Extraction of Programs from Proofs using Postulated Axioms

Anton Setzer

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

10 October 2011
1. Agda in 5 Slides

2. Real Number Computations in Agda

3. Theory of Program Extraction

Conclusion
1. Agda in 5 Slides

2. Real Number Computations in Agda

3. Theory of Program Extraction

Conclusion
Agda

- Agda is a theorem prover based on Martin-Löf’s intuitionistic type theory.
- Proofs and programs are treated the same:
  
  \[
  n : \mathbb{N} \\
  n = \text{exp} \ 5 \ 20 \\
  p : A \land B \\
  p = \langle \cdots , \cdots \rangle
  \]

- For historic reasons types denoted by keyword Set.
- 3 main constructs:
  - dependent function types,
  - algebraic data types,
  - coalgebraic data types.
Dependent Function Types

- \((x : A) \rightarrow B\)

  type of functions mapping \(a : A\) to an element of type \(B[x := a]\).

- E.g.

  \[
  \text{matmult} : (n \ m \ k : \mathbb{N}) \rightarrow \text{Mat}(n, m) \rightarrow \text{Mat}(m, k) \rightarrow \text{Mat}(n, k)
  \]

  \[
  \text{matmult} \ n \ m \ k \ A \ B = \cdots
  \]
Algebraic data types

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

\[
\begin{align*}
\text{zero} & : \mathbb{N} \\
\text{succ} & : \mathbb{N} \rightarrow \mathbb{N}
\end{align*}
\]

Functions defined by pattern matching

\[
\begin{align*}
f : \mathbb{N} & \rightarrow \mathbb{N} \\
f \text{ zero} & = 5 \\
f \text{ (suc zero)} & = 12 \\
f \text{ (suc (suc } n \text{ )}) & = (f n) \times 20
\end{align*}
\]
Coalgebraic data types

Syntax as I would like it to be:

```agda
coalg Stream : Set where
  head   : Stream → ℕ
  tail   : Stream → Stream

inc : ℕ → Stream
head (inc n) = n
tail (inc n) = inc (n + 1)
```
Further Elements of Agda

- Postulated functions (functions without a definition)
  
  \[
  \text{postulate } \text{false} : \bot
  \]

- Hidden arguments
  
  \[
  \text{cons} : \{X : \text{Set}\} \to X \to \text{List } X \to \text{List } X
  \]

  \[
  l : \text{List } \mathbb{N}
  \]

  \[
  l = \text{cons } 0 \text{ nil}
  \]
1. Agda in 5 Slides

2. Real Number Computations in Agda

3. Theory of Program Extraction

Conclusion
Question by Ulrich Berger:
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

Approach of Ulrich Berger transferred to Agda:
Axiomatize the real numbers abstractly. E.g.

postulate \( \mathbb{R} \) : Set
postulate \( \_ == \_ \) : \( \mathbb{R} \to \mathbb{R} \to \text{Set} \)
postulate \( \_ + \_ \) : \( \mathbb{R} \to \mathbb{R} \to \mathbb{R} \)
postulate commutative : \( (r \ s : \mathbb{R}) \to r + s == s + r \)
...
Computational Numbers

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

```agda
data \( \mathbb{N} \) : Set where
  zero : \( \mathbb{N} \)
  suc  : \( \mathbb{N} \rightarrow \mathbb{N} \)

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

_ \ast _ : \( \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \)
...

data \( \mathbb{Z} \) : Set where
  ...

data \( \mathbb{Q} \) : Set where
```
Embedding of \( \mathbb{N}, \mathbb{Z}, \mathbb{Q} \) into \( \mathbb{R} \)

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

  \[
  \begin{align*}
  N2R : \mathbb{N} & \rightarrow \mathbb{R} \\
  N2R \text{ zero} & = 0_R \\
  N2R \ (\text{suc } n) & = N2R \ n +_R 1_R \\
  \\
  Z2R : \mathbb{Z} & \rightarrow \mathbb{R} \\
  \ldots \\
  \\
  Q2R : \mathbb{Q} & \rightarrow \mathbb{R} \\
  \ldots 
  \end{align*}
  \]

- We obtain a link between computational types and the postulated type \( \mathbb{R} \):

Cauchy Reals

data CauchyReal (r : ℝ) : Set where
  cauchyReal : (f : Q⁺ → Q)
  → ((q : Q⁺) → |Q2ℝ (f q) − ℝ r|_ℝ < ℝ Q⁺ 2ℝ r)
  → CauchyReal r
Show \texttt{CauchyReal} closed under certain operations:

\[
\text{lemma} : (r, s : \mathbb{R}) \to \text{CauchyReal } r \to \text{CauchyReal } s \\
\to \text{CauchyReal } (r \ast_{\mathbb{R}} s)
\]

Extract from \texttt{Cauchy Reals} their approximations:

\[
\text{extract} : \{ r : \mathbb{R} \} \to \text{CauchyReal } r \to \mathbb{Q}^+ \to \mathbb{Q}
\]

If we have \( r : \mathbb{R} \) and \( p : \text{CauchyReal } r \), then for \( q : \mathbb{Q}^+ \)

\[
\text{extract } p \ q : \mathbb{Q}
\]

is an approximation of \( r \) up to \( q \). 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
  \[
  0.111(-1)0(-1)01(-1)\cdots
  \]
Coalgebraic Definition of Signed Digit Real Numbers (SD)

data Digit : Set where
   −1₀d 0₀d 1₀d : Digit

coalg SD : ℝ → Set where
   ∈[−1, 1] : \{ r : ℝ \} → SD r → r ∈ ℝ [−1, 1]
digit : \{ r : ℝ \} → SD r → Digit
tail : \{ r : ℝ \} → (p : SD r) → SD (2 * ℝ r − ℝ (digit p))
Proof of “$1_R = 0.1_d1_d1_d1_d \cdots$”

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

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

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

tail \quad (1_{SD} \ r \ q) = 1_{SD} \ (2_R *_R r -_R 1_R) \ \cdots$$

Proofs of \cdots can be

- inferred purely logically from axioms about $\mathbb{R}$ (using automated theorem proving?)
- added as postulated axioms.
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 \( \mathbb{Q} \cap [-1, 1], + \cap [-1, 1], *, \frac{\pi}{10} \cdots \)

- 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. Agda in 5 Slides

2. Real Number Computations in Agda

3. Theory of Program Extraction

Conclusion
Problem with Program Extraction

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

- Example 1

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

\[
a : A
\]

\[
a = \cdots
\]

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

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

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

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

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

postulate ax : \( A \land B \)

\( f : A \rightarrow B \rightarrow \mathbb{B} \)
\( f \ a \ b = \cdots \)

\( g : A \land B \rightarrow \mathbb{B} \)
\( g \langle a, b \rangle = f \ a \ b \)

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

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

\[ r_0 : \mathbb{R} \]
\[ r_0 = 1_{\mathbb{R}} \]

\[ r_1 : \mathbb{R} \]
\[ r_1 = 1_{\mathbb{R}} + 0_{\mathbb{R}} \]

postulate ax : \( r_0 = r_1 \)
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 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 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).
3. 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 \( \text{length}(t) \) 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.
1. Agda in 5 Slides

2. Real Number Computations in Agda

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.
- 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, and totality.