Extraction of Programs from Proofs using Postulated Axioms
or
Ideal and Concrete Objects in Type Theory

Anton Setzer

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

10 October 2011
1. Type Theory

2. Formalising $\mathbb{R}$

3. Theory of Program Extraction

Conclusion
1. Type Theory

2. Formalising $\mathbb{R}$

3. Theory of Program Extraction

Conclusion
Main Type Theoretic Setting

- We use a variant of Martin-Löf Type Theory based essentially on the syntax of the theorem prover Agda.
  - Abstract formulation work in progress.
- In Martin-Löf Type Theory types denoted by keyword Set.
- Three main constructs:
  - dependent function types,
  - algebraic data types,
  - coalgebraic data types.
Dependent Function Types

- $(x : A) \to B$
- Essentially $\prod x : A. B$ (subtle differences).
- Formula $\forall x : A. B$ represented by $(x : A) \to B$. 
Algebraic data types

data \mathbb{N} : \text{Set}

\begin{align*}
0 & : \mathbb{N} \\
S & : \mathbb{N} \rightarrow \mathbb{N}
\end{align*}

Functions defined by pattern matching

\begin{align*}
f & : \mathbb{N} \rightarrow \mathbb{N} \\
f \quad 0 & = 5 \\
f \quad (S \ 0) & = 12 \\
f \quad (S \ (S \ n)) & = (f \ n) \times 20
\end{align*}
Coalgebraic data types

\[
\text{coalg \ Stream : Set where}
\]
\[
\text{head : Stream → N}
\]
\[
\text{tail : Stream → Stream}
\]
\[
\text{inc : N → Stream}
\]
\[
\text{head (inc n) = n}
\]
\[
\text{tail (inc n) = inc (n + 1)}
\]
(Non-Agda syntax)
Postulates

postulate  $A : \mathbb{N} \rightarrow \text{Set}$

postulate  $f : \mathbb{N} \rightarrow \mathbb{N}$

postulate  $\text{lem} : f 1 == f (f 0)$
1. Type Theory

2. Formalising $\mathbb{R}$

3. Theory of Program Extraction

Conclusion
2. Formalising $\mathbb{R}$

Standard Approach to Formulation of $\mathbb{R}$

- Define

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}$

in a standard way.

- Define using these constructs
  - the Cauchy Reals
    \[
    \Sigma a : \mathbb{N} \to \mathbb{Q}.\text{Cauchy}(a)
    \]
  - or any other representation of constructive real numbers.

- Problem: Need to prove theorems constructively, even if they have no computational content.
2. Formalising $\mathbb{R}$

**Ideal Objects**

- Use of ideal and concrete objects.
- Use approach by Berger transferred to type theory:
  Axiomatize the real numbers abstractly. E.g.

  
  postulate $\mathbb{R}$ : Set  
  postulate $0_{\mathbb{R}}, 1_{\mathbb{R}}$ : $\mathbb{R}$  
  postulate $\_ == \_ : \mathbb{R} \to \mathbb{R} \to \text{Set}$  
  postulate $\_ + \_ : \mathbb{R} \to \mathbb{R} \to \mathbb{R}$  
  postulate commutative : $(r \ s : \mathbb{R}) \to r + s == s + r$

...
Concrete Objects

- Formulate \( \mathbb{N}, \mathbb{Z}, \mathbb{Q} \) as standard computational data types.

\[
\begin{align*}
\text{data } \mathbb{N} &: \text{Set where} \\
0 &: \mathbb{N} \\
S &: \mathbb{N} \rightarrow \mathbb{N} \\
- + - &: \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \\
n + 0 &= n \\
n + S \, m &= S \, (n + m) \\
_* _* &: \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \\
\ldots \\
\text{data } \mathbb{Z} &: \text{Set where} \\
\ldots \\
\text{data } \mathbb{Q} &: \text{Set where}
\end{align*}
\]
Embedding of \( \mathbb{N}, \mathbb{Z}, \mathbb{Q} \) into \( \mathbb{R} \):

- Embed \( \mathbb{N}, \mathbb{Z}, \mathbb{Q} \) into \( \mathbb{R} \):

  \[
  \begin{align*}
  \text{embed}_\mathbb{N} : \mathbb{N} & \to \mathbb{R} \\
  \text{embed}_\mathbb{N} 0 &= 0_\mathbb{R} \\
  \text{embed}_\mathbb{N} (S\ n) &= \text{embed}_\mathbb{N} n + 1_\mathbb{R}
  \end{align*}
  \]

  \[
  \begin{align*}
  \text{embed}_\mathbb{Z} : \mathbb{Z} & \to \mathbb{R} \\
  \ldots
  \end{align*}
  \]

  \[
  \begin{align*}
  \text{embed}_\mathbb{Q} : \mathbb{Q} & \to \mathbb{R} \\
  \ldots
  \end{align*}
  \]
Signed Digit Representations

- Signed digit (SD) representable real numbers in \([-1, 1]\) are of the form

\[
  r = 0. \underbrace{1\ldots 1}_{d} \underbrace{(-1)0(-1)01(-1)\ldots}_{2* r-d}
\]

- So

\[
r \in \text{SD} \iff r \in [-1, 1] \land \exists d \in \{-1, 0, 1\}.2 * r - d \in \text{SD}
\]

\(\text{SD} = \) largest fixed point fulfilling that equation.

- Formulation in type theory:

\[
\text{coalg SD} : \mathbb{R} \to \text{Set} \quad \text{where}
\]

\[
\in [-1, 1] : (r : \mathbb{R}) \to \text{SD } r \quad \to \quad r \in [-1, 1]
\]

\[
digit : (r : \mathbb{R}) \to \text{SD } r \quad \to \quad \{-1, 0, 1\}
\]

\[
tail : (r : \mathbb{R}) \to (p : \text{SD } r) \quad \to \quad \text{SD } (2 \mathbb{R} \ast r - \text{digit } r \ p)
\]
Extraction of Programs

- From

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

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

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

- Then we extract the first \( n \) digits of any real number formed using these operations.

- Has been done (excluding \( \frac{\pi}{10} \)) in Agda, program extraction can be executed feasibly.
3. Theory of Program Extraction

1. Type Theory

2. Formalising $\mathbb{R}$

3. Theory of Program Extraction

Conclusion
Problem with Program Extraction

- We don’t want that

  \[ d : \text{Digit} \]
  \[ d = \cdots \]

  and evaluation of \( d \) to normal form has result

  \[ ax1 (ax2 (ax3 \cdots)) \]

- We want that \( d \) evaluates to \(-1\) or \(0\) or \(1\).
Example 1

postulate \( \text{ax} : B \lor C \)

\[
\begin{align*}
  f : B \lor C & \rightarrow \mathbb{B} \\
  f \ (\text{inl} \ x) & = \text{tt} \\
  f \ (\text{inr} \ x) & = \text{ff}
\end{align*}
\]

\((f \ \text{ax})\) in normal form, doesn't start with a constructor
Example 2

\textbf{postulate ax : } A \land B \\
\textbf{f : } A \land B \rightarrow \mathbb{B} \\
\textbf{f } \langle a, b \rangle = \cdots \\
(f \text{ ax} ) \text{ in normal form doesn’t start with a constructor}
Example 3

postulate ax : r0 == r1

transfer : (r s : ℝ) → r == s → SD r → SD s
transfer r r refl p = p

firstdigit : (r : ℝ) → SD r → Digit
firstdigit r a = · · ·

p : SD r0
p = · · ·

q : SD r1
q = transfer r0 r1 ax

q' : Digit
q' = firstdigit r1 q

NF of q' doesn’t start with a constructor
Problem occurs because an element of an algebraic data type was introduced by a postulate and eliminated by an elimination rule for that type.

Restriction needed: 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 result type of postulated 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 \( \text{length}(t) \) that \( t \) starts with a constructor.
- Let \( t = f \ t_1 \cdots t_n \), \( f \) 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.
3. Theory of Program Extraction

Reduction of Nested Pattern Matching to Simple Pattern Matching

Difficult proof in the thesis of Chi Ming Chuang.
3. Theory of Program Extraction

Logic for Ideal Objects

- The following fulfils our conditions:

  postulate \( \_ \lor' \_ \) : \( \text{Set} \rightarrow \text{Set} \rightarrow \text{Set} \)
  postulate excluded_middle : \( (X: \text{Set}) \rightarrow X \lor' \neg X \)
  postulate \( \lor'_\text{elim} \) : \( (A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow A \lor' B \rightarrow C \)
  (for postulated \( C \))
Negated Axioms

- Not allowed (using $\neg A = A \rightarrow \bot$

  postulate $a : \neg A$
  postulate $b : A$

  $c : \bot \rightarrow \mathbb{N}$
  $c ()$

  $d : \mathbb{N}$
  $d = c (a \ b)$

- However: If the type theory used $\not \vdash p : \bot$ and every postulated type has result type postulate type or $\bot$, then conclusions of theorem fulfilled.
1. Type Theory

2. Formalising $\mathbb{R}$

3. Theory of Program Extraction

Conclusion
Conclusion

- If result types of postulated constants are postulated types, then closed elements of algebraic types evaluate to constructor normal form.
- Makes develop of programs much easier (by postulating axioms or proving them using ATP).
- Axiomatic treatment of $\mathbb{R}$.
- Program extraction for proofs with real number computations works very well.
- Possible application to type theory with partial and total objects.