Extraction of Programs from Proofs about Real Numbers in Dependent Type Theory

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

August 24, 2010

1. Introduction
2. Restrictions and assumptions about Agda
3. Proof Part 1: Proof of Theorem assuming simple pattern matching
4. Proof Part 2: Reduction to simple pattern matching
5. Conclusion

Goal

- We want use dependent type theory for extracting programs from intuitionistic proofs about real numbers.
  - System to be used is **Agda**
- We want to use the fact that in dependent type theory proofs and programs are the same.
- Therefore if we have

$$p : \forall x : A.\exists y : B.\varphi\ x,\ y$$

  we get a function

$$f := \lambda x.\pi_0\ (p\ x) : A \rightarrow B$$

  s.t.

$$\lambda x.\pi_1\ (p\ x) : \forall x : A.\varphi\ x\ (f\ x)$$

- Question: What happens if we add axioms, e.g. axioms formalising the real numbers.

Real Number Computations

- For formalising real numbers we follow the approach by Berger.
- For axiomatising the real numbers we postulate

$$\mathbb{R} : \mathrm{Set}$$

  together with certain operations and their properties.
- We will define coalgebraically

$$\mathrm{SignedDigit} : \mathbb{R} \rightarrow \mathrm{Set}$$

  the set of real numbers which have a signed digit representation, i.e. which can be written as

$$0.d_0 d_1 d_2 \cdots$$

  where $d_i \in \{-1, 0, 1\}$.
  (They are necessarily elements of the interval $[-1, 1]$).

Streams

- Let Stream be the data type of signed digit streams.
- We can define

$$\mathrm{toStream} : (r : \mathbb{R}) \to \mathrm{SignedDigit}\ r \to \mathrm{Stream}$$

  which determines for an element $r : \mathbb{R}$ s.t. $\mathrm{SignedDigit}\ r$ holds its signed digit representation.
- We can define

$$\mathrm{toList} : \mathrm{Stream} \to \mathbb{N} \to \mathrm{List\ Digit}$$

  which determines for a stream s and $n : \mathbb{N}$ the list of the first n digits of s.

Real Number Computations

- We will show that the signed digits are closed under certain operations e.g.

  $\forall r, s : \mathbb{R}.\mathrm{SignedDigit}\ r \rightarrow \mathrm{SignedDigit}\ s \rightarrow \mathrm{SignedDigit}\ (\mathrm{av}\ r\ s)$
  $\forall r, s : \mathbb{R}.\mathrm{SignedDigit}\ r \rightarrow \mathrm{SignedDigit}\ s \rightarrow \mathrm{SignedDigit}\ (r * s)$
  $\mathrm{SignedDigit}\ \frac{\sqrt{2}}{2}$

  and potentially more complicated operations.
  (Here av is the average function

  $$\mathrm{av}\ r\ s = \frac{r+s}{2}$$

  Since elements of $\mathrm{SignedDigit}$ are in $[-1, 1]$ signed digit are not closed under $+$; however, they are closed under under av).

Real Number Computations

- Therefore we can determine certain $r : \mathbb{R}$ s.t.

$$p : \mathrm{SignedDigit}\ r$$

holds.

- Then

$$q : \mathrm{toList}\ (\mathrm{toStream}\ r\ p)\ n$$

is the list of the first n digits of r.

- We would like that q evaluates to

$$[d_0, \ldots, d_{n-1}]$$

for some $d_i : \mathrm{Digit}$, so in ordinary mathematics

$$r = 0.d_0 \cdots d_{n-1} \cdots$$

Real Number Computations

- For instance we could find d_i s.t.

$$\frac{\sqrt{2}+\sqrt{2}}{4} = 0.d_0 \cdots d_{n-1} \cdots$$

- Our approach should be extensible to more advanced functions carried out by Ulrich Berger.
- Problem: Evaluation of q might make use of the axioms used which are just postulates.

Example 1

- Assume we introduce the axiom

$$\text{postulate axiom1} : \neg\ (0 \# 0)$$

which is

$$\text{postulate axiom1} : 0 \# 0 \to \bot$$

- Let's axiomatise errnoeously as well

$$\text{postulate wrongAxiom} : 0 \# 0$$

- We can define

$$\begin{array}{lll} \text{lemma} & : & \bot \to \text{Digit} \\ \text{lemma} & () & \end{array}$$

- Now

$$\text{lemma (axiom1 wrongAxiom)} : \text{Digit}$$

doesn't normalise.

Example 2

- Assume the correct axiom

$$\mathrm{axiom2} : -0 == 0$$

- The equality is defined in Agda (using a hidden argument $\{A : \mathrm{Set}\}$) as

$$\begin{aligned}&\mathrm{data}\ \_ == \_\ \{A : \mathrm{Set}\}\ (a : A) : A \to \mathrm{Set}\ \mathrm{where}\\ &\quad \mathrm{refl} : a == a\end{aligned}$$

  $\_ == \_$ means that the arguments of $==$ are written before and after it (infix).
  $a == b$ is defined for all $a, b : A$ by having $\mathrm{refl} : a == a$ for all $a : A$.

- Define by case distinction on $==$

$$\begin{aligned}&\mathrm{transfer} : (P : \mathbb{R} \to \mathrm{Set}) \to (r, s : \mathbb{R}) \to r == s \to P\ r \to P\ s\\ &\mathrm{transfer}\ P\ r\ r\ \mathrm{refl}\ p = p\end{aligned}$$

Example 2

$\mathrm{transfer} : (P : \mathbb{R} \to \mathrm{Set}) \to (r, s : \mathbb{R}) \to r == s \to P\ r \to P\ s$
$\mathrm{transfer}\ P\ r\ r\ \mathrm{refl}\ p = p$

- Let $P : \mathbb{R} \to \mathrm{Set}$, $P\ r = \mathrm{Digit}$.
- Then

$$q := \mathrm{transfer}\ P\ {-0}\ 0\ \mathrm{axiom2}\ 0 : \mathrm{Digit}$$

but doesn't normalise, since $\mathrm{axiom2}$ doesn't normalise to a constructor of $-0 == 0$.

1. Introduction

2. Restrictions and assumptions about Agda

3. Proof Part 1: Proof of Theorem assuming simple pattern matching

4. Proof Part 2: Reduction to simple pattern matching

5. Conclusion

Restrictions on Language of Agda (Types)

For simplicity we restrict our language.
We have as types

- postulated types

$$\text{postulate } A : B \rightarrow C \rightarrow \text{Set}$$

- non-indexed (but possibly parametrized) algebraic and coalgebraic data types

$$\begin{array}{l}\text{(co)data } A\ (B : \text{Set})\ (n : \mathbb{N}) : \text{Set where} \\ \quad C_0 : A\ B\ n \rightarrow A\ B\ n \\ \quad C_1 : \mathbb{N} \rightarrow A\ B\ n \\ \quad \ldots\end{array}$$

  - So $A\ B\ n$ refers only to $A\ B\ n$.

Restrictions on Language of Agda (Types)

- restricted indexed algebraic and coalgebraic data types

$$
\begin{array}{l}
\text{(co)data}\ A\ (B : \text{Set}) : (n : \mathbb{N}) \to \text{Set where} \\
\quad C_0 : (n : \mathbb{N}) \to A\ B\ 0 \to A\ B\ n \\
\quad C_1 : (n : \mathbb{N}) \to A\ B\ (n+3) \to A\ B\ n \\
\quad \cdots
\end{array}
$$

  - So $A\ B\ n$ can refer to $A\ B\ n'$ for other n' but n is first argument of constructor (constructors are uniform in n).
- The equality type $\_ == \_$ which is the only generalised indexed inductive definition allowed:

$$
\begin{array}{l}
\text{data}\ \_ == \_\ \{A : \text{Set}\}\ (a : A) : A \to \text{Set where} \\
\quad \text{refl} : a == a
\end{array}
$$

Restrictions on Language of Agda (Types)

- Dependent function types

$$(a_1 : A_1) \to (a_2 : A_2) \to \cdots \to A_n$$

- Types defined in the same way as functions below.
- Not allowed in this setting:
  - other generalised indexed inductive definitions,
  - induction-recursion,
  - induction-induction,
  - record types.

Restrictions on Language of Agda (Functions)

- We have postulated functions

$$\mathrm{postulate}\ f : (a_1 : A_1) \to \cdots \to A_n$$

- We have directly defined functions

$$\begin{aligned} &f : (a_1 : A_1) \to \cdots \to A_{n+1} \\ &f\ a_1 \cdots a_n = s \end{aligned}$$

- We have functions defined by possibly deep pattern matching e.g.

$$\begin{aligned} &f : (a : A) \to (b : B) \to C \\ &f\ (\mathrm{C}_1\ (\mathrm{C}_2\ x))\ (\mathrm{C}_3\ y) = s \\ &f\ (\mathrm{C}_1\ (\mathrm{C}_2'\ x))\ () \end{aligned}$$

(second line absurdity pattern, assuming $B[a := \mathrm{C}_1\ (\mathrm{C}_2'\ x)]$ is a directly empty algebraic data type (no constructor)).

Restrictions on Language of Agda (Functions)

- Not allowed:
  - let and where-expressions (can be reduced easily).
  - No with-expressions (can be reduced as well).

Restrictions on Language of Agda (Functions)

- Functions can be defined mutually.
- Functions can be defined recursively.
  - Termination checker of Agda imposes restrictions.
  - We assume that Agda with these restrictions is normalising.
  - The theory of coalgebras (represented by codata) is not fully worked out in Agda yet, but a satisfactory solution is possible.
- That functions defined by pattern matching have complete pattern matching is guaranteed by the coverage checker.

Assumptions about Agda

- We assume termination and coverage checked Agda code is normalising and coverage complete.

Specific Restricitions on Agda code

- Postulated functions have as result type equalities or postulated types.
  - Therefore postulated axioms which imply negations are not allowed:

    $$\mathrm{axiom1} : \neg\,(0 \# 0)$$

    stands for

    $$\mathrm{axiom1} : 0 \# 0 \to \bot$$

    which has as result type an algebraic data type ($\bot$ which is the empty algebraic data type)
- Functions defined by case distinction on equalities have as result type only equalities or postulated types.
  - So when using postulated functions and equalities we stay within equalities and postulated types.

Theorem

- Assume Agda code with these restrctions.
- Assume $r : A$ in normal form, where A is an algebraic data type.
- Then r starts with a constructor.

Especially,

- If $r : \mathrm{List\ Digit}$, r in normal form, then $r = [d_1, \ldots, d_n]$ for some n and $d_i \in \{-1, 0, 1\}$.

1. Introduction

2. Restrictions and assumptions about Agda

3. Proof Part 1: Proof of Theorem assuming simple pattern matching

4. Proof Part 2: Reduction to simple pattern matching

5. Conclusion

Proof

- Assume we have only simple pattern matching for functions with result types non-generalised algebraic/coalgebraic data types, i.e. functions are defined by pattern matching have only complete non-nested patterns on one argument:

$$
\begin{array}{l}
f : (a_1 : A_1) \to \cdots \to (a_k : A_k) \to \cdots \to (a_n : A_n) \to A_{n+1} \\
f\ x_1 \cdots x_{k-1}\ (C_1\ y^1_1 \cdots y^1_{n_1})\ x_{k+1} \cdots x_n = s_1 \\
\cdots \\
f\ x_1 \cdots x_{k-1}\ (C_l\ y^l_1 \cdots y^l_{n_l})\ x_{k+1} \cdots x_n = s_1
\end{array}
$$

or

$$
\begin{array}{l}
f : (a_1 : A_1) \to \cdots \to (a_k : A_k) \to \cdots \to (a_n : A_n) \to A_{n+1} \\
f\ x_1 \cdots x_{k-1}\ ()\ x_{k+1} \cdots x_n
\end{array}
$$

Proof of Part 1

- Induction on length of r.
- Assume $r : A$ in normal form, A algebraic data type.
- Show r starts with a constructor.
- Let $r = f\ r_1 \cdots r_n$.
  - Assume f is not a constructor.
  - f cannot be a postulated function or defined by case distinction on an equality.
  - f cannot be directly defined.
  - So f is defined by pattern matching on one argument say argument No. i.
    - By IH r_i starts with a constructor.
    - So r reduces in one step, is not in normal form, a contradiction.

1. Introduction

2. Restrictions and assumptions about Agda

3. Proof Part 1: Proof of Theorem assuming simple pattern matching

4. Proof Part 2: Reduction to simple pattern matching

5. Conclusion

Theorem

- Agda code following the assumptions can be reduced to
  - normalising and coverage complete Agda code
  - fulfilling the assumptions and
  - using only simple pattern matching for functions having result types non-generalised (co)algebraic data types.

Proof

- Assume a function which has no simple pattern matching:

$$
\begin{array}{l}
f : (x_1 : B_1) \to \cdots \to (x_n : B_n) \to A \\
f\ x_1 \cdots x_{k-1}\ r_k^1\ \cdots r_n^1 = s_1 \\
\cdots \\
f\ x_1 \cdots x_{k-1}\ r_k^l\ \cdots r_n^l = s_l
\end{array}
$$

where one of r_k^i is not a variable.

Step 1

- Replace if r_k^i is a variable this by having a simple pattern matching on that argument:
  Assume B_k has constructors $\mathrm{C}_1, \ldots, \mathrm{C}_l$ (we assume here the easier case of non-indexed inductive definitions).
  Assume r_k^1 is a variable.
  Replace the above by

$$
\begin{array}{l}
f : (x_1 : B_1) \to \cdots \to (x_n : B_n) \to A \\
f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_1\ y_1^1\ \cdots\ y_{n_1}^1)\ r_k^1\ \cdots r_n^1 = s_1[\cdots] \\
\cdots \\
f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_l\ y_1^l\ \cdots\ y_{n_l}^l)\ \ r_k^1\ \cdots r_n^1 = s_1[\cdots] \\
f\ x_1 \cdots x_{k-1}\ r_k^2\ \cdots r_n^1 = s_2 \\
\cdots \\
f\ x_1 \cdots x_{k-1}\ r_k^l\ \cdots r_n^l = s_l
\end{array}
$$

Step 2

- Assume Step 1 has been carried out so that no variables occur in column k.

Step 2

- Assume we have

$$
\begin{array}{l}
f : (x_1 : B_1) \to \cdots \to (x_n : B_n) \to A \\
f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_1\ s_1^{1,1} \cdots s_{n_1}^{1,1})\ r_{k+1}^{1,1}\ \cdots r_n^{1,1} = t^{1,1} \\
\cdots \\
f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_1\ s_1^{1,j} \cdots s_{n_1}^{1,j})\ r_{k+1}^{1,j}\ \cdots r_n^{1,j} = t^{j,1} \\
f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_2\ s_1^{2,1} \cdots s_{n_2}^{2,1})\ r_{k+1}^{2,1}\ \cdots r_n^{2,1} = t^{2,1} \\
\cdots \\
f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_2\ s_1^{2,j'} \cdots s_{n_2}^{2,j'})\ r_{k+1}^{2,j'}\ \cdots r_n^{2,j'} = t^{2,j'} \\
\cdots \\
f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_l\ s_1^{l,1} \cdots s_{n_l}^{l,1})\ r_{k+1}^{l,1}\ \cdots r_n^{l,1} = t^{l,1} \\
\cdots \\
f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_l\ s_1^{l,j'} \cdots s_{n_l}^{l,j'})\ r_{k+1}^{l,j'}\ \cdots r_n^{l,j'} = t^{l,j'}
\end{array}
$$

Step 2

- Replace this by defining mutually

$$
\begin{aligned}
& f : (x_1 : B_1) \to \cdots \to (x_n : B_n) \to A \\
& f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_1\ y_1 \cdots y_{n_1})\ x_{k+1} \cdots x_n \\
& \qquad\qquad = g_1\ x_1 \cdots x_{k-1}\ y_1 \cdots y_{n_1}\ x_{k+1} \cdots x_n \\
& \cdots \\
& f\ x_1 \cdots x_{k-1}\ (\mathrm{C}_l\ y_1 \cdots y_{n_l})\ x_{k+1} \cdots x_n \\
& \qquad\qquad = g_l\ x_1 \cdots x_{k-1}\ y_1 \cdots y_{n_l}\ x_{k+1} \cdots x_n \\
& \cdots \\
& g_i : \cdots \\
& g_i\ x_1 \cdots x_{k-1}\ s_1^{i,1} \cdots s_{n_i}^{i,1}\ r_{k+1}^{i,1} \cdots r_n^{i,1} = t^{i,1}[\cdots] \\
& \cdots \\
& g_i\ x_1 \cdots x_{k-1}\ s_1^{i,j''} \cdots s_{n_i}^{i,j''}\ r_{k+1}^{i,j''} \cdots r_n^{i,j''} = t^{i,j''}[\cdots]
\end{aligned}
$$

Termination of this Procudure

- Difficulty: find a well-founded measure for Agda code such that after carrying out several steps 1 and one step 2 the measure is reduced.
- Problem: Step 1 increases the length of the pattern matching.

1. Introduction

2. Restrictions and assumptions about Agda

3. Proof Part 1: Proof of Theorem assuming simple pattern matching

4. Proof Part 2: Reduction to simple pattern matching

5. Conclusion

Conclusion

- We can extract in Agda programs from proofs using postulated axioms, if restrictions are applied.
- Chi Ming Chuang has shown that signed digit reals are closed under av and $*$ and contain the rationals.
- We could obtain programs normalising to signed digit representations for some real numbers.
- In order to execute them the compiled version of Agda needed to be used.