Undecidability of Equality for Codata Types

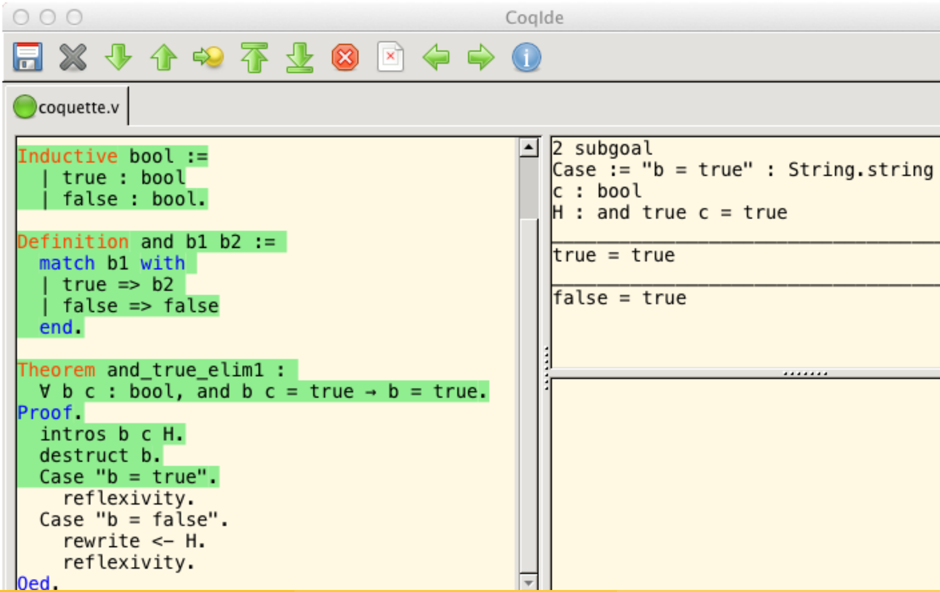
Ulrich Berger and Anton Setzer

Swansea University

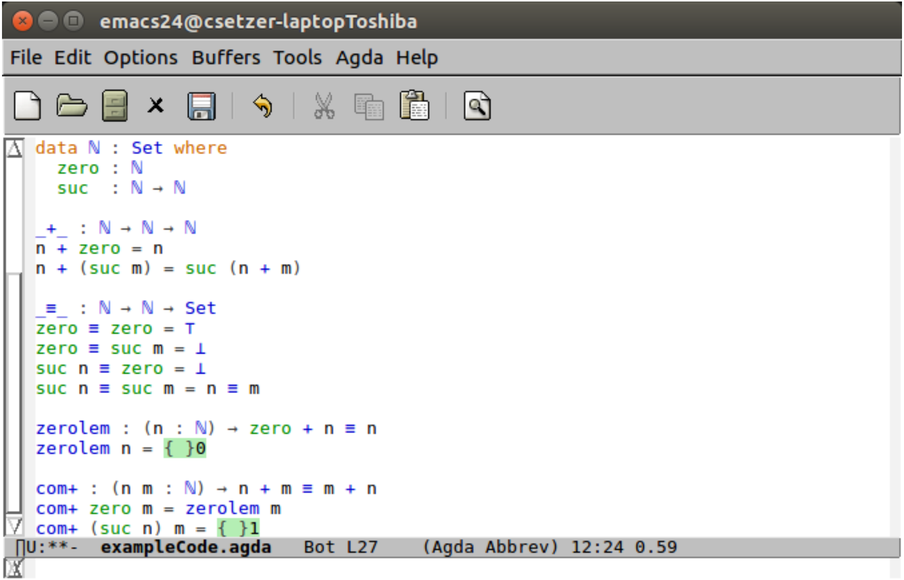
CMCS'18 Thessaloniki, Greece

15 April 2018

Goal Directed Theorem Prover (Here Coq)



Theorems as Functional Programs with Holes (Agda)



Need for Decidability of Equality

- Agda's approach requires decidability of type checking.
- Type checking for dependently typed programs relies on a decidable equality:

$$\lambda X.\lambda x.x : \Pi_{X:A\to\mathrm{Set}}(X\ a \to X\ b) \Leftrightarrow a \text{ and } b \text{ are equal elements of } A$$

Three Equalities in Agda

- **Definitional equality** - **decidable** equality used during type checking.

$$f = g : \mathbb{N} \rightarrow \mathbb{N} \Leftrightarrow f, g \text{ are “equivalent” programs.}$$

- **User-defined equalities.**
  - Can be **undecidable**.
  - Can be used to prove correctness of programs.
  - For coalgebras the standard choice is **bisimilarity** defined coinductively.
- **Propositional equality**.
  - Generic equality type based on definitional equality.
  - Not relevant for this talk.

Codata Types

- Algebraic data types introduce **least** fixed points:

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

- Codata types introduce **largest** fixed point:

$$\begin{array}{l} \text{codata Stream} : \text{Set where} \\ \quad \_{::}\_ : \mathbb{N} \to \text{Stream} \to \text{Stream} \end{array}$$

$$\begin{array}{l} \text{fun2Stream} : (\mathbb{N} \to \mathbb{N}) \to \text{Stream} \\ \text{fun2Stream } f = f\ 0 :: \text{fun2Stream } (f \circ \text{suc}) \end{array}$$

- Infinite terms + non normalisation unless we restrict expansion:

$$\begin{array}{lcl} \text{fun2Stream } f & = & f\ 0 :: \text{fun2Stream } (f \circ \text{suc}) \\ & = & f\ 0 :: f\ 1 :: \text{fun2Stream } (f \circ \text{suc}^2) \\ & = & f\ 0 :: f\ 1 :: f\ 2 :: \text{fun2Stream } (f \circ \text{suc}^3) \end{array}$$

Problems of Codata Types

- This implies that if for some n

$$\begin{array}{c} \forall k < n \,.\, f\ k = g\ k \\ f \circ \mathrm{suc}^n = g \circ \mathrm{suc}^n \end{array}$$

  then

$$\begin{array}{rcl} \mathrm{fun2Stream}\ f & = & f\ 0 :: f\ 1 :: \cdots :: f\ (n-1) :: \mathrm{fun2Stream}\ (f \circ \mathrm{suc}^n) \\ & = & g\ 0 :: g\ 1 :: \cdots :: g\ (n-1) :: \mathrm{fun2Stream}\ (g \circ \mathrm{suc}^n) \\ & = & \mathrm{fun2Stream}\ g \end{array}$$

- But this makes the equality **undecidable**.

Problems of Codata Types

- Definition of functions by pattern matching:

$$\begin{aligned}&\mathrm{inc} : \mathrm{Stream} \to \mathrm{Stream}\\&\mathrm{inc}\ (n :: s) = (n+1) :: \mathrm{inc}\ s\end{aligned}$$

  - Assumes every $s : \mathrm{Stream}$ is of the form $s = n :: s'$ for some t.
  - We will see that this results in undecidability of equality.
- Problem was fixed in Coq and early versions of Agda by applying special restrictions on when to expand the defining equations for fun2Stream. Resulted in **subject-reduction problem**

Coalgebras as Observations + Copattern Matching

- New approach (Abel, Pientka, Setzer, Thibodeau, POPL'13):
- Coinductive Types defined by **observations**:

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

- Elements of Stream defined by **copattern matching**:

$$
\begin{array}{l}
\mathrm{fun2Stream} : (\mathbb{N} \to \mathbb{N}) \to \mathrm{Stream} \\
\begin{array}{lcl}
\mathrm{head}\ (\mathrm{fun2Stream}\ f) & = & f\ 0 \\
\mathrm{tail}\ \ (\mathrm{fun2Stream}\ f) & = & \mathrm{fun2Stream}\ (f \circ \mathrm{suc})
\end{array}
\end{array}
$$

  - $(\mathrm{fun2Stream}\ f)$ is in normal form, if f in normal form.
  - Reductions are only carried out after applying head or tail to it.

Constructor as Defined Operation

- $\_::\_$ is not a constructor but defined by copattern matching:

$$\begin{array}{lll} \multicolumn{3}{l}{\_::\_ : \mathbb{N} \to \mathrm{Stream} \to \mathrm{Stream}} \\ \mathrm{head}\ (n :: s) & = & n \\ \mathrm{tail}\ \ (n :: s) & = & s \end{array}$$

- We don't have

$$s = \mathrm{head}\ s :: \mathrm{tail}\ s$$

Applications of the Copattern Approach

Examples of projects of using copattern matching for proving theorems in Agda

- With Chuang: Representation of **constructive reals** using coalgebras. (PhD thesis Chi Ming Chuang).
- With Bashar Igried: **CSP-Agda**.
  - Representation of the **process algebra CSP** in Agda in a coalgebraic way.
  - Proof of algebraic laws using trace semantics, stable failures semantics, failures divergences infinite traces semantics, bisimilarity, and divergence respecting weak bisimilarity.
- With Peter Hancock **IO monad** as coalgebra.
- With Andreas Abel and Stephan Adelsberger: Representations of **objects** and **GUIs** as coalgebras. (Abel, Adelsberger, Setzer, J Functional Programming 2017)

Encoding of Streams

Definition

(a) An encoding of streams $(\mathrm{Stream}, \mathrm{head}, \mathrm{tail}, ==)$ is given by:

1. A subset $\mathrm{Stream} \subseteq \mathbb{N}$.
2. An equivalence relation $== \subseteq \mathrm{Stream} \times \mathrm{Stream}$ written infix.
3. Functions $\mathrm{head} : \mathrm{Stream} \to \mathbb{N}$, $\mathrm{tail} : \mathrm{Stream} \to \mathrm{Stream}$ that are congruences.

(b) An encoding of streams is injective if $\langle \mathrm{head}, \mathrm{tail} \rangle$ is injective i.e.

$$\forall s, s' : \mathrm{Stream} \, . \, \mathrm{head}(s) = \mathrm{head}(s') \wedge \mathrm{tail}(s) == \mathrm{tail}(s') \to s == s'$$

(c) An encoding of streams is universal if it allows to define functions by primitive corecursion.

(c) An encoding of streams is coiteratively universal if it allows to define functions by primitive coiteration.

Equalities Extending $==$

Definition

Assume an encoding of streams.

$$
\begin{aligned}
s ==_{<\omega} t &\Leftrightarrow \exists n.(\forall i < n.(s)_i = (t)_i) \wedge \mathrm{tail}^n(s) == \mathrm{tail}^n(t) \\
s \sim t &\Leftrightarrow \forall i \in \mathbb{N} \,.\, (s)_i = (t)_i
\end{aligned}
$$

Injectivity does not imply Bisimilarity

Lemma

(a) $==_{<\omega}$ *is the least injective equivalence relation containing* $==$ *and respecting* $\mathrm{head}, \mathrm{tail}$*.*

(b) $== \subseteq ==_{<\omega} \subseteq \sim$.

(c) *For the standard model of streams in Agda we have that* $== \neq ==_{<\omega} \neq \sim$.

Decidable Streams Not Determined by head, tail

Theorem

(a) *Every injective universal encoding of streams has an undecidable equality.*

(b) *The same applies to injective coiteratively universal encodings.*

Decidable Streams Not Always of Form $\mathrm{cons}(n, s)$

Corollary

(a) *Assume a universal or coiteratively universal encoding of streams together with a* cons *function respecting equalities. If*

$$\forall s : \mathrm{Stream}\,.\, s == \mathrm{cons}(\mathrm{head}(s), \mathrm{tail}(s))$$

then $==$ *is undecidable.*

(b) *Assume* cons *as in (a). Assume*

$$\forall s : \mathrm{Stream}, n : \mathbb{N}\,.\, \mathrm{head}(\mathrm{cons}(n, s)) = n \wedge \mathrm{tail}(\mathrm{cons}(n, s)) == s$$
$$\forall s : \mathrm{Stream}\,.\, \exists n, s'\,.\, s == \mathrm{cons}(n, s')$$

Then $==$ *is undecidable.*

(c) $==_{<\omega}$ *and* $\sim$ *are both undecidable.*

Proof of Main Theorem

- A proof of undecidability of $\sim$ is easy since extensional equality on $\mathbb{N} \to \mathbb{N}$ is undecidable by undecidability of Turing halting problem.
- We cannot use this fact, since in general $==_{<\omega} \neq \sim$.
- Instead we use the following theorem from computability theory, where $\{e\}$ is the partial function defined by the eth Turing Machine:

Theorem

(Rosser, Kleene, Novikov, Trakhtenbrot)
Let $A := \{e \mid \{e\} \simeq 0\}$ *and* $B := \{e \mid \{e\} \simeq 1\}$.
Then A *and* B *are recursively inseparable:*
There is no (total) computable function $f : \mathbb{N} \to \{0, 1\}$ *such that*
$\forall e \in A \,.\, f(e) = 0$ *and* $\forall e \in B \,.\, f(e) = 1$

Proof of Main Theorem

- Assume a universal injective encoding of streams.
- We define $f : \mathbb{N} \to \mathrm{Stream}$ mapping Turing Machines with code e to streams as follows.
  If we had codata types the definition would be:
  - If e terminates after k steps with result r

    $$f(e) = \underbrace{0 :: 0 :: \cdots :: 0}_{k} :: r :: r :: r :: \cdots$$

  - If e never terminates then

    $$f(e) = 0 :: 0 :: \cdots$$

Proof of Main Theorem

- $f(e) = g(e, 0)$ where

$$
\begin{array}{rcl}
\mathrm{head}(g(e,n)) & = & 0 \\
\mathrm{tail}(g(e,n)) & = & \begin{cases} g(e, n+1) & \text{if } e \text{ has not terminated} \\ & \text{after } k \text{ steps} \\ \mathrm{const}(r) & \text{if } e \text{ has terminated} \\ & \text{after } k \text{ steps with result } r \end{cases}
\end{array}
$$

  where $\mathrm{const}(r)$ is a fixed constant stream returning always r.
- g defined by primitive corecursion.
  It can be defined with some extra effort by primitive coiteration.
- It is crucial that after having terminated we give back the same stream $\mathrm{const}(r)$, not only a stream bisimilar to $\mathrm{const}(r)$.

Proof of Main Theorem

- Assume that the encoding of streams is injective.
- If $\{e\} \simeq 0$, then $f(e) == \mathrm{const}(0)$.
- If $\{e\} \simeq 1$ then $\neg(f(e) == \mathrm{const}(0))$.
- So if $==$ were decidable, the function

$$\lambda e.f(e) == \mathrm{const}(0)$$

would separate $\{e \mid \{e\} \simeq 0\}$ from $\{e \mid \{e\} \simeq 1\}$, a contradiction.

Conclusion

- Decidable type checking requires **decidable definitional equality**.
- With decidable equality we **cannot assume** for weakly final coalgebras that
  - **streams are determined by head and tail**
  - or that **every stream is of the form cons(n,s)**.
- Proof using advanced result from computability theorem, not just undecidability of halting problem for Turing Machines.
- **Codata approach implicitly assumes** that every stream is of the form $\mathrm{cons}(n, s)$, resulting in an undecidable equality.

Conclusion

- Problem of codata types can be fixed by defining coinductive types by observations and copattern matching.
  - However streams are not always of the form $\mathrm{cons}(n, s)$.
- Defining coalgebras by observations and copattern matching has been used in Agda successfully for **large scale implementation and verification** of processes, IO programs, objects and GUIs.
- In Agda there exist a **musical approach** to codata types, which can be considered as syntactic sugar for coalgebras while behaving as close as possible to codata types.
  - Currently not much used.
  - See discussion in CMCS'18 paper.

One Referee: Is the paper nothing but another nail in the coffin of the co-data approach?

Coalgebras to the Rescue