Non-wellfounded trees in Homotopy Type Theory

Benedikt Ahrens 1  Paolo Capriotti 2  Régis Spadotti 1

1Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier
2School of Computer Science, University of Nottingham

TYPES 2015
Main result

Coinductive types—in the form of M-types—are derivable in HoTT.

- Details in an article with same name (arXiv:1504.02949)
- Agda code on https://hott.github.io/M-types/
Outline

1. What are non-wellfounded trees?

2. Overview of coinductive types in extensions of type theory

3. Outline of the construction of a type of streams

4. The construction in HoTT: difficulties compared to ETT
Outline

1. What are non-wellfounded trees?

2. Overview of coinductive types in extensions of type theory

3. Outline of the construction of a type of streams

4. The construction in HoTT: difficulties compared to ETT
### Coinductive types

- types of potentially infinite data structures
- only a finite part is “visible” at any moment

### M-types

- a particular class of coinductive types
- elements can be visualized as trees of a given shape
- typical examples of coinductive types are M-types:
  - streams
  - colists
Non-wellfounded trees in pictures

Specification of an $M$-type

- a type $A$ of labels for nodes
- a family of types $(B(a))_{a \in A}$ specifying branching

\[
\begin{align*}
& a, b, c \in A \\
& B(a) = \{\} \\
& B(b) = \{1, 2\} \\
& B(c) = \{1, 2, 3\}
\end{align*}
\]
Example: streams

Take $A$ a type and $B := \lambda (a : A).1$. A tree of that shape is

$$a_0 \xrightarrow{1} a_1 \xrightarrow{1} a_2 \xrightarrow{1} \ldots$$
Example: streams

Take \( A \) a type and \( B := \lambda(a : A).1 \). A tree of that shape is

\[
\begin{array}{ccccccc}
  a_0 \quad 1 & a_1 \quad 1 & a_2 \quad 1 & \ldots
\end{array}
\]

Given a stream \( a \), we can decompose it:

\[
\begin{array}{ccccccc}
  a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots
\end{array}
\]
Example: streams

Take $A$ a type and $B := \lambda (a : A).1$. A tree of that shape is

$$
a_0 \overset{1}{\rightarrow} a_1 \overset{1}{\rightarrow} a_2 \overset{1}{\rightarrow} \ldots
$$

Given a stream $a$, we can decompose it:

$$
\begin{array}{c}
\circ \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots \\
\text{head}
\end{array}
$$
Example: streams

Take $A$ a type and $B := \lambda (a : A).1$. A tree of that shape is

$$a_0 \xrightarrow{1} a_1 \xrightarrow{1} a_2 \xrightarrow{1} \ldots$$

Given a stream $a$, we can decompose it:

- head
- tail
Example: streams

Take $A$ a type and $B := \lambda(a : A).1$. A tree of that shape is

$$
a_0 \quad \overset{1}{\longrightarrow} \quad a_1 \quad \overset{1}{\longrightarrow} \quad a_2 \quad \overset{1}{\longrightarrow} \quad \ldots
$$

Given a stream $a$, we can decompose it:

The tail of a stream is again a stream.
What is an M-type, formally?

Two possible ways to formally specify the type of streams:

**externally: via typing rules**

\[
\begin{align*}
A : U & \quad \vdash \quad \text{Stream}A : U \\
\text{head}_A t : A & \quad \vdash \quad \text{head}_A t : \text{Stream}A \\
\text{tail}_A t : \text{Stream}A & \quad \vdash \quad \text{tail}_A t : \text{Stream}A
\end{align*}
\]

**internally: via a universal property**

- as a “terminal coalgebra”: as the pair

\[
(\text{Stream}A, \langle \text{head}, \text{tail} \rangle : \text{Stream}A \to A \times \text{Stream}A)
\]

such that ...
Universal property of streams

...for any pair

\[(T, \langle h, t \rangle : T \to A \times T)\]

there is exactly one \( f : T \to \text{Stream}A \) such that

\[
\begin{array}{ccc}
T & \xrightarrow{\langle h, t \rangle} & A \times T \\
| f \downarrow | & & | 1 \times f \downarrow \\
\text{Stream}A & \xrightarrow{\langle \text{head}, \text{tail} \rangle} & A \times \text{Stream}A
\end{array}
\]

- Call this characterisation “internal”, because is expressible within type theory.
- For any two \((\text{Stream}A, \langle \text{head}, \text{tail} \rangle)\) and \((\text{Stream}'A, \langle \text{head}', \text{tail}' \rangle)\), the carriers are isomorphic.
• In type theory with enough extensionality, the type \( \mathbb{N} \rightarrow A \) satisfies the universal property of \( \text{Stream}A \).
• More generally, can \( M \)-types be built from other type constructors?
1. What are non-wellfounded trees?

2. Overview of coinductive types in extensions of type theory

3. Outline of the construction of a type of streams

4. The construction in HoTT: difficulties compared to ETT
### Martin-Löf type theory (MLTT)

<table>
<thead>
<tr>
<th>Type theory</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inhabitant</td>
<td>(a : A)</td>
</tr>
<tr>
<td>Dependent type</td>
<td>(x : A \vdash B(x))</td>
</tr>
<tr>
<td>Sigma type</td>
<td>(\sum_{(x:A)} B(x))</td>
</tr>
<tr>
<td>Product type</td>
<td>(\prod_{(x:A)} B(x))</td>
</tr>
<tr>
<td>Identity type</td>
<td>(a =_A b)</td>
</tr>
<tr>
<td>Natural numbers</td>
<td>(\mathbb{N})</td>
</tr>
</tbody>
</table>
Construction of coinductive types in extensions of MLTT

with Identity Reflection (extensional TT)

- *Containers - constructing strictly positive types*, Abbott, Altenkirch & Ghani 2005

with Uniqueness of Identity Proofs & FunExt

- *Indexed containers*, Altenkirch + 4, unpublished
Coinductive types in MLTT + something

something = Identity Reflection or UIP

- a type $A$ looks like a (constructive) set
- a universe $\mathcal{U}$ of types has the structure of a topos (roughly)
- construction of terminal coalgebras for polynomial functors on a topos well-known

something = Univalence

- a tower $(A, =_A, =_=A, \ldots)$ looks like an $\infty$-groupoid
- a universe $\mathcal{U}$ has the structure of an “$\infty$-topos”
- this work: construction of terminal coalgebras for polynomial functors on $\infty$-toposes

POST-TALK EDIT: the last point is to be read as an analogy, not as a precise statement. It should have been more clearly marked as such.
Remarks about extensions of MLTT

- For this talk, HoTT = MLTT + Univalence
- No HITs employed in this work
- UIP (as an axiom for all types) and Univalence are incompatible
- Univalence entails function extensionality
- Types satisfying UIP form a “subuniverse” of HoTT
Outline

1. What are non-wellfounded trees?

2. Overview of coinductive types in extensions of type theory

3. Outline of the construction of a type of streams

4. The construction in HoTT: difficulties compared to ETT
In this section

Sketch of classical construction of streams

- applies to ETT (mentioned work) and to HoTT (our work)
- differences between ETT and HoTT are discussed later
How to construct streams

In this section

Sketch of classical construction of streams

• applies to ETT (mentioned work) and to HoTT (our work)
• differences between ETT and HoTT are discussed later

Goal

Using the type constructors of MLTT, define a type $\text{Stream}A$ and maps

$$\langle \text{head}, \text{tail} \rangle : \text{Stream}A \to A \times \text{Stream}A$$

s.t. for any $(T, \langle h, t \rangle : T \to A \times T)$, there is a unique “good”

$$f : T \to \text{Stream}A$$
Approximating streams by a sequence of growing lists

A stream over $A$ is one of...

- a map $\mathbb{N} \to A$,

\[ a_0, a_1, a_2, \ldots \]

- the “limit” of a sequence of finite approximations, i.e., of elements of $A^n$

\[ (a_0), (a_0, a_1), (a_0, a_1, a_2), \ldots \]

in tree form

```
      a_2
     /\  \\
    a_1 /  \\
   /   /  \\
a_0 /   /  \\
   \   \  \\
    \   \  \\
a_0 /   /  \\
   \   \  \\
    \   \  \\
a_0 /   /  \\
   \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\
    \   \  \\...
The latter point of view generalizes from streams to arbitrary trees:
The latter point of view generalizes from streams to arbitrary trees:
The latter point of view generalizes from streams to arbitrary trees:
The latter point of view generalizes from streams to arbitrary trees:
Sequence of lists and projections

\[
\begin{align*}
1 & \xleftarrow{!} A \xleftarrow{l_1} A^2 \xleftarrow{l_2} A^3 \xleftarrow{l_3} \ldots \\
tt & \xleftarrow{\_} a_0 \xleftarrow{\_} (a_0, a_1) \xleftarrow{\_} (a_0, a_1, a_2) \xleftarrow{\_} \ldots
\end{align*}
\]
Sequence of lists and projections

\[
\begin{array}{ccccccc}
1 & \leftarrow & A & \leftarrow & A^2 & \leftarrow & A^3 & \leftarrow & \ldots \\
\hline
tt & \leftarrow & a_0 & \leftarrow & (a_0, a_1) & \leftarrow & (a_0, a_1, a_2) & \leftarrow & \ldots
\end{array}
\]

Definition

A **stream** over \( A \) consists of

- \((x_n : A^n)_{n : \mathbb{N}}\) with
- \(l_n(x_{n+1}) = x_n\)

\[
\text{Stream} A := \sum_{(x : \prod_{(n : \mathbb{N})} A^n)} \prod_{(n : \mathbb{N})} l_n(x_{n+1}) = x_n
\]
Universal property for streams

Definition

\[
\text{Stream} A := \sum_{(x : \prod_{(n : \mathbb{N})} A^n)} \prod_{(n : \mathbb{N})} l_n(x_{n+1}) = x_n
\]

Theorem

\text{Stream} A has the universal property of streams. I.e. we have coalgebra structure

\text{maps } \langle \text{head, tail} \rangle : \text{Stream} A \to A \times \text{Stream} A

universal property

\text{for any } (T, h : T \to A, t : T \to T), \text{ there is a unique “good” } f : T \to \text{Stream} A
Proof of the theorem

Coalgebra structure

- \( \text{head}\left((tt, (a_0), (a'_0, a_1), \ldots), p\right) := a_0 : A \)

- \( \text{tail}\left((tt, (a_0), (a'_0, a_1), ((a'_0, a_1)'), a_2), \ldots), p\right) := \left((tt, (a_1), (a'_1, a_2), ((a'_1, a_2)'), a_3), \ldots), \bar{p}\right) \)

- \( \langle \text{head}, \text{tail} \rangle \) is an equivalence

Universal property

Given \( T, h : T \to A \) and \( t : T \to T \), define \( f : T \to \text{Stream}A \ldots \)
Outline

1. What are non-wellfounded trees?
2. Overview of coinductive types in extensions of type theory
3. Outline of the construction of a type of streams
4. The construction in HoTT: difficulties compared to ETT
Differences between the constructions in ETT and HoTT

- In general, terms of identity type are non-trivial in HoTT
- cannot “ignore” them or equate them to reflexivity

When are two coalgebra morphisms equal?

<table>
<thead>
<tr>
<th></th>
<th>ETT</th>
<th>HoTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ETT</td>
<td>when the carriers are equal—use UIP to equate second components</td>
<td>also need the respective diagrams to commute in the same way</td>
</tr>
</tbody>
</table>
Proof-relevant identities

Proving two morphisms into streams equal involves comparing them pointwise, i.e., comparing streams:

When are two streams \((x, p)\) and \((x', p')\) equal?

- **in ETT** when \(x_n = x'_n\) for any \(n\)
- **in HoTT** also need that \(p : \prod_{n : \mathbb{N}} l_n(x_{n+1}) = x_n\) is equal to \(p' : \prod_{n : \mathbb{N}} l_n(x'_{n+1}) = x'_n\) modulo transport
Proof-relevant identities

Proving two morphisms into streams equal involves comparing them pointwise, i.e., comparing streams:

When are two streams \((x, p)\) and \((x', p')\) equal?

- in ETT when \(x_n = x'_n\) for any \(n\)
- in HoTT also need that \(p : \prod_{(n : \mathbb{N})} l_n(x_{n+1}) = x_n\) is equal to \(p' : \prod_{(n : \mathbb{N})} l_n(x'_{n+1}) = x'_n\) modulo transport

Calculations on paths are made feasible by

- decomposing them into small steps
- steps are often instances of general lemmas about paths, sigma types, etc.
A typical proof involving many paths

\[
L^P_
\simeq \sum_{(w:\prod_{(n:\mathbb{N})} \sum_{(a:A)} B(a) \to X_n)} \prod_{(n:\mathbb{N})} (P \pi_n) w_{n+1} = w_n
\]

\[
\simeq \sum_{(a:\prod_{(n:\mathbb{N})} A)} \sum_{(u:\prod_{(n:\mathbb{N})} B(a_n) \to X_n)} \prod_{(n:\mathbb{N})} (a_{n+1}, \pi_n \circ u_{n+1}) = (a_n, u_n)
\]

\[
\simeq \sum_{(a:\prod_{(n:\mathbb{N})} A)} \sum_{(p:\prod_{(n:\mathbb{N})} a_{n+1} = a_n)} \sum_{(u:\prod_{(n:\mathbb{N})} B(a_n) \to X_n)} \prod_{(n:\mathbb{N})} (p_n)_*(\pi_n \circ u_{n+1}) = u_n
\]

\[
\simeq \sum_{(a:A)} \sum_{(u:\prod_{(n:\mathbb{N})} B(a) \to X_n)} \prod_{(n:\mathbb{N})} \pi_n \circ u_{n+1} = u_n
\]

\[
\simeq \sum_{a:A} B(a) \to L \simeq PL
\]
Concluding remarks

Main result

Coinductive types—in the form of $M$-types—are **derivable** in HoTT.

Computation rule

only holds propositionally in the Agda implementation, due to lack of computational interpretation of UA/FunExt

Extensionality

bisimilarity is identity for the constructed $M$-types
Concluding remarks

Main result

Coinductive types—in the form of $M$-types—are derivable in HoTT.

Computation rule

only holds propositionally in the Agda implementation, due to lack of computational interpretation of UA/FunExt

Extensionality

bisimilarity is identity for the constructed $M$-types

The end