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

Agda

- Agda is a theorem prover based on Martin-Löf's intuitionistic type theory.
- Proofs and programs are treated the same:

$$n : \mathbb{N}$$
$$n = \text{exp } 5\ 20$$

$$p : A \wedge B$$
$$p = \langle \cdots , \cdots \rangle$$

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

Dependent Function Types

- $$(x : A) \to B$$

  type of functions mapping $a : A$ to an element of type $B[x := a]$.
- E.g.

  $$\mathrm{matmult} : (n\ m\ k : \mathbb{N}) \to \mathrm{Mat}(n, m) \to \mathrm{Mat}(m, k) \to \mathrm{Mat}(n, k)$$
  $$\mathrm{matmult}\ n\ m\ k\ A\ B = \cdots$$

Algebraic data types

$$
\begin{aligned}
&\mathrm{data}\ \mathbb{N} : \mathrm{Set} \\
&\quad \mathrm{zero} \quad : \quad \mathbb{N} \\
&\quad \mathrm{succ} \quad : \quad \mathbb{N} \to \mathbb{N}
\end{aligned}
$$

Functions defined by pattern matching

$$
\begin{array}{llcl}
f : \mathbb{N} \to \mathbb{N} & & & \\
f & \mathrm{zero} & = & 5 \\
f & (\mathrm{suc}\ \mathrm{zero}) & = & 12 \\
f & (\mathrm{suc}\ (\mathrm{suc}\ n\ )) & = & (f\ \ n) * 20
\end{array}
$$

Coalgebraic data types

Syntax as I would like it to be:

$$
\begin{array}{l}
\text{coalg Stream : Set where} \\
\quad \begin{array}{lcl}
\text{head} & : & \text{Stream} \to \mathbb{N} \\
\text{tail} & : & \text{Stream} \to \text{Stream}
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\text{inc} : \mathbb{N} \to \text{Stream} \\
\begin{array}{lcl}
\text{head (inc } n) & = & n \\
\text{tail (inc } n) & = & \text{inc } (n+1)
\end{array}
\end{array}
$$

Further Elements of Agda

- Postulated functions (functions without a definition)

$$\text{postulate } \mathsf{false} : \bot$$

- Hidden arguments

$$\text{cons} : \{X : \text{Set}\} \to X \to \text{List } X \to \text{List } X$$

$$l : \text{List } \mathbb{N}$$
$$l = \text{cons } 0 \text{ nil}$$

Program Extraction in Agda

- 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

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

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

Computational Numbers

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

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

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

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

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

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

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

- Embed $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ 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 and the postulated type $\mathbb{R}$:

Cauchy Reals

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

Program Extraction for Cauchy Reals

- Show $\mathrm{CauchyReal}$ closed under certain operations:

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

- Extract from Cauchy Reals their approximations:

$$\mathrm{extract} : \{r : \mathbb{R}\} \to \mathrm{CauchyReal}\ r \to \mathbb{Q}^{+} \to \mathbb{Q}$$

- If we have $r : \mathbb{R}$ and $p : \mathrm{CauchyReal}\ r$, then for $q : \mathbb{Q}^{+}$

$$\mathrm{extract}\ p\ q : \mathbb{Q}$$

  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)

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

$$
\begin{array}{lllllll}
\multicolumn{7}{l}{\text{coalg SD} : \mathbb{R} \to \text{Set where}} \\
\quad \in[-1,1] & : & \{r : \mathbb{R}\} & \to & \text{SD}\ r & \to & r \in_{\mathbb{R}} [-1,1] \\
\quad \text{digit} & : & \{r : \mathbb{R}\} & \to & \text{SD}\ r & \to & \text{Digit} \\
\quad \text{tail} & : & \{r : \mathbb{R}\} & \to & (p : \text{SD}\ r) & \to & \text{SD}\ (2_{\mathbb{R}} *_{\mathbb{R}} r -_{\mathbb{R}} (\text{digit}\ p))
\end{array}
$$

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.

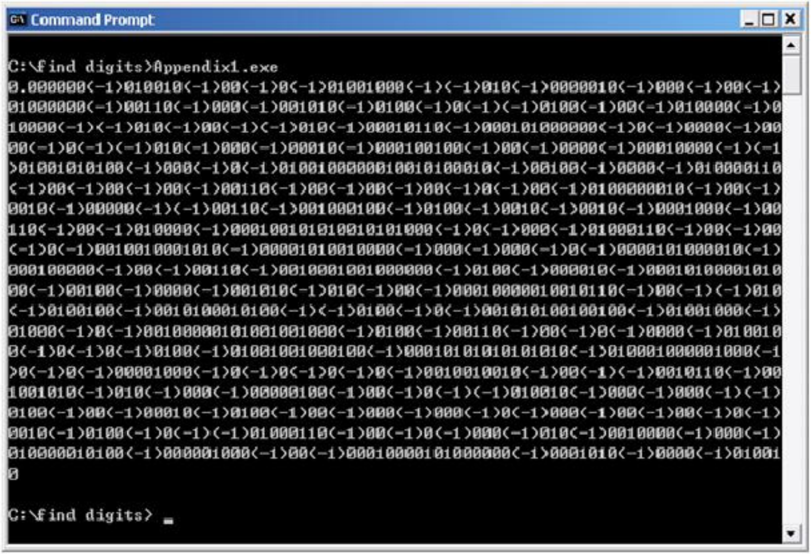
Extraction of Programs

- 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 $\frac{29}{37} * \frac{29}{3998}$



Problem with Program Extraction

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

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

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

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

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

- Axioms with computational content should not be allowed.

Example 2

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

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

$$g : A \wedge B \to \mathbb{B}$$
$$g\ \langle a, b \rangle = f\ a\ b$$

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

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

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

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

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

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

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

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

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

$q' : \text{Digit}$
$q' = \text{firstdigit}\ r_1\ q$

NF of q' doesn't start with a constructor

Problem actually occurred.

Main Restriction

- If A is a postulated constant then either
  - $A : (x_1 : B_1) \to \cdots \to (x_n : B_n) \to \mathrm{Set}$ or
  - $A : (x_1 : B_1) \to \cdots \to (x_n : B_n) \to A' \; t_1 \cdots t_n$ 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).

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.

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.