Extraction of Programs from Proofs using Postulated Axioms

Anton Setzer
Swansea University, Swansea UK
(Joint work with Chi Ming Chuang)

April 11, 2011

Question by Ulrich Berger

- Can you extract programs from proofs in Agda.
- Obvious because of Axiom of Choice – ?
  From
  $$p : (x : A) \to \exists\, [y : B]\, \varphi(y)$$
  we get of course
  $$\begin{aligned} f &= \lambda x.\pi_0(f\ x) : A \to B \\ p &= \lambda x.\pi_1(f\ x) : (x : A) \to \varphi(f\ x) \end{aligned}$$
- However what happens in the presence of axioms?

Abstract Real Numbers

- Situation different in presence of axioms.
- Approach of Ulrich Berger transferred to Agda:
  Axiomatize the real numbers abstractly. E.g.

$$
\begin{array}{l}
\text{postulate } \mathbb{R} : \text{Set} \\
\text{postulate } \_ == \_ : \mathbb{R} \to \mathbb{R} \to \mathbb{R} \\
\text{postulate } \_ + \_ : \mathbb{R} \to \mathbb{R} \to \mathbb{R} \\
\text{postulate commutative} : (r\ s : \mathbb{R}) \to r + s == s + r \\
\ldots
\end{array}
$$

Computational Numbers

- Formulate $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ as usual

$$
\begin{aligned}
&\text{data } \mathbb{N} : \text{Set where} \\
&\quad \text{zero} \quad : \quad \mathbb{N} \\
&\quad \text{suc} \quad : \quad \mathbb{N} \to \mathbb{N}
\end{aligned}
$$

$$
\begin{aligned}
&\_ + \_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
&n \quad + \quad \text{zero} \quad = \quad n \\
&n \quad + \quad \text{suc } m \quad = \quad \text{suc } (n + m)
\end{aligned}
$$

$$
\begin{aligned}
&\_ * \_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
&\ldots
\end{aligned}
$$

$$
\begin{aligned}
&\text{data } \mathbb{Z} : \text{Set where} \\
&\quad \ldots
\end{aligned}
$$

$$
\text{data } \mathbb{Q} : \text{Set where}
$$

Embedding of $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ into $\mathbb{R}$

$$
\begin{array}{lll}
\mathbb{N}2\mathbb{R} : \mathbb{N} \to \mathbb{R} & & \\
\mathbb{N}2\mathbb{R} \quad \text{zero} & = & 0_{\mathbb{R}} \\
\mathbb{N}2\mathbb{R} \quad (\text{suc } n) & = & \mathbb{N}2\mathbb{R}\ n +_{\mathbb{R}} 1_{\mathbb{R}}
\end{array}
$$

$$
\mathbb{Z}2\mathbb{R} : \mathbb{Z} \to \mathbb{R}
$$

$\ldots$

$$
\mathbb{Q}2\mathbb{R} : \mathbb{Q} \to \mathbb{R}
$$

$\ldots$

Cauchy Reals

$$
\begin{aligned}
&\mathrm{data\ CauchyReal}\ (r : \mathbb{R}) : \mathrm{Set\ where} \\
&\quad \mathrm{cauchyReal} : (f : \mathbb{N} \to \mathbb{Q}) \\
&\qquad\qquad \to (p : (n : \mathbb{N}) \to |\mathbb{Q}2\mathbb{R}\ (f\ \ n) -_{\mathbb{R}} r|_{\mathbb{R}} <_{\mathbb{R}} 2_{\mathbb{R}}^{-n}) \\
&\qquad\qquad \to \mathrm{CauchyReal}\ r
\end{aligned}
$$

Signed Digit Representations

- We can consider as well the real numbers with signed digit representations.
- Signed digit representable real numbers in $[-1, 1]$ are of the form

$$0.111(-1)0(-1)01(-1)\cdots$$

  In general

$$0.d_0 d_1 d_2 d_3 \cdots$$

  where $d_i \in \{-1, 0, 1\}$.
- Signed digit needed because even the first digit of an unsigned digit representation can in general not be determined.

Signed Digit Representations

- Consider for easy of presentation decimal numbers.
- Assume a sequence of approximations of a real number, starting with

$$0.9, 0.99, 0.999, 0.9999, \cdots$$

  it might at any time switch to

$$1.0000001$$

  in which case first digits are 1.0
  or to

$$0.9999998$$

  in which case first digits are 0.9.
- With first digits 0.9 we can represent numbers in the interval

$$[0.9000000\cdots, 0.9999999\cdots] = [0.9, 1.0]$$

- With first digits 1.0 we can represent

$$[1.00000000\cdots, 1.09999999\cdots] = [1.0, 1.1]$$

Signed Digit Representations

- The choice between 0.9 and 1.0 is the choice

$$r \leq 1.0 \vee r \geq 1.0$$

  which is undecidable.
- With signed digits we can modify our decisions:
- With first digit 0.9 we can obtain numbers in interval

$$[0.9(-9)(-9)(-9)\cdots, 0.9999999\cdots] = [0.8, 1.0]$$

- With first digit 1.0 we can obtain numbers in interval

$$[1.0(-9)(-9)(-9)\cdots, 1.0999999\cdots] = [0.9, 1.1]$$

- The choice between 0.9 and 1.0 is the choice

$$r \leq 1.0 \vee r \geq 0.9$$

  which is decidable.

Coinductive Definition of Binary Signed Digit Real Numbers

$$
\begin{aligned}
&\text{data Digit : Set where} \\
&\quad -1_{\mathrm{d}}\ 0_{\mathrm{d}}\ 1_{\mathrm{d}} : \text{Digit}
\end{aligned}
$$

$$
\begin{aligned}
&\text{data SignedDigit} : \mathbb{R} \to \text{Set where} \\
&\quad \text{signedDigit} : (r : \mathbb{R}) \\
&\qquad \to (r \in [-1, 1]) \\
&\qquad \to (d : \text{Digit}) \\
&\qquad \to \infty\ (\text{SignedDigit}\ (2_{\mathbb{R}} *_{\mathbb{R}} r - \text{digit2}\mathbb{R}\ d)) \\
&\qquad \to \text{SignedDigit}\ r
\end{aligned}
$$

Conversion Functions

$$\text{cauchy2SignedDigit} : (r : \mathbb{R}) \to r \in [-1,1] \to \text{CauchyReal } r \\ \to \text{SignedDigit } r$$

...

$$\text{signedDigit2Cauchy} : (r : \mathbb{R}) \to \text{SignedDigit } r \to \text{CauchyReal } r$$

...

$$\text{signedDigit2Stream} : (r : \mathbb{R}) \to \text{SignedDigit } r \to \text{Stream Digit}$$

...

$$\text{streamToSignedDigit} : \text{Stream Digit} \to \exists\, [r : \mathbb{R}]\ (\text{SignedDigit } r)$$

...

– – Requires completeness axiom for $\mathbb{R}$

Conversion Functions

$\text{streamToList} : \{A : \text{Set}\} \to \text{Stream}\ A \to \mathbb{N} \to \text{List}\ A$

$--$ determine first n elements

$\ldots$

Generating Real Numbers

Prove:

$$\mathbb{Q}2\mathrm{Cauchy} : (q : \mathbb{Q}) \to \mathrm{CauchyReal}\ (\mathbb{Q}2\mathbb{R}\ q)$$
$$\ldots$$
$$\begin{aligned}\mathrm{closure}{+} : (r\ s : \mathbb{R}) &\to \mathrm{CauchyReal}\ r \to \mathrm{CauchyReal}\ s \\ &\to \mathrm{CauchyReal}\ (r + s)\end{aligned}$$
$$\ldots$$
$$\begin{aligned}\mathrm{closure}{*} : (r\ s : \mathbb{R}) &\to \mathrm{CauchyReal}\ r \to \mathrm{CauchyReal}\ s \\ &\to \mathrm{CauchyReal}\ (r * s)\end{aligned}$$
$$\ldots$$
$$\begin{aligned}\mathrm{cauchyComplete} : &(f : \mathbb{N} \to \mathbb{R}) \\ &(p : (n : \mathbb{N}) \to \mathrm{CauchyReal}\ (f\ n)) \\ &(q : (n\ m : \mathbb{N}) \to (n \geq m) \to |f\ n -_{\mathbb{R}} f\ m|_{\mathbb{R}} <_{\mathbb{R}} 2_{\mathbb{R}}^{-n}) \\ &\to \exists\ [r : \mathbb{R}]\ ((n : \mathbb{N}) \to |f\ n -_{\mathbb{R}} r|_{\mathbb{R}} \leq_{\mathbb{R}} 2_{\mathbb{R}}^{-n})\end{aligned}$$

Extraction of Programs

Plugging these functions we can now obtain

- Obtain a singed digit representation of rational numbers.

$$\mathrm{l} : (n : \mathbb{N}) \to \text{List Digit}$$
$$\mathrm{l}\ n = \mathbb{Q}\text{2ListDigit}\ (+\ 1\ /\ 3)\ p\ n$$

  so l 10 evaluates to

$$1_{\mathrm{d}} :: -1_{\mathrm{d}} :: 1_{\mathrm{d}} :: -1_{\mathrm{d}} :: 1_{\mathrm{d}} :: -1_{\mathrm{d}} :: 1_{\mathrm{d}} :: -1_{\mathrm{d}} :: 1_{\mathrm{d}} :: -1_{\mathrm{d}}$$

- Determine addition (move precisely average), multiplication for signed digit streams.
- Determine from a Cauchy Sequence for e.g. $\frac{\pi}{10}$ its signed digit representation (not done yet).

Problem of Program Extraction

- Because of postulates it is not guaranteed that each program reduces to canonical head normal form.
- Example 1

$$\text{postulate decide}_\pi : \pi \leq_{\mathbb{R}} 3.14 \vee 3.14 \leq_{\mathbb{R}} \pi$$

$$\begin{aligned}
&\text{lem} : (r\ s : \mathbb{R}) \rightarrow (r \leq_{\mathbb{R}} s \vee s \leq_{\mathbb{R}} r) \rightarrow \text{Bool} \\
&\text{lem}\ r\ s\ (\text{inl}\ _) = \text{true} \\
&\text{lem}\ r\ s\ (\text{inr}\ _) = \text{false}
\end{aligned}$$

$\text{lem}\ \pi\ 3.14\ \text{decide}_\pi$ is non-canonical element in NF

- Example 2 (something like this actually occurred)

$$\mathrm{postulate\ lem}_\pi : -1_{\mathbb{R}} \leq_{\mathbb{R}} \pi/10 \wedge \pi/10 \leq_{\mathbb{R}} 1$$

$$p : \mathrm{CauchyReal}\ \pi/10$$
$$p = \cdots$$

$$\mathrm{cauchy2SignedDigit} : (r : \mathbb{R}) \to -1 \leq_{\mathbb{R}} r \to r \leq_{\mathbb{R}} 1 \to \mathrm{CauchyReal}\ r \to \mathrm{SignedDigit}\ r$$
$$\mathrm{cauchy2SignedDigit}\ r\ p\ q\ q' = \cdots$$

$$\mathrm{cauchy2SignedDigit}' : (r : \mathbb{R}) \to (-1 \leq_{\mathbb{R}} r \wedge r \leq_{\mathbb{R}} 1) \to \mathrm{CauchyReal}\ r \to \mathrm{SignedDigit}\ r$$
$$\mathrm{cauchy2SignedDigit}'\ r\ (\mathrm{and}\ p\ q)\ q' = \mathrm{cauchy2SignedDigit}\ r\ p\ q\ q'$$

$$q : \mathrm{List\ Digit}$$
$$q = \mathrm{signedDigitToList}\ 10\ \pi/10\ (\mathrm{cauchy2SignedDigit}'\ \pi/10\ \mathrm{lem}_\pi\ p)$$
$$- - q \text{ doesn't reduce to } d_0 :: d_1 :: \cdots$$

Problem of Program Extraction

- Example 3 (something like this actually occurred)

$$\text{postulate lem} : (r : \mathbb{R}) \to r == r +_{\mathbb{R}} 0_{\mathbb{R}}$$

$$\text{transfer} : (r\ s : \mathbb{R}) \to r == s \to \text{CauchyReal}\ r \to \text{CauchyReal}\ s$$
$$\text{transfer}\ r\ r\ \text{refl}\ p = p$$

$$\text{1IsCauchy} : \text{CauchyReal}\ 1_{\mathbb{R}}$$
$$\text{1IsCauchy} = \cdots$$

$$\text{transfer}\ 1_{\mathbb{R}}\ (r +_{\mathbb{R}} 0_{\mathbb{R}})\ \text{lem}\ \text{1IsCauchy} : \text{CauchyReal}\ (r +_{\mathbb{R}} 0_{\mathbb{R}})$$
$-$ $-$ doesn't reduce to canonical normal form

- Can be avoided by proving transfer by guarded recursion into $\text{CauchyReal}\ s$

Theorem

- Assume some healthy conditions (e.g. strong normalisation, confluence, elements starting with different constructors are different).
- Assume no record types or indexed inductive definitions are used (probably can be removed).
- Assume result type of axioms is always a postulated type.
- Then every closed term which is an element of an algebraic data type is in canonical normal form (starts with a constructor).

Proof Assuming Simple Pattern Matching

- Assume $t : A$, t closed and in NF, A algebraic.
- Show by induction on length of t that t starts with a constructor.
- Then $t = f\ t_1 \cdots t_n$, f function symbol or constructor.
- f cannot be postulated or directly defined.
- If f is defined by pattern matching on say t_i.
  - By IH t_i starts with a constructor.
  - t has a reduction, wasn't in NF
- So f is a constructor.

Reduction of Nested Pattern Matching to Simple Pattern Matching

Difficult proof in the thesis of Chi Ming Chuang.

1. Real Number Computations in Agda

2. Theory of Program Extraction

Extensions

Evaluation

Extensions

- Negated axioms such as $\neg(0_{\mathbb{R}} == 1_{\mathbb{R}})$ are currently forbidden
  - Have form $0_{\mathbb{R}} == 1_{\mathbb{R}} \to \bot$ where $\bot$ is algebraic data type.
  - Causes problems since they are needed (e.g. when using the reciprocal function).
  - Without negated axioms the theory was trivially consistent (interpret all postulate sets as one element sets).
  - With negated axioms it could be inconsistent
    - E.g. take axioms which have consequences $0_{\mathbb{R}} == 1_{\mathbb{R}}$ and $\neg(0_{\mathbb{R}} == 1_{\mathbb{R}})$.)
    - Then we get a proof $p : \bot$ and therefore

$$\mathrm{efq}\ p : \mathbb{N}$$

is noncanonical in NF.

Theorem (Negated Axioms)

- Assume conditions as before.
- Assume result type of axioms is always a postulated type or a negated postulated type.
- Assume the Agda code doesn't prove $\perp$.
- Then every closed term which is an element of an algebraic data type is in canonical normal form (starts with a constructor).

More Extensions

- We could separate our algebraic data types into those for which we want to use their computational content and those for which we don't use their content.
- Assume we never derive using case distinction on a non-computational data type an element of a computational data type.
- Then axioms with result type non-computational data types could be allowed, e.g.

$$\mathrm{tertiumNonDatur} : A \vee_{\mathrm{non-computational}} \neg A$$

1. Real Number Computations in Agda

2. Theory of Program Extraction

Extensions

Evaluation

Easy Proofs

- Axiomatized theory allows to proof easily big theorems, if one is only interested in the computational content.
- In an experiment we introduced axioms such as

$$\mathrm{ax} : (r : \mathbb{R}) \to (q : \mathbb{Q}) \to |\mathbb{Q}2\mathbb{R}\; q -_{\mathbb{R}} r| <_{\mathbb{R}} 2_{\mathbb{R}}^{-2} \to q \leq_{\mathbb{Q}} 1/4_{\mathbb{Q}} \\ \to r \leq_{\mathbb{R}} 1/2_{\mathbb{R}}$$

- In fact the more is postulated the faster the program (and the easier one can see what is computed).

Separation of Logic and Computation

- Postulates allow us to have a two-layered theory with
  - computational part (using non-postulated types)
  - an a logic part (using postulated types).

Useful for Programming with Dependent Types

- This could be very useful for programming with dependent types.
  - Postuluate axioms with no computational content.
  - Possibly prove them using automated theorem provers (approach by Bove,Dybjer et. al.).
  - Concentrate in programming on computational part.

Experiments carried out

- In about 6 hours I developed a framework using Cauchy Reals, Signed Digit Reals, conversion into streams and lists form scratch.
  - Allowed the compuation of the first 10 digits of rational numbers in $[-1, 1]$.
- Framework is easy to use since most proofs are replaced by postulates.
- Chi Ming Chuang showed closure of signed digit reals under average and multiplication using more efficient direct calculations and full proofs of most theorems needed.
- Was able to calculated fast the first 1000 digits of rational numbers.

Extraction of the Actual Algorithm

- In most cases the algorithm is not visible.
- Can be made explicit if functions defined by pattern matching are given by their recursion operators.
- Maybe reflection could offer a possibility to get around this restriction.

Conclusion

- Framework which allows to reduce the burden of proofs while programming.
- Allows the integration of advanced ATP tools for proving non-computational theorems.
- Axiomatic treatment of $\mathbb{R}$ seems to be appropriate.
- Algorithm not yet visible when case distinction is used.