Verified Programming of Turing Machines in Coq

Yannick Forster　Fabian Kunze　Maximilian Wuttke

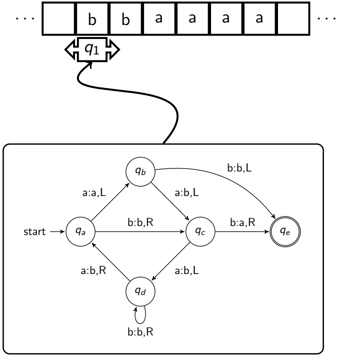
Saarland University, Saarland Informatics Campus

January 20
CPP 2020

Why Turing Machines?

Traditional basis of theory of computation & complexity:

- Simple model of computation
- Straightforward notion of time & space consumption



Finite Control

Turing machines are difficult

Hard to construct and verify:

- Not inherently compositional:
  - States & transition function
  - Number of tapes
  - Different alphabets
- Data not structured: strings over alphabet

Turing machines are difficult

Hard to construct and verify:

- Not inherently compositional:
  - States & transition function
  - Number of tapes
  - Different alphabets
- Data not structured: strings over alphabet

Therefore no mechanisation of complexity-theoretic results available like:

- $\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$
- Cook-Levin theorem: SAT is NP complete
- Time & space hierarchy theorem

Our vision: Complexity theory in call-by-value λ-Calculus L

Why use L?

- Tools for verification and running time analysis in L (Forster&Kunze (2019))

L induces same notion of computability:

- **Church-Turing thesis**: L and TMs can simulate each other

Complexity theory possible?

- **Invariance thesis** justifies this: L and TMs can simulate each other with polynomial time overhead and constant factor space overhead.
  - Proof: *The Weak Call-By-Value λ-Calculus is Reasonable for Both Time and Space* POPL Wednesday 10:30, Forster&Kunze&Roth (2020) (non-mechanised)

Mechanisation of the POPL result will require *resource-aware* verification of TMs[1]

---

[1] Church-Turing already mechanised using the results of this paper

Related Work: Mechanised verification of Turing machines

Mechanised universal Turing machines (correctness & termination):

- In Matita: Asperti&Ricciotti (2013,2015)
- In Isabelle/HOL, using a Hoare logic: Xu&Zhang&Urban (2013)

We extend Asperti&Ricciotti's verification approach with time & space analysis, in Coq

Contribution

A framework to:

- Construct Turing machines
- Verify correctness & termination
- Verify/deduce time and space complexity

Case studies:

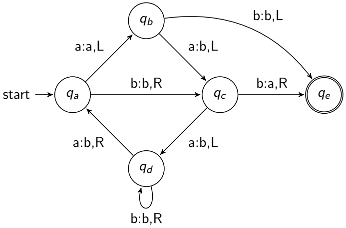
- Addition and multiplication of numbers
- Several operations on lists
- Turing machine interpreter (Universal TM)
- Multi-tape to single-tape translation

Constructing Machines

Example: 2 tapes, Copy symbol left of head on tape 0 to tape 1.

$$\begin{array}{l} \mathsf{CopyLeft} : \mathsf{TM}^2_\Sigma := \\ \quad \Uparrow_{[0]}(\mathsf{Move}\,\mathsf{L})\,; \\ \quad \mathsf{Switch}\ (\Uparrow_{[0]} \mathsf{Read}) \\ \qquad (\,\lambda f.\ \mathsf{match}\ f\ \mathsf{with} \\ \qquad\qquad\quad \lfloor s \rfloor \Rightarrow\ \Uparrow_{[1]}(\mathsf{Write}\,s) \\ \qquad\qquad \mid\ \ \emptyset \Rightarrow \mathsf{Nop}\,) \end{array}$$

$\Longrightarrow$



- Shallowly embedded language constructs machines
- Primitive machines: Move d, Read, Write s
- Control flow combinators: $M_1; M_2$, Switch M $(\lambda f.M_f)$
- Tape selection: $\Uparrow_I M$

Approach: Abstraction Layers

0. Multi-tape Turing machines (definition and semantics)
1. Labelled Turing machines & specification predicates
2. Control-flow & lifting operators
3. Registers: tape contain encodable types ($\mathbb{N}$, lists, ...)

Layer 0: Bare Turing Machines

From Asperti&Ricciotti:

- n-tape Turing machines $M : \mathsf{TM}^n_\Sigma$ over alphabet Σ.
- Semantics: evaluation in k steps, $M(q, t) \triangleright^k (q', t')$
- $t : \mathsf{Tape}_\Sigma$ is canonical, blank-free representation of tapes
    - Used space visible in resulting tape since no deallocation can happen

Layer 1: Labelled Machines & Relations

We only care *what* can be computed:

- Labelled machine $M : \mathsf{TM}^n_\Sigma(F)$ over finite Type F to hide state names:

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

Layer 1: Labelled Machines & Relations

We only care *what* can be computed:

- Labelled machine $M : \mathsf{TM}_\Sigma^n(F)$ over finite Type F to hide state names:

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

- Realisation of $R : \mathsf{Tape}_\Sigma^n \to (F \times \mathsf{Tape}_\Sigma^n) \to \mathbb{P}$:

$$M \vDash R \ := \ \forall t\ q\ t'.\ M(t) \triangleright (q, t') \to R\ t\ (\mathsf{lab}_M\ q, t')$$

Can include space consumption.

Layer 1: Labelled Machines & Relations

We only care *what* can be computed:

- Labelled machine $M : \mathsf{TM}_\Sigma^n(F)$ over finite Type F to hide state names:

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

- Realisation of $R : \mathsf{Tape}_\Sigma^n \to (F \times \mathsf{Tape}_\Sigma^n) \to \mathbb{P}$:

$$M \vDash R \ := \ \forall t\ q\ t'.\ M(t) \triangleright (q, t') \to R\ t\ (\mathsf{lab}_M\ q, t')$$

  Can include space consumption.

- Termination in $T : \mathsf{Tape}_\Sigma^n \to \mathbb{N} \to \mathit{Prop}$:

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

  Includes time analysis.

Layer 2: Basic Machines

The “assembly commands” of our (shallowly embedded) language:

Layer 2: Basic Machines

The “assembly commands” of our (shallowly embedded) language:

- Write symbol s to the tape:

$$\mathsf{Write}\ s \vDash^{\mathrm{c}} \lambda\ t\ t'.\ t'[0] = \mathsf{tape\_write}\ s\ t[0]$$

Layer 2: Basic Machines

The “assembly commands” of our (shallowly embedded) language:

- Write symbol s to the tape:

$$\mathsf{Write}\ s \vDash^{\mathrm{c}} \lambda\ t\ t'.\ t'[0] = \mathsf{tape\_write}\ s\ t[0]$$

- Read the current symbol:

$$\mathsf{Read} \vDash^{\mathrm{c}} \lambda\ t\ (\ell, t').\ \ell = \mathsf{current}\ (t[0]) \land t = t'$$

Layer 2: Basic Machines

The “assembly commands” of our (shallowly embedded) language:

- Write symbol s to the tape:

$$\text{Write } s \vDash^{\mathrm{c}} \lambda\ t\ t'.\ t'[0] = \text{tape\_write } s\ t[0]$$

- Read the current symbol:

$$\text{Read} \vDash^{\mathrm{c}} \lambda\ t\ (\ell, t').\ \ell = \text{current } (t[0]) \land t = t'$$

- Move d

Layer 2: Control Flow Combinators

$$\frac{M_1 \vDash R_1 \qquad M_2 \vDash R_2}{M_1; M_2 \vDash R_1 \circ R_2}$$

Layer 2: Control Flow Combinators

$$\frac{M_1 \vDash R_1 \qquad M_2 \vDash R_2}{M_1; M_2 \vDash R_1 \circ R_2}$$

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

Layer 2: Control Flow Combinators

$$\frac{M_1 \vDash R_1 \qquad M_2 \vDash R_2}{M_1; M_2 \vDash R_1 \circ R_2}$$

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

- Conditional: If M_1 Then M_2 Else M_3
- Loop doWhile $M : \mathsf{TM}(F)$ (for $M : \mathsf{TM}(\mathsf{option}\, F)$)

Layer 2: Liftings

- Combine 2-tape machine M and 5-tape machine N:

$$(\Uparrow_{[2,4]} M); N$$

- Other lifting to change alphabet
- Realisation/termination: relations can be lifted as well.

Layer 3: Compound data in tapes

- Type X encodable in alphabet Σ means there exists injective function $\varepsilon : X \to \Sigma^*$
- Notion $t \simeq_k x$ means t contains encoding of x and $\leq k$ other used cells
- Encodable: $\mathbb{N}$, pairs, lists, inductive data types...
- Constructor and destructor machines, e.g. for $\mathbb{N}$:
  - ConstrO: Write 0 to tape
  - ConstrS: Increment number encoded on tape
  - CaseNat : TM($\mathbb{B}$): destruct number and return occured constructor.

How to use all this?

Recall Example:

CopyLeft : $\mathsf{TM}_\Sigma^2 := \Uparrow_{[0]} \mathsf{Move\ L}; \mathsf{Switch}\ (\Uparrow_{[0]} \mathsf{Read})(\lambda f.\ \mathsf{match}\ f\ \mathsf{with} \ldots$

We want CopyLeft $\vDash$ CopyLeftRel

How to use all this?

Recall Example:

CopyLeft : $\mathsf{TM}_\Sigma^2 := \Uparrow_{[0]}$ Move L; Switch ($\Uparrow_{[0]}$ Read)($\lambda f.$ match f with …

We want CopyLeft $\vDash$ CopyLeftRel
Our framework and automation reduces all this to relational inclusion:

$$
\begin{aligned}
&\Uparrow_{[0]}(\lambda\ t_0\ t_1.\ t_1[0] = \mathsf{tape\_move}\ \mathsf{L}\ t_0[0])\\
&\circ (\lambda\ t_1\ ((\ell' : \mathbb{1}), t_3).\\
&\quad \exists t_2\ (\ell : \mathsf{option}(\Sigma)).\ (\Uparrow_{[0]}(\lambda\ t_1\ (\ell, t_2).\ \ell = \mathsf{current}(t[0]) \wedge t_2 = t_1))\ t_1\ (\ell, t_2)\\
&\qquad \wedge(\mathsf{match}\ \ell\ \mathsf{with}\\
&\qquad\quad \lfloor s \rfloor\ \Rightarrow \mathit{LiftTapes}(\lambda\ t_2\ t_3.\ t_3[0] = \mathsf{tape\_write}\ s\ t_2[0])[1]\ t_2\ t_3\\
&\qquad\quad |\,\emptyset\ \Rightarrow t_3 = t_2))\\
&\subseteq \mathsf{CopyLeftRel}
\end{aligned}
$$

How to use all this?

Recall Example:

$\mathsf{CopyLeft} : \mathsf{TM}^2_\Sigma := \Uparrow_{[0]} \mathsf{Move\ L}; \mathsf{Switch}\ (\Uparrow_{[0]} \mathsf{Read})(\lambda f.\ \mathsf{match}\ f\ \mathsf{with} \ldots$

We want $\mathsf{CopyLeft} \vDash \mathsf{CopyLeftRel}$
Our framework and automation reduces all this to relational inclusion:

$$\begin{array}{l}
\Uparrow_{[0]}(\lambda\ t_0\ t_1.\ t_1[0] = \mathsf{tape\_move\ L}\ t_0[0]) \\
\circ\ (\lambda\ t_1\ ((\ell' : \mathbb{1}), t_3). \\
\quad \exists t_2\ (\ell : \mathsf{option}(\Sigma)).\ (\Uparrow_{[0]}(\lambda\ t_1\ (\ell, t_2).\ \ell = \mathsf{current}(t[0]) \land t_2 = t_1))\ t_1\ (\ell, t_2) \\
\qquad \land(\mathsf{match}\ \ell\ \mathsf{with} \\
\qquad\quad \lfloor s \rfloor\ \Rightarrow \mathit{LiftTapes}(\lambda\ t_2\ t_3.\ t_3[0] = \mathsf{tape\_write}\ s\ t_2[0])[1]\ t_2\ t_3 \\
\qquad\quad |\, \emptyset\ \Rightarrow t_3 = t_2)) \\
\subseteq \mathsf{CopyLeftRel}
\end{array}$$

And similar for termination/time.

Main improvements compared to Asperti&Ricciotti

- Explicit handling of time and space
- Explicit notion of encoded datatypes (Layer 3)
- Labelled machines

Case studies

- Addition and multiplication of numbers [2]
- Several list operations
- Turing machine interpreter (Universal TM)
- Multi-tape to single-tape translation (without layer 3)

---

[2] unary, but binary in files

Universal 1-tape Turing Machine

Theorem

There exists a universal Turing machine Univ_Σ *that, given an encoded single-tape machine* M *over* Σ *and an encoded input tape* t_M*, simulates* M *on* t_M *with polynomial time overhead and constant-factor space overhead.*

Universal 1-tape Turing Machine

Theorem

There exists a universal Turing machine Univ_Σ *that, given an encoded single-tape machine* M *over* Σ *and an encoded input tape* t_M*, simulates* M *on* t_M *with polynomial time overhead and constant-factor space overhead.*

Actually two theorems, e.g. for correctness/space:

$$
\begin{array}{l}
\mathrm{Univ}_\Sigma \vDash \lambda t\ t'. \\
\quad \forall (M : \mathrm{TM}^1_{\Sigma_M})\ (t_M : \mathrm{Tape}_{\Sigma_M})\ (q : Q_M)\ (s_1\ s_2\ s_3\ s_4\ s_5 : \mathbb{N}). \\
\quad\quad t[0] \simeq t_M \to t[1] \simeq_{s_1} \delta_M \to t[2] \simeq_{s_2} q \to \\
\quad\quad \mathrm{isVoid}_{s_3}(t[3]) \to \mathrm{isVoid}_{s_4}(t[4]) \to \mathrm{isVoid}_{s_5}(t[5]) \to \\
\quad\quad \exists (t'_M : \mathrm{Tape}_{\Sigma_M})\ (q' : Q_M).\ M(q, t_M) \rhd (q', t'_M) \land \\
\quad\quad\quad t'[0] \simeq t'_M \ \land\ t'[1] \simeq_{s_1} \delta_M \ \land\ t[2] \simeq_{(2+|Q_M|+\max\ c_M\ s_2)} q' \ \land \\
\quad\quad\quad \mathrm{isVoid}_{\max\ c_M\ s_3}(t'[3]) \land \mathrm{isVoid}_{\max\ c_M\ s_4}(t'[4]) \ \land \mathrm{isVoid}_{\max\ c_M\ s_5}(t'[5]) \ \text{, where } c_M := |\varepsilon(\delta_M)| + 1
\end{array}
$$

And similarly for time.

Mechanisation in Coq

Useful features:

- Tactics and Ltac (proof state unmanageable otherwise)
- Existential variables
- smpl plugin[3] by Sigurd Schneider:
    - Automated forward reasoning and simplification
    - Extendable by hints
    - Should be part of Coq's `auto`

Lessons learned:

- We should have used mathcomp for finite types.
- `Inductive tape (Sigma:`~~`fin`~~`Type) := ...`

---

[3] `https://github.com/sigurdschneider/smpl`

Mechanisation in Coq (2)

- Paper hyperlinks to Coq development
- 5 min compile time
- Total: 19,100 loc
- Universal machine: 1844 lines

Lesson learned

The project started October 2016. Now, the verifications are possible, yet very tedious.

Lesson learned

The project started October 2016. Now, the verifications are possible, yet very tedious.

Formalising[4] complexity theoretical results,

(like Cook-Levin theorem, $P \subseteq NP \subseteq PSPACE \subseteq EXP \dots$)

with Turing machines is inherently infeasable.

---

[4] not only mechanising

Lesson learned

The project started October 2016. Now, the verifications are possible, yet very tedious.

Formalising[4] complexity theoretical results,

(like Cook-Levin theorem, $P \subseteq NP \subseteq PSPACE \subseteq EXP \ldots$)

with **Turing machines** is inherently infeasable.

---

[4] not only mechanising

Conclusion

Future Work:

- Mechanise call-by-value λ-calculus interpreters needed for invariance thesis:

  Yannick Forster & Fabian Kunze & Marc Roth;
  The Weak Call-By-Value λ-Calculus is Reasonable for Both Time and Space;
  POPL 2020, Wednesday 10:30

- Investigate complexity theory in the call-by-value λ-calculus:
  - Time hierarchy
  - Cook-Levin theorem (SAT is NP-complete)

Framework used in the library for undecidable problems to reduce halting problem from L to TMs: CoqPL(Saturday 16:05), `github.com/uds-psl/coq-library-undecidability`

Conclusion

Future Work:

- Mechanise call-by-value λ-calculus interpreters needed for invariance thesis:

  Yannick Forster & Fabian Kunze & Marc Roth;
  The Weak Call-By-Value λ-Calculus is Reasonable for Both Time and Space;
  POPL 2020, Wednesday 10:30

- Investigate complexity theory in the call-by-value λ-calculus:
  - Time hierarchy
  - Cook-Levin theorem (SAT is NP-complete)

Framework used in the library for undecidable problems to reduce halting problem from L to TMs: CoqPL(Saturday 16:05), `github.com/uds-psl/coq-library-undecidability`

Thanks

Lines of Code

Component		Spec	Proof
Libraries (finite types, retractions, etc.)		2157	2716
L_0&L_1	Semantics of (labelled) TMs	644	288
	Primitive Machines	178	48
L_2	Lifts	367	195
	Combinators	561	541
	Simple machines	598	432
L_3	Encodable types	282	179
	Tape-containment	171	43
	(De-) constructors	501	511
	CopyValue, Reset, Compare	538	548
	Refinement on alphabet-lift	139	93
`Ltac automation` (for L_2 and L_3)		156	15
Complexity	Time (infrastructure and machines)	268	279
	Space (infrastructure and machines)	259	193
Add and Mult		222	276
M_1	Encoding	76	65
	Implementation	783	1186
	Time bounds	236	371
Univ	Lookup in association list	169	120
	Implementation	324	333
	Time bounds	146	212
	Space bounds	240	300
MU (single-tape Univ for multi-tape machines)		408	637
Total (excl. libraries)		7327	6957

5 minutes @ Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz machine.

Semantics TM

$$\begin{aligned} M : \mathsf{TM}^n_\Sigma := (Q &: \mathsf{finType}, \\ s &: Q, \\ h &: Q \to \mathbb{B}, \\ \delta &: Q \times (\mathsf{option}(\Sigma))^n \to Q \times (\mathsf{option}(\Sigma) \times \mathsf{Move})^n) \end{aligned}$$