A Framework for Extraction of Programs from Proofs Using Postulated Axioms

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

16 September 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.

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

Computational Numbers

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

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

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

$$\begin{array}{l} \_ * \_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\ \ldots \end{array}$$

$$\begin{array}{l} \text{data } \mathbb{Z} : \text{Set where} \\ \quad \ldots \end{array}$$

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

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

$$
\begin{array}{llcl}
\multicolumn{4}{l}{\mathbb{N}2\mathbb{R} : \mathbb{N} \to \mathbb{R}} \\
\mathbb{N}2\mathbb{R} & \text{zero} & = & 0_{\mathbb{R}} \\
\mathbb{N}2\mathbb{R} & (\text{suc } n) & = & \mathbb{N}2\mathbb{R}\ n +_{\mathbb{R}} 1_{\mathbb{R}}
\end{array}
$$

$$
\begin{array}{l}
\mathbb{Z}2\mathbb{R} : \mathbb{Z} \to \mathbb{R} \\
\ldots
\end{array}
$$

$$
\begin{array}{l}
\mathbb{Q}2\mathbb{R} : \mathbb{Q} \to \mathbb{R} \\
\ldots
\end{array}
$$

- We obtain a link between computational types $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ and the postulated type $\mathbb{R}$.

Cauchy Reals

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

Program Extraction for Cauchy Reals

- Show $\mathrm{CauchyReal}$ closed under $+$, $*$, other operations.

$$\begin{aligned} \mathrm{lemma} : (r\ s : \mathbb{R}) \to \mathrm{CauchyReal}\ r \to \mathrm{CauchyReal}\ s \\ \to \mathrm{CauchyReal}\ (r * s) \end{aligned}$$

- Using this show $p : \mathrm{CauchyReal}\ r$ for some r.
  - E.g. for $r = \mathbb{Q}2\mathbb{R}\ q$.
- Define

$$f : (r : \mathbb{R}) \to (p : \mathrm{CauchyReal}\ r) \to \mathbb{N} \to \mathbb{Q}$$

  which extracts the Cauchy sequence in p.
- If we have $r : \mathbb{R}$; $\quad p : \mathrm{CauchyReal}\ r$; $\quad n : \mathbb{N}$ then

$$f\ r\ p\ n : \mathbb{Q}$$

  is an approximation of r up to 2^{-n}. Can be computed in Agda.

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.

Coalgebraic Definition of Signed Digit Real Numbers (SD)

$$\begin{array}{l} \mathrm{data\ Digit} : \mathrm{Set\ \ where} \\ \quad -1_{\mathrm{d}}\ 0_{\mathrm{d}}\ 1_{\mathrm{d}} : \mathrm{Digit} \end{array}$$

$$\begin{array}{llllllll} \multicolumn{8}{l}{\mathrm{coalg\ SD} : \mathbb{R} \to \mathrm{Set\ \ where}} \\ \quad \in[-1,1] & : & \{r : \mathbb{R}\} & \to & \mathrm{SD}\ r & \to & r \in_{\mathbb{R}} [-1,1] \\ \quad \mathrm{digit} & : & \{r : \mathbb{R}\} & \to & \mathrm{SD}\ r & \to & \mathrm{Digit} \\ \quad \mathrm{tail} & : & \{r : \mathbb{R}\} & \to & (p : \mathrm{SD}\ r) & \to & \mathrm{SD}\ (2_{\mathbb{R}} *_{\mathbb{R}} r -_{\mathbb{R}} (\mathrm{digit}\ p)) \end{array}$$

Proof of "$1_{\mathbb{R}} = 0.1_{\mathrm{d}}1_{\mathrm{d}}1_{\mathrm{d}}1_{\mathrm{d}}\cdots$"

$$
\begin{array}{llcl}
\multicolumn{4}{l}{1_{\mathrm{SD}} : (r : \mathbb{R}) \to (r ==_{\mathbb{R}} 1_{\mathbb{R}}) \to \mathrm{SD}\ r} \\
\in[-1,1] & (1_{\mathrm{SD}}\ r\ q) & = & \cdots \\
\mathrm{digit} & (1_{\mathrm{SD}}\ r\ q) & = & 1_{\mathrm{d}} \\
\mathrm{tail} & (1_{\mathrm{SD}}\ r\ q) & = & 1_{\mathrm{SD}}\ (2_{\mathbb{R}} *_{\mathbb{R}} r -_{\mathbb{R}} 1_{\mathbb{R}})\ \cdots
\end{array}
$$

Proofs of $\cdots$ can be

- inferred purely logically from axioms about $\mathbb{R}$ (using automated theorem proving?)
- added as postulated axioms.

Proof of "$0_{\mathbb{R}} = 0.1_{\mathrm{d}}(-1_{\mathrm{d}})(-1_{\mathrm{d}})(-1_{\mathrm{d}})\cdots$"

$$
\begin{array}{llllll}
\multicolumn{6}{l}{0_{\mathrm{SD}} : (r : \mathbb{R}) \to (r ==_{\mathbb{R}} 0_{\mathbb{R}}) \to \mathrm{SD}\ r} \\
\in[-1,1] & & (0_{\mathrm{SD}}\ r\ q) & = & \cdots \\
\mathrm{digit} & & (1_{\mathrm{SD}}\ r\ q) & = & 1_{\mathrm{d}} \\
\in[-1,1] & (\mathrm{tail} & (0_{\mathrm{SD}}\ r\ q)) & = & \cdots \\
\mathrm{digit} & (\mathrm{tail} & (0_{\mathrm{SD}}\ r\ q)) & = & -1_{\mathrm{d}} \\
\mathrm{tail} & (\mathrm{tail} & (0_{\mathrm{SD}}\ r\ q)) & = & \mathrm{tail}\ (0_{\mathrm{SD}}\ (2_{\mathbb{R}} *_{\mathbb{R}} r -_{\mathbb{R}} 0_{\mathbb{R}}))
\end{array}
$$

Extraction of Programs

- From

$$p : \mathrm{SD}\ r$$

  one can extract the first n digits of r.
- Show e.g. closure of SD under $\mathbb{Q} \cap [-1, 1]$, $+ \cap [-1, 1]$, $*$, $\frac{\pi}{10} \cdots$
- Then we extract the first n digits of any real number formed using these operations.
- Has been done (excluding $\frac{\pi}{10}$) in Agda.

First 1000 Digits of $\frac{29}{37} * \frac{29}{3998}$

Command Prompt

```
C:\find digits>Appendix1.exe
0.000000<-1>010010<-1>00<-1>0<-1>01001000<-1><-1>010<-1>0000010<-1>000<-1>00<-1>
01000000<-1>00110<-1>000<-1>001010<-1>0100<-1>0<-1><-1>0100<-1>00<-1>010000<-1>0
10000<-1><-1>010<-1>00<-1><-1>010<-1>00010110<-1>000101000000<-1>0<-1>0000<-1>00
00<-1>0<-1><-1>010<-1>000<-1>00010<-1>000100100<-1>00<-1>0000<-1>00010000<-1><-1
>0100101010 0<-1>000<-1>0<-1>01001000000100101000010<-1>00100<-1>0000<-1>01000011 0
<-1>00<-1>00<-1>00<-1>00110<-1>00<-1>00<-1>00<-1>0<-1>00<-1>0100000010<-1>00<-1>
0010<-1>00000<-1><-1>00110<-1>001000100<-1>0100<-1>0010<-1>0010<-1>0001000<-1>00
110<-1>00<-1>010000<-1>0001001010100101010 00<-1>0<-1>000<-1>01000110<-1>00<-1>00
<-1>0<-1>0010010001010<-1>0000101001000 0<-1>000<-1>000<-1>0<-1>00001010000 10<-1>
000100000<-1>00<-1>00110<-1>0010001001000000<-1>0100<-1>000010<-1>0001010000101 0
00<-1>00100<-1>0000<-1>001010<-1>010<-1>00<-1>000100000100101 10<-1>00<-1><-1>010
<-1>0100100<-1>0010100010100<-1><-1>0100<-1>0<-1>0010101001001 00<-1>01001000<-1>
01000<-1>0<-1>00100000101001001000<-1>0100<-1>00110<-1>00<-1>0<-1>0000<-1>010010
0<-1>0<-1>0<-1>0100<-1>0100100100010 0<-1>0001010101010 10<-1>01000100000100 0<-1
>0<-1>0<-1>00001000<-1>0<-1>0<-1>0<-1>0<-1>001001001 0<-1>00<-1><-1>0010110<-1>00
1001010<-1>010<-1>000<-1>00000100<-1>00<-1>0<-1><-1>010010<-1>000<-1>000<-1><-1>
0100<-1>00<-1>00010<-1>0100<-1>00<-1>000<-1>000<-1>0<-1>000<-1>00<-1>00<-1>0<-1>
0010<-1>0100<-1>0<-1><-1>01000110<-1>00<-1>0<-1>000<-1>010<-1>0010000<-1>000<-1>
0100000101 00<-1>000001000<-1>00<-1>00010000101000000<-1>0001010<-1>0000<-1>01001
0

C:\find digits> _
```

Problem with Program Extraction

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

$$\mathrm{postulate\ ax} : (x : A) \to B[x] \vee C[x]$$

$$a : A$$
$$a = \cdots$$

$$f : B[a] \vee C[a] \to \mathbb{B}$$
$$f\ (\mathrm{inl}\ x) = \mathrm{tt}$$
$$f\ (\mathrm{inr}\ x) = \mathrm{ff}$$

$f\ (\mathrm{ax}\ a)$ in Normal form, doesn't start with a constructor

- Axioms with computational content should not be allowed.

Example 2

$$\mathrm{postulate}\ \mathrm{ax} : A \wedge B$$

$$f : A \to B \to \mathbb{B}$$
$$f\ a\ b = \cdots$$

$$g : A \wedge B \to \mathbb{B}$$
$$g\ (\mathrm{p}\ a\ b) = f\ a\ b$$

$g\ \mathrm{ax}$ in normal form doesn't start with a constructor

- Problem actually occurred.
- Axioms with result type algebraic data types are not allowed.

Example 3

$$r0 : \mathbb{R}$$
$$r0 = 1_{\mathbb{R}}$$

$$r1 : \mathbb{R}$$
$$r1 = 1_{\mathbb{R}} +_{\mathbb{R}} 0_{\mathbb{R}}$$

$$\text{postulate ax} : r0 == r1$$

postulate ax : $r0 == r1$

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

$f : (r : \mathbb{R}) \to \mathrm{SD}\ r \to \mathrm{Digit}$
$f\ r\ a = \cdots$

$p : \mathrm{SD}\ r_0$
$p = \cdots$

$q : \mathrm{SD}\ r_1$
$q = \mathrm{transfer}\ r_0\ r_1\ \mathrm{ax}$

$q' : \mathrm{Digit}$
$q' = f\ r_1\ q$

NF of q' doesn't start with a constructor

Problem actually occurred.

Main Restriction

- If A is a postulated constant then either
  - $A : (x_1 : B_1) \rightarrow \cdots \rightarrow (x_n : B_n) \rightarrow \mathrm{Set}$ or
  - $A : (x_1 : B_1) \rightarrow \cdots \rightarrow (x_n : B_n) \rightarrow A' \; t_1 \cdots t_n$ where A' is a postulated constant.
- Essentially: postulated constants have result type a postulated type.

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 in normal form 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 in normal form, A algebraic data type.
- Show by induction on $\mathrm{length}(t)$ that t starts with a constructor:
  - We have $t = f\ t_1 \cdots t_n$, 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.

Extensions

- Negated axioms such as $\neg(0_{\mathbb{R}} == 1_{\mathbb{R}})$ are currently forbidden
  - Have form $0_{\mathbb{R}} == 1_{\mathbb{R}} \to \perp$ where $\perp$ is algebraic data type.
  - Causes problems since they are needed (e.g. when using the reciprocal function).
  - Without negated axioms the theory is 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 $\neg(0_{\mathbb{R}} == 1_{\mathbb{R}})$.)
  - In case of an inconsistency we would get a proof $p : \perp$ and therefore

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

is noncanonical of $\mathbb{N}$ 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 $\bot$.
- 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.

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

Addition of Coalgebraic Types

- Original proof didn't include coalgbraic types.
- With coalgebraic types additional complication:
  t can be of the form
  $$\mathrm{elim}\ t_1$$
  for an eliminator elim of a coalgebraic type.
- Extend the theorem by proving simultaneously:
  - If A algebraic, t closed term in NF, $t : A$, then t starts with a constructor.
  - If A coalgebriac, t closed term, $t : A$, and elim is an eliminator of A, then $\mathrm{elim}\ t$ has a reduction.

Easy Proofs

- Axiomatized theory allows to easily prove big theorems by postulating them, as long as we are 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.

Idea: Type Theory with Partial and Total Objects

- One could postulate
  - types of partial elements,
  - constants operating on those types,
  - equations for those constants .
- Then one can
  - define predicates on those partial elements corresonding to the total elements,
  - and show that certain partial elements are total or have other properties.

Example

$$
\begin{array}{lll}
\text{postulate} & \mathbb{N}_{\text{partial}} & : \ \text{Set} \\
\text{postulate} & \_==\_ & : \ \mathbb{N}_{\text{partial}} \to \mathbb{N}_{\text{partial}} \to \text{Set} \\
\text{postulate} & \text{zero} & : \ \mathbb{N}_{\text{partial}} \\
\text{postulate} & \text{succ} & : \ \mathbb{N}_{\text{partial}} \to \mathbb{N}_{\text{partial}} \\
\text{postulate} & \text{f} & : \ \mathbb{N}_{\text{partial}} \to \mathbb{N}_{\text{partial}} \\
\text{postulate} & \text{lemf0} & : \ \text{f zero} == \cdots \\
\text{postulate} & \text{lemfs} & : \ (n : \mathbb{N}_{\text{partial}}) \to \text{f } (\text{succ } n) == \cdots
\end{array}
$$

$$
\begin{array}{l}
\text{data } \mathbb{N} : \mathbb{N}_{\text{partial}} \to \text{Set where} \\
\quad \text{zerop} : \mathbb{N} \text{ zero} \\
\quad \text{succp} : (n : \mathbb{N}_{\text{partial}}) \to \mathbb{N}\ n \to \mathbb{N}\ (\text{succ } n) \\
\quad \text{eqp} : (n\ m : \mathbb{N}_{\text{partial}}) \to \mathbb{N}\ n \to n == m \to \mathbb{N}\ m
\end{array}
$$

$$
\begin{array}{l}
\text{lemma} : (n : \mathbb{N}_{\text{partial}}) \to \mathbb{N}\ n \to \mathbb{N}\ (f\ n) \\
\text{lemma } n\ p = \cdots
\end{array}
$$

Idea: Type Theory with Partial and Total Objects

- One could develop a system with a “total” and “partial” mode.
  - In total mode equations for partial objects are postulated axioms.
  - In partial mode no total types can be used.
  - In partial mode, equations for partial objects become definitional equations.
    - E.g. $f\ \mathrm{zero}$ can be evaluated in “partial mode”.

Framework for Agda

- Proofs rely on some understanding of the behaviour Agda.
- Not fully mathematical proofs.
- Instead develop
    - a logical framework with reductions,
    - an extension of this framework by constants and reduction rules,
    - an abstract notion of
        - postulated types and constants,
        - algebraic and coalgebraic types
        - functions defined by nested pattern matching (including elimination patterns)
    - and show that the theorem holds in this framework.

Conclusion

- If result types of postulated constants are postulated types, then closed elements of algebraic types evaluate to constructor normal form.
- Reduces the need burden of proofs while programming (by postulating axioms or proving them using ATP).
- Axiomatic treatment of $\mathbb{R}$.
- Program extracton for proofs with real number computations works very well.
- Applications to programming with dependent types in general.
- Possible solution for type theory with partiality and totality.