Ideal objects and abstract machines

Thomas Powell

Technische Universität Darmstadt

Workshop: Proofs and Computation
part of the Trimester: Types, Sets and Constructions

Hausdorff Research Institute for Mathematics, Bonn
5 July 2018

Outline

1. The problem.
2. A brief sketch of some ideas from the last few months.
3. Open problems

Outline

1. The problem.
2. A brief sketch of some ideas from the last few months.
3. Open problems (which include the original problem).

Motivation

Lots of people study

$$\textsc{Proofs} \mapsto \textsc{Programs}$$

There are many well known techniques for extracting programs from proofs e.g.

- Epsilon calculus & substitution method
- Functional intepretation
- Many variants of realizability
- ...

Motivation

Lots of people study

$$\textsc{Proofs} \mapsto \textsc{Programs}$$

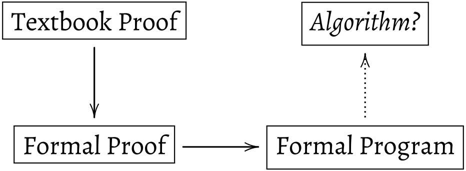
There are many well known techniques for extracting programs from proofs e.g.

- Epsilon calculus & substitution method
- Functional intepretation
- Many variants of realizability
- ...

On a **small scale**, these tools gives us a clear insight into the computational meaning of proofs.

On a **large scale**, they produce programs which are usually incomprehensible.

What is the issue?



What is the issue?

$$
\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[d] & \boxed{\textit{Algorithm?}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, dotted]
\end{tikzcd}
$$

Why do we care?

What is the issue?

$$\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[d] & \boxed{\textit{Algorithm?}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, dotted]
\end{tikzcd}$$

Why do we care?

- The programs from proofs paradigm should also work on a high level.

What is the issue?

$$\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[d] & \boxed{\textit{Algorithm?}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, dotted]
\end{tikzcd}$$

Why do we care?

- The programs from proofs paradigm should also work on a high level.
- For certain applications, it would be good know how formally extracted programs behave.

What is the issue?

$$
\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[d] & \boxed{\textit{Algorithm?}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, dotted]
\end{tikzcd}
$$

Why do we care?

- The programs from proofs paradigm should also work on a high level.
- For certain applications, it would be good know how formally extracted programs behave.
- Trying to connect proof theoretic techniques with ideas from the theory of algorithms e.g.
  - automata
  - flowcharts
  - state machines

  could lead to new ideas.

One attempt

Idea behind ***T.P.*** *A functional interpretation with state (LICS 18)*:

$$
\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[d] & \boxed{\textit{Algorithm encoded in state}} \\
\boxed{\text{Enriched Formal Proof}} \arrow[r] & \boxed{\text{Formal Program with Global State}} \arrow[u]
\end{tikzcd}
$$

One attempt

Idea behind ***T.P.*** *A functional interpretation with state (LICS 18)*:

$$
\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[d] & \boxed{\textit{Algorithm encoded in state}} \\
\boxed{\text{Enriched Formal Proof}} \arrow[r] & \boxed{\text{Formal Program with Global State}} \arrow[u]
\end{tikzcd}
$$

Some downsides:

- Translation quite complicated
- Still need to fully formalise proof
- State only captures some aspects of underlying algorithm

One attempt

Idea behind ***T.P.*** *A functional interpretation with state (LICS 18)*:

$$\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[d] & \boxed{\textit{Algorithm encoded in state}} \\
\boxed{\text{Enriched Formal Proof}} \arrow[r] & \boxed{\text{Formal Program with Global State}} \arrow[u]
\end{tikzcd}$$

Some downsides:

- Translation quite complicated
- Still need to fully formalise proof
- State only captures some aspects of underlying algorithm

N. B. There are some upsides too, which I will present next week!

Another direction

$$
\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[r] \arrow[d, squiggly] & \boxed{\textit{Algorithm}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, squiggly]
\end{tikzcd}
$$

Another direction

$$\begin{tikzcd}
\text{Textbook Proof} \arrow[r] \arrow[d, squiggly] & \textit{Algorithm} \\
\text{Formal Proof} \arrow[r] & \text{Formal Program} \arrow[u, squiggly]
\end{tikzcd}$$

Key idea:

- Use formal proof theoretic techniques as **tools**...
- ... to be combined with **human intuition**.
- Program extraction should mimic the style of ordinary mathematics.

Another direction

$$
\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[r] \arrow[d, rightsquigarrow] & \boxed{\textit{Algorithm}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, rightsquigarrow]
\end{tikzcd}
$$

Key idea:

- Use formal proof theoretic techniques as **tools**...
- ... to be combined with **human intuition**.
- Program extraction should mimic the style of ordinary mathematics.

Key question: **What is an algorithm?**

What is an algorithm?

$$\begin{array}{c} \boxed{\text{Description with Pen \& Paper}} \\ \uparrow \\ \boxed{\textbf{Abstract State Machine}} \\ \uparrow \\ \boxed{\text{Formal calculus (e.g. System T)}} \end{array}$$

State machines

At its simplest, a state machine consists of

- A set S of *states*
- A *transition relation* $\rhd \subseteq S \times S$.

A **computation** is a sequence $s_0 \rhd s_1 \rhd \ldots \rhd s_{n-1}$.

State machines

At its simplest, a state machine consists of

- A set S of *states*
- A *transition relation* $\rhd \subseteq S \times S$.

A **computation** is a sequence $s_0 \rhd s_1 \rhd \ldots \rhd s_{n-1}$.

- State machines are very good at describing how programs work.

- Our choice of state machine will depend on **what we are trying to describe**, and on **which level of abstraction**.

A state machine for Π_3 formulas

States encode the following structure:

$$(\text{control}, \text{input}, \text{oracle query}, \text{oracle answer}) \in C \times A \times X \times (Y + \{\square\})$$

A state machine for Π_3 formulas

States encode the following structure:

$$(\text{control}, \text{input}, \text{oracle query}, \text{oracle answer}) \in C \times A \times X \times (Y + \{\square\})$$

Among the states we identify

- Initial states $(c_0, a, x_0, \square)$;
- Query states $(c, a, x, \square)$ with $c \in C^?$;
- End states (c, a, x, y) with $c \in C^!$.

A state machine for Π_3 formulas

States encode the following structure:

$$(\text{control}, \text{input}, \text{oracle query}, \text{oracle answer}) \in C \times A \times X \times (Y + \{\square\})$$

Among the states we identify

- Initial states $(c_0, a, x_0, \square)$;
- Query states $(c, a, x, \square)$ with $c \in C^?$;
- End states (c, a, x, y) with $c \in C^!$.

Our transition relation comprises

- Normal transitions $(c, a, x, y) \rhd (c', a, x', y/\square)$;
- Oracle transitions $(c, a, x, \square) \rhd (c, a, x, y)$.

The no-counterexample interpretation

A state machine computes a Π_3 formula $\forall a \in A \exists x \in X \forall y \in Y P(a,x,y)$ if

$$(c_0, a, x_0, \Box) \rhd^* (c \in C^!, a, x, y) \text{ with } P(a,x,y)$$

for any input a and any oracle.

The no-counterexample interpretation

A state machine computes a Π_3 formula $\forall a \in A \exists x \in X \forall y \in Y P(a,x,y)$ if

$$(c_0, a, x_0, \square) \triangleright^* (c \in C^!, a, x, y) \text{ with } P(a,x,y)$$

for any input a and any oracle.

Theorem (Rough statement)

There is a computable functional witnessing the n.c.i. of $A :\equiv \forall a \exists x \forall y P(a,x,y)$ *iff there is a state machine which computes* A*.*

The no-counterexample interpretation

A state machine computes a Π_3 formula $\forall a \in A \exists x \in X \forall y \in Y P(a,x,y)$ if

$$(c_0, a, x_0, \square) \rhd^* (c \in C^!, a, x, y) \text{ with } P(a,x,y)$$

for any input a and any oracle.

Theorem (Rough statement)

There is a computable functional witnessing the n.c.i. of $A :\equiv \forall a \exists x \forall y P(a,x,y)$ iff there is a state machine which computes A.

Proof.

$\Leftarrow$: Define $\Phi(a,f) := x$ where

$$(c_0, a, x_0, \square) \rhd^* (c, a, x, y)$$

is a computation on oracle f (*there are a few additional details*).

$\Rightarrow$: There is an oracle Turing machine, and hence state machine, which simulates Φ. $\square$

The no-counterexample interpretation

A state machine computes a Π_3 formula $\forall a \in A \exists x \in X \forall y \in Y P(a,x,y)$ if

$$(c_0, a, x_0, \square) \rhd^* (c \in C^!, a, x, y) \text{ with } P(a,x,y)$$

for any input a and any oracle.

Theorem (Rough statement)

There is a computable functional witnessing the n.c.i. of $A :\equiv \forall a \exists x \forall y P(a,x,y)$ *iff there is a state machine which computes* A*.*

Proof.

$\Leftarrow$: Define $\Phi(a,f) := x$ where

$$(c_0, a, x_0, \square) \rhd^* (c, a, x, y)$$

is a computation on oracle f (*there are a few additional details*).

$\Rightarrow$: There is an oracle Turing machine, and hence state machine, which simulates Φ. $\square$

Idea. State machines make certain properties of the functional explicit.

So how do we extract algorithms from proofs?

$$
\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[r] \arrow[d, rightsquigarrow] & \boxed{\textit{Abstract State Machine}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, rightsquigarrow]
\end{tikzcd}
$$

So how do we extract algorithms from proofs?

$$\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[r] \arrow[d, rightsquigarrow] & \boxed{\textit{Abstract State Machine}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, rightsquigarrow]
\end{tikzcd}$$

We have several options:

- Write down an algorithm directly (works well for easy proofs or clever people).

So how do we extract algorithms from proofs?

$$
\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[r] \arrow[d, squiggly] & \boxed{\textit{Abstract State Machine}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, squiggly]
\end{tikzcd}
$$

We have several options:

- Write down an algorithm directly (works well for easy proofs or clever people).
- Convert a formally extracted program into a suitable ASM (might still be difficult to understand).

So how do we extract algorithms from proofs?

$$\begin{tikzcd}
\boxed{\text{Textbook Proof}} \arrow[r] \arrow[d, squiggly] & \boxed{\textit{Abstract State Machine}} \\
\boxed{\text{Formal Proof}} \arrow[r] & \boxed{\text{Formal Program}} \arrow[u, squiggly]
\end{tikzcd}$$

We have several options:

- Write down an algorithm directly (works well for easy proofs or clever people).
- Convert a formally extracted program into a suitable ASM (might still be difficult to understand).
- Develop some operations on algorithms which reflect **key mathematical lemmas**, which allow us to convert simple machines into more complicated ones.

A machine based interpretation of dependent choice

Suppose we have a machine $(S, \triangleright)$ with $S \subseteq C \times X^* \times X \times Y_\square$ which computes

$$\forall a \in X^* \exists x \in X \forall y \in Y \; P(a, x, y).$$

We want to convert this to a machine $(S^\star, \blacktriangleright)$ which computes

$$\exists f^{\mathbb{N} \to X} \forall n, y P(\bar{f} n, f(n), y).$$

A machine based interpretation of dependent choice

Suppose we have a machine $(S, \triangleright)$ with $S \subseteq C \times X^* \times X \times Y_\Box$ which computes

$$\forall a \in X^* \exists x \in X \forall y \in Y \ P(a, x, y).$$

We want to convert this to a machine $(S^\star, \blacktriangleright)$ which computes

$$\exists f^{\mathbb{N} \to X} \forall n, y P(\bar{f}n, f(n), y).$$

States $S^\star$ have the form

$$(\underbrace{\ \sigma\ }_{\text{control}}, \underbrace{a \mid b}_{\text{query}}, \underbrace{n, y}_{\text{answers}}) \subseteq C^* \times (X^* \times X^*_\Box) \times (\mathbb{N}_\Box \times Y_\Box)$$

A machine based interpretation of dependent choice

Suppose we have a machine $(S, \triangleright)$ with $S \subseteq C \times X^* \times X \times Y_\Box$ which computes

$$\forall a \in X^* \exists x \in X \forall y \in Y \; P(a, x, y).$$

We want to convert this to a machine $(S^\star, \blacktriangleright)$ which computes

$$\exists f^{\mathbb{N}\to X} \forall n, y P(\bar{f} n, f(n), y).$$

States $S^\star$ have the form

$$(\underbrace{\sigma}_{\text{control}}, \underbrace{a \mid b}_{\text{query}}, \underbrace{n, y}_{\text{answers}}) \subseteq C^* \times (X^* \times X^*_\Box) \times (\mathbb{N}_\Box \times Y_\Box)$$

- The main object being computed is a pair of finite sequences $a \mid b$ which represent the choice sequence $a :: b :: 0, 0, \ldots$. We view b as the current 'completion' of a.

A machine based interpretation of dependent choice

Suppose we have a machine $(S, \triangleright)$ with $S \subseteq C \times X^* \times X \times Y_\square$ which computes

$$\forall a \in X^* \exists x \in X \forall y \in Y \; P(a, x, y).$$

We want to convert this to a machine $(S^\star, \blacktriangleright)$ which computes

$$\exists f^{\mathbb{N} \to X} \forall n, y P(\bar{f} n, f(n), y).$$

States $S^\star$ have the form

$$(\underbrace{\sigma}_{\text{control}}, \underbrace{a \mid b}_{\text{query}}, \underbrace{n, y}_{\text{answers}}) \subseteq C^* \times (X^* \times X^*_\square) \times (\mathbb{N}_\square \times Y_\square)$$

- The main object being computed is a pair of finite sequences $a \mid b$ which represent the choice sequence $a :: b :: 0, 0, \ldots$. We view b as the current 'completion' of a.

- We now have two oracles, which take the approximation $a \mid b$ and return the desired length and depth respectively, which eventually need to be satisfied by our approximation.

You don't need to understand this!

If $(c, a, x, \square)$ a query state then

You don't need to understand this!

If $(c, a, x, \square)$ a query state then

- $(\sigma :: c, a :: x \mid \square, \square, \square) \blacktriangleright (\sigma :: c, a :: x \mid \square, n, \square)$ $\boxed{\text{check length}}$

You don't need to understand this!

If $(c, a, x, \square)$ a query state then

- $(\sigma :: c, a :: x \mid \square, \square, \square) \blacktriangleright (\sigma :: c, a :: x \mid \square, n, \square)$ $\boxed{\text{check length}}$

- If $n < |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, \square)$ $\boxed{\text{length good}}$

You don't need to understand this!

If $(c, a, x, \square)$ a query state then

- $(\sigma :: c, a :: x \mid \square, \square, \square) \blacktriangleright (\sigma :: c, a :: x \mid \square, n, \square)$ $\boxed{\text{check length}}$

- If $n < |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, \square)$ $\boxed{\text{length good}}$

- If $n \geq |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c :: c_0, a :: x :: x_0 \mid \square, \square, \square)$ $\boxed{\text{length bad}}$

You don't need to understand this!

If $(c, a, x, \square)$ a query state then

- $(\sigma :: c, a :: x \mid \square, \square, \square) \blacktriangleright (\sigma :: c, a :: x \mid \square, n, \square)$ $\boxed{\text{check length}}$

- If $n < |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, \square)$ $\boxed{\text{length good}}$

- If $n \geq |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c :: c_0, a :: x :: x_0 \mid \square, \square, \square)$ $\boxed{\text{length bad}}$

- $(\sigma :: c, a :: x \mid [], n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, y)$ $\boxed{\text{check depth}}$

You don't need to understand this!

If $(c, a, x, \square)$ a query state then

- $(\sigma :: c, a :: x \mid \square, \square, \square) \blacktriangleright (\sigma :: c, a :: x \mid \square, n, \square)$ $\boxed{\text{check length}}$

- If $n < |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, \square)$ $\boxed{\text{length good}}$

- If $n \geq |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c :: c_0, a :: x :: x_0 \mid \square, \square, \square)$ $\boxed{\text{length bad}}$

- $(\sigma :: c, a :: x \mid [], n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, y)$ $\boxed{\text{check depth}}$

If $(c, a, x, y) \triangleright (c', a, x', y/\square)$ then

You don't need to understand this!

If $(c, a, x, \square)$ a query state then

- $(\sigma :: c, a :: x \mid \square, \square, \square) \blacktriangleright (\sigma :: c, a :: x \mid \square, n, \square)$ $\boxed{\text{check length}}$

- If $n < |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, \square)$ $\boxed{\text{length good}}$

- If $n \geq |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c :: c_0, a :: x :: x_0 \mid \square, \square, \square)$ $\boxed{\text{length bad}}$

- $(\sigma :: c, a :: x \mid [], n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, y)$ $\boxed{\text{check depth}}$

If $(c, a, x, y) \triangleright (c', a, x', y/\square)$ then

- $(\sigma :: c, a :: x \mid b, n, y) \blacktriangleright (\sigma :: c', a :: x' \mid b/\square, n/\square, y/\square)$ $\boxed{\text{compute element}}$

You don't need to understand this!

If $(c, a, x, \square)$ a query state then

- $(\sigma :: c, a :: x \mid \square, \square, \square) \blacktriangleright (\sigma :: c, a :: x \mid \square, n, \square)$ $\boxed{\text{check length}}$

- If $n < |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, \square)$ $\boxed{\text{length good}}$

- If $n \geq |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c :: c_0, a :: x :: x_0 \mid \square, \square, \square)$ $\boxed{\text{length bad}}$

- $(\sigma :: c, a :: x \mid [], n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, y)$ $\boxed{\text{check depth}}$

If $(c, a, x, y) \triangleright (c', a, x', y/\square)$ then

- $(\sigma :: c, a :: x \mid b, n, y) \blacktriangleright (\sigma :: c', a :: x' \mid b/\square, n/\square, y/\square)$ $\boxed{\text{compute element}}$

If (c, a, x, y) an end state then

You don't need to understand this!

If $(c, a, x, \square)$ a query state then

- $(\sigma :: c, a :: x \mid \square, \square, \square) \blacktriangleright (\sigma :: c, a :: x \mid \square, n, \square)$ $\boxed{\text{check length}}$

- If $n < |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, \square)$ $\boxed{\text{length good}}$

- If $n \geq |a|$ then $(\sigma :: c, a :: x \mid \square, n, \square) \blacktriangleright (\sigma :: c :: c_0, a :: x :: x_0 \mid \square, \square, \square)$ $\boxed{\text{length bad}}$

- $(\sigma :: c, a :: x \mid [], n, \square) \blacktriangleright (\sigma :: c, a :: x \mid [], n, y)$ $\boxed{\text{check depth}}$

If $(c, a, x, y) \triangleright (c', a, x', y/\square)$ then

- $(\sigma :: c, a :: x \mid b, n, y) \blacktriangleright (\sigma :: c', a :: x' \mid b/\square, n/\square, y/\square)$ $\boxed{\text{compute element}}$

If (c, a, x, y) an end state then

- $(\sigma :: c, a :: x \mid b, n, y) \blacktriangleright (\sigma, a \mid x :: b, n, y)$ $\boxed{\text{element computed}}$

The following are quite easy to prove

Theorem (Rough statement)

We have

$$([],[] \mid \square,\square,\square) \blacktriangleright^* ([],[] \mid b,n,y)$$

with

- $n < |b|$;
- $\forall m < |b|\ P(\bar{b}m, b(m), y)$.

The following are quite easy to prove

Theorem (Rough statement)

We have

$$([], [] \mid \square, \square, \square) \blacktriangleright^* ([], [] \mid b, n, y)$$

with

- $n < |b|$;
- $\forall m < |b| \; P(\bar{b}m, b(m), y)$.

Corollary

The machine $(S^\star, \blacktriangleright)$ *with start state* $([], [] \mid \square, \square, \square)$ *computes*

$$\exists f \forall n, y \; P(\bar{f}n, f(n), y).$$

A real example

Theorem

Let R be a commutative ring with $0 \neq 1$. Suppose that r lies in the intersection of all prime ideals of R. Then r is nilpotent i.e. $\exists e > 0 (r^e = 0)$.

A real example

Theorem

Let R be a commutative ring with $0 \neq 1$. Suppose that r lies in the intersection of all prime ideals of R. Then r is nilpotent i.e. $\exists e > 0(r^e = 0)$.

Proof.

Suppose that r is not nilpotent. Define

$$\Sigma := \{I \subset R \mid I \text{ is an ideal satisfying } \forall e > 0(r^e \notin I)\}.$$

Then $\{0\} \in \Sigma$ (by our assumption), and Σ is chain-complete w.r.t. inclusion, so by Zorn's lemma it has a maximal element M.

A real example

Theorem

Let R be a commutative ring with $0 \neq 1$. Suppose that r lies in the intersection of all prime ideals of R. Then r is nilpotent i.e. $\exists e > 0 (r^e = 0)$.

Proof.

Suppose that r is not nilpotent. Define

$$\Sigma := \{I \subset R \mid I \text{ is an ideal satisfying } \forall e > 0 (r^e \notin I)\}.$$

Then $\{0\} \in \Sigma$ (by our assumption), and Σ is chain-complete w.r.t. inclusion, so by Zorn's lemma it has a maximal element M.

We show that M is prime: If $m, n \notin M$ then $M + (m)$ and $M + (n)$ are proper extensions of M, so by maximality there exist $e_1, e_2 > 0$ such that $r^{e_1} \in M + (m)$ and $r^{e_2} \in M + (n)$. Therefore

$$r^{e_1+e_2} \in M + (mn)$$

and so $M + (mn) \notin \Sigma$, which means that $mn \notin M$. Since $r^1 \notin M$, r cannot lie in the intersection of all prime ideals. $\square$

A state machine which computes maximal ideals

For countable commutative rings $R := \{r_n \ : \ n \in \mathbb{N}\}$ (with w.l.o.g. $r_0 = 0$), this proof can be formalised using DC via a standard trick in reverse math.

A state machine which computes maximal ideals

For countable commutative rings $R := \{r_n \ : \ n \in \mathbb{N}\}$ (with w.l.o.g. $r_0 = 0$), this proof can be formalised using DC via a standard trick in reverse math.

There is a corresponding state machine:

$$(\sigma, \underbrace{a \mid b}_{\text{maximal } M \in \Sigma}, \underbrace{n, \langle [y_1, \ldots, y_k, y], e \rangle}_{m_1 y_1 + \ldots + m_k y_k + r_n y = r^e}) \text{ with } a(i) = \langle \underbrace{\chi(i)}_{\mathbb{B}}, \underbrace{\vec{y}(i)}_{R^*}, \underbrace{e(i)}_{\mathbb{N}_{>0}} \rangle$$

A state machine which computes maximal ideals

For countable commutative rings $R := \{r_n \ : \ n \in \mathbb{N}\}$ (with w.l.o.g. $r_0 = 0$), this proof can be formalised using DC via a standard trick in reverse math.

There is a corresponding state machine:

$$(\sigma, \underbrace{a \mid b}_{\text{maximal } M \in \Sigma}, \underbrace{n, \langle [y_1, \ldots, y_k, y], e \rangle}_{m_1 y_1 + \ldots + m_k y_k + r_n y = r^e}) \text{ with } a(i) = \langle \underbrace{\chi(i)}_{\mathbb{B}}, \underbrace{\vec{y}(i)}_{R^*}, \underbrace{e(i)}_{\mathbb{N}_{>0}} \rangle$$

- Either $\chi(i) = 1$ and $r_i \in M$, or $\chi(i) = 0$ and $r_i \notin M$, in which case $\langle \vec{y}(i), e(i) \rangle$ act as *evidence* for the exclusion of r_i i.e.

$$m_1 y(i)_1 + \ldots + m_k y(i)_k + r_i y(i) = r^{e(i)}$$

A state machine which computes maximal ideals

For countable commutative rings $R := \{r_n \;:\; n \in \mathbb{N}\}$ (with w.l.o.g. $r_0 = 0$), this proof can be formalised using DC via a standard trick in reverse math.

There is a corresponding state machine:

$$(\sigma, \underbrace{a \mid b}_{\text{maximal } M \in \Sigma}, \underbrace{n, \langle [y_1, \ldots, y_k, y], e\rangle}_{m_1 y_1 + \ldots + m_k y_k + r_n y = r^e}) \text{ with } a(i) = \langle \underbrace{\chi(i)}_{\mathbb{B}}, \underbrace{\vec{y}(i)}_{R^*}, \underbrace{e(i)}_{\mathbb{N}_{>0}} \rangle$$

- Either $\chi(i) = 1$ and $r_i \in M$, or $\chi(i) = 0$ and $r_i \notin M$, in which case $\langle \vec{y}(i), e(i)\rangle$ act as *evidence* for the exclusion of r_i i.e.

$$m_1 y(i)_1 + \ldots + m_k y(i)_k + r_i y(i) = r^{e(i)}$$

- Oracle queries provide evidence that for a given M encoded by $a \mid b$, we have

$$r \in M \vee M \text{ not a prime ideal}$$

which is converted into evidence for excluding r_n for some $n \in \mathbb{N}$.

A constructive proof

Theorem

$$([], \underbrace{[] \mid \square}_{M:=R}, \square, \square) \blacktriangleright^* ([], \underbrace{[] \mid b}_{M\subset R}, n, \langle [y_1, \ldots, y_k, y], e\rangle)$$

where $r_0 = 0 \notin M$. *In other words,* $b(0) = \langle 0, [y_0], e(0)\rangle$ *with* $e(0) > 0$ *and*

$$r^{e(0)} = r_0 y_0 = 0.$$

A constructive proof

Theorem

$$([\,], \underbrace{[\,] \mid \square}_{M:=R}, \square, \square) \blacktriangleright^* ([\,], \underbrace{[\,] \mid b}_{M \subset R}, n, \langle [y_1, \ldots, y_k, y], e \rangle)$$

where $r_0 = 0 \notin M$. *In other words,* $b(0) = \langle 0, [y_0], e(0) \rangle$ *with* $e(0) > 0$ *and*

$$r^{e(0)} = r_0 y_0 = 0.$$

- **Classical.** There exists a maximal element $M \in \Sigma$, hence $r^e = 0$ for some $e > 0$ by contradiction.

A constructive proof

Theorem

$$([], \underbrace{[] \mid \Box}_{M:=R}, \Box, \Box) \blacktriangleright^* ([], \underbrace{[] \mid b}_{M\subset R}, n, \langle [y_1, \ldots, y_k, y], e\rangle)$$

where $r_0 = 0 \notin M$. *In other words,* $b(0) = \langle 0, [y_0], e(0)\rangle$ *with* $e(0) > 0$ *and*

$$r^{e(0)} = r_0 y_0 = 0.$$

- **Classical.** There exists a maximal element $M \in \Sigma$, hence $r^e = 0$ for some $e > 0$ by contradiction.

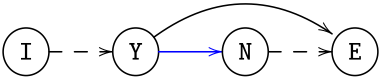
- **Computational.** There exists a machine with which, relative to an oracle witnessing that $r \in P$ for all prime ideals P, builds an approximation to a maximal $M \in \Sigma$ by starting with R and gradually excluding elements:

$$R = M_0 \supset M_1 \supset M_2 \supset \ldots \supset M_k = \text{‘maximal’}$$

Eventually 0 is excluded, hence we have found some $e > 0$ with $r^e = 0$.

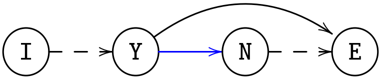
We can even generate a diagram of the control flow

Input machine $\mathcal{M}$ for single elements:

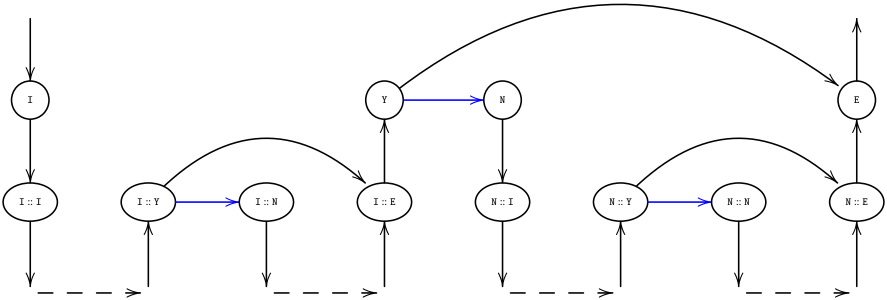


We can even generate a diagram of the control flow

Input machine $\mathcal{M}$ for single elements:



Output machine $\mathcal{M}^\star$ for whole set:



Open questions

This is work in progress with plenty of open questions. In particular:

Open questions

This is work in progress with plenty of open questions. In particular:

- Can we improve traditional techniques by understanding how they act as algorithms?

Open questions

This is work in progress with plenty of open questions. In particular:

- Can we improve traditional techniques by understanding how they act as algorithms?
- Large scale: Can we better understand computational content of complicated non-constructive proofs in e.g. WQO theory?

Open questions

This is work in progress with plenty of open questions. In particular:

- Can we improve traditional techniques by understanding how they act as algorithms?
- Large scale: Can we better understand computational content of complicated non-constructive proofs in e.g. WQO theory?
- Small scale: Can we automatically extract machines which implement e.g. sorting algorithms from proofs?

Open questions

This is work in progress with plenty of open questions. In particular:

- Can we improve traditional techniques by understanding how they act as algorithms?
- Large scale: Can we better understand computational content of complicated non-constructive proofs in e.g. WQO theory?
- Small scale: Can we automatically extract machines which implement e.g. sorting algorithms from proofs?
- We focused on the n.c.i. interpretation. What about the full functional interpretations i.e. oracle machines in all finite types? What would be a suitable computational model for types > 2.

Open questions

This is work in progress with plenty of open questions. In particular:

- Can we improve traditional techniques by understanding how they act as algorithms?
- Large scale: Can we better understand computational content of complicated non-constructive proofs in e.g. WQO theory?
- Small scale: Can we automatically extract machines which implement e.g. sorting algorithms from proofs?
- We focused on the n.c.i. interpretation. What about the full functional interpretations i.e. oracle machines in all finite types? What would be a suitable computational model for types > 2.
- Can we give a formal geometric characterisation of what's going on via some kind of graphs?

Open questions

This is work in progress with plenty of open questions. In particular:

- Can we improve traditional techniques by understanding how they act as algorithms?
- Large scale: Can we better understand computational content of complicated non-constructive proofs in e.g. WQO theory?
- Small scale: Can we automatically extract machines which implement e.g. sorting algorithms from proofs?
- We focused on the n.c.i. interpretation. What about the full functional interpretations i.e. oracle machines in all finite types? What would be a suitable computational model for types > 2.
- Can we give a formal geometric characterisation of what's going on via some kind of graphs?

THANK YOU!