Termination Checking in the Presence of Nested Inductive and Coinductive Types

Thorsten Altenkirch
(joint work with Nils Anders Danielsson)

School of Computer Science
University of Nottingham

July 16, 2010

Context

- Dependently typed programming, e.g. Agda, Epigram, Coq, ...
- Totality?
  - Soundness as a logic
  - Efficient code (don't run proofs)
- Two approaches:
  1. Reduce to a total core language
     Epigram?, Coq ?
  2. Use a partial language and a termination checker
     Agda, Coq?

This talk

- Adding coinductive types to Agda
- Mixed inductive-coinductive definitions
- Simple but powerful extension of the termination checker (due to Andreas Abel).
- Easy to define inductive types nested inside coinductive types ($\nu\mu$).
- Impossible to define coinductive types nested inside inductive types ($\mu\nu$) directly.
- Is this a (serious) issue?
- If so, how can we fix it?

Foetus

- Andreas Abel's master thesis
- Closely related to size change termination (N. Jones et al)

mutual

$$
\begin{aligned}
& f : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
& f\ m\ zero \quad\ = m \\
& f\ m\ (suc\ n) = f\ m\ n + g\ m \\
& g : \mathbb{N} \to \mathbb{N} \\
& g\ zero \quad\ = zero \\
& g\ (suc\ n) = f\ n\ n
\end{aligned}
$$

$$
f \to f : \quad \begin{pmatrix} = & < \\ ? & < \end{pmatrix} \qquad f \to g : \quad \begin{pmatrix} = \\ ? \end{pmatrix} \qquad g \to f : \quad \begin{pmatrix} < & < \end{pmatrix}
$$

Coinductive Definitions in Agda

Streams:

$$\textbf{data}\ \mathit{Stream}\ (A : \mathit{Set}) : \mathit{Set}\ \textbf{where}$$
$$\_ :: \_ : A \to \infty(\mathit{Stream}\ A) \to \mathit{Stream}\ A$$

Categorically: $\mathrm{Stream}\, A = \nu X.A \times X$

Force and Delay:

$$\flat \quad : \{A : \mathit{Set}\} \to \infty A \to A$$
$$\sharp\_ : \{A : \mathit{Set}\} \to A \to \infty A$$

Corecursive programs

$$\mathit{from} : \mathbb{N} \to \mathit{Stream}\ \mathbb{N}$$
$$\mathit{from}\ n = n :: \sharp \mathit{from}\ (\mathit{suc}\ n)$$

$$\mathit{mapStream} : \forall \{A\ B\} \to (A \to B) \to \mathit{Stream}\ A \to \mathit{Stream}\ B$$
$$\mathit{mapStream}\ f\ (a :: as) = f\ a :: \sharp(\mathit{mapStream}\ f\ (\flat as))$$

Functional representation of streams

$$Stream' : Set \rightarrow Set$$
$$Stream'\ A = \mathbb{N} \rightarrow A$$

$$from' : \mathbb{N} \rightarrow (Stream'\ \mathbb{N})$$
$$from'\ n\ 0 = n$$
$$from'\ n\ (suc\ m) = from'\ (suc\ n)\ m$$

$$mapStream' : \{A\ B : Set\} \rightarrow (A \rightarrow B) \rightarrow (Stream'\ A) \rightarrow (Stream'\ B)$$
$$mapStream'\ f\ as\ 0 = f\ (as\ 0)$$
$$mapStream'\ f\ as\ (suc\ n) = mapStream'\ f\ (\lambda\ i \rightarrow as\ (suc\ i))\ n$$

Using subsets (Σ) such a representation (as an ω-limit) exists for all coinductive types.

Extending the termination checker

The translation suggests:

- Coinductive types introduce an additional (invisible) argument.
- Any use of $\sharp$ reduces this argument.
- $\flat$ does not preserve the structural order.

$$\begin{aligned}&\mathit{from} : \mathbb{N} \to \mathit{Stream}\ \mathbb{N}\\ &\mathit{from}\ n = n :: \sharp\mathit{from}\ (\mathit{suc}\ n)\end{aligned}$$

$$\mathit{from} \to \mathit{from} : \quad (< \quad ?)$$

$$\begin{aligned}&\mathit{mapStream} : \forall\{A\ B\} \to (A \to B) \to \mathit{Stream}\ A \to \mathit{Stream}\ B\\ &\mathit{mapStream}\ f\ (a :: as) = f\ a :: \sharp(\mathit{mapStream}\ f\ (\flat as))\end{aligned}$$

$$\mathit{mapStream} \to \mathit{mapStream} : \quad (< \quad = \quad ?)$$

Mixed induction/coinduction

Stream Processors:

$$
\begin{array}{l}
\textbf{data}\ SP\ (A\ B : Set) : Set\ \textbf{where} \\
\quad get : (A \to SP\ A\ B) \to SP\ A\ B \\
\quad put : B \to \infty(SP\ A\ B) \to SP\ A\ B
\end{array}
$$

Categorical interpretation:
$\mathrm{SP}\,A\,B = \nu X.\mu Y.A \to Y + B \times X$

In general:
data $D = F\ (\infty D)\ D$ corresponds to
$D = \nu X.\mu Y.F\,X\,Y$.

Semantics of SP

$$
\begin{array}{l}
\textbf{data}\ SP\ (A\ B : Set) : Set\ \textbf{where} \\
\quad get : (A \to SP\ A\ B) \to SP\ A\ B \\
\quad put : B \to \infty(SP\ A\ B) \to SP\ A\ B
\end{array}
$$

Semantics of stream processors:

$$
\begin{array}{ll}
[\![\_]\!] : \{A\ B : Set\} \to SP\ A\ B \to Stream\ A \to Stream\ B & \\
[\![\ get\ f\ ]\!]\ (a :: as) & = [\![f\ a]\!]\ (\flat as) \\
[\![\ put\ b\ sp\ ]\!]\ as & = b :: \sharp\ [\![\ \flat sp\ ]\!]\ as
\end{array}
$$

Extended Call graph

$$
[\![\_]\!] \to [\![\_]\!] : \quad \begin{pmatrix} = & = & = & < & ? \\ < & = & = & ? & = \\ < & = & = & ? & ? \end{pmatrix}
$$

Composition of SPs

Data driven:

$$
\begin{array}{l}
\_ \ggg_? \_ : \forall\{A\ B\ C\} \to SP\ A\ B \to SP\ B\ C \to SP\ A\ C \\
get\ f \ggg_? tq = get\ (\lambda\ a \to f\ a \ggg_? tq) \\
put\ a\ sp \ggg_? get\ f = \flat sp \ggg_? f\ a \\
put\ a\ sp \ggg_? put\ b\ tq = put\ b\ (\sharp put\ a\ sp \ggg_? \flat tq)
\end{array}
$$

Demand Driven

$$
\begin{array}{l}
\_ \ggg_! \_ : \forall\{A\ B\ C\} \to SP\ A\ B \to SP\ B\ C \to SP\ A\ C \\
get\ g \ggg_! get\ f = get\ (\lambda\ a \to g\ a \ggg_! get\ f) \\
put\ b\ sp \ggg_! get\ f = \flat sp \ggg_! f\ b \\
sp \ggg_! put\ c\ tq = put\ c\ (\sharp(sp \ggg_! \flat tq))
\end{array}
$$

- Both are accepted by the extended termination checker.
- Try to implement them using the categorical combinators.

From $\nu\mu$ to $\mu\nu$?

data *ZO* : *Set* **where**
$0, : ZO \to ZO$
$1, : \infty\, ZO \to ZO$

$ZO = \nu X.\mu Y.(0 : Y) + (1 : X)$

$01^{\omega} : ZO$
$01^{\omega} = 0, (1, (\sharp 01^{\omega}))$

From $\nu\mu$ to $\mu\nu$?

$$\begin{aligned} ZO' &= \mu Y.\nu X.(0 : Y) + (1 : X) \\ &= \mu Y.O\,X \\ &\text{with} \quad O\,X = \nu X.0 : Y + 1 : X \end{aligned}$$

data $O\ (X : Set) : Set$ **where**
  $0, : X \to O\ X$
  $1, : \infty\ (O\ X) \to O\ X$
data $ZO' : Set$ **where**
  $emb : O\ ZO' \to ZO'$

But we can still define:

$01^{\omega} : ZO'$
$01^{\omega} = emb\ (1, (\sharp 0, 01^{\omega}))$

No fold!

$$
\begin{array}{l}
\textit{mutual} \\
\quad \textit{fold} : \forall \{A\} \to (O\ A \to A) \to ZO' \to A \\
\quad \textit{fold}\ f\ (\textit{emb}\ x) = f\ (\textit{mapfold}\ f\ x) \\
\quad \textit{mapfold} : \forall \{A\} \to (O\ A \to A) \to O\ ZO' \to O\ A \\
\quad \textit{mapfold}\ f\ (0, x) = 0, (\textit{fold}\ f\ x) \\
\quad \textit{mapfold}\ f\ (1, x) = 1, (\sharp \textit{mapfold}\ f\ (\flat x))
\end{array}
$$

is not accepted by the termination checker.
The problem is that $\flat$ doesn't preserve the structural order.
Otherwise we could derive a diverging program:

$$
\begin{array}{l}
\textit{foo} : O\ ZO' \to ZO' \\
\textit{foo}\ (0, x) = x \\
\textit{foo}\ (1, x) = \textit{emb}\ (\flat x) \\
\textit{bar} : ZO' \\
\textit{bar} = \textit{fold}\ \textit{foo}\ 01^{\omega}
\end{array}
$$

$\mu\nu$?

- Our attempt to define a $\mu\nu$-type by parametrisation fails.
- We can define infinite elements which shouldn't be there.
- We cannot define fold (or induction) for the μ-type.
- What is going on?

Domain-theoretic explanation ?

- ∞A is interpreted as $A_\perp$ (lifting).
- Recursive datatypes as solutions to (strictly positive) domain equations.
- The termination checker identifies the *total elements* in the domain.
- $ZO = \nu X.\mu Y.0 : Y + 1 : X$ is interpreted as $\mathrm{rec} X.\mathrm{rec} Y.0 : Y + 1 : X_\perp$.
- $ZO' = \mu Y.\nu X.0 : Y + 1 : X$ is interpreted as $\mathrm{rec} Y.\mathrm{rec} X.0 : Y + 1 : X_\perp$.
- In general we have $\mathrm{rec}\ X.\mathrm{rec}\ Y.T\ X\ Y \simeq \mathrm{rec}\ Y.\mathrm{rec}\ X.T\ X\ Y$
- Since the domains are isomorphic, they have the same total elements.
- How to define the total elements for a (strictly positive) domain equation in general?

Explanation by translation (simplified)

- We can explain parametrized types by mutual types.

$$
\begin{array}{l}
\textbf{data}\ O\ (X : Set) : Set\ \textbf{where} \\
\quad 0, : X \to O\ X \\
\quad 1, : \infty\ (O\ X) \to O\ X \\
\textbf{data}\ ZO' : Set\ \textbf{where} \\
\quad emb : O\ ZO' \to ZO'
\end{array}
$$

becomes

$$
\begin{array}{l}
mutual \\
\quad \textbf{data}\ O\_ZO'' : Set\ \textbf{where} \\
\quad\quad 0, : ZO'' \to O\_ZO'' \\
\quad\quad 1, : \infty\ (O\_ZO'') \to O\_ZO'' \\
\quad \textbf{data}\ ZO'' : Set\ \textbf{where} \\
\quad\quad emb : O\_ZO'' \to ZO''
\end{array}
$$

- It is easy to see that *ZO* and ZO'' are isomorphic.

So what?

- We cannot easily define $\mu\nu$ types.
- There are extensionally isomorphic functional encodings.

$$\begin{array}{l} \textbf{data}\ \mathit{Tree} : \mathit{Set}\ \textbf{where} \\ \quad \mathit{leaf} : \mathit{Tree} \\ \quad \mathit{node} : \mathit{Stream}\ \mathit{Tree} \to \mathit{Tree} \end{array}$$

can be encoded as

$$\begin{array}{l} \textbf{data}\ \mathit{Tree}' : \mathit{Set}\ \textbf{where} \\ \quad \mathit{leaf} : \mathit{Tree}' \\ \quad \mathit{node} : (\mathbb{N} \to \mathit{Tree}') \to \mathit{Tree}' \end{array}$$

- This also shows that data types may not preserve extensional isomorphism.
- Maybe *Tree* should be forbidden by saying that *Tree* doesn't appear strictly positive in *Stream Tree*.

Keiko and Tarmo's encoding

- Coq doesn't permit nested datatypes at all. (Not even $\mu\mu$).
- To represent $\nu\mu$ they use left Kan extensions. I.e. FD is replaced by $\Sigma Y.(Y \to D) \times FY$.
- Can we use the same trick to encode $\mu\nu$ in Agda? (Switching off the universe checker).

  - **data** $O\ (X : Set) : Set$ **where**
    $0, : X \to O\ X$
    $1, : \infty\ (O\ X) \to O\ X$

    data $ZO : Set$ **where**
    $emb : \forall\{X\} \to (X \to ZO) \to O\ X \to ZO$

- Indeed, *fold* is definable for this encoding!
- But so is $01\ \omega$.
- Indeed, Agda's termination checker is unsound if we allow impredicativity (unlike Coq's).

The last slide

- Nested ν-types are not treated properly by Agda's termination checker.
- One solution is to outlaw them (we would be still better than Coq).
- One can still use an extensionally isomorphic functional encoding.
- Or can we fix the termination checker?
- One idea is to combine parity games with size change termination.