NICTA

Mechanised Computability Theory

Michael Norrish

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Motivation

$(x, A_1)::\Gamma \vdash_\Sigma y : B_1$ and $(x, A_1)::\Gamma \vdash_\Sigma A_2 = B_2[y:=x] : type$. Moreover, in this case we cannot use the strong version of the inversion lemma to avoid this problem, because x is already in use in the context.

Although their proof looks rigorous and detailed, here Harper and Pfenning appear to employ implicit "without loss of generality" reasoning about inversion and renaming that is not easy to formalize directly. Instead we needed to carefully show that:

LEMMA 39. *If* $(x, A_1)::\Gamma \vdash_\Sigma M\, x : A_2$ *and* $x \# M$ *then* $\Gamma \vdash_\Sigma M : \Pi x{:}A_1.\, A_2$.

PROOF. The proof proceeds by applying validity and inversion principles, as already discussed. One subtle freshness side-condition is the fact that x is fresh for $\Pi y{:}B_1.\, B_2$, and this is proved by translating to the algorithmic typechecking system and using Lemma 38. □

Strong extensionality then follows essentially as in HP05 , using Lemma 39 to fill the gap we identified.

THEOREM 11 (STRONG EXTENSIONALITY). *If* $(x, A_1)::\Gamma \vdash_\Sigma Mx = Nx : A_2$ *and* $x \# (M, N)$ *then* $\Gamma \vdash_\Sigma M = N : \Pi x{:}A_1.\, A_2$.

Mechanizing the Metatheory of LF.

Urban, Cheney, Berghofer. ACM Transactions on Computational Logic, 12:2, 2011.

3.7 Decidability

HP05 also sketches proofs of the decidability of the algorithmic judgments (and hence also the definitional system). Reasoning about decidability within Isabelle/HOL is not straightforward because Isabelle/HOL is based on classical logic. Thus, unlike constructive logics or type theories, we cannot infer decidability of P simply by proving $P \vee \neg P$. Furthermore, given a relation R definable in Isabelle/HOL, it is not clear how best to formalize the informal statement "R is decidable."

As a sanity check, we have shown that weak head reduction is strongly normalizing for well-formed terms. We write $M\Downarrow$ to indicate that M is strongly normalizing under weak head reduction. This proof uses techniques and definitions from the example formalization of strong normalization for the simply-typed lambda calculus in the Nominal Datatype Package.

THEOREM 12. *If* $\Gamma \vdash_\Sigma M : A$ *then* $M\Downarrow$.

PROOF. We first show the standard property that if $M\,N\Downarrow$ then $M\Downarrow$. We then show that if $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$, then $M\Downarrow$ by induction on derivations. The main result follows by reflexivity and Theorem 1. □

Turning now to the issue of formalizing decidability properties in Isabelle/HOL, we considered the following options.

Formalizing computability theory. It should be possible to define Turing machines (or some other universal model of computation) within Isabelle/HOL and derive enough of the theory of computation to be able to prove that the algorithmic equivalence and typechecking relations are decidable. It appears to be an open question how to formalize proofs of decidability in Isabelle/HOL, especially for algorithms over complex data structures such as nominal datatypes.

Motivation

Mechanizing the Metatheory of LF.

Urban, Cheney, Berghofer. ACM Transactions on Computational Logic, 12:2, 2011.

$(x, A_1)::\Gamma \vdash_\Sigma y : B_1$ and $(x, A_1)::\Gamma \vdash_\Sigma A_2 = B_2[y:=x] : type$. Moreover, in this case we cannot use the strong version of the inversion lemma to avoid this problem, because x is already in use in the context.

Although their proof looks rigorous and detailed, here Harper and Pfenning appear to employ implicit "without loss of generality" reasoning about inversion and renaming that is not easy to formalize directly. Instead we needed to carefully show that:

LEMMA 39. *If* $(x, A_1)::\Gamma \vdash_\Sigma M\, x : A_2$ *and* $x \# M$ *then* $\Gamma \vdash_\Sigma M : \Pi x{:}A_1.\, A_2$.

PROOF. The proof proceeds by applying validity and inversion principles, as already dis… …dition is the fact that x is fres… for $\Pi y{:}B_1$. … system and …

Strong ext… fill the gap …

THEOREM … (M, N) *the* …

3.7 Decida…

HP05 also … hence als… abelle/HO… logic. Thu… ability of … in Isabelle… "R is decidable."

hence also the definitional system). Reasoning about decidability within Isabelle/HOL is not straightforward because Isabelle/HOL is based on classical logic. Thus, unlike constructive logics or type theories, we cannot infer decidability of P simply by proving $P \vee \neg P$. Furthermore, given a relation R definable in Isabelle/HOL, it is not clear how best to formalize the informal statement "R is decidable."

As a sanity check, we have shown that weak head reduction is strongly normalizing for well-formed terms. We write $M\Downarrow$ to indicate that M is strongly normalizing under weak head reduction. This proof uses techniques and definitions from the example formalization of strong normalization for the simply-typed lambda calculus in the Nominal Datatype Package.

THEOREM 12. *If* $\Gamma \vdash_\Sigma M : A$ *then* $M\Downarrow$.

PROOF. We first show the standard property that if $M\,N\Downarrow$ then $M\Downarrow$. We then show that if $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$, then $M\Downarrow$ by induction on derivations. The main result follows by reflexivity and Theorem 1. □

Turning now to the issue of formalizing decidability properties in Isabelle/HOL, we considered the following options.

Formalizing computability theory. It should be possible to define Turing machines (or some other universal model of computation) within Isabelle/HOL and derive enough of the theory of computation to be able to prove that the algorithmic equivalence and typechecking relations are decidable. It appears to be an open question how to formalize proofs of decidability in Isabelle/HOL, especially for algorithms over complex data structures such as nominal datatypes.

Motivation

Mechanizing the Metatheory of LF.

Urban, Cheney, Berghofer. ACM Transactions on Computational Logic, 12:2, 2011.

$(x, A_1)::\Gamma \vdash_\Sigma y : B_1$ and $(x, A_1)::\Gamma \vdash_\Sigma A_2 = B_2[y:=x] : type$. Moreover, in this case we cannot use the strong version of the inversion lemma to avoid this problem, because x is already in use in the context.

Although their proof looks rigorous and detailed, here Harper and Pfenning appear to employ implicit "without loss of generality" reasoning about inversion and renaming that is not easy to formalize directly. Instead we needed to carefully show that:

LEMMA 39. *If* $(x, A_1)::\Gamma \vdash_\Sigma M\, x : A_2$ *and* $x \# M$ *then* $\Gamma \vdash_\Sigma M : \Pi x{:}A_1.\, A_2$.

PROOF. The proof proceeds by applying validity and inversion principles, as already dis… …btle fres… …dition is the fact that x is fres… for $\Pi y{:}B_1$. … system and …

Strong ext… fill the gap …

THEOREM … (M, N) *the* …

3.7 Decida…

HP05 also … hence als… abelle/HC… logic. Thu… ability of … in Isabelle… "R is decidable."

As a sanity check, we have shown that weak head reduction is strongly normalizing for well-formed terms. We write $M\downarrow$ to indicate that M is strongly normalizi… nitions fro… typed lam…

THEOREM …

PROOF. … show tha… result fo…

Turnin… abelle/HC…

Formal… machine… and deri… rithmic … an open … cially for algorithms over complex data structures such as nominal datatypes.

hence also the definitional system). Reasoning about decidability within Isabelle/HOL is not straightforward because Isabelle/HOL is based on classical logic. Thus, unlike constructive logics or type theories, we cannot infer decidability of P simply by proving $P \vee \neg P$. Furthermore, given a relation R definable in Isabelle/HOL, it is not clear how best to formalize the informal statement "R is decidable."

Formalizing computability theory. It should be possible to define Turing machines (or some other universal model of computation) within Isabelle/HOL and derive enough of the theory of computation to be able to prove that the algorithmic equivalence and typechecking relations are decidable. It appears to be an open question how to formalize proofs of decidability in Isabelle/HOL, especially for algorithms over complex data structures such as nominal datatypes.

A Top-Down Exposition

We're not quite at the point of being able to do the *Mechanizing LF* example.

So:

- What can we do?
- And how do we get there?

Goals and Ambitions

NICTA

Some basic results from standard computability theory:

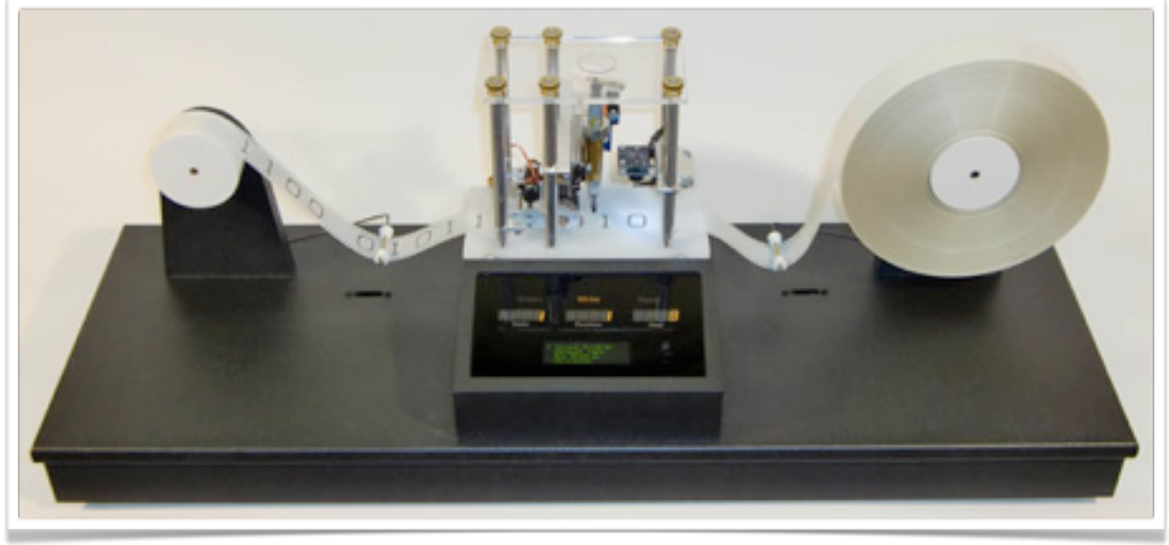
- recursive and recursively enumerable (r.e.) sets
- undecidability results, such as Rice's Theorem
- existence of Universal Machines

Question 1: What Model?

NICTA

Turing Machines

- yuck. Fiddly to define, even fiddlier constructions required to do basic arithmetic and recursion.



Register Machines

- slightly less yuck.
- But still fiddly.
- Some existing work: Zammit's PhD showed that register machines could compute the recursive functions

Question 1: What Model?

NICTA

Recursive functions

- That is: zero, successor, projection, composition, primitive recursion, and minimisation
- Clean!
- Related work:
  - Harrison and O'Connor used recursive functions in proofs of Gödel Incompleteness
  - Paulson and Szasz mechanised proof that Ackermann is not primitive recursive

Recursive Functions

But all is not rosy.

One of our desired results is

$$\text{r.e.}(S_1) \wedge \text{r.e.}(S_2) \Rightarrow \text{r.e.}(S_1 \cup S_2)$$

The argument is that there is a machine that can **dovetail** the machines enumerating S_1 and S_2.

Dovetailing Rec. Functions

NICTA

To dovetail functions, you have to be able to run them for a fixed number of “steps”.

Recursive functions only work over $\mathbb{N}$; we would have to

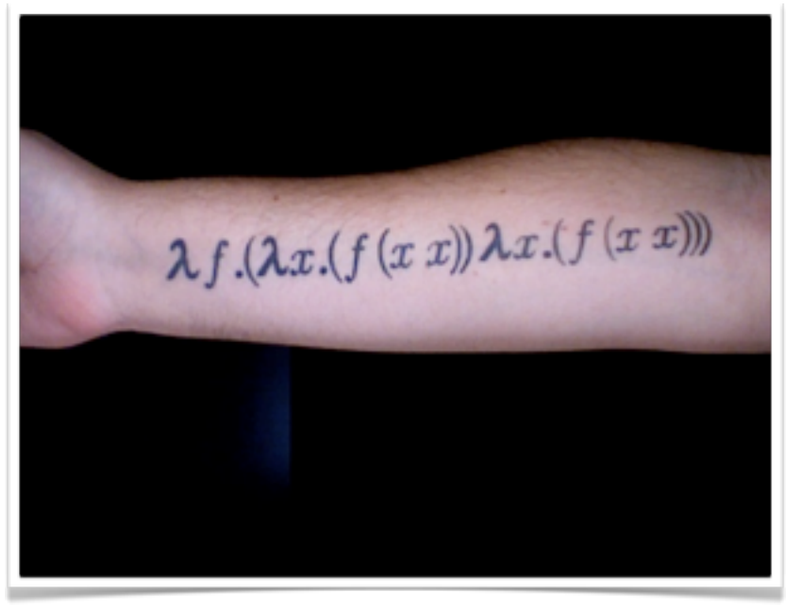
- encode all functions as numbers
- write an “step-counting” interpreter for them
- and do it all as a recursive function
- *i.e.,* a Universal Machine for recursive functions...

Question 1: What Model?

NICTA

The λ-Calculus

- Clean and expressive
- Already have extensive mechanisation in HOL4
- Know how to do recursive functions:
  - Church numerals
  - Y combinator for minimisation



Computing with λ-Terms

NICTA

Basic problem is non-determinism.

Luckily, normal order reduction guarantees finding of normal forms:

$$\frac{}{(\lambda v.\ M) \diamond N \ \rightarrow_n \ M[v := N]} \qquad \frac{M_1 \ \rightarrow_n \ M_2 \quad \neg \texttt{is\_abs}\ M_1}{M_1 \diamond N \ \rightarrow_n \ M_2 \diamond N}$$

$$\frac{M_1 \ \rightarrow_n \ M_2}{(\lambda v.\ M_1) \ \rightarrow_n \ (\lambda v.\ M_2)}$$

Computing with λ-Terms

Basic problem is non-determinism.

Luckily, normal order reduction guarantees finding of normal forms:

$$\frac{}{(\lambda v.\ M) \diamond N \ \rightarrow_n \ M[v := N]} \qquad \frac{M_1 \ \rightarrow_n \ M_2 \quad \neg\texttt{is\_abs}\ M_1}{M_1 \diamond N \ \rightarrow_n \ M_2 \diamond N}$$

$$\frac{M_1 \ \rightarrow_n \ M_2}{(\lambda v.\ M_1) \ \rightarrow_n \ (\lambda v.\ M_2)} \qquad \frac{N_1 \ \rightarrow_n \ N_2 \quad \texttt{bnf}\ M \quad \neg\texttt{is\_abs}\ M}{M \diamond N_1 \ \rightarrow_n \ M \diamond N_2}$$

We Can "Run" λ-Terms

NICTA

Within the logic:

```
while (t not in beta-normal form) do
  t := normal-order-reduct-of ( t )
```

Can prove that this "computes" the normal form of a term, if it has one.

Still need to show this is really computable.

We Can "Run" λ-Terms

NICTA

Within the logic:

```
while (t not in beta-normal form) do
  t := normal-order-reduct-of ( t )
```

Can prove that this "computes" the normal form of a term, if it ha

DEPENDS ON PROOF OF THE STANDARDISATION THEOREM

Still need to show this is really computable.

Running λ-Terms Inside Themselves?

NICTA

Recall: we have to do this to be able to build the Universal Machine....

Running λ-Terms Inside Themselves?

NICTA

1. How do we represent λ-terms inside themselves?
   - Use de Bruijn terms!

2. Huh?
   - de Bruijn terms are an algebraic type; we can "Church encode" them just like numbers, pairs and lists

“Church” de Bruijn Terms

NICTA

$(\lambda x.\ x\ y) \quad \approx \quad (\mathrm{dLAM}\ (\mathrm{dAPP}\ (\mathrm{dV}\ 0)\ (\mathrm{dV}\ 1))$

$\approx \quad (\lambda v\ c\ a.\ a\ (c\ (v\ \ulcorner 0 \urcorner)\ (v\ \ulcorner 1 \urcorner)))$

On top of this foundation, write computable functions to perform:

- substitution
- redex-finding
- perform *n* normal order reduction steps

Thus, a Universal Machine

NICTA

$\Phi\ m\ n$ — In-logic calculation of bnf of machine m applied to n

$\mathtt{UM} \diamond \ulcorner m \otimes n \urcorner$ — λ-term taking a Church-encoded pair of m and n

$$\vdash \Phi\ m\ n = \mathtt{NONE} \iff \mathtt{bnf\_of}\ (\mathtt{UM} \diamond \ulcorner m \otimes n \urcorner) = \mathtt{NONE}$$

$$\vdash \Phi\ m\ n = \mathtt{SOME}(p) \iff \mathtt{bnf\_of}\ (\mathtt{UM} \diamond \ulcorner m \otimes n \urcorner) = \mathtt{SOME}(\ulcorner p \urcorner)$$

Thus, a Universal Machine

NICTA

$\Phi\ m\ n$ — In-logic calculation of bnf of machine m applied to n

$\mathtt{UM} \diamond \ulcorner m \otimes n \urcorner$ — λ-term taking a Church-encoded pair of m and n

$\vdash \Phi\ m\ n = \mathtt{NONE} \iff \mathtt{bnf\_c}$

$\vdash \Phi\ m\ n = \mathtt{SOME}(p) \iff$

$\mathtt{bnf\_of}\ (\mathtt{UM} \diamond \ulcorner m \otimes n \urcorner) = \mathtt{SOME}(\ulcorner p \urcorner)$

Depends on proof of the Isomorphism of de Bruijn and "normal" λ-terms

Thus, Desired Results

Or at least a selection thereof:

$$\vdash \texttt{recursive}\ s \Rightarrow \texttt{re}\ s$$
$$\vdash \texttt{re}\ s \wedge \texttt{re}\ t \Rightarrow \texttt{re}\ (s \cap t)$$
$$\vdash \texttt{re}\ s \wedge \texttt{re}\ t \Rightarrow \texttt{re}\ (s \cup t)$$
$$\vdash \texttt{re}\ s \wedge \texttt{re}\ (\texttt{COMPL}\ s) \Rightarrow \texttt{recursive}\ s$$

Others include:

Rice's Theorem, Recursion Theorem...

Enter Paranoid Doubts

Proved results suggest mechanised maths is right.

But, we can make it yet more convincing.

Recursive Functions (II)

It is fairly straightforward to show that the λ-terms can implement the recursive functions.

- **Not** as easy as I expected though: the partiality introduced by minimisation is fiddly

What about the other way ‘round?

λ-Terms as Numbers

Recursive functions only manipulate numbers.

de Bruijn terms are a countable set.

Used this in Universal Machine construction

- UM took index into enumeration of all terms

Encoding dB Terms

NICTA

This is the invertible map from terms to numbers.

$\vdash$ dBnum (dV i) = 3 $\times$ i

$\vdash$ dBnum (dAPP M N) = 3 $\times$ (dBnum M $\otimes$ dBnum N) + 1

$\vdash$ dBnum (dABS M) = 3 $\times$ dBnum M + 2

The inverse uses (natural number) division and modulus.

Flavours of Recursion I

NICTA

Substitution is primitive recursive over the structure of terms.

When the term has been encoded as a number, the corresponding recursion is not primitive

- the recursive calls are to numbers that are much smaller (not just the predecessor).

Flavours of Recursion II

NICTA

Substitution on de Bruijn terms changes the other parameters when the recursion passes through an abstraction:

$$\texttt{nsub}\ M\ i\ (\texttt{dLAM}\ N) \quad = \quad \texttt{nsub}\ (\texttt{lift}\ M\ 0)\ (i+1)\ N$$

The primitive recursion allowed in Recursive Functions keeps other parameters unchanged...

Avoid Minimisation

We could use minimisation to implement these recursions.

By avoiding it, we show that all operations except the search for the normal form are **primitive recursive**.

... effectively Kleene's Normal Form theorem

More Desirable Results

Theorem 4. *There exists a recursive function* `recPhi` *of type*

$$\texttt{num list} \rightarrow \texttt{num option}$$

which emulates Φ*:*

$$\vdash \texttt{recfn recPhi 2}$$
$$\vdash \texttt{recPhi } [i; n] = \Phi\ i\ n$$

What of *Mechanizing LF* ?

NICTA

Recall:

> *Formalizing computability theory.* It should be possible to define Turing machines (or some other universal model of computation) within Isabelle/HOL and derive enough of the theory of computation to be able to prove that the algorithmic equivalence and typechecking relations are decidable. It appears to be an open question how to formalize proofs of decidability in Isabelle/HOL, especially for algorithms over complex data structures such as nominal datatypes.

λ-terms are not as complicated as LF terms.

Nonetheless they are a nominal datatype.

Conclusions

I have mechanised a pile of basic computability theory.

Classical, non-constructive systems (like the HOLs) now have a chance to reason about computability.

See the paper for many more technical details on how it was done.