Verified Programming of Turing Machines In Coq

Final Bachelor Talk

Maxi Wuttke

Saarland University

Programming Systems Lab

September 14, 2018

Advisor: Yannick Forster

Supervisor: Prof. Dr. Gert Smolka

Motivation

- Turing machines build traditional foundation of the theory of computation and complexity
- simple (but not quite simplistic)
- many different models
- usually not formally verified

Overview: Abstraction Levels

- **Level 0:** multi-tape Turing machines
  - unstructured; non-compositional; huge amount of states; low-level ☹

Overview: Abstraction Levels

- **Level 0:** multi-tape Turing machines
  - unstructured; non-compositional; huge amount of states; low-level ☹
- **Level 1:** labelled Turing machines
  - semantical predicates for correctness & time complexity
  - "label" all internal states

Overview: Abstraction Levels

- **Level 0:** multi-tape Turing machines
  - unstructured; non-compositional; huge amount of states; low-level ☹
- **Level 1:** labelled Turing machines
  - semantical predicates for correctness & time complexity
  - “label” all internal states
- **Level 2:** control-flow & lifting operators
  - imperative language; structured; compositional
  - no need to refer to internal states

Overview: Abstraction Levels

- **Level 0:** multi-tape Turing machines
  - unstructured; non-compositional; huge amount of states; low-level ☹
- **Level 1:** labelled Turing machines
  - semantical predicates for correctness & time complexity
  - "label" all internal states
- **Level 2:** control-flow & lifting operators
  - imperative language; structured; compositional
  - no need to refer to internal states
- **Level 3:** generalised register machine
  - imperative language with values
  - use each tape as a register for an arbitrary encodable type

Overview: Abstraction Levels

- **Level 0:** multi-tape Turing machines
  - unstructured; non-compositional; huge amount of states; low-level ☹
- **Level 1:** labelled Turing machines
  - semantical predicates for correctness & time complexity
  - "label" all internal states
- **Level 2:** control-flow & lifting operators
  - imperative language; structured; compositional
  - no need to refer to internal states
- **Level 3:** generalised register machine
  - imperative language with values
  - use each tape as a register for an arbitrary encodable type
- **Level 4:** call-by-value λ-calculus
  - functional language ☺

Level 0: Multi-Tape Turing Machines

- Tape_Σ: Type of tapes over alphabet Σ

[A. Asperti, W. Ricciotti, 2015]

Level 0: Multi-Tape Turing Machines

- Tape_Σ: Type of tapes over alphabet Σ
- TM^n_Σ: Type of n-tape Turing machines over finite alphabet Σ
  - finite type of states Q
  - initial state $init : Q$
  - final states $halt : Q \to \mathbb{B}$
  - transition function $\delta : Q \times (\mathcal{O}(\Sigma))^n \to Q \times (\mathcal{O}(\Sigma) \times \mathsf{Move})^n$

[A. Asperti, W. Ricciotti, 2015]

Level 0: Multi-Tape Turing Machines

- Tape_Σ: Type of tapes over alphabet Σ
- TM^n_Σ: Type of n-tape Turing machines over finite alphabet Σ
  - finite type of states Q
  - initial state $init : Q$
  - final states $halt : Q \to \mathbb{B}$
  - transition function $\delta : Q \times (\mathcal{O}(\Sigma))^n \to Q \times (\mathcal{O}(\Sigma) \times \mathsf{Move})^n$
- $M(t) \triangleright^k (q, t')$: M terminates in k steps in the configuration (q, t'), given the input tapes t

[A. Asperti, W. Ricciotti, 2015]

Level 1: Labelled Turing Machines

- $M : \mathsf{TM}^n_\Sigma(L)$: pair of a machine and a state labelling function:

$$(M' : \mathsf{TM}^n_\Sigma,\ lab : Q_{M'} \to L)$$

Level 1: Labelled Turing Machines

- $M : \mathsf{TM}_\Sigma^n(L)$: pair of a machine and a state labelling function:

$$(M' : \mathsf{TM}_\Sigma^n,\ lab : Q_{M'} \to L)$$

- **Correctness predicate:** Let $M : \mathsf{TM}_\Sigma^n(L)$ and $R \subseteq \mathsf{Tape}_\Sigma^n \times (L \times \mathsf{Tape}_\Sigma^n)$.

$$M \vDash R \ := \ \forall t\ q\ t'.\ M(t) \triangleright^* (q, t') \to R\ t\ (lab_M\ q, t')$$

Level 1: Labelled Turing Machines

- $M : \mathsf{TM}^n_\Sigma(L)$: pair of a machine and a state labelling function:

$$(M' : \mathsf{TM}^n_\Sigma,\ lab : Q_{M'} \to L)$$

- **Correctness predicate:** Let $M : \mathsf{TM}^n_\Sigma(L)$ and $R \subseteq \mathsf{Tape}^n_\Sigma \times (L \times \mathsf{Tape}^n_\Sigma)$.

$$M \vDash R \ := \ \forall t\ q\ t'.\ M(t) \triangleright^* (q, t') \to R\ t\ (lab_M\ q, t')$$

Lemma (Monotonicity)

If $M \vDash R'$ and $R' \subseteq R$, then $M \vDash R$.

Level 1: Labelled Turing Machines

- $M : \mathsf{TM}^n_\Sigma(L)$: pair of a machine and a state labelling function:

$$(M' : \mathsf{TM}^n_\Sigma,\ lab : Q_{M'} \to L)$$

- **Correctness predicate:** Let $M : \mathsf{TM}^n_\Sigma(L)$ and $R \subseteq \mathsf{Tape}^n_\Sigma \times (L \times \mathsf{Tape}^n_\Sigma)$.

$$M \vDash R \ := \ \forall t\ q\ t'.\ M(t) \triangleright^* (q, t') \to R\ t\ (lab_M\ q, t')$$

Lemma (Monotonicity)

If $M \vDash R'$ and $R' \subseteq R$, then $M \vDash R$.

(There is also a notion for running time)

Primitive Machines

Machines that terminate after 0 or 1 transitions, e.g.:

- $\mathsf{Write}(s) : \mathsf{TM}^1_\Sigma(1)$, s.t.
  $\mathsf{Write}(s) \vDash (\lambda t\ (\_, t').\ t'[0] = \mathsf{wr}\ (t'[0])\ s)$
- $\mathsf{Read} : \mathsf{TM}^1_\Sigma(\mathcal{O}(\Sigma))$, s.t.
  $\mathsf{Read} \vDash (\lambda t\ (\ell, t').\ t' = t\ \wedge\ \ell = \mathsf{current}(t[0]))$

Level 2: Control-Flow

Sequential composition:

Let $M_1 : \mathsf{TM}^n_\Sigma(L_1)$ and $M_2 : \mathsf{TM}^n_\Sigma(L_2)$, then $M_1; M_2 : \mathsf{TM}^n_\Sigma(L_2)$.

Level 2: Control-Flow

Sequential composition:

Let $M_1 : \mathrm{TM}^n_\Sigma(L_1)$ and $M_2 : \mathrm{TM}^n_\Sigma(L_2)$, then $M_1; M_2 : \mathrm{TM}^n_\Sigma(L_2)$.

Lemma

If $M_1 \vDash R_1$ and $M_2 \vDash R_2$, then with $R|_y := \lambda x\ z.\ R\ x\ (y, z)$

$$M_1; M_2 \vDash \bigcup_{\ell : L_1} (R_1|_\ell \circ R_2)$$

Level 2: Control-Flow

Sequential composition:

Let $M_1 : \mathsf{TM}^n_\Sigma(L_1)$ and $M_2 : \mathsf{TM}^n_\Sigma(L_2)$, then $M_1; M_2 : \mathsf{TM}^n_\Sigma(L_2)$.

Lemma

If $M_1 \vDash R_1$ and $M_2 \vDash R_2$, then *with $R|_y := \lambda x\ z.\ R\ x\ (y, z)$*

$$M_1; M_2 \vDash \bigcup_{\ell : L_1} (R_1|_\ell \circ R_2)$$

Conditional:

Let $M_1 : \mathsf{TM}^n_\Sigma(\mathbb{B})$ and $M_2, M_3 : \mathsf{TM}^n_\Sigma(L)$, then
If M_1 Then M_2 Else $M_3 : \mathsf{TM}^n_\Sigma(L)$.

Level 2: Control-Flow

Sequential composition:

Let $M_1 : \mathsf{TM}^n_\Sigma(L_1)$ and $M_2 : \mathsf{TM}^n_\Sigma(L_2)$, then $M_1; M_2 : \mathsf{TM}^n_\Sigma(L_2)$.

Lemma

If $M_1 \vDash R_1$ and $M_2 \vDash R_2$, then *with $R|_y := \lambda x\ z.\ R\ x\ (y, z)$*

$$M_1; M_2 \vDash \bigcup_{\ell : L_1} (R_1|_\ell \circ R_2)$$

Conditional:

Let $M_1 : \mathsf{TM}^n_\Sigma(\mathbb{B})$ and $M_2, M_3 : \mathsf{TM}^n_\Sigma(L)$, then
If M_1 Then M_2 Else $M_3 : \mathsf{TM}^n_\Sigma(L)$.

Lemma

If $M_1 \vDash R_1$, $M_2 \vDash R_2$, and $M_3 \vDash R_3$, then

$$\text{If } M_1 \text{ Then } M_2 \text{ Else } M_3 \vDash (R_1|_{\mathsf{true}} \circ R_2) \cup (R_1|_{\mathsf{false}} \circ R_3)$$

Level 2: Control-Flow

"Do-While" Loop:
Let $M : \mathsf{TM}^n_\Sigma(\mathcal{O}(L))$, then While $M : \mathsf{TM}^n_\Sigma(L)$.

Level 2: Control-Flow

"Do-While" Loop:
Let $M : \mathsf{TM}^n_\Sigma(\mathcal{O}(L))$, then While $M : \mathsf{TM}^n_\Sigma(L)$.

Lemma (Correctness of While M)

If $M \vDash R$, then While $M \vDash \mathit{WhileRel}\ R$*, which is defined inductively:*

$$\frac{R\ t\ (\lfloor \ell \rfloor, t')}{\mathit{WhileRel}\ R\ t\ (\ell, t')} \qquad \frac{R\ t\ (\emptyset, t') \qquad \mathit{WhileRel}\ R\ t'\ (\ell, t'')}{\mathit{WhileRel}\ R\ t\ (\ell, t'')}$$

Level 2: Lifting

- Problem: How to combine machines with different number of tapes and alphabets?

$$\frac{M_1 : \mathsf{TM}^{n}_{\Sigma}(L_1) \qquad M_2 : \mathsf{TM}^{n}_{\Sigma}(L_2)}{M_1; M_2 : \mathsf{TM}^{n}_{\Sigma}(L_2)}$$

Level 2: Lifting

- Problem: How to combine machines with different number of tapes and alphabets?

$$\frac{M_1 : \mathrm{TM}_{\Sigma}^{n}(L_1) \qquad M_2 : \mathrm{TM}_{\Sigma}^{n}(L_2)}{M_1; M_2 : \mathrm{TM}_{\Sigma}^{n}(L_2)}$$

- Solution: Two **lifting** operators:
  - Tapes-lift (increase the number of tapes)
  - Alphabet-lift (increase the alphabet)

Level 2: Lifting

- Problem: How to combine machines with different number of tapes and alphabets?

$$\frac{M_1 : \mathrm{TM}_{\Sigma}^{n}(L_1) \qquad M_2 : \mathrm{TM}_{\Sigma}^{n}(L_2)}{M_1 ; M_2 : \mathrm{TM}_{\Sigma}^{n}(L_2)}$$

- Solution: Two **lifting** operators:
  - Tapes-lift (increase the number of tapes)
  - Alphabet-lift (increase the alphabet)
  - Lift all sub-machines to the same number of tapes and alphabet, before applying the control-flow operators

Tapes-lift

- Let $M : \mathsf{TM}_\Sigma^m(L)$ and $I : \mathbb{F}_m \hookrightarrow \mathbb{F}_n$.
- Then $\Uparrow_I M : \mathsf{TM}_\Sigma^n$.

Tapes-lift

- Let $M : \mathsf{TM}^m_\Sigma(L)$ and $I : \mathbb{F}_m \hookrightarrow \mathbb{F}_n$.
- Then $\Uparrow_I M : \mathsf{TM}^n_\Sigma$.

Lemma

If $M \vDash R$, then $\Uparrow_I M \vDash \Uparrow_I R$ with

$$\Uparrow_I R \ := \ \lambda t\ (I, t').\ R\ (I^{-1}\ t)\ (I, I^{-1}\ t') \wedge \\ (\forall i \notin \operatorname{img} I.\ t'[i] = t[i])$$

Level 3: Encoding Values

- Idea: think of tapes as **registers**, that
  - may *contain* a value of an arbitrary encodable type,
  - or contain no value

Level 3: Encoding Values

- Idea: think of tapes as **registers**, that
  - may *contain* a value of an arbitrary encodable type,
  - or contain no value
- A type X is encodable on Σ, if there is a function $\mathit{encode} : X \to \mathcal{L}(\Sigma)$.
  - For example, $\mathit{encode}(n) := \mathsf{S}^n +\!\!+ [\mathsf{O}]$

Level 3: Encoding Values

- Idea: think of tapes as **registers**, that
  - may *contain* a value of an arbitrary encodable type,
  - or contain no value
- A type X is encodable on Σ, if there is a function $encode : X \to \mathcal{L}(\Sigma)$.
  - For example, $encode(n) := \mathsf{S}^n +\!\!+ [\mathsf{O}]$
  - Obvious problem: ambiguity

Level 3: Encoding Values

- Idea: think of tapes as **registers**, that
  - may *contain* a value of an arbitrary encodable type,
  - or contain no value
- A type X is encodable on Σ, if there is a function $encode : X \to \mathcal{L}(\Sigma)$.
  - For example, $encode(n) := \mathsf{S}^n +\!\!+ [\mathsf{O}]$
  - Obvious problem: ambiguity
- Inductive types Σ_X to *minimally encode* X on, e.g.
  - $\Sigma_{\mathbb{N}} ::= \mathsf{S} \mid \mathsf{O}$
  - $\Sigma_{X+Y} ::= \mathsf{INL} \mid \mathsf{INR} \mid (x : \Sigma_X) \mid (y : \Sigma_Y)$
    (if X is minimally encodable on Σ_X and Y on Σ_Y)

Level 3: Value-Containment

Let X be encodable on Σ.

Definition (Σ^+)

Σ^+ ::= START | STOP | UNKNOWN | $(x : \Sigma_X)$

Definition (tape-containment)

Let $t : \text{Tape}_{\Sigma^+}$ and $x : X$.

$$t \simeq x \ := \ \exists ls.\ t = (ls\ \underset{\uparrow}{\text{START}}\ encode(x)\ \text{STOP})$$

Level 3: Value-Containment

Let X be encodable on Σ.

Definition (Σ^+)

$\Sigma^+ ::= \mathsf{START} \mid \mathsf{STOP} \mid \mathsf{UNKNOWN} \mid (x : \Sigma_X)$

Definition (tape-containment)

Let $t : \mathsf{Tape}_{\Sigma^+}$ and $x : X$.

$$t \simeq x \;:=\; \exists ls.\ t = (ls\ \underset{\uparrow}{\mathsf{START}}\ encode(x)\ \mathsf{STOP})$$

We write $t \simeq_f x$ if X is minimally encodable on Σ_X and $f : \Sigma_X \hookrightarrow \Sigma$.

Level 3: Value-Containment

Let X be encodable on Σ.

Definition (Σ^+)

$\Sigma^+ ::= \mathsf{START} \mid \mathsf{STOP} \mid \mathsf{UNKNOWN} \mid (x : \Sigma_X)$

Definition (tape-containment)

Let $t : \mathsf{Tape}_{\Sigma^+}$ and $x : X$.

$$t \simeq x \;:=\; \exists ls.\; t = (ls\ \underset{\uparrow}{\mathsf{START}}\ \mathit{encode}(x)\ \mathsf{STOP})$$

We write $t \simeq_f x$ if X is minimally encodable on Σ_X and $f : \Sigma_X \hookrightarrow \Sigma$.

Definition (right tape)

$$\mathit{isRight}(t) \;:=\; \exists s\ ls.\; t = (ls\ \underset{\uparrow}{s})$$

Level 3: Value-Manipulating Machines

Level 3: Value-Manipulating Machines

Auxiliary Machines

- Reset : $\mathsf{TM}^1_\Sigma(1)$, s.t. Reset $\vDash (\lambda t\ (\_, t').\ \forall x.\ t[0] \simeq x \rightarrow \mathit{isRight}\ t'[0])$

Level 3: Value-Manipulating Machines

Auxiliary Machines

- Reset : $\mathsf{TM}^1_\Sigma(1)$, s.t. Reset $\vDash (\lambda t\ (\_, t').\ \forall x.\ t[0] \simeq x \rightarrow \mathit{isRight}\ t'[0])$
- Copy : $\mathsf{TM}^2_\Sigma(1)$, s.t.
  Copy $\vDash (\lambda t\ (\_, t').\ \forall x.\ t[0] \simeq x \rightarrow \mathit{isRight}\ t[1] \rightarrow t'[0] \simeq x \land t'[1] \simeq x)$

Level 3: Value-Manipulating Machines

Auxiliary Machines

- Reset : $\mathsf{TM}^1_\Sigma(1)$, s.t. Reset $\vDash (\lambda t\ (\_, t').\ \forall x.\ t[0] \simeq x \to \mathit{isRight}\ t'[0])$
- Copy : $\mathsf{TM}^2_\Sigma(1)$, s.t.
  Copy $\vDash (\lambda t\ (\_, t').\ \forall x.\ t[0] \simeq x \to \mathit{isRight}\ t[1] \to t'[0] \simeq x \land t'[1] \simeq x)$
- Translate $f_1\ f_2$: $\mathsf{TM}^1_\Sigma(1)$ for $f_1, f_2 : \Sigma_X \hookrightarrow \Sigma$, s.t.
  Translate $f_1\ f_2 \vDash \big(\lambda t\ (\_, t').\ \forall x.\ t[0] \simeq_{f_1} x \to t'[0] \simeq_{f_2} x\big)$

Level 3: Value-Manipulating Machines

Auxiliary Machines

- Reset : $\mathsf{TM}^1_\Sigma(1)$, s.t. $\mathsf{Reset} \vDash (\lambda t\ (\_, t').\ \forall x.\ t[0] \simeq x \to \mathit{isRight}\ t'[0])$
- Copy : $\mathsf{TM}^2_\Sigma(1)$, s.t.
  $\mathsf{Copy} \vDash (\lambda t\ (\_, t').\ \forall x.\ t[0] \simeq x \to \mathit{isRight}\ t[1] \to t'[0] \simeq x \land t'[1] \simeq x)$
- Translate $f_1\ f_2$: $\mathsf{TM}^1_\Sigma(1)$ for $f_1, f_2 : \Sigma_X \hookrightarrow \Sigma$, s.t.
  $\mathsf{Translate}\ f_1\ f_2 \vDash \left(\lambda t\ (\_, t').\ \forall x.\ t[0] \simeq_{f_1} x \to t'[0] \simeq_{f_2} x\right)$

Constructors & Deconstructors

- $\mathsf{ConstrO} \vDash (\lambda t\ (\_, t').\ \mathit{isRight}\ t[0] \to t'[0] \simeq 0)$
- $\mathsf{ConstrS} \vDash (\lambda t\ (\_, t').\ \forall n.\ t[0] \simeq n \to t'[0] \simeq S\ n)$
- CaseNat : $\mathsf{TM}^1_{\Sigma_\mathbb{N}}(\mathbb{B})$

Towards A Call-By-Value λ-Calculus Interpreter

- Goal: build a Turing machine that simulates the weak call-by-value λ-calculus with De Bruijn terms (aka. "L")

[F. Kunze, Y. Forster, G. Smolka, 2018]

Towards A Call-By-Value λ-Calculus Interpreter

- Goal: build a Turing machine that simulates the weak call-by-value λ-calculus with De Bruijn terms (aka. "*L*")
- Intermediate abstract machine: instead of β-substitution, manage *closures* (variable bindings & term)

[F. Kunze, Y. Forster, G. Smolka, 2018]

Towards A Call-By-Value λ-Calculus Interpreter

- Goal: build a Turing machine that simulates the weak call-by-value λ-calculus with De Bruijn terms (aka. "L")
- Intermediate abstract machine: instead of β-substitution, manage *closures* (variable bindings & term)
- Bindings: implemented as a linked list of closures ("heap")

$$
\begin{aligned}
Com &::= \mathsf{VAR}(n : \mathbb{N}) \mid \mathsf{APP} \mid \mathsf{LAM} \mid \mathsf{RET} \\
Pro &:= \mathcal{L}(Com) \\
Clos &:= \mathbb{N} \times Pro \\
Heap &:= \mathcal{L}\,(\mathcal{O}(Clos \times \mathbb{N}))
\end{aligned}
$$

[F. Kunze, Y. Forster, G. Smolka, 2018]

Towards A Call-By-Value λ-Calculus Interpreter

- Goal: build a Turing machine that simulates the weak call-by-value λ-calculus with De Bruijn terms (aka. "L")
- Intermediate abstract machine: instead of β-substitution, manage *closures* (variable bindings & term)
- Bindings: implemented as a linked list of closures ("heap")

$$\begin{aligned} Com &::= \mathrm{VAR}(n : \mathbb{N}) \mid \mathrm{APP} \mid \mathrm{LAM} \mid \mathrm{RET} \\ Pro &:= \mathcal{L}(Com) \\ Clos &:= \mathbb{N} \times Pro \\ Heap &:= \mathcal{L}\,(\mathcal{O}(Clos \times \mathbb{N})) \end{aligned}$$

- Configurations (T, V, H): *control closure* stack T, *argument closure* stack V, heap H

[F. Kunze, Y. Forster, G. Smolka, 2018]

Heap Machine: APP Rule

If the first control closure is $(a, \text{APP} :: P)$:

Heap Machine: APP Rule

If the first control closure is $(a, \text{APP} :: P)$:

- pop two closures from the argument stack: g and (b, Q)

Heap Machine: APP Rule

If the first control closure is $(a, \mathrm{APP} :: P)$:

- pop two closures from the argument stack: g and (b, Q)
- push new closure (g, b) to the heap

Heap Machine: APP Rule

If the first control closure is $(a, \text{APP} :: P)$:

- pop two closures from the argument stack: g and (b, Q)
- push new closure (g, b) to the heap
- push (c, Q) to the control stack (c is the address to the new heap entry)

Heap Machine Simulator: Step

Step : $\mathrm{TM}^{11}_{\Sigma^+}(\mathcal{O}(1))$ simulates single steps of heap machines:

Lemma (Correctness of Step)

Step $\vDash$ *StepRel with*

$$
\begin{aligned}
&\mathit{StepRel} := \\
&\quad \lambda t\ (l, t').\ \forall T\ V\ H.\ t[0] \simeq T \to t[1] \simeq V \to t[2] \simeq H \to (\forall (i : \mathbb{F}_8).\ \mathit{isRight}\ t[3+i]) \to \\
&\qquad \text{if } l = \emptyset \text{ then } \exists T'\ V'\ H'.\ (T, V, H) \succ (T', V', H')\ \wedge \\
&\qquad\qquad\qquad t'[0] \simeq T' \wedge t'[1] \simeq V' \wedge t'[2] \simeq H' \wedge \Big(\forall (i : \mathbb{F}_8).\ \mathit{isRight}\ t'[3+i]\Big) \\
&\qquad\qquad \text{else } \ \mathit{halt}(T, V, H)
\end{aligned}
$$

Heap Machine Simulator: Step

Step : $\mathrm{TM}^{11}_{\Sigma^+}(\mathcal{O}(1))$ simulates single steps of heap machines:

Lemma (Correctness of Step)

Step $\vDash$ *StepRel with*

$$
\begin{aligned}
&\mathit{StepRel} := \\
&\quad \lambda t\ (l, t').\ \forall T\ V\ H.\ t[0] \simeq T \to t[1] \simeq V \to t[2] \simeq H \to (\forall (i : \mathbb{F}_8).\ \mathit{isRight}\ t[3+i]) \to \\
&\qquad \text{if } l = \emptyset \text{ then } \exists T'\ V'\ H'.\ (T, V, H) \succ (T', V', H')\ \wedge \\
&\qquad\qquad\qquad t'[0] \simeq T' \wedge t'[1] \simeq V' \wedge t'[2] \simeq H' \wedge \left(\forall (i : \mathbb{F}_8).\ \mathit{isRight}\ t'[3+i]\right) \\
&\qquad\qquad \text{else } \ \mathit{halt}(T, V, H)
\end{aligned}
$$

(We also have a running time relation for Step.)

Heap Machine Simulator: Loop

Define Loop := While Step.

Lemma (Correctness of Loop)

Loop $\vDash$ *LoopRel with*

$$\mathit{LoopRel} := \lambda t\ (\_, t').\ \forall T\ V\ H.\ t[0] \simeq T \to t[1] \simeq V \to t[2] \simeq H \to (\forall (i : \mathbb{F}_8).\ \mathit{isRight}\ t[3+i]) \to \exists T'\ V'\ H'.\ (T, V, H) \rhd^* (T', V', H')$$

Heap Machine Simulator: Loop

Define Loop := While Step.

Lemma (Correctness of Loop)

Loop ⊨ *LoopRel with*

$$\mathit{LoopRel} := \lambda t\ (\_, t').\ \forall T\ V\ H.\ t[0] \simeq T \to t[1] \simeq V \to t[2] \simeq H \to (\forall (i : \mathbb{F}_8).\ \mathit{isRight}\ t[3+i]) \to \exists T'\ V'\ H'.\ (T, V, H) \rhd^* (T', V', H')$$

(We also have a running time relation for Loop.)

Heap Machine: Halting Problem

Theorem (Halting problem reduction)

The halting problem of heap machines reduces to the halting problem of multi-tape Turing machines.

Conclusion

- We have a framework for programming and verifying multi-tape Turing machines in Coq
- We made programming structural and compositional
- The notion of value-containment gives the advantages of register machines (but we are not restricted to natural numbers)
- As a case-study, we programmed a simulator for the heap machine

Related Work

- A. Asperti, W. Ricciotti
  Formalizing Turing Machines
  WoLLIC 2012
- A. Asperti, W. Ricciotti
  A formalization of multi-tape Turing machines
  Theoretical Computer Science, 2015
- Xu, Jian and Zhang, Xingyuan and Urban, Christian
  Mechanising Turing Machines and Computability Theory in Isabelle/HOL
  ITP 2013
- Alberto Ciaffaglione
  Towards Turing computability via coinduction
  Science of Computer Programming, 2016
- F. Kunze, Y. Forster, G. Smolka
  Formal Small-step Verification of a Call-by-value Lambda Calculus Machine
  arXiv preprint, 2018

Future Work

- Show that the running time function of Loop is polynomial in the size of the encoding of the initial state
- Enrich correctness relations with commitments over space-usage
- Formalise reduction from multi-tape Turing machines to single-tape Turing machines and to Turing machines with binary alphabet

Future Work

- Show that the running time function of Loop is polynomial in the size of the encoding of the initial state
- Enrich correctness relations with commitments over space-usage
- Formalise reduction from multi-tape Turing machines to single-tape Turing machines and to Turing machines with binary alphabet

Thank you!

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

Code Complexity 1

Module	Spec	Proof
Preliminary (incl. loop and relations)	176	84
Definition of Turing machines	430	194
Primitive Machines	122	34
Control-flow operators	425	383
Lifting	362	193
Simple Machines	380	278
Value containment	394	119
Copying and writing values	411	288
Alphabet-Lift with values	133	147
Deconstructors and constructors	486	482
Notations and tactics for compound or programmed machines	165	15
MapSum	47	110
Addition and Multiplication machines	181	298
List functions machines	326	456
Heap machine simulator	981	1040
Total	5019	4121

Code Complexity 2

Library code lines: 153 spec and 2638 proof.

- discrete & finite types
- retractions (injective function with partial inverse function)
- inhabited types

Loop has:

- 30 symbols
- 11537 states

A Relational Notion For Time Complexity

Let $M : \mathsf{TM}^n_\Sigma$ and $T \subseteq \mathsf{Tape}^n_\Sigma \times \mathbb{N}$.

$$M \downarrow T \;\; := \;\; \forall t\ k.\ T\ t\ k \to \exists c.\ M(t) \triangleright^k c$$

A Relational Notion For Time Complexity

Let $M : \mathsf{TM}^n_\Sigma$ and $T \subseteq \mathsf{Tape}^n_\Sigma \times \mathbb{N}$.

$$M \downarrow T \;:=\; \forall t\ k.\ T\ t\ k \to \exists c.\ M(t) \triangleright^k c$$

Lemma (Anti-monotonicity)

If $M \downarrow T'$ and $T \subseteq T'$, then $M \downarrow T$.

Some Running Time Relations

Lemma (Running time of $M_1; M_2$)

If $M_1 \vDash R_1$, $M_1 \downarrow T_1$, *and* $M_2 \downarrow T_2$, *then*

$$M_1; M_2 \downarrow (\lambda t\ k.\ \exists k_1\ k_2.\ T_1\ t\ k_1\ \wedge\ 1 + k_1 + k_2 \leq k\ \wedge\ \forall t'\ \ell.\ R_1\ t\ (\ell, t') \rightarrow T_2\ t'\ k_2)$$

Lemma (Running time of While M)

If $M \vDash R$ *and* $M \downarrow T$, *then* While $M \downarrow \mathit{WhileT}\ R\ T$, *which is defined co-inductively:*

$$\frac{T\ t\ k_1 \qquad \forall \ell\ t'.\ R\ t\ (\lfloor \ell \rfloor, t') \rightarrow k_1 \leq k \qquad \forall t'.\ R\ t\ (\emptyset, t') \rightarrow \exists k_2.\ \mathit{WhileT}\ R\ T\ t'\ k_2 \wedge 1 + k_1 + k_2 \leq k}{\mathit{WhileT}\ R\ T\ t\ k}$$