a, t').R\ (i_s^{-1} t)\ (a, i_s^{-1} t')$$

Lemma (Correctness of the Σ-Lift)

If $M \models R$, then $\Uparrow_{i;s} M \models \Uparrow_{i;s} R$.

Admissible Typing Rules for Combinators and Lifts

$$\frac{M_1 : \mathrm{PTM}^n_\Sigma(F_1) \qquad M_2 : \mathrm{PTM}^n_\Sigma(F_2)}{M_1;\ M_2 : \mathrm{PTM}^n_\Sigma(F_2)}$$

$$\frac{M_1 : \mathrm{PTM}^n_\Sigma(\mathbb{B}) \qquad M_2, M_3 : \mathrm{PTM}^n_\Sigma(F_2)}{\mathsf{If}\ M_1\ M_2\ M_3 : \mathrm{PTM}^n_\Sigma(F_2)}$$

$$\frac{M : \mathrm{PTM}^n_\Sigma(\mathbb{B} \times F)}{\mathsf{While}\ M : \mathrm{PTM}^n_\Sigma(F)}$$

$$\frac{M : \mathrm{PTM}^n_\Sigma(F_1) \qquad M' : F_1 \to \mathrm{PTM}^n_\Sigma(F_2)}{\mathsf{Match}\ M\ M' : \mathrm{PTM}^n_\Sigma(F_2)}$$

$$\frac{i : \mathrm{Fin}_m \hookrightarrow \mathrm{Fin}_n \qquad M : \mathrm{PTM}^m_\Sigma(F)}{\Uparrow_i M : \mathrm{PTM}^n_\Sigma(F)}$$

$$\frac{i : \Sigma \hookrightarrow \Gamma \qquad s : \Sigma^n \qquad M : \mathrm{PTM}^n_\Sigma(F)}{\Uparrow_{i;s} M : \mathrm{PTM}^n_\Gamma(F)}$$

Coq implementation

	Lines Spec	Lines Proof
Prelim (`Fin`, `Vector`, `finType`, etc.)	2183	2644
Relations	179	168
TM, $\models$, $\downarrow$	522	255
Σ-Lift	106	330
n-Lift	308	247
Combinators	309	405
Basic machines	235	112
Computation	382	292
Match $\mathbb{N}$	45	88
CopyValue	509	608
Add, Mult (wip)	183	321
Σ	4961	5470