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
1. Real Number Computations in Agda

2. Theory of Program Extraction

3. Extensions

4. Applications

5. More Formal Proof

Conclusion
1. Real Number Computations in Agda

2. Theory of Program Extraction

3. Extensions

4. Applications

5. More Formal Proof

Conclusion
Can you extract programs from proofs in Agda?

Obvious because of Axiom of Choice?

From

\[ p : (x : A) \rightarrow \exists [y : B] \varphi(y) \]

we get of course

\[ f = \lambda x. \pi_0(f x) : A \rightarrow B \]
\[ p = \lambda x. \pi_1(f x) : (x : A) \rightarrow \varphi(f x) \]

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.

\[
\begin{align*}
\text{postulate} & \quad \mathbb{R} : \text{Set} \\
\text{postulate} & \quad _{==} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \text{Set} \\
\text{postulate} & \quad _{+} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\
\text{postulate} & \quad \text{commutative} : (r, s : \mathbb{R}) \rightarrow r + s = s + r \\
\ldots
\end{align*}
\]
Formulate \( \mathbb{N}, \mathbb{Z}, \mathbb{Q} \) as usual

```agda
data \text{N} : \text{Set} where
  \text{zero} : \text{N}
  \text{suc} : \text{N} \rightarrow \text{N}

_+_ : \text{N} \rightarrow \text{N} \rightarrow \text{N}
\n\n\text{n + zero} = \text{n}
\n\text{n + suc m} = \text{suc (n + m)}

_*_ : \text{N} \rightarrow \text{N} \rightarrow \text{N}
\n\n\ldots
\n\n\text{data \text{Z} : \text{Set} where}
\n\ldots
\n\n\text{data \text{Q} : \text{Set} where}
```
Embedding of \( \mathbb{N}, \mathbb{Z}, \mathbb{Q} \) into \( \mathbb{R} \)

\[
\begin{align*}
N2R : \mathbb{N} &\rightarrow \mathbb{R} \\
N2R \; \text{zero} &\quad = \quad 0_R \\
N2R \; (\text{suc } n) &\quad = \quad N2R \; n + R \; 1_R
\end{align*}
\]

\[
Z2R : \mathbb{Z} \rightarrow \mathbb{R}
\]

\[
\ldots
\]

\[
Q2R : \mathbb{Q} \rightarrow \mathbb{R}
\]

\[
\ldots
\]

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

data CauchyReal (r : ℝ) : Set where
  cauchyReal : (f : ℕ → ℚ)
              → (p : (n : ℕ) → |Q2R (f n) − R r|_R < R 2⁻ⁿ)
              → CauchyReal r
Program Extraction for Cauchy Reals

- Show `CauchyReal` closed under `+`, `*`, other operations.
  
  ```
  lemma : (r s : ℝ) → CauchyReal r → CauchyReal s → CauchyReal (r * s)
  ```

- Using this show `p : CauchyReal r` for some `r`.
  - E.g. for `r = Q2R q`.

- Define
  
  ```
  f : (r : ℝ) → (p : CauchyReal r) → ℕ → ℚ
  ```

  which extracts the Cauchy sequence in `p`.

- If we have `r : ℝ`; `p : CauchyReal r`; `n : ℕ` then
  
  ```
  f r p n : 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
  
  \[
  0.d_0d_1d_2d_3\cdots
  \]

  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)

data Digit : Set where
  \(-1_d\ 0_d\ 1_d : Digit\)

coalg SD : \(\mathbb{R} \rightarrow Set\) where
  \(\in [-1, 1] : \{ r : \mathbb{R} \} \rightarrow SD r \rightarrow r \in \mathbb{R} [-1, 1]\)
  digit : \{ r : \mathbb{R} \} \rightarrow SD r \rightarrow Digit
  tail : \{ r : \mathbb{R} \} \rightarrow (p : SD r) \rightarrow SD (2 \mathbb{R} \ast \mathbb{R} r - \mathbb{R} (\text{digit} p))\)
Proof of “$1^\mathbb{R} = 0.1d1d1d1d\cdots$”

$1_{SD} : (r : \mathbb{R}) \rightarrow (r =_{\mathbb{R}} 1^\mathbb{R}) \rightarrow SD\ r$

$\in[-1,1]$ $(1_{SD}\ r\ q) = \cdots$

digit $(1_{SD}\ r\ q) = 1_d$

tail $(1_{SD}\ r\ q) = 1_{SD}\ (2^\mathbb{R} *_{\mathbb{R}} r\ -_{\mathbb{R}} 1^\mathbb{R})\ \cdots$

Proofs of \cdots\ can be

- inferred purely logically from axioms about $\mathbb{R}$ (using automated theorem proving?)
- added as postulated axioms.
Proof of \( 0_R = 0.1_d(-1_d)(-1_d)(-1_d) \cdots \)
Extraction of Programs

- From

\[ p : \text{SD} \ r \]

one can extract the first \( n \) digits of \( r \).

- Show e.g. closure of \( \text{SD} \) under \( \mathbb{Q} \cap [-1, 1], + \cap [-1, 1], \times, \frac{\pi}{10} \cdots \)

- 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 $\frac{29}{37} \times \frac{29}{3998}$
1. Real Number Computations in Agda

2. Theory of Program Extraction

3. Extensions

4. Applications

5. More Formal Proof

Conclusion
Problem with Program Extraction

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

\[
\text{postulate } \text{ax} : (x : A) \to B[x] \lor C[x]
\]

\[
a : A
\]

\[
a = \cdots
\]

\[
f : B[a] \lor C[a] \to \mathbb{B}
\]

\[
f (\text{inl } x) = \texttt{tt}
\]

\[
f (\text{inr } x) = \texttt{ff}
\]

\[
f (\text{ax } a) \text{ in Normal form, doesn’t start with a constructor}
\]

- Axioms with computational content should not be allowed.
Example 2

postulate ax : \( A \land B \)

\[ f : A \rightarrow B \rightarrow \mathbb{B} \]
\[ f\ a\ b = \cdots \]

\[ g : A \land B \rightarrow \mathbb{B} \]
\[ g (p \ a \ b) = f\ a\ b \]

\( g\ ax \) in normal form doesn’t start with a constructor

- Problem actually occurred.
- Axioms with result type algebraic data types are not allowed.
Example 3

\[
\begin{align*}
  r0 &: \mathbb{R} \\
  r0 &= 1_{\mathbb{R}} \\
  r1 &: \mathbb{R} \\
  r1 &= 1_{\mathbb{R}} +_{\mathbb{R}} 0_{\mathbb{R}} \\
  \text{postulate ax} &: r0 \equiv r1
\end{align*}
\]
postulate ax : $r_0 == r_1$

transfer : $(r \ s : \mathbb{R}) \rightarrow r == s \rightarrow \text{SD } r \rightarrow \text{SD } s$

transfer $r \ r \ \text{refl} \ p = p$

$f : (r : \mathbb{R}) \rightarrow \text{SD } r \rightarrow \text{Digit}$

$f \ r \ a = \cdots$

$p : \text{SD } r_0$

$p = \cdots$

$q : \text{SD } r_1$

$q = \text{transfer } r_0 \ r_1 \ \text{ax}$

$q' : \text{Digit}$

$q' = f \ r_1 \ q$

NF of $q'$ doesn’t start with a constructor

Problem actually occurred.
Main Restriction

- If $A$ is a postulated constant then either
  - $A : (x_1 : B_1) \rightarrow \cdots \rightarrow (x_n : B_n) \rightarrow \text{Set}$ 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).
2. Theory of Program Extraction

Proof Assuming Simple Pattern Matching

- Assume $t : A$, $t$ closed in normal form, $A$ algebraic data type.
- Show by induction on 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

Difficulty proof in the thesis of Chi Ming Chuang.
3. Extensions

1. Real Number Computations in Agda

2. Theory of Program Extraction

3. Extensions

4. Applications

5. More Formal Proof

Conclusion
Extensions

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

\[
\text{tertiumNonDatur} : A \lor_{\text{non-computational}} \neg A
\]
Addition of Coalgebraic Types

- Original proof didn’t include coalgebraic types.
- With coalgebraic types additional complication: 
  \[ t \text{ can be of the form } \text{elim } t_1 \]
  for an eliminator \text{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 \( A \) coalgebraic, \( t \) closed term, \( t : A \), and \text{elim} is an eliminator of \( A \), then \text{elim} \( t \) has a reduction.
1. Real Number Computations in Agda

2. Theory of Program Extraction

3. Extensions

4. Applications

5. More Formal Proof

Conclusion
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

\[
\text{ax : } (r : \mathbb{R}) \rightarrow (q : \mathbb{Q}) \rightarrow |\mathbb{Q} R q - R r| <_R 2^{-2} \rightarrow q \leq_\mathbb{Q} 1/4_\mathbb{Q} \\
\rightarrow r \leq_R 1/2_R
\]

- 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).
4. Applications

Useful for Programming with Dependent Types

- This could be very useful for programming with dependent types.
  - Postulate 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 computation 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 corresponding to the total elements,
  - and show that certain partial elements are total or have other properties.
Example

postulate $\mathbb{N}_{\text{partial}} : \text{Set}$
postulate $\_ == \_ : \mathbb{N}_{\text{partial}} \rightarrow \mathbb{N}_{\text{partial}} \rightarrow \text{Set}$
postulate $\text{zero} : \mathbb{N}_{\text{partial}}$
postulate $\text{succ} : \mathbb{N}_{\text{partial}} \rightarrow \mathbb{N}_{\text{partial}}$
postulate $f : \mathbb{N}_{\text{partial}} \rightarrow \mathbb{N}_{\text{partial}}$
postulate $\text{lemf0} : f \text{ zero} == \cdots$
postulate $\text{lemfs} : (n : \mathbb{N}_{\text{partial}}) \rightarrow f (\text{succ } n) == \cdots$
data $\mathbb{N} : \mathbb{N}_{\text{partial}} \rightarrow \text{Set}$ where

$\text{zerop} : \mathbb{N} \text{ zero}$
$\text{succp} : (n : \mathbb{N}_{\text{partial}}) \rightarrow \mathbb{N} \ n \rightarrow \mathbb{N} \ (\text{succ } n)$
$\text{eqp} : (n \ m : \mathbb{N}_{\text{partial}}) \rightarrow \mathbb{N} \ n \rightarrow n == m \rightarrow \mathbb{N} \ m$

$\text{lemma} : (n : \mathbb{N}_{\text{partial}}) \rightarrow \mathbb{N} \ n \rightarrow \mathbb{N} \ (f \ n)$
$\text{lemma } n \ p = \cdots$
4. Applications

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. $f \text{ zero}$ can be evaluated in “partial mode”.
1. Real Number Computations in Agda

2. Theory of Program Extraction

3. Extensions

4. Applications

5. More Formal Proof

Conclusion
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.
1. Real Number Computations in Agda

2. Theory of Program Extraction

3. Extensions

4. Applications

5. More Formal Proof

Conclusion
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 extraction 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.