Formalising multi-tape Turing machines in Coq

First Bachelor Seminar Talk

Maxi Wuttke

Saarland University

Programming Systems Lab

December 21, 2017

Advisor: Yannick Forster

Supervisor: Prof. Dr. Gert Smolka

Motivation: Finish the reduction chain

$$\mathsf{L} \to m\mathsf{TM} \to \mathsf{TM} \to \mathsf{PCP}$$

[1] Yannick Forster, Edith Heiter, Gert Smolka: Verification of PCP-Related Computational Reductions in Coq; 2017

[2] Yannick Forster and Gert Smolka: Weak Call-by-Value Lambda Calculus as a Model of Computation in Coq; 2017

[3] Yannick Forster, Fabian Kunze, Marc Roth: The strong invariance thesis for a lambda-calculus; LOLA 2017

Motivation: Finish the reduction chain

$$\mathrm{L} \to m\mathrm{TM} \to \mathrm{TM} \to \mathrm{PCP}$$

- Reduction of TM $\to$ PCP formalised in Coq[1]

[1] Yannick Forster, Edith Heiter, Gert Smolka: Verification of PCP-Related Computational Reductions in Coq; 2017

[2] Yannick Forster and Gert Smolka: Weak Call-by-Value Lambda Calculus as a Model of Computation in Coq; 2017

[3] Yannick Forster, Fabian Kunze, Marc Roth: The strong invariance thesis for a lambda-calculus; LOLA 2017

Motivation: Finish the reduction chain

$$\mathrm{L} \to m\mathrm{TM} \to \mathrm{TM} \to \mathrm{PCP}$$

- Reduction of TM $\to$ PCP formalised in Coq[1]
- Computational theory of L formalised in Coq[2]

[1] Yannick Forster, Edith Heiter, Gert Smolka: Verification of PCP-Related Computational Reductions in Coq; 2017

[2] Yannick Forster and Gert Smolka: Weak Call-by-Value Lambda Calculus as a Model of Computation in Coq; 2017

[3] Yannick Forster, Fabian Kunze, Marc Roth: The strong invariance thesis for a lambda-calculus; LOLA 2017

Motivation: Finish the reduction chain

$$\mathrm{L} \to m\mathrm{TM} \to \mathrm{TM} \to \mathrm{PCP}$$

- Reduction of TM $\to$ PCP formalised in Coq[1]
- Computational theory of L formalised in Coq[2]
- Now we formalise multi-tape Turing machines

[1] Yannick Forster, Edith Heiter, Gert Smolka: Verification of PCP-Related Computational Reductions in Coq; 2017

[2] Yannick Forster and Gert Smolka: Weak Call-by-Value Lambda Calculus as a Model of Computation in Coq; 2017

[3] Yannick Forster, Fabian Kunze, Marc Roth: The strong invariance thesis for a lambda-calculus; LOLA 2017

Motivation: Finish the reduction chain

$$\mathrm{L} \to m\mathrm{TM} \to \mathrm{TM} \to \mathrm{PCP}$$

- Reduction of TM $\to$ PCP formalised in Coq[1]
- Computational theory of L formalised in Coq[2]
- Now we formalise multi-tape Turing machines
- Goal: build a multi-tape Turing machine that simulates L[3]

[1] Yannick Forster, Edith Heiter, Gert Smolka: Verification of PCP-Related Computational Reductions in Coq; 2017

[2] Yannick Forster and Gert Smolka: Weak Call-by-Value Lambda Calculus as a Model of Computation in Coq; 2017

[3] Yannick Forster, Fabian Kunze, Marc Roth: The strong invariance thesis for a lambda-calculus; LOLA 2017

Introduction: Turing machines

Pros:

- are easy to understand / imagine;
- are the de-facto standard model of computation

Introduction: Turing machines

Pros:

- are easy to understand / imagine;
- are the de-facto standard model of computation

Cons:

- not compositional;
- encoding data is tedious;
- formal reasoning is tedious;
- completely unstructured

Introduction: Related Work

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

Definitions

Let Σ be a finite type and $n : \mathbb{N}$.

Definition (n-tape Turing machine over Σ)

An n-tape Turing machine over Σ is a record mTM Σ n:

- Q is the finite type of states;
- $\gamma : Q \times \mathcal{O}(\Sigma)^n \to Q \times (\mathcal{O}(\Sigma) \times \text{Move})^n$ (transition function);
- $s : Q$ (start state);
- $h : Q \to \mathbb{B}$ (halting states)

Where Move $:= \{L, R, N\}$.

Definitions

Definition (Tape)

A tape over Σ is either: $[\underset{\uparrow}{\ }]$, $[\underset{\uparrow}{\ } \; r \; R]$, $[R \; \underset{\uparrow}{m} \; L]$, or $[L \; l \; \underset{\uparrow}{\ }]$,

Where $r, l : \Sigma$ and $R, L : \mathcal{L}(\Sigma)$.

Definitions

Definition (Tape)

A tape over Σ is either: $[\underset{\uparrow}{\ }]$, $[\underset{\uparrow}{\ }\ r\ R]$, $[R\ \underset{\uparrow}{m}\ L]$, or $[L\ l\ \underset{\uparrow}{\ }]$,

Where $r, l : \Sigma$ and $R, L : \mathcal{L}(\Sigma)$.

$$
\begin{aligned}
\text{Tape} := \ & \text{niltape} \\
& \mid \text{leftof}\ (r : \Sigma)\ (R : \mathcal{L}(\Sigma)) \\
& \mid \text{midtape}\ (L : \mathcal{L}(\Sigma))\ (m : \Sigma)\ (R : \mathcal{L}(\Sigma)) \\
& \mid \text{rightof}\ (l : \Sigma)\ (L : \mathcal{L}(\Sigma)).
\end{aligned}
$$

Definitions

Definition (Tape)

A tape over Σ is either: $[\underset{\uparrow}{\ }]$, $[\underset{\uparrow}{\ } \; r \; R]$, $[R \; \underset{\uparrow}{m} \; L]$, or $[L \; l \; \underset{\uparrow}{\ }]$,

Where $r, l : \Sigma$ and $R, L : \mathcal{L}(\Sigma)$.

$$\begin{aligned} \mathsf{Tape} := \; & \mathsf{niltape} \\ & \mid \mathsf{leftof}\ (r : \Sigma)\ (R : \mathcal{L}(\Sigma)) \\ & \mid \mathsf{midtape}\ (L : \mathcal{L}(\Sigma))\ (m : \Sigma)\ (R : \mathcal{L}(\Sigma)) \\ & \mid \mathsf{rightof}\ (l : \Sigma)\ (L : \mathcal{L}(\Sigma)). \end{aligned}$$

Definition (Configuration)

A configuration is a record $\mathsf{Conf}_M := \{\mathsf{state} : Q_M;\ \ \mathsf{tapes} : \mathsf{Tape}^n\}$.

Definitions

Definition (Step function)

With the functions

$$\begin{aligned} \text{wr\_mv} &: \text{Tape} \to (\mathcal{O}(\Sigma) \times \text{Move}) \to \text{Tape} \\ \text{current} &: \text{Tape} \to \mathcal{O}(\Sigma) \end{aligned}$$

we define the **step function** $\text{step}_M : \text{Conf}_M \to \text{Conf}_M$:

$$\begin{aligned} \text{step}_M\ (\text{tapes}, q) := \text{let}\ (q', \text{actions}) := \gamma_M(q, \text{map current tapes})\ \text{in} \\ (q', \text{map}_2\ \text{wr\_mv tapes actions}) \end{aligned}$$

Definitions

Definition (Execution)

$$\text{loop } k\ f\ h\ s := \begin{cases} \lfloor s \rfloor & h(s) = \text{true} \\ \emptyset & h(s) = \text{false} \wedge k = 0 \\ \text{loop } (k-1)\ f\ h\ (f(s)) & h(s) = \text{false} \wedge k > 0 \end{cases}$$

Definitions

Definition (Execution)

$$\text{loop } k\ f\ h\ s := \begin{cases} \lfloor s \rfloor & h(s) = \text{true} \\ \emptyset & h(s) = \text{false} \wedge k = 0 \\ \text{loop } (k-1)\ f\ h\ (f(s)) & h(s) = \text{false} \wedge k > 0 \end{cases}$$

$$\text{iconf}_M\ t_{in} := (\text{tapes}, s_M)$$

$$\text{hconf}_M\ (\text{tapes}, q) := h_M(q)$$

$$\text{exec}_M\ t_{in}\ k := \text{loop } k\ \text{step}_M\ \text{hconf}\ (\text{initc}_M\ t_{in})$$

$$\text{exec}_M : \text{Tape}^n \to \mathbb{N} \to \mathcal{O}(\text{Conf})$$

Definitions

Definition (Execution)

$$\text{loop } k\ f\ h\ s := \begin{cases} \lfloor s \rfloor & h(s) = \text{true} \\ \emptyset & h(s) = \text{false} \wedge k = 0 \\ \text{loop } (k-1)\ f\ h\ (f(s)) & h(s) = \text{false} \wedge k > 0 \end{cases}$$

$$\text{iconf}_M\ t_{in} := (\text{tapes}, s_M)$$

$$\text{hconf}_M\ (\text{tapes}, q) := h_M(q)$$

$$\text{exec}_M\ t_{in}\ k := \text{loop } k\ \text{step}_M\ \text{hconf}\ (\text{initc}_M\ t_{in})$$

$$\text{exec}_M : \text{Tape}^n \to \mathbb{N} \to \mathcal{O}(\text{Conf})$$

We write $\mathrm{M}(\mathrm{t_{in}}) \triangleright^{\mathrm{n}} \mathrm{c_{out}}$, if $\text{exec}_M\ t_{in}\ n = \lfloor c_{out} \rfloor$.

Correctness of machines

Definition (Realisation, $M \Vdash_f R$)

Let $R \subseteq (\text{Tape}^n) \times (F \times \text{Tape}^n)$ and $f : Q_M \to F$, for a finite F.

$$\text{M} \Vdash_{\text{f}} \text{R} := \forall (t_{in} : \text{Tape}^n)\ (i : \mathbb{N})\ (q_{out} : Q_M)\ (t_{out} : \text{Tape}^n).$$
$$M(t_{in}) \triangleright^i (q_{out}, t_{out}) \to (t_{in}, (f(q_{out}), t_{out})) \in R.$$

Then we say that M **realises** R.

Correctness of machines

Definition (Realisation, $M \Vdash_f R$)

Let $R \subseteq (\text{Tape}^n) \times (F \times \text{Tape}^n)$ and $f : Q_M \to F$, for a finite F.

$$\text{M} \Vdash_\text{f} \text{R} := \forall (t_{in} : \text{Tape}^n)\ (i : \mathbb{N})\ (q_{out} : Q_M)\ (t_{out} : \text{Tape}^n).$$
$$M(t_{in}) \triangleright^i (q_{out}, t_{out}) \to (t_{in}, (f(q_{out}), t_{out})) \in R.$$

Then we say that M **realises** R.

For **partitioned machines** $M : \left\{ M' : \text{mTM}\ \Sigma\ n;\ f : Q_{M'} \to F \right\}$, we write $\text{M} \Vdash \text{R}$.

Termination of machines

Definition (Termination, $M \downarrow T$)

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

$$\mathsf{M} \downarrow \mathsf{T} := \forall (t_{in},\ i) \in T.\ \exists c_{out},\ M(t_{in}) \rhd^i c_{out}$$

We say that M **terminates in** T.

Relations

Let $R, S \subseteq X \times Y$; $T \subseteq Y \times Z$; $U \subseteq X \times X$; $V \subseteq X \times (F \times Y)$; and $\Phi : F \to \mathcal{P}(X \times Y)$.

Standard relation combinators:

$$R \cap S := \{(x,y) \mid (x,y) \in R \wedge (x,y) \in S\}$$
$$R \cup S := \{(x,y) \mid (x,y) \in R \vee (x,y) \in S\}$$
$$R \circ T := \{(x,z) \mid \exists y,\ (x,y) \in R \wedge (y,z) \in T\}$$
$$\mathrm{Id}_X := \{(x,x) \mid x : X\}$$

Reflexive transitive closure:

$$\frac{}{(x,x) \in U^*} \qquad \frac{(x,y) \in U \qquad (y,z) \in U^*}{(x,z) \in U^*}$$

Big union:

$$\bigcup_{a:F} \Phi(a) := \{(x,y) \mid \exists a, (x,y) \in (\Phi\ a)\}$$

Parametrise relations:

$$\Uparrow R := \{(x,(a,y)) \mid a : F \wedge (x,y) \in R\}$$
$$\Downarrow R := \{((x,a),y) \mid a : F \wedge (x,y) \in R\}$$

Relational if:

$$R\ \phi\ S := \{(x,(\mathsf{true},y)) \mid (x,y) \in R\} \cup \{(x,(\mathsf{false},y)) \mid (x,y) \in S\}$$

Restriction: (for a fixed $a : F$)

$$V|_a := \{(x,z) \mid (x,(a,z)) \in V\}$$

Combinators

- Implement programming primitives as operators on Turing machines:
  - match
  - sequential composition
  - if then else
  - do-while
- Verify them using correctness relations

Sequential composition and If Then Else

Let M_1 be parametrised over F and M_2 over F'.

Corollary (Correctness of sequential composition)

If $M_1 \Vdash R_1$ *and* $M_2 \Vdash R_2$, *then* $(M_1;\ M_2) \Vdash R_1 \circ \Downarrow R_2$.

Sequential composition and If Then Else

Let M_1 be parametrised over F and M_2 over F'.

Corollary (Correctness of sequential composition)

If $M_1 \Vdash R_1$ *and* $M_2 \Vdash R_2$, *then* $(M_1;\ M_2) \Vdash R_1 \circ \Downarrow R_2$.

Let M_1 be parametrised over $\mathbb{B}$ and M_2 and M_3 over F.

Corollary (Correctness of boolean if)

If $M_1 \Vdash R_1$ *and* $M_2 \Vdash R_2$ *and* $M_3 \Vdash R_3$, *then*
$(\mathit{If}\ M_1\ \mathit{Then}\ M_2\ \mathit{Else}\ M_3) \Vdash (R_1|_{\mathsf{true}}) \circ R_2 \cup (R_1|_{\mathsf{false}}) \circ R_3$.

Machine transformations

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

Machine transformations

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

Solution: Two operations on machines:

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

n-Lift

Let $f : \mathsf{Fin}_m \hookrightarrow \mathsf{Fin}_n$ be an injection between tape indexes.

Let M be an m-tape F-partitioned Turing machine parametrised over Σ, and R be a F-parametrised relation.

Define an n-tape Turing machine $\mathsf{Lift}_f\ M$ and a relation $\mathsf{Lift}_f\ R$.

$$\begin{aligned}\mathsf{Lift}_f\ R := \left\{(t_{in}, (a, t_{out})) \;\middle|\; \left(f^{-1}(t_{in}), (a, f^{-1}(t_{out}))\right) \in R\right\} \cap \\ \Uparrow \left\{(t_{in}, t_{out}) \;\middle|\; \forall i \notin \mathsf{img}\ f.\ t_{in}[i] = t_{out}[i]\right\}\end{aligned}$$

(Where $f^{-1} : \mathsf{Tape}^n \to \mathsf{Tape}^m$)

Lemma (Correctness of the n-Lift)

$$M \Vdash R \to (\mathit{Lift}_f\ M) \Vdash (\mathit{Lift}_f\ R)$$

Σ-Lift

Let $f : \Sigma \hookrightarrow \Gamma$ be an injection between alphabets.

Let default : Σ.

Let M be an n-tape F-partitioned Turing machine parametrised over Σ, and R be a F-parametrised relation.

Define an n-tape Turing machine $\text{Lift}_{f,\text{default}}\ M$ and a relation $\text{Lift}_{f,\text{default}}\ R$.

$$\text{Lift}_{f,\text{default}}\ R := \left\{(t_{in}, (a, t_{out})) \;\middle|\; (f^{-1}_{\text{default}}(t_{in}), (a, f^{-1}_{\text{default}}(t_{out}))) \in R\right\}$$

Lemma (Correctness of the Σ-Lift)

$$M \Vdash R \to (Lift_{f,default}\ M) \Vdash (Lift_{f,default}\ R)$$

Encoding

Definition (Encodable types)

A type X is *encodable* over Σ, if there exists a function $\text{encode} : X \to \mathcal{L}\ \Sigma$, s.t.

$$\forall (v_1\ v_2 : X)\ (r_1\ r_2 : \mathcal{L}(\Sigma)).$$
$$\text{encode}(x) +\!\!+ r_1 = \text{encode}(v_2) +\!\!+ r_2 \to v_1 = v_2 \wedge r_1 = r_2$$

Encoding

Definition (Encodable types)

A type X is *encodable* over Σ, if there exists a function $\text{encode} : X \to \mathcal{L}\ \Sigma$, s.t.

$$\forall (v_1\ v_2 : X)\ (r_1\ r_2 : \mathcal{L}(\Sigma)).$$
$$\text{encode}(x) +\!\!+ r_1 = \text{encode}(v_2) +\!\!+ r_2 \to v_1 = v_2 \wedge r_1 = r_2$$

Definition (Tape encodes value)

We write Σ^+ instead of $\Sigma + \mathbb{B}$.
For a X encodable over Σ, a Σ^+-tape t **encodes** a value $x : X$, if

$$t = [(r_1 +\!\!+ [\text{inr}\,\text{true}])\ (\underset{\uparrow}{\text{inl}\,y})\ (\text{map}\ \ \text{inl}\ \ ys +\!\!+ \text{inr}\,\text{false} :: r_2)]$$

for $y :: ys = \text{encode}(x)$, and some $r_1, r_2 \in \mathcal{L}\ \Sigma^+$.

Calculate functions

Definition (Calculate function)

A machine M *computes* a function $f : X \to Y$ from $i < n$ to $j < n$, if $M \Vdash \mathsf{FunRel}_{f,i,j}$ with

$$\mathsf{FunRel}_{f,i,j} := \Uparrow\left\{(t_{in}, t_{out}) \;\middle|\; \forall x : X,\ \mathsf{encodes}(t_{in}[i], x) \to \mathsf{encodes}(t_{out}[j], f(x))\right\}$$

Calculate functions

Definition (Calculate function)

A machine M *computes* a function $f : X \to Y$ from $i < n$ to $j < n$, if $M \Vdash \mathsf{FunRel}_{f,i,j}$ with

$$\mathsf{FunRel}_{f,i,j} := \Uparrow\left\{(t_{in}, t_{out}) \;\middle|\; \forall x : X,\ \mathsf{encodes}(t_{in}[i], x) \to \mathsf{encodes}(t_{out}[j], f(x))\right\}$$

Extendable to multiple tapes.

Example: De Morgan machine

Assume AND : mTM $\mathbb{B}^+$ 2 that computes *andb* : $\mathbb{B} \to \mathbb{B} \to \mathbb{B}$ from 0 and 1 to 1.

Assume NOT : mTM $\mathbb{B}^+$ 1 that computes *notb* : $\mathbb{B} \to \mathbb{B}$.

$$\text{OR} := \text{Lift}_{[0 \mapsto 0]}\ \text{NOT};\ \text{Lift}_{[0 \mapsto 1]}\text{NOT};\ \text{AND};\ \text{Lift}_{[0 \mapsto 1]}\ \text{NOT}$$

Example: De Morgan machine

Assume AND : mTM $\mathbb{B}^+$ 2 that computes $andb : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ from 0 and 1 to 1.

Assume NOT : mTM $\mathbb{B}^+$ 1 that computes $notb : \mathbb{B} \to \mathbb{B}$.

$$\mathsf{OR} := \mathsf{Lift}_{[0 \mapsto 0]}\ \mathsf{NOT};\ \mathsf{Lift}_{[0 \mapsto 1]}\mathsf{NOT};\ \mathsf{AND};\ \mathsf{Lift}_{[0 \mapsto 1]}\ \mathsf{NOT}$$

Fact: OR computes $orb : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ from 0 and 1 to 1.

Proof: Using the correctness lemmas about Lift and sequential composition we get a relation R' that M realises. Show $R' \subseteq \mathsf{FunRel}_{orb,i,j}$. This follows with the De Morgan law.

Example: De Morgan machine 2

Fact:

$$OR' := \mathsf{Lift}_{[\mathsf{true}\mapsto\mathsf{false},\mathsf{false}\mapsto\mathsf{true}]}\ \mathsf{AND}$$

also computes *orb* from 0 and 1 to 1.

Example: De Morgan machine 2

Fact:

$$OR' := \mathsf{Lift}_{[\mathsf{true}\mapsto\mathsf{false},\mathsf{false}\mapsto\mathsf{true}]}\ \mathsf{AND}$$

also computes *orb* from 0 and 1 to 1.

a	$\bar{a}$	b	$\bar{b}$	$a \wedge b$	$\bar{a} \wedge \bar{b}$	$\overline{\bar{a} \wedge \bar{b}} = a \vee b$
0	1	0	1	0	1	0
0	1	1	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	1

Other results

- Termination relations
- match
- while
- mirror
- Basic machines (move, read, write, nop)
- **Fact:** All finite functions can be computed in 1 step.

Conclusion

What we have

- Formal definition of Turing machines and a framework of correctness predicates;
- Construction and verification of combinators and transformations;
- Formal definition of encoding and computation;
- Implementation of some basic and compound machines

Future work

- Verify compound machines with while, e.g. the projection functions for tuples and lists;
- Build towards an interpreter for the weak CBV lambda calculus L with time and space analysis;
- Build a Universal Turing machine

Backup Slides

This Are The Backup Slides!

Thank you!

Related Work

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

Yannick Forster and Gert Smolka

Weak Call-by-Value Lambda Calculus as a Model of Computation in Coq

ITP 2017

Yannick Forster, Fabian Kunze, Marc Roth

The strong invariance thesis for a lambda-calculus

LOLA 2017

Yannick Forster, Edith Heiter, Gert Smolka

Verification of PCP-Related Computational Reductions in Coq

(Pre-print)

Match

The idea

Given:

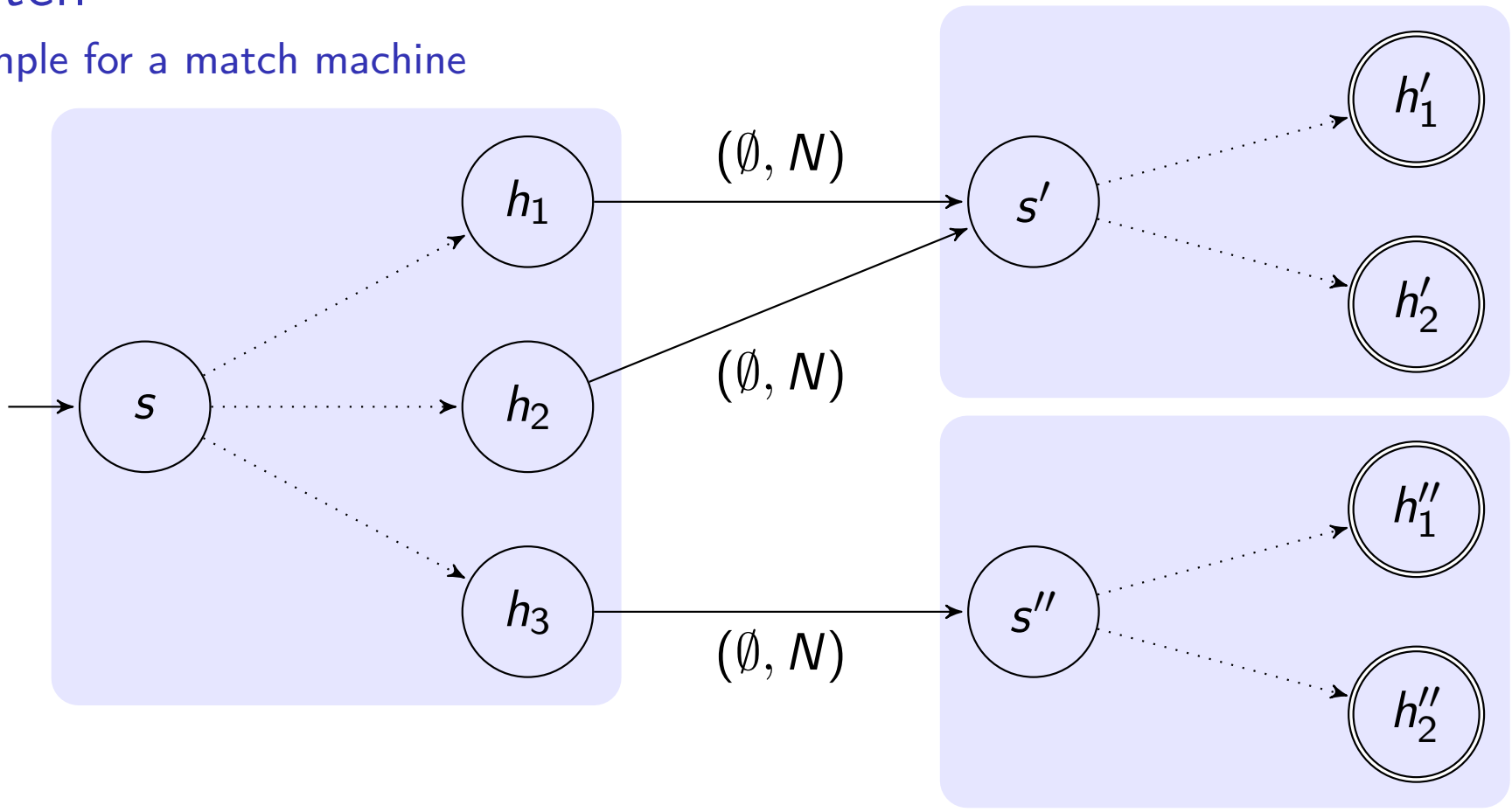
- Two finite types F and F',
- A machine M and a partitioning function $p : Q_M \to F$,
- For all $a : F$, a machine M'_a with a partitioning function $p'ay : Q_{M'_a} \to F'$

Do:

1. Execute a machine M.
2. Depending on the state of termination q_h:
   - Execute the machine $M'_{p(q_h)}$.

Match

Example for a match machine



$p(h_1) = \text{true}$ $p(h_2) = \text{true}$ $p(h_3) = \text{false}$

Match

Definition

$$Q_{\mathsf{match}} := Q_M + \left\{ y : F \;\&\; Q_{\left(M'_y\right)} \right\}$$

$$\gamma_{\mathsf{match}}(\mathsf{inl}\, q, \mathsf{symbols}) := \begin{cases} \gamma_M(q, \mathsf{symbols}) & h_M(q) = \mathsf{false} \\ \left(s_{\left(M'_{p(q)}\right)}, (\emptyset, N)^n \right) & h_M(q) = \mathsf{true} \end{cases}$$

$$\gamma_{\mathsf{match}}(\mathsf{inr}(a, q), \mathsf{symbols}) := \gamma_{\left(M'_a\right)}(q, \mathsf{symbols})$$

$$h_{\mathsf{match}}\ (\mathsf{inl}\, q) := \mathsf{false}$$
$$h_{\mathsf{match}}\ (\mathsf{inr}(a, q)) := h_{\left(M'_a\right)}(q)$$

$$p_{\mathsf{match}}\ (\mathsf{inl}\, q) := \ldots$$
$$p_{\mathsf{match}}\ (\mathsf{inr}(a, q)) := p'_a(q)$$

Match

Correctness

Lemma (Correctness of match)

If $M \Vdash R$ *and for each* $a : F$, $M'_a \Vdash R'_a$, *then*

$$match \Vdash \bigcup_{y:F} \left((R|_y) \circ R'_a \right)$$

Sequential composition and If Then Else

Definition (Sequential composition)

$$M_1;\ M_2 := \mathsf{match}\ M_1\ (\lambda y.\, M_2)$$

Definition (Boolean if)

$$\mathsf{If}\ M_1\ \mathsf{Then}\ M_2\ \mathsf{Else}\ M_3 := \mathsf{match}\ (M_1)\left(\lambda b.\begin{cases} M_2 & b = \mathsf{true} \\ M_3 & b = \mathsf{false} \end{cases}\right)$$

While

Idea:

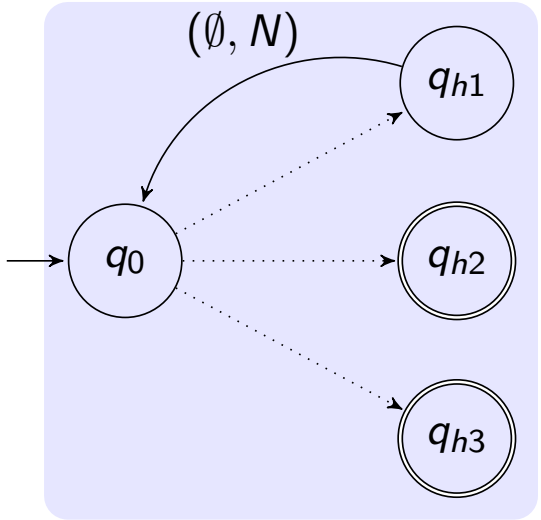
- Let M be parametrised over $\mathbb{B} \times F$.
- Repeat M until terminates in a rejecting state
- Partition function returns the F part

While

Idea:

- Let M be parametrised over $\mathbb{B} \times F$.
- Repeat M until terminates in a rejecting state
- Partition function returns the F part

Example:



$\#1(p(q_{h1})) = \text{true}$

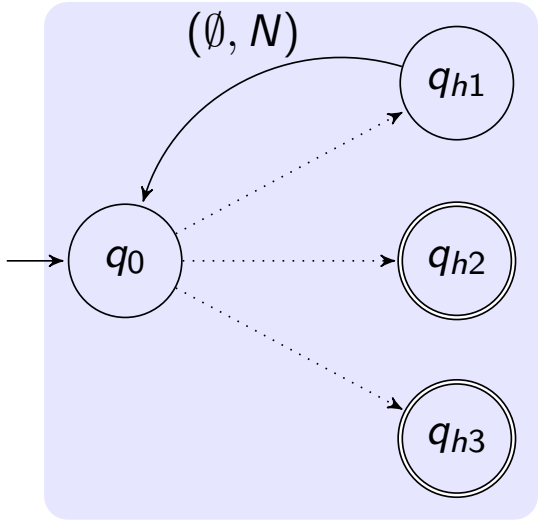
$\#1(p(q_{h2})) = \#1(p(q_{h3})) = \text{false}$

While

Idea:

- Let M be parametrised over $\mathbb{B} \times F$.
- Repeat M until terminates in a rejecting state
- Partition function returns the F part

Example:



$\#1(p(q_{h1})) = \text{true}$

$\#1(p(q_{h2})) = \#1(p(q_{h3})) = \text{false}$

Lemma (Correctness of while)

If $M \Vdash R$, *then*

$$\mathit{while} \Vdash \left(\bigcup_{y:F} R|_{(\text{true},y)} \right)^* \circ \left\{ (t,(y,t')) \mid (t,((\text{false},y),t')) \in R \right\}$$

Encoding functions

$$
\begin{aligned}
&\mathsf{encode}_{\Sigma+\Gamma+\mathbb{B}}\ (\mathsf{inl}\, x) := \mathsf{inr}\, \mathsf{false} :: \mathsf{map}\ (\mathsf{inl} \circ \mathsf{inr})\ \mathsf{encode}(x)\\
&\mathsf{encode}_{\Sigma+\Gamma+\mathbb{B}}\ (\mathsf{inl}\, y) := \mathsf{inr}\, \mathsf{true} :: \mathsf{map}\ (\mathsf{inl} \circ \mathsf{inl})\ \mathsf{encode}(y)\\
&\mathsf{encode}_{\Sigma+\mathbb{B}}\ (\mathsf{nil}) := \mathsf{inr}\, \mathsf{false}\\
&\mathsf{encode}_{\Sigma+\mathbb{B}}\ (x :: xs) := \mathsf{inr}\, \mathsf{true} :: \mathsf{map}\ \ \mathsf{inl}\ \ \mathsf{encode}_{\Sigma}(x) +\!\!+ \mathsf{encode}_{\Sigma+\mathbb{B}}(xs)\\
&\mathsf{encode}_{\Sigma+\Gamma}\ (x, y) := \mathsf{map}\ \ \mathsf{inl}\ \ \mathsf{encode}_{\Sigma}(x) +\!\!+ \mathsf{map}\ \ \mathsf{inr}\ \ \mathsf{encode}_{\Gamma}(y)\\
&\mathsf{encode}_{\mathbb{B}}\ (0) := [\mathsf{false}]\\
&\mathsf{encode}_{\mathbb{B}}\ (\mathsf{S}\ n) := \mathsf{true} :: \mathsf{encode}_{\mathbb{B}}(n)
\end{aligned}
$$

n-lift

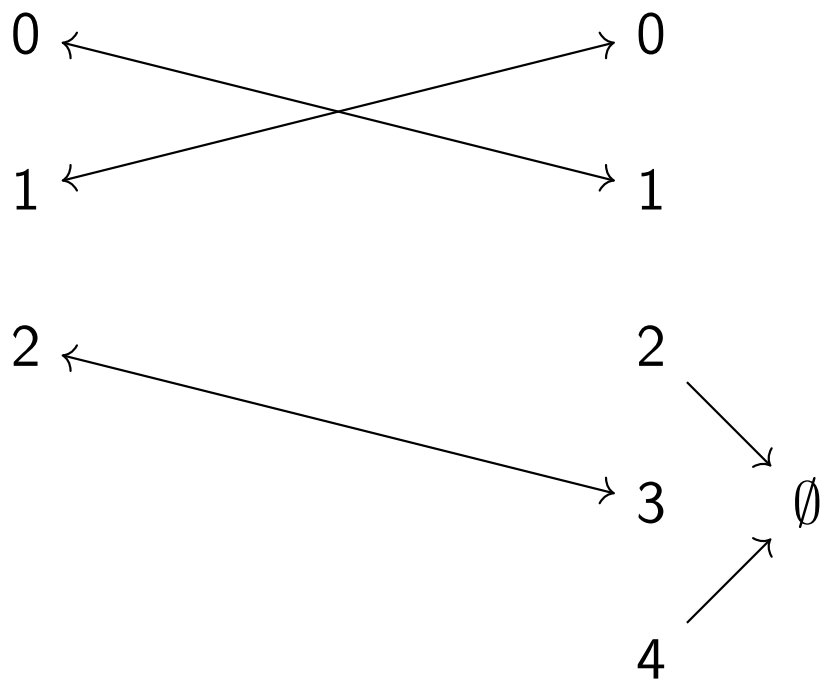


Figure: Tape retract encoded as the index-vector $[1, 0, 3]$.

Σ-lift

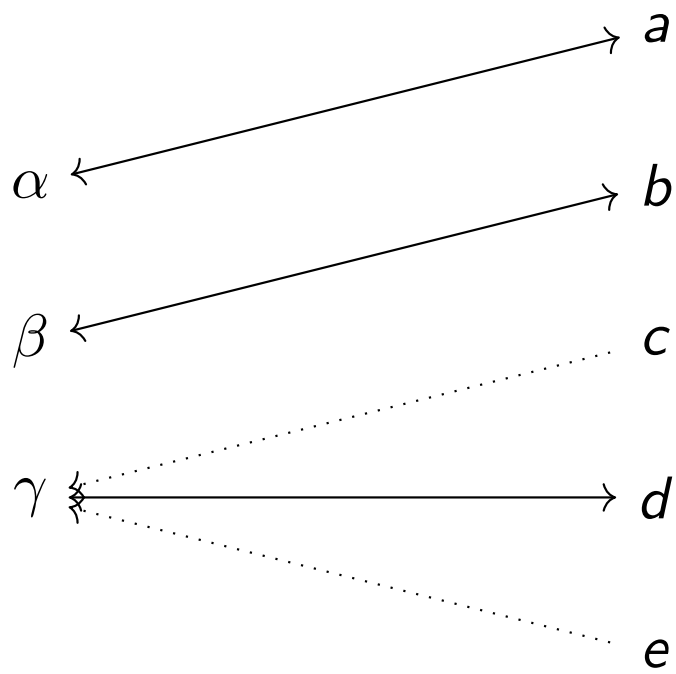


Figure: Example alphabet retract; γ is the default symbol in Σ.

Basic machines

Classes of 1-tape machines for each basic action (together with partitioning functions):

- Move head in the direction D
- Write one symbol
- Read one symbol
- nop

Basic machines

Classes of 1-tape machines for each basic action
(together with partitioning functions):

- Move head in the direction D
- Write one symbol
- Read one symbol
- nop
- Write a list of symbols (defined per recursion over the string)

Coq code dimensions

	Lines Spec	Lines Proof
Prelim	235+	286+
Relations/Retracts	475	269
Def. nTM	215	85
Σ-Lift	132	129
n-Lift	153	195
Basic machines	142	69
Encoding	170	164
Computation	241	289
Combinators	158	354
Finite and $\mathbb{B}$ TM	112	203
Σ	2033	2042