Extraction of Programs from Proofs using Postulated Axioms

Anton Setzer
Swansea University, Swansea UK
(Joint work with Chi Ming Chuang)
Talk given at JAIST, Japan

22 January 2015

Agda

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

$$\begin{array}{lcl} n & : & \mathbb{N} \\ n & = & \exp\ 5\ 20 \end{array}$$

$$\begin{array}{lcl} p & : & A \wedge B \\ p & = & \langle \cdots , \cdots \rangle \end{array}$$

- Programs and proofs are defined recursively.
- In order to obtain soundness, elements of proofs need to be terminating. Otherwise we could prove falsity:

$$\begin{array}{l} p : \bot \\ p = p \end{array}$$

Termination of programs guaranteed by a termination checker based on strongly extended primitive recursion.

Framework of Agda

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

Dependent Function Types and ∀-Quantifier

- Dependent function type

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

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

- E.g.

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

- Main example of dependent function type is ∀-quantifier:

$$(x : A) \to \varphi$$

is type of functions mapping $x : A$ to a proof of φ,
i.e. type of proofs of $\forall x.\varphi$.
So $(x : A) \to \varphi$ stands for $\forall x.\varphi$.

Algebraic data types

$$
\begin{array}{l}
\mathrm{data}\ \mathbb{N} : \mathrm{Set} \\
\quad 0 \quad : \quad \mathbb{N} \\
\quad \mathrm{suc} \quad : \quad \mathbb{N} \to \mathbb{N}
\end{array}
$$

Functions defined by pattern matching

$$
\begin{array}{lrcl}
f : \mathbb{N} \to \mathbb{N} & & & \\
f & 0 & = & 5 \\
f & (\mathrm{suc}\ 0) & = & 12 \\
f & (\mathrm{suc}\ (\mathrm{suc}\ n)) & = & (f\ n) * n
\end{array}
$$

Equality

Equality type is algebraic type indexed over pairs of elements of set A
There is on proof $\mathrm{refl} : x == x$.

$$
\begin{array}{l}
\mathrm{data}\ \_ == \_\ \{X : \mathrm{Set}\} : X \to X \to \mathrm{Set}\ \mathrm{where} \\
\quad \mathrm{refl} : \{x : X\} \to x == x
\end{array}
$$

$$
\begin{array}{lll}
\mathrm{transferEq} : (X : \mathrm{Set}) & & \\
 & \to & (Y : X \to \mathrm{Set}) \\
 & \to & (x : X) \\
 & \to & (y : X) \\
 & \to & (x == y) \\
 & \to & Y\ x \\
 & \to & Y\ y \\
\multicolumn{3}{l}{\mathrm{transferEq}\ X\ Y\ x\ x\ \mathrm{refl}\ y = y}
\end{array}
$$

Coalgebraic data types

Syntax as AS would like it to be:

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

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

Syntax in Agda

- Agda allows hidden arguments

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

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

  No deep theory behind – anything is legal as long as the theorem prover can determine a unique solution to hidden arguments.

- Agda has mixfix symbols.
  Syntax example if_then_else_
  Again: anything is allowed as long as the parser can parse it uniquely.

- Postulated functions (functions without a definition)

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

Dependent Product

One example of an algebraic data type:

$$\mathrm{data}\ \exists\ (A : \mathit{Set})\ (\varphi : A \to \mathrm{Set}) : \mathrm{Set}$$
$$\langle \_, \_ \rangle \quad : \quad (a : A) \to \varphi\ a \to \exists\ A\ \varphi$$

Projections

$$\pi_0 : \{A : \mathrm{Set}\} \to \{\varphi : A \to \mathrm{Set}\} \to \exists\ A\ \varphi \to A$$
$$\pi_0\ \langle a, b \rangle = a$$

$$\pi_1 : \{A : \mathrm{Set}\} \to \{\varphi : A \to \mathrm{Set}\} \to (x : \exists\ A\ \varphi) \to \varphi\ (\pi_0\ x)$$
$$\pi_1\ \langle a, b \rangle = b$$

Question by Ulrich Berger

- Can you extract programs from proofs in Agda?
- Obvious because of Axiom of Choice?
  From

  $$p : (x : A) \to \exists\ B\ \varphi$$

  we get of course

  $$\begin{aligned} f &= \lambda x.\pi_0\ (p\ x) : A \to B \\ q &= \lambda x.\pi_1\ (p\ x) : (x : A) \to \varphi\ (f\ x) \end{aligned}$$

- However what happens in the presence of axioms?

Real Numbers as Ideal Objects

- Situation different in presence of axioms.
- Approach of Ulrich Berger transferred to Agda:
  Axiomatice the real numbers abstractly. E.g.

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

Computational Numbers as Concrete Objects

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

$$\begin{array}{l}\text{data } \mathbb{N} : \text{Set where} \\ \quad \begin{array}{lcl} 0 & : & \mathbb{N} \\ \text{suc} & : & \mathbb{N} \to \mathbb{N} \end{array}\end{array}$$

$$\begin{array}{l}\_+\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\ \begin{array}{lclcl} n & + & 0 & = & n \\ n & + & \text{suc}\ m & = & \text{suc}\ (n+m) \end{array}\end{array}$$

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

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

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

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

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

$$
\mathbb{Z}2\mathbb{R} : \mathbb{Z} \to \mathbb{R}
$$

$\cdots$

$$
\mathbb{Q}2\mathbb{R} : \mathbb{Q} \to \mathbb{R}
$$

$\cdots$

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

Cauchy Reals

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

Program Extraction for Cauchy Reals

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

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

- 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 $r : \mathbb{R}$; $p : \mathrm{CauchyReal}\ r$; $n : \mathbb{N}$ then

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

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

Problem of Program Extraction

- Problem is that definition of f was referring to postulated axioms.
- So we might obtain

$$f\ r\ p\ n = \mathrm{lemma35}\ (\mathrm{lemma16}\ 3)\ 5$$

- We want that even though we use postulated axioms $f\ r\ p\ n$ reduces to a computational real number, i.e. $(1/2)$.

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)

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

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

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

$$1_{\mathrm{SD}} : (r : \mathbb{R}) \to (r ==_{\mathbb{R}} 1_{\mathbb{R}}) \to \mathrm{SD}\ r$$

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

Proofs of $\cdots$ can be

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

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

$$
\begin{array}{llcl}
\multicolumn{4}{l}{0_{\mathrm{SD}} : (r : \mathbb{R}) \to (r ==_{\mathbb{R}} 0_{\mathbb{R}}) \to \mathrm{SD}\ r} \\
\in[-1,1] & (0_{\mathrm{SD}}\ r\ q) & = & \cdots \\
\mathrm{digit} & (0_{\mathrm{SD}}\ r\ q) & = & -1_{\mathrm{d}} \\
\mathrm{tail} & (0_{\mathrm{SD}}\ r\ q)) & = & 1_{\mathrm{SD}}\ (2_{\mathbb{R}} *_{\mathbb{R}} r -_{\mathbb{R}} (-1_{\mathbb{R}}))\ \cdots
\end{array}
$$

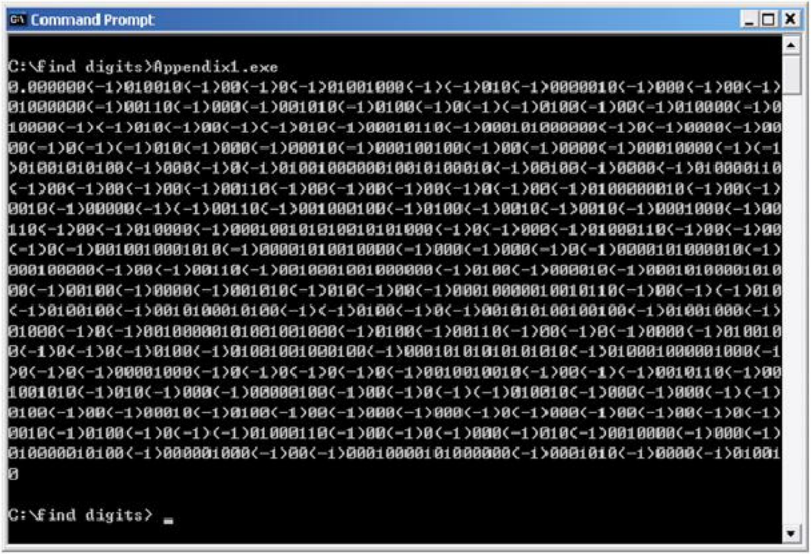
Extraction of Programs

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



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

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

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

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

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

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

Example 3

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

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

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

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

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

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

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

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

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

NF of q' doesn't start with a constructor

Problem actually occurred.

Work around Problem of Equality

- Instead of defining

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

define

$$p : (r : \mathbb{R}) \to (r == r_0) \to \mathrm{SD}\ r$$

Conditions for Correctness

- We will define conditions which guarantee that every closed term in normal form which is an element of an algebraic data type is in **canonical normal form** (starts with a constructor).

General Assumptions about Agda Code

- Agda code is **strongly normalising**.
- Agda code is **confluent**.
- **No** occurrence of **record types**, **let**- and **where**-expressions.
- Apart from the identity type, all **algebraic data types** are **non-indexed** and we have **no inductive-recursive definitions**.
- **No coalgebraic types** (work in progress to include them).
- Functions defined in Agda by pattern matching have
  - a **coverage complete pattern matching** (all cases provided)
  - all **patterns** are **disjoint**.

General Assumptions about Agda Code

- Agda code is **consistent**, i.e.:
  - If Agda proves $A = B : \mathrm{Set}$ then
    - if one is algebraic data type the other one is algebraic data type with same definition (up to equality)
    - if one is of the form $(x : B) \to C$ so is the other with equal types
  - If $t : C\ t_1 \cdots t_n : B$ where B is algebraic, then C is a constructor of B and t_i are of appropriate types.
  - If $C\ t_1 \cdots t_n = C'\ t_1 \cdots t'_m$ then $C = C'$, $n = m$, $t_i = t'_i$.

Main Restriction on Agda Code

- 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 or an equality.
- The same applies to functions f defined by case distinction on equalities.

Main Theorem

Theorem (Main Theorem)

- *Assume the above conditions.*
- *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
    $$t = f\ t_1 \cdots t_n$$
    where f function symbol or constructor.
  - f cannot be postulated or directly defined.
  - f cannot be defined by case distinction on an equality.
  - If f is defined by pattern matching on an algebraic data type say t_i.
    - By IH t_i starts with a constructor.
    - t has a reduction, wasn't in NF.
  - So f is a constructor.

Properties of Agda Code

- Agda code has the **normal form property** if every closed normal term which is an element of an algebraic data type starts with a constructor.
- Agda code **$\mathcal{A}'$ extends Agda code $\mathcal{A}$** ($\mathcal{A} \subseteq \mathcal{A}'$)

  if all judgements derivable in $\mathcal{A}$ are derivable in $\mathcal{A}'$ as well.
- Assume $\mathcal{A} \subseteq \mathcal{A}'$.
  $\mathcal{A}'$ induces the head normal form property on $\mathcal{A}$ if
  - whenever B is an algebraic data type
  - s.t. $\mathcal{A} \vdash t : B$
  - and t has in $\mathcal{A}'$ a normal form starting with a constructor,
  - then t has in $\mathcal{A}$ a normal form starting with the same constructor.

Properties of Agda Code

- Assume $\mathcal{A} \subseteq \mathcal{A}'$.
  - $\mathcal{A} \subseteq \mathcal{A}'$ **induces the coverage completeness property**, iff: if $\mathcal{A}$ is coverage complete with disjoint patterns so is $\mathcal{A}'$.
  - $\mathcal{A} \subseteq \mathcal{A}'$ **induces the strong normalisation property**, iff: if $\mathcal{A}$ is strongly normalising, so is $\mathcal{A}'$.
  - $\mathcal{A} \subseteq \mathcal{A}'$ **induces the consistency property**, iff: if $\mathcal{A}$ is consistent, so is $\mathcal{A}'$.

Theorem (Unnesting of Pattern Matching)

Theorem (Unnesting of Pattern Matching)

- *Assume $\mathcal{A}$ is Agda code fulfilling the above restrictions.*
- *Then there exists $\mathcal{A} \subseteq \mathcal{A}'$ s.t.*
  - *$\mathcal{A}'$ has simple pattern matching only,*
  - *$\mathcal{A} \subseteq \mathcal{A}'$ induces the head normal form property,*
  - *$\mathcal{A} \subseteq \mathcal{A}'$ induces coverage completeness, strong normalisation and consistency properties.*

Example Reduction to Simple Pattern Matching

Original code:

$$
\begin{array}{lcll}
\_ - \_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} & & & \\
m & - & 0 & = & m \\
0 & - & (\mathrm{suc}\ n) & = & 0 \\
(\mathrm{suc}\ m) & - & (\mathrm{suc}\ n) & = & m - n
\end{array}
$$

Make sure lines make case distinction on first argument:

$$
\begin{array}{lcll}
\_ - \_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} & & & \\
0 & - & 0 & = & 0 \\
(\mathrm{suc}\ n) & - & 0 & = & \mathrm{suc}\ n \\
0 & - & (\mathrm{suc}\ n) & = & 0 \\
(\mathrm{suc}\ m) & - & (\mathrm{suc}\ n) & = & m - n
\end{array}
$$

Example Reduction to Simple Pattern Matching

$$
\begin{array}{llll}
\_ - \_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
0 & - & 0 & = & 0 \\
(\mathrm{suc}\ n) & - & 0 & = & \mathrm{suc}\ n \\
0 & - & (\mathrm{suc}\ n) & = & 0 \\
(\mathrm{suc}\ n) & - & (\mathrm{suc}\ m) & = & n - m
\end{array}
$$

Reorder lines:

$$
\begin{array}{llll}
\_ - \_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
0 & - & 0 & = & 0 \\
0 & - & (\mathrm{suc}\ n) & = & 0 \\
(\mathrm{suc}\ n) & - & 0 & = & \mathrm{suc}\ n \\
(\mathrm{suc}\ n) & - & (\mathrm{suc}\ m) & = & n - m
\end{array}
$$

Example Reduction to Simple Pattern Matching

Make case distinction on first argument only and delegate it to auxiliary functions e and f:

$$
\begin{array}{l}
\mathrm{mutual} \\
\quad \_-\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
\quad \begin{array}{llll} 0 & - \quad m & = & e\ m \\ (\mathrm{suc}\ n) & - \quad m & = & f\ n\ m \end{array} \\
\\
\quad e : \mathbb{N} \to \mathbb{N} \\
\quad \begin{array}{llll} e & 0 & = & 0 \\ e & (\mathrm{suc}\ n) & = & 0 \end{array} \\
\\
\quad f : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
\quad \begin{array}{lllll} f & n & 0 & = & \mathrm{suc}\ n \\ f & n & (\mathrm{suc}\ m) & = & n - m \end{array}
\end{array}
$$

Example 2 Reduction to Simple Pattern Matching

Original code:

$$
\begin{array}{lll}
f : \mathbb{N} \to \mathbb{N} & & \\
f \quad 0 & = & 5 \\
f \quad (\mathrm{suc}\ 0) & = & 12 \\
f \quad (\mathrm{suc}\ (\mathrm{suc}\ n)) & = & (f\ \ n) * n
\end{array}
$$

Reduct:

$$
\begin{array}{l}
\text{mutual} \\
\quad \begin{array}{lll}
f : \mathbb{N} \to \mathbb{N} & & \\
f \quad 0 & = & 5 \\
f \quad (\mathrm{suc}\ n) & = & g\ \ n \\
 & & \\
g : \mathbb{N} \to \mathbb{N} & & \\
g \quad 0 & = & 12 \\
g \quad (\mathrm{suc}\ n) & = & (f\ \ n) * n
\end{array}
\end{array}
$$

Termination of the Reductions

- If $\mathcal{A}$ is Agda code, f a function of $\mathcal{A}$ with pattern matching terms

$$\mathrm{m}^{\mathcal{A}}(f) := \begin{cases} 0 & \text{if } f \text{ has simple pattern} \\ \text{sum of length of all patterns of } f & \text{otherwise} \end{cases}$$

- Let for Agda code $\mathcal{A}$

$$m(\mathcal{A}) = \{|\mathrm{m}^{\mathcal{A}}(f) \mid f \text{ function symbol defined by pattern matching in } \mathcal{A}$$

where $\{| \cdots |\}$ denotes a multiset.

Main Difficulty

- Show that each reduction step induces the properties mentioned before.

Proof of Main Theorem

- First reduce Agda code to simple pattern matching using Theorem on Unnesting of Pattern Matching.
- Then use the above proof for Agda code having simple pattern matching.

Extensions

- Negated axioms such as $\neg(0_\mathbb{R} == 1_\mathbb{R})$ are currently forbidden
  - Have form $0_\mathbb{R} == 1_\mathbb{R} \to \bot$ 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 $0_\mathbb{R} == 1_\mathbb{R}$ and $\neg(0_\mathbb{R} == 1_\mathbb{R})$.)
  - In case of an inconsistency we would get a proof $p : \bot$ and therefore

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

is non-canonical 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.

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

Addition of Coalgebraic Types

- Original proof didn't include coalgebraic 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 coalgebraic, t closed term, $t : A$, and elim is an eliminator of A, then $\mathrm{elim}\ t$ has a reduction.

Easy Proofs

- Acclimatised 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.
  - 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

$$
\begin{array}{llll}
\text{postulate} & \mathbb{N}_{\text{partial}} & : & \text{Set} \\
\text{postulate} & \_==\_ & : & \mathbb{N}_{\text{partial}} \to \mathbb{N}_{\text{partial}} \to \text{Set} \\
\text{postulate} & 0 & : & \mathbb{N}_{\text{partial}} \\
\text{postulate} & \text{suc} & : & \mathbb{N}_{\text{partial}} \to \mathbb{N}_{\text{partial}} \\
\text{postulate} & \text{f} & : & \mathbb{N}_{\text{partial}} \to \mathbb{N}_{\text{partial}} \\
\text{postulate} & \text{lemf0} & : & \text{f } 0 == \cdots \\
\text{postulate} & \text{lemfs} & : & (n : \mathbb{N}_{\text{partial}}) \to \text{f } (\text{suc } n) == \cdots
\end{array}
$$

$\text{data } \mathbb{N} : \mathbb{N}_{\text{partial}} \to \text{Set where}$

$\quad \text{zerop} : \mathbb{N}\ 0$

$\quad \text{succp} : (n : \mathbb{N}_{\text{partial}}) \to \mathbb{N}\ n \to \mathbb{N}\ (\text{suc } n)$

$\quad \text{eqp} : (n\ m : \mathbb{N}_{\text{partial}}) \to \mathbb{N}\ n \to n == m \to \mathbb{N}\ m$

$\text{lemma} : (n : \mathbb{N}_{\text{partial}}) \to \mathbb{N}\ n \to \mathbb{N}\ (f\ n)$

$\text{lemma } n\ p = \cdots$

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.