Bachelor thesis
Computer Science

Radboud University

Equality of infinite objects

Author:
Robin Munsterman
s4070968

First supervisor/assessor:
prof. dr. J.H. Geuvers
herman@cs.ru.nl

Second assessor:
prof. dr. H. Zantema
h.zantema@tue.nl

June 27, 2013

Abstract

How do you prove equality of infinite objects? We consider two basic types: streams and trees. Streams are infinite lists of data elements. We discuss two specific methods of comparing streams, bisimulation and circular coinduction. Next we describe infinite binary trees and make new definitions for bisimulation and circular coinduction in order to compare binary trees. We conclude with a transformation function between streams and trees.

Contents

Chapter 1

Introduction

Infinite objects have interesting properties compared to their finite counterparts. Firstly, how do you describe infinite objects? Secondly, how do you compare infinite objects? To describe an infinite object we use equations that the object should satisfy. We try to find equations that fully specify that one object. To be able to talk about these notions we define models which contain our infinite objects and in which the equations should hold. The simplest example of infinite objects are streams: infinite sequences of data elements. These can be described using input and output at any position in the stream. But comparing streams is a little harder. We generally can compare two lists by comparing each element in those lists. If we extend this to streams then for all streams p, q we have

$$p = q \leftrightarrow \forall n \in \mathbb{N}\ (p_n = q_n) \tag{1}$$

where the lowercase n denotes the position in the stream. However, a naive algorithm that would compare two streams this way would never terminate. A better way is to prove equality of all elements recursively. A well-known method is induction:

$$\begin{cases} p_0 = q_0 \\ p_n = q_n \rightarrow p_{n+1} = q_{n+1} \end{cases}$$

in which $p_n = q_n$ is the induction hypothesis. The implication should hold for all n. However, this method fails to prove the equality of infinite objects in which there is no direct relation between each subsequent pair of elements. What we need is a stronger induction hypothesis. If for example two streams repeat after two positions ($s_n = s_{n+2}$) we only need to check the equality of those two positions and add those to the induction hypothesis. From there we can derive the equality of the rest of the streams. In non-repeating streams there are other ways, for example making a recursive specification for which we only need to check the equality of all cases in the recursion.

In this paper we start with a specification of streams. We discuss three ways of comparing streams: induction, bisimulation and circular coinduction. We prove their correctness by showing that induction, bisimulation and circular coinduction are equivalent proving methods. Next we describe infinite binary trees and extend the equality rules of the one-dimensional bisimulation and circular coinduction to be able to compare two-dimensional trees. We will again prove that the same relations between the proving methods hold and we define a function that first transforms the trees into streams before comparing them.

Chapter 2

Streams and stream equality

2.1 A specification of streams

This section is largely based on section 2 of [13]. Streams are infinite lists of elements. We consider two basic types: streams s and data elements d. The set of all data elements is D. Next we want to have functions to operate on streams or data elements. We consider three basic functions:

- : of type $d \times s \to s$, concatenating a data element to a stream.
- `hd` of type $s \to d$, giving the first element ('head') of a stream.
- `tl` of type $s \to s$, giving the stream minus the first element ('tail') of a stream.

Other specified functions have types $d^m \times s^n \to d \mid d^m \times s^n \to s$ for $m, n \geq 0$, we call this set of functions F. Finally we have two sets of variables: $x_i \in X_d$ of type d and $\sigma_i \in X_s$ of type s.

Definition 2.1. The set of all terms is $D \cup \{:, \texttt{hd}, \texttt{tl}\} \cup F \cup X_d \cup X_s$. If $u_1, \ldots, u_m$ are terms of type d and $t_1, \ldots, t_n$ are terms of type s and $f \in F \cup \{:, \texttt{hd}, \texttt{tl}\}$ with signature $f\colon d^m \times s^n \to d \mid f\colon d^m \times s^n \to s$ then $f(u_1, \ldots, u_m, t_1, \ldots, t_n)$ is a term.

Now that we know how to define terms we want to compare two terms.

Definition 2.2. For comparing two terms we define an stream equation as a pair of terms (l, r), both of type s. A stream specification E is a set of equations, each written as $l = r$.

To be able to prove an equation we need to know the semantics of the functions. Therefore we define a stream model.

Definition 2.3. A stream model M over $D \cup F$ consists of the set of all streams $S = D^\omega$ and the set of functions and constants $[F]$ of which every $[f]$ is an interpretation of function $f \in F$.

For the function interpretations $[f]$ it is required that $[f]\colon D^m \times S^n \to D$ if $f\colon d^m \times s^n \to d$ and $[f]\colon D^m \times S^n \to S$ if $f\colon d^m \times s^n \to s$. The interpretations of the function symbols in F are free to choose.

Definition 2.4. We define the fixed interpretations of the basic functions $:, \texttt{hd}, \texttt{tl}$ as follows:

- $\forall u \in D, \forall s \in S, \forall n \in \mathbb{N}.$ $\begin{cases} [:](u,s)(0) = u \\ [:](u,s)(n+1) = s_n \end{cases}$
- $\forall s \in S.\ [\texttt{hd}](s) = s_0$
- $\forall s \in S, \forall n \in \mathbb{N}.\ [\texttt{tl}](s)(n) = s_{n+1}$

With these fixed interpretations $[:], [\texttt{hd}], [\texttt{tl}]$ we can make the following stream specification E_b:

$$E_b = \begin{cases} \texttt{hd}(x : \sigma) = x \\ \texttt{tl}(x : \sigma) = \sigma \\ \texttt{hd}(\sigma) : \texttt{tl}(\sigma) = \sigma \end{cases}$$

Because the interpretations of $:, \texttt{hd}, \texttt{tl}$ are fixed, E_b holds in every model, therefore for all stream models $M \models E_b$.

We say that $E \vdash l = r$ if $l = r$ can be derived from the equations in E using the rules of equational logic.

Definition 2.5. The rules of equational logic are:

- $\vdash t = t$ (reflexivity)
- $t_1 = t_2 \vdash t_2 = t_1$ (symmetry)
- $t_1 = t_2, t_2 = t_3 \vdash t_1 = t_3$ (transitivity)
- $t_1 = t'_1, \ldots, t_n = t'_n \vdash f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n)$ (equality of arguments)
- $t_1 = t_2 \vdash t_1[x/u] = t_2[x/u]$ (substitution)

The rules of equational logic are sound for stream models: if $E \vdash l = r$, then this is true in all stream models. Therefore we can say $E \models l = r$.

Definition 2.6. $E \models l = r \ :=\ M \models E \Rightarrow M \models l = r$

Equational logic is also complete for stream models: if $E \models l = r$, then $E \vdash l = r$. For a new set of equations E' we can prove $E \models E'$ if $E \models l = r$ for every $l = r \in E'$. This concludes the definition of streams and stream equality.

2.2 Bisimulation

The idea of making a bisimulation to prove the equality of two streams is similar to induction as defined in Equation (1), but with a larger hypothesis space. This makes it a very good method to prove equality of infinite objects. The following definition originates from [10, 2].

Definition 2.7. For $\mathcal{R}$, a binary relation on streams, we say that it is a bisimulation for all streams s, t if we have:

$$\mathcal{R}(s,t) \Rightarrow \mathtt{hd}(s) = \mathtt{hd}(t) \qquad \text{(bisim-head)}$$
$$\mathcal{R}(s,t) \Rightarrow \mathcal{R}(\mathtt{tl}(s), \mathtt{tl}(t)) \qquad \text{(bisim-tail)}$$

We define equality of streams s, t by bisimulation as:

$$\exists \mathcal{R}(\mathcal{R} \text{ is a bisimulation} \wedge \mathcal{R}(s,t)) \qquad (2)$$

In the following theorem we use a subscript 1 to describe the induction equality of Equation (1).

Theorem 2.8.

$$s =_1 t \Leftrightarrow s =_2 t$$

Proof $\Leftarrow$ We assume $s =_2 t$, so we have a bisimulation $\mathcal{R}$ with $\mathcal{R}(s,t)$. We need to prove that for all streams s, t we have $\forall n \in \mathbb{N}\ (s_n = t_n)$, for which we use induction on the position in the stream.
For $n = 0$: $s_0 = \mathtt{hd}(s) =_2 \mathtt{hd}(t) = t_0$, where $=_2$ follows from Equation (bisim-head) in Definition 2.7.
For $n > 0$: we have the induction hypothesis $s_n = t_n$. Left to prove: $s_{n+1} = t_{n+1}$.
We have $\mathcal{R}(s,t)$, so from Equation (bisim-tail) we also have $\mathcal{R}(\mathtt{tl}(s), \mathtt{tl}(t))$ and therefore $(\mathtt{tl}(s))_n =_2 (\mathtt{tl}(t))_n$ from Equation (2). Now we can prove $s_{n+1} = (\mathtt{tl}(s))_n =_2 (\mathtt{tl}(t))_n = t_{n+1}$.

Proof $\Rightarrow$ We assume $s =_1 t$, so according to Equation (1) we have $\forall n \in \mathbb{N}\ (s_n = t_n)$. We need to prove that there exists a bisimulation $\mathcal{R}$ with $\mathcal{R}(s,t)$. For this we can choose equality (1) from our assumption, so $\mathcal{R}(s,t) := (s =_1 t)$. Now we need to prove that $=_1$ is a bisimulation for all streams s, t with $\mathcal{R}(s,t)$.
According to Equation (bisim-head) the heads must be equal. $\mathtt{hd}(s) = s_0 =_1 t_0 = \mathtt{hd}(t)$.
According to Equation (bisim-tail) the tails must be in $\mathcal{R}$. For all streams s, t we have $\forall n \in \mathbb{N}\ (s_n = t_n)$. This means we also have $\forall n \in \mathbb{N}\ ((\mathtt{tl}(s))_n = (\mathtt{tl}(t))_n)$, so $\mathtt{tl}(s) = \mathtt{tl}(t)$. From Equation (bisim-tail) we conclude $\mathcal{R}(\mathtt{tl}(s), \mathtt{tl}(t))$ and thereby we have proven $\mathcal{R}$ to be a bisimulation.

Example 2.9. In this example we will prove equality of two binary streams by defining a bisimulation. For this we will extend $\mathcal{R}$ until we can prove both requirements as defined in Definition 2.7. Our stream specification is:

$$\begin{aligned} \texttt{zeros} &= 0 : \texttt{zeros} \\ \texttt{ones} &= 1 : \texttt{ones} \\ \texttt{alt} &= 0 : 1 : \texttt{alt} \\ \texttt{zip}(x : s, t) &= x : \texttt{zip}(t, s) \end{aligned}$$

To prove: $\texttt{alt} = \texttt{zip}(\texttt{zeros}, \texttt{ones})$. For this we make a bisimulation $\mathcal{R}$ with the set $\{(\texttt{alt}, \texttt{zip}(\texttt{zeros}, \texttt{ones}))\} \subseteq \mathcal{R}$.

According to (bisim-head) we must have $\texttt{hd}(\texttt{alt}) = \texttt{hd}(\texttt{zip}(\texttt{zeros}, \texttt{ones}))$. This follows from:

$$\begin{aligned} \texttt{hd}(\texttt{alt}) &= \texttt{hd}(0 : 1 : \texttt{alt}) \\ &= 0 \\ &= \texttt{hd}(0 : \texttt{zip}(\texttt{ones}, \texttt{zeros})) \\ &= \texttt{hd}(\texttt{zip}(0 : \texttt{zeros}, \texttt{ones})) \\ &= \texttt{hd}(\texttt{zip}(\texttt{zeros}, \texttt{ones})) \end{aligned}$$

The second requirement is $\mathcal{R}(\texttt{tl}(\texttt{alt}), \texttt{tl}(\texttt{zip}(\texttt{zeros}, \texttt{ones})))$ according to (bisim-tail). We can rewrite these two terms:

$$\begin{aligned} \texttt{tl}(\texttt{alt}) &= \texttt{tl}(0 : 1 : \texttt{alt}) \\ &= 1 : \texttt{alt} \end{aligned}$$

and

$$\begin{aligned} \texttt{tl}(\texttt{zip}(\texttt{zeros}, \texttt{ones})) &= \texttt{tl}(0 : \texttt{zip}(\texttt{ones}, \texttt{zeros})) \\ &= \texttt{zip}(\texttt{ones}, \texttt{zeros}) \end{aligned}$$

This pair is not defined in $\mathcal{R}$ yet, so we extend $\mathcal{R}$ to $\{(\texttt{alt}, \texttt{zip}(\texttt{zeros}, \texttt{ones})), (1 : \texttt{alt}, \texttt{zip}(\texttt{ones}, \texttt{zeros}))\}$.

For this new pair (bisim-head) follows from:

$$\begin{aligned} \texttt{hd}(1 : \texttt{alt}) &= 1 \\ &= \texttt{hd}(1 : \texttt{zip}(\texttt{zeros}, \texttt{ones})) \\ &= \texttt{hd}(\texttt{zip}(\texttt{ones}, \texttt{zeros})) \end{aligned}$$

Now we can prove (bisim-tail) as well, by using a pair from $\mathcal{R}$:

$$\begin{aligned} \texttt{tl}(1 : \texttt{alt}) &= \texttt{alt} \\ &=_{\mathcal{R}} \texttt{zip}(\texttt{zeros}, \texttt{ones}) \\ &= \texttt{tl}(1 : \texttt{zip}(\texttt{zeros}, \texttt{ones})) \\ &= \texttt{tl}(\texttt{zip}(\texttt{ones}, \texttt{zeros})) \end{aligned}$$

This proves $\mathcal{R}$ is een bisimulation, so we have $\texttt{alt} = \texttt{zip}(\texttt{zeros}, \texttt{ones})$.

Example 2.10. In this example we will prove equality of two natural streams by using a successor function s. Our stream specification is:

$$\begin{aligned} \texttt{from}(x) &= x : \texttt{from}(s(x)) \\ \texttt{from2}(x) &= x : \texttt{from2}(s(s(x))) \\ \texttt{zip}(x : s, t) &= x : \texttt{zip}(t, s) \end{aligned}$$

To prove: $\texttt{from}(x) = \texttt{zip}(\texttt{from2}(x), \texttt{from2}(s(x)))$. That is, if we make an interpretation $[s](n) = n+1$ for $n \in \mathbb{N}$ we have to prove that the `zip` of even and odd numbers is equal to $\mathbb{N}$. For this we make a bisimulation $\mathcal{R}$ with the set $\{(\texttt{from}(x), \texttt{zip}(\texttt{from2}(x), \texttt{from2}(s(x)))) \mid x \in \{s^n 0 \mid n \in \mathbb{N}\}\} \subseteq \mathcal{R}$. For $\mathcal{R}$ to be a bisimulation we need to prove (bisim-head) and (bisim-tail) from Definition 2.7.

Definition (bisim-head) states:
$\texttt{hd}(\texttt{from}(x)) = \texttt{hd}(\texttt{zip}(\texttt{from2}(x), \texttt{from2}(s(x))))$. This follows from:

$$\begin{aligned} \texttt{hd}(\texttt{from}(x)) &= \texttt{hd}(x : \texttt{from}(s(x))) \\ &= x \\ &= \texttt{hd}(x : \texttt{zip}(\texttt{from2}(s(x)), \texttt{from2}(s(s(x))))) \\ &= \texttt{hd}(\texttt{zip}(x : \texttt{from2}(s(s(x))), \texttt{from2}(s(x)))) \\ &= \texttt{hd}(\texttt{zip}(\texttt{from2}(x), \texttt{from2}(s(x)))) \end{aligned}$$

Definition (bisim-tail) states:
$\{(\texttt{hd}(\texttt{from}(x)), \texttt{hd}(\texttt{zip}(\texttt{from2}(x), \texttt{from2}(s(x))))))\} \subseteq \mathcal{R}$. This follows from:

$$\begin{aligned} \texttt{tl}(\texttt{from}(x)) &= \texttt{tl}(x : \texttt{from}(s(x))) \\ &= \texttt{from}(s(x)) \\ &=_{\mathcal{R}} \texttt{zip}(\texttt{from2}(s(x)), \texttt{from2}(s(s(x)))) \\ &= \texttt{tl}(x : \texttt{zip}(\texttt{from2}(s(x)), \texttt{from2}(s(s(x))))) \\ &= \texttt{tl}(\texttt{zip}(x : \texttt{from2}(s(s(x))), \texttt{from2}(s(x)))) \\ &= \texttt{tl}(\texttt{zip}(\texttt{from2}(x), \texttt{from2}(s(x)))) \end{aligned}$$

This proves $\mathcal{R}$ is een bisimulation, so $\texttt{from}(x) = \texttt{zip}(\texttt{from2}(x), \texttt{from2}(s(x)))$.

2.3 Circular coinduction

Circular coinduction is another method to compare two streams and is quite similar to bisimulation. Let E be a stream specification. We want to prove $E \models l = r$, for which by Definition 2.6 it suffices to prove that for every stream model M, if $M \models E$ then $M \models l = r$.

Theorem 2.11. *Let* $\mathtt{fr}$ *be a function of type* $s \to d$ *that is 'fresh', i.e.* $\mathtt{fr}$ *does not occur in* E, l, r*. For every stream model* M*, if* $M \models E$ *and the following two properties hold, we may conclude* $M \models l = r$*.*

$$M \models \mathtt{hd}(l) = \mathtt{hd}(r)$$
$$M \models \mathtt{fr}(l) = \mathtt{fr}(r) \Rightarrow M \models \mathtt{fr}(\mathtt{tl}(l)) = \mathtt{fr}(\mathtt{tl}(r))$$

Corollary 2.12. *If the following two rules hold, then* $E \vdash l = r$*.*

$$E \vdash \mathtt{hd}(l) = \mathtt{hd}(r) \qquad \text{(coind-head)}$$
$$E \cup (\mathtt{fr}(l) = \mathtt{fr}(r)) \vdash \mathtt{fr}(\mathtt{tl}(l)) = \mathtt{fr}(\mathtt{tl}(r)) \qquad \text{(coind-tail)}$$

These rules can be extended to prove a set of equations E' as is done in Theorem 3.1 from [13]. This will give more possibilities in proving equality, as the assumption in rule (coind-tail) can be used to prove the right side $\mathtt{fr}(\mathtt{tl}(l)) = \mathtt{fr}(\mathtt{tl}(r))$ for another equation in E'.

In the following theorem we use $=_1$ to describe the induction equality of Equation (1) and $=_3$ to describe the coinduction equality of Corollary 2.12.

Theorem 2.13.

$$s =_1 t \Leftrightarrow s =_3 t$$

Proof We can prove $E \vdash l = r$ by induction on the position n. Our assumptions are (coind-head) and (coind-tail). Let M be an arbitrary stream model for which we have $M \models E$.
For $n = 0$ we have $l_0 = \mathtt{hd}(l) = \mathtt{hd}(r) = r_0$, by (coind-head).
For $n \geq 0$ we have the induction hypothesis $l_n = r_n$. Now we're left to prove $l_{n+1} = r_{n+1}$. We can make an interpretation for $\mathtt{fr}$. Let us define $\mathtt{fr}(s) = s_n$, for which the assumption $\mathtt{fr}(l) = \mathtt{fr}(r)$ in (coind-tail) becomes the induction hypothesis. Then we obtain: $l_{n+1} = (\mathtt{tl}(l))_n = \mathtt{fr}(\mathtt{tl}(l)) = \mathtt{fr}(\mathtt{tl}(r)) = (\mathtt{tl}(r))_n = r_{n+1}$, by (coind-tail).

Because the coinduction rules translate to induction, a proof of coinduction given the induction rules is trivial, as it will use the same equalities in the proof above.

Comparison with bisimulation Corollary 2.12 is in fact a less general case of Definition 2.7, because the $\mathtt{fr}$ function outputs a single data element. Therefore Equation (coind-tail) only compares elements, not streams. It is more like the inductive step. However, the extension to prove a set E' instead of a single equation makes it stronger than normal induction.
If we combine Theorem 2.13 with Theorem 2.8 we should be able to prove $s =_2 t \Leftrightarrow s =_3 t$. We leave this proof open, as we were unable to complete it in time.

Example 2.14. In this example we will prove equality of two streams by circular coinduction and bisimulation to show the similarities with term rewriting. Let E be the following stream specification:

$$\begin{aligned}\mathtt{odd}(x : s) &= \mathtt{even}(s)\\ \mathtt{even}(x : s) &= x : \mathtt{odd}(s)\\ \mathtt{zip}(x : s, t) &= x : \mathtt{zip}(t, s)\end{aligned}$$

To prove: $E \models \mathtt{zip}(\mathtt{even}(s), \mathtt{odd}(s)) = s$.

Circular coinduction From Corollary 2.12 it follows that we need to prove both rules. Equation (coind-head) states $\mathtt{hd}(\mathtt{zip}(\mathtt{even}(s), \mathtt{odd}(s))) = \mathtt{hd}(s)$ and follows from:

$$\begin{aligned}\mathtt{hd}(\mathtt{zip}(\mathtt{even}(s), \mathtt{odd}(s))) &= \mathtt{hd}(\mathtt{zip}(\mathtt{hd}(s) : \mathtt{odd}(\mathtt{tl}(s)), \mathtt{odd}(s)))\\ &= \mathtt{hd}(\mathtt{hd}(s) : \mathtt{zip}(\mathtt{odd}(s), \mathtt{odd}(\mathtt{tl}(s)))\\ &= \mathtt{hd}(s)\end{aligned}$$

Equation (coind-head) states $\mathtt{fr}(\mathtt{tl}(\mathtt{zip}(\mathtt{even}(s), \mathtt{odd}(s)))) = \mathtt{fr}(\mathtt{tl}(s))$ using induction hypothesis $\mathtt{fr}(\mathtt{zip}(\mathtt{even}(s), \mathtt{odd}(s))) = \mathtt{fr}(s)$. This follows from:

$$\begin{aligned}\mathtt{fr}(\mathtt{tl}(\mathtt{zip}(\mathtt{even}(s), \mathtt{odd}(s)))) &= \mathtt{fr}(\mathtt{tl}(\mathtt{zip}(\mathtt{hd}(s) : \mathtt{odd}(\mathtt{tl}(s)), \mathtt{odd}(s))))\\ &= \mathtt{fr}(\mathtt{tl}(\mathtt{hd}(s) : \mathtt{zip}(\mathtt{odd}(s), \mathtt{odd}(\mathtt{tl}(s)))))\\ &= \mathtt{fr}(\mathtt{zip}(\mathtt{odd}(s), \mathtt{odd}(\mathtt{tl}(s))))\\ &= \mathtt{fr}(\mathtt{zip}(\mathtt{even}(\mathtt{tl}(s)), \mathtt{odd}(\mathtt{tl}(s))))\\ &=_{\mathrm{CH}} \mathtt{fr}(\mathtt{tl}(s))\end{aligned}$$

In the last step we use the coinduction hypothesis which holds for all streams s. Because both circular coinduction requirements hold we prove that $E \models \mathtt{zip}(\mathtt{even}(s), \mathtt{odd}(s)) = s$.

Bisimulation Choose $\mathcal{R} = \{(\mathtt{zip}(\mathtt{even}(s), \mathtt{odd}(s)), s)\}$ for all streams s. We need to prove both requirements from Definition 2.7.
Equation (bisim-head) follows from:

$$\begin{aligned}\mathtt{hd}(\mathtt{zip}(\mathtt{even}(s), \mathtt{odd}(s))) &= \mathtt{hd}(\mathtt{zip}(\mathtt{hd}(s) : \mathtt{odd}(\mathtt{tl}(s)), \mathtt{odd}(s)))\\ &= \mathtt{hd}(\mathtt{hd}(s) : \mathtt{zip}(\mathtt{odd}(s), \mathtt{odd}(\mathtt{tl}(s)))\\ &= \mathtt{hd}(s)\end{aligned}$$

Equation (bisim-tail) follows from:

$$\begin{aligned}\mathtt{tl}(\mathtt{zip}(\mathtt{even}(s),\mathtt{odd}(s))) &= \mathtt{tl}(\mathtt{zip}(\mathtt{hd}(s):\mathtt{odd}(\mathtt{tl}(s)),\mathtt{odd}(s)))\\ &= \mathtt{tl}(\mathtt{hd}(s):\mathtt{zip}(\mathtt{odd}(s),\mathtt{odd}(\mathtt{tl}(s))))\\ &= \mathtt{zip}(\mathtt{odd}(s),\mathtt{odd}(\mathtt{tl}(s)))\\ &= \mathtt{zip}(\mathtt{even}(\mathtt{tl}(s)),\mathtt{odd}(\mathtt{tl}(s)))\\ &=_{\mathcal{R}} \mathtt{tl}(s)\end{aligned}$$

By proving both requirements we conclude that $\mathcal{R}$ is a bisimulation and therefore we know that $\mathtt{zip}(\mathtt{even}(s),\mathtt{odd}(s)) = s$.

Example 2.15. In this example we prove a more advanced equality by bisimulation, which can't be solved with normal circular coinduction. Let E be the following stream specification:

$$\begin{aligned}\mathtt{zip}(x:s,t) &= x:\mathtt{zip}(t,s)\\ \mathtt{ones} &= 1:\mathtt{zip}(\mathtt{ones},\mathtt{ones})\end{aligned}$$

To prove: $E \models \mathtt{ones} = 1:\mathtt{ones}$ with a bisimulation. Again, we start with the equation we need to prove: $\mathcal{R} = \{(\mathtt{ones}, 1:\mathtt{ones})\}$.

Then the first requirement (bisim-head) in Definition 2.7 follows from:
$\mathtt{hd}(\mathtt{ones}) = \mathtt{hd}(1:\mathtt{zip}(\mathtt{ones},\mathtt{ones})) = 1 = \mathtt{hd}(1:\mathtt{ones})$.

The second requirement (bisim-tail) isn't fulfilled yet, so we rewrite both terms first:
$\mathtt{tl}(\mathtt{ones}) = \mathtt{tl}(1:\mathtt{zip}(\mathtt{ones},\mathtt{ones})) = \mathtt{zip}(\mathtt{ones},\mathtt{ones})$ and
$\mathtt{tl}(1:\mathtt{ones}) = \mathtt{ones}$.
Now we add $(\mathtt{zip}(\mathtt{ones},\mathtt{ones}),\mathtt{ones})$ to $\mathcal{R}$ and check both requirements again.

(bisim-head) for the new pair follows from:
$\mathtt{hd}(\mathtt{zip}(\mathtt{ones},\mathtt{ones})) = \mathtt{hd}(\mathtt{zip}(1:\mathtt{zip}(\mathtt{ones},\mathtt{ones}),\mathtt{ones})) =$
$\mathtt{hd}(1:\mathtt{zip}(\mathtt{ones},\mathtt{zip}(\mathtt{ones},\mathtt{ones}))) = 1 = \mathtt{hd}(1:\mathtt{zip}(\mathtt{ones},\mathtt{ones})) = \mathtt{hd}(\mathtt{ones})$.

Again, (bisim-tail) for the new pair isn't fulfilled yet, so we rewrite both terms first:
$\mathtt{tl}(\mathtt{zip}(\mathtt{ones},\mathtt{ones})) = \mathtt{tl}(\mathtt{zip}(1:\mathtt{zip}(\mathtt{ones},\mathtt{ones}),\mathtt{ones})) =$
$\mathtt{tl}(1:\mathtt{zip}(\mathtt{ones},\mathtt{zip}(\mathtt{ones},\mathtt{ones}))) = \mathtt{zip}(\mathtt{ones},\mathtt{zip}(\mathtt{ones},\mathtt{ones}))$ and
$\mathtt{tl}(\mathtt{ones}) = \mathtt{tl}(1:\mathtt{zip}(\mathtt{ones},\mathtt{ones})) = \mathtt{zip}(\mathtt{ones},\mathtt{ones})$.

Next we would need to add $(\mathtt{zip}(\mathtt{ones},\mathtt{zip}(\mathtt{ones},\mathtt{ones})),\mathtt{zip}(\mathtt{ones},\mathtt{ones}))$ to $\mathcal{R}$ and check both requirements again, but from the previous two steps it's clear that we would need to keep adding terms with an increasing number of $\mathtt{zip}$ terms in them. So we need to make a recursive specification T

to define all pairs in $\mathcal{R}$:

$$T := \texttt{ones} \mid 1 : T \mid \texttt{zip}(T, T)$$
$$\mathcal{R} = \{(s, t) \mid s, t \in T\}$$

We can now prove (bisim-head) and (bisim-tail) for a bisimulation by induction on T. Because $\mathcal{R}$ relates all terms in T we prove $\mathcal{R}$ is a bisimulation by proxy: $\forall s, t \in T$ ($\texttt{hd}(s) = \texttt{hd}(t)$) and $\forall s \in T$ ($\texttt{tl}(s) \in T$).

Base cases: $s, t \in \{\texttt{ones}, 1 : \sigma\}$ with $\sigma \in T$
$\texttt{hd}(\texttt{ones}) = \texttt{hd}(1 : \texttt{zip}(\texttt{ones}, \texttt{ones})) = 1 = \texttt{hd}(1 : \sigma)$
$\texttt{tl}(\texttt{ones}) = \texttt{tl}(1 : \texttt{zip}(\texttt{ones}, \texttt{ones})) = \texttt{zip}(\texttt{ones}, \texttt{ones}) \in T$
$\texttt{tl}(1 : \sigma) = \sigma \in T$

Now we prove the third case in which s or t is $\texttt{zip}(\sigma, \tau)$ with $\sigma, \tau \in T$, for which we need the induction hypothesis $\texttt{hd}(\sigma) = \texttt{hd}(\tau)$, $\texttt{tl}(\sigma) \in T$, $\texttt{tl}(\tau) \in T$.
$\texttt{hd}(\texttt{zip}(\sigma, \tau)) = \texttt{hd}(\texttt{hd}(\sigma) : \texttt{zip}(\tau, \texttt{tl}(\sigma))) = \texttt{hd}(\sigma) = \texttt{hd}(\tau)$
$\texttt{tl}(\texttt{zip}(\sigma, \tau)) = \texttt{tl}(\texttt{hd}(\sigma) : \texttt{zip}(\tau, \texttt{tl}(\sigma))) = \texttt{zip}(\tau, \texttt{tl}(\sigma)) \in T$

2.4 Similarities and differences

Previous examples have shown that there are many similarities between proving equality with bisimulation and circular coinduction, but also a few differences. We have shown that the circular coinduction rules are more specific compared to the bisimulation rules. This means that if there exists a proof by circular coinduction a similar proof by bisimulation is straightforward. One could take all coinduction hypotheses in a proof by coinduction, remove the freeze functions and add the equations as pairs to $\mathcal{R}$, which would make $\mathcal{R}$ a correct bisimulation. But how would bisimulations with a recursive definition like in Example 2.15 translate back to coinduction?

For this we need an extension to circular coinduction called "special contexts", as described in [8, 13]. Indeed we can prove $\texttt{zip}(\Box, \tau)$ to be a special context, for which one coinduction hypothesis is sufficient to prove the equation.

According to Theorem 4.5 in [13] first we need to specify a stream specification $E' \subseteq E$ of which we want to prove special contexts. Let us choose $E' = \{\texttt{zip}(x : \sigma, \tau) = x : \texttt{zip}(\tau, \sigma)\}$. For all contexts in E' to be special we must prove E' is both guarded and exhaustive. E' is guarded, because we have for all equations that the basic functions `hd` and `tl` do not occur in both terms, the right side of the equations has : as the outer function and the left side of the equation only has : and variables in the arguments. E' is also exhaustive, because all subterms consisting of precisely one non-basic function are defined on the left side of an equation. In this case we only have subterm $\texttt{zip}(\tau, \sigma)$ which is defined on the left side of the same

equation. These two requirements prove that all contexts in E' are special, which are $\texttt{zip}(\square, \tau)$, $\texttt{zip}(\sigma, \square)$ and $\texttt{zip}(\square, \square)$.

If we look at Example 2.14 again we can prove that $\texttt{even}(\square)$ and $\texttt{odd}(\square)$ are not special contexts. If $E' = \{\texttt{even}(x, \sigma) = x : \texttt{odd}(\sigma)\}$ it is not exhaustive, because $\texttt{odd}(\sigma)$ is left undefined. However, if we extend E' to $\{\texttt{even}(x, \sigma) = x : \texttt{odd}(\sigma), \texttt{odd}(x : \sigma) = \texttt{even}(\sigma)\}$ it is not guarded anymore, because the second equation does not have : as the outer function on the right side.

A more intuitive way is by looking at the input and output size for a limited input. The output size should be at least as much as the input size. In the case of `even` and `odd` only half of the input is outputted.

Chapter 3

Trees and tree equality

3.1 A specification of infinite binary trees

Infinite binary trees are trees with two subtrees per node and no leaf nodes. We will not discuss trees with more than two subtrees per node or trees with finite paths. We consider two basic types: trees t and data elements d. The set of all data elements is D. Next we want to have functions to specify a tree or an element. We consider four basic functions:

- `n` of type $d \times t \times t \to t$, joining two trees with a new root element.
- `r` of type $t \to d$, giving the data element of the root node.
- `lt` of type $t \to t$, giving the left subtree of the root node.
- `rt` of type $t \to t$, giving the right subtree of the root node.

Other specified functions have types $d^m \times t^n \to d \mid d^m \times t^n \to t$ for $m, n \geq 0$, we call this set of functions F. Finally we have two sets of variables: $x_i \in X_d$ of type d and $\tau_i \in X_t$ of type t.

Definition 3.1. The set of all terms is $D \cup \{\texttt{n}, \texttt{r}, \texttt{lt}, \texttt{rt}\} \cup F \cup X_d \cup X_t$. If $u_1, \ldots, u_m$ are terms of type d and $t_1, \ldots, t_n$ are terms of type t and $f \in F \cup \{\texttt{n}, \texttt{r}, \texttt{lt}, \texttt{rt}\}$ with signature $f\colon d^m \times t^n \to d \mid f\colon d^m \times t^n \to t$, then $f(u_1, \ldots, u_m, t_1, \ldots, t_n)$ is a term.

Now that we know how to define terms we want to compare two terms, which goes the same as with streams.

Definition 3.2. For comparing two terms we define an equation as a pair of terms (l, r) of type t. A tree specification is a set of equations E, written as $l = r$.

To be able to prove an equation we need to know the semantics of the functions. Therefore we define a tree model.

Definition 3.3. A tree model M over $D \cup F$ consists of the set of all trees $T = D^{\{0,1\}^*}$ and the set of functions and constants $[F]$ of which every $[f]$ is an interpretation of function $f \in F$.

In this definition $\{0,1\}^*$ is the set of all finite paths in the infinite binary tree. A path is a sequence of taking the left or right subtree, here coded with $\{0,1\}$. A $\tau \in T$ is a function with signature $\tau : \{0,1\}^* \to D$, giving the data element after a certain path.

A function interpretation $[f]$ in the tree model specifies what trees are defined by function f. It is required that $[f] \colon D^m \times T^n \to D$ if $f \colon d^m \times t^n \to d$ and $[f] \colon D^m \times T^n \to T$ if $f \colon d^m \times t^n \to t$.The interpretations of the function symbols in F are free to choose.

Definition 3.4. We define the fixed interpretations of the basic functions $\mathtt{n}, \mathtt{r}, \mathtt{lt}, \mathtt{rt}$ as follows:

- $\forall u \in D, \forall t_1, t_2 \in T, \forall p \in \{0,1\}^*$ $\begin{cases} [n](u, t_1, t_2)(\epsilon) = u \\ [n](u, t_1, t_2)(0p) = t_1(p) \\ [n](u, t_1, t_2)(1p) = t_2(p) \end{cases}$
- $\forall t \in T$ $([\mathtt{r}](t) = t(\epsilon))$
- $\forall t \in T, \forall p \in \{0,1\}^*$ $([\mathtt{lt}](t)(p) = t(0p))$
- $\forall t \in T, \forall p \in \{0,1\}^*$ $([\mathtt{rt}](t)(p) = t(1p))$

With these fixed interpretations $[\mathtt{n}], [\mathtt{r}], [\mathtt{lt}], [\mathtt{rt}]$ we can make the following tree specification E_b:

$$E_b = \begin{cases} \mathtt{r}(\mathtt{n}(x, \tau_1, \tau_2)) = x \\ \mathtt{lt}(\mathtt{n}(x, \tau_1, \tau_2)) = \tau_1 \\ \mathtt{rt}(\mathtt{n}(x, \tau_1, \tau_2)) = \tau_2 \\ \mathtt{n}(\mathtt{r}(\tau), \mathtt{lt}(\tau), \mathtt{rt}(\tau)) = \tau \end{cases}$$

Because the interpretations of $\mathtt{n}, \mathtt{r}, \mathtt{lt}, \mathtt{rt}$ are fixed, E_b holds in every model, therefore for all tree models $M \models E_b$.

We have the same rules of equational logic for trees as for streams (Definition 2.5), which are sound and complete for models. We can denote the same definition $E \models l = r$ for trees as in Definition 2.6. But how to define equality on trees? If we have for all paths that the data elements are equal, then the trees must be equal.

$$t_1 =_1 t_2 \leftrightarrow \forall s \in \{0,1\}^* \ (t_1(s) = t_2(s)) \qquad \text{(tree-1)}$$

3.2 Bisimulation

Bisimulation for infinite binary trees should work the same as for infinite streams, only each node has two pointers instead of one. Again, we define $\mathcal{R}$, now as a relation on trees.

Definition 3.5. For $\mathcal{R}$, a binary relation on trees, we say that it is a bisimulation for all trees t_1, t_2 if we have:

$$\begin{aligned} \mathcal{R}(t_1, t_2) &\Rightarrow \mathtt{r}(t_1) = \mathtt{r}(t_2) && \text{(bisim-root)} \\ \mathcal{R}(t_1, t_2) &\Rightarrow \mathcal{R}(\mathtt{lt}(t_1), \mathtt{lt}(t_2)) && \text{(bisim-left)} \\ \mathcal{R}(t_1, t_2) &\Rightarrow \mathcal{R}(\mathtt{rt}(t_1), \mathtt{rt}(t_2)) && \text{(bisim-right)} \end{aligned}$$

The notion of equality by bisimulation is the same as in Equation (2), so for all infinite binary trees t_1, t_2 we define:

$$t_1 =_2 t_2 \leftrightarrow \exists \mathcal{R}(\mathcal{R} \text{ is a bisimulation} \wedge \mathcal{R}(t_1, t_2)) \tag{2}$$

Example 3.6. In this example we prove equality of two trees by defining a bisimulation. Let E be the following tree specification:

$$\begin{aligned} \mathtt{evenl}(\mathtt{n}(x, \tau_1, \tau_2)) &= \mathtt{n}(x, \mathtt{oddl}(t_1), t_2) \\ \mathtt{oddl}(\mathtt{n}(x, \tau_1, \tau_2)) &= \mathtt{evenl}(\tau_1) \\ \mathtt{zipl}(\mathtt{n}(x, \tau_1, \tau_2), \upsilon) &= \mathtt{n}(x, \mathtt{zipl}(\upsilon, \tau_1), \tau_2) \end{aligned}$$

To prove: $E \models \mathtt{zipl}(\mathtt{evenl}(t), \mathtt{oddl}(t)) = t$. For this we make a bisimulation $\mathcal{R}$ with the set $\{(\mathtt{zipl}(\mathtt{evenl}(t), \mathtt{oddl}(t)), t)\} \in \mathcal{R}$.

Definition 3.5 states that we must prove three rules. Firstly, according to (bisim-root) we must have $\mathtt{r}(\mathtt{zipl}(\mathtt{evenl}(t), \mathtt{oddl}(t))) = \mathtt{r}(t)$. This follows from:

$$\begin{aligned} \mathtt{r}(\mathtt{zipl}(\mathtt{evenl}(t), \mathtt{oddl}(t))) &= \mathtt{r}(\mathtt{zipl}(\mathtt{n}(\mathtt{r}(t), \mathtt{oddl}(\mathtt{lt}(t)), \mathtt{rt}(t)), \mathtt{oddl}(t))) \\ &= \mathtt{r}(\mathtt{n}(\mathtt{r}(t), \mathtt{zipl}(\mathtt{oddl}(t), \mathtt{oddl}(\mathtt{lt}(t))), \mathtt{rt}(t))) \\ &= \mathtt{r}(t) \end{aligned}$$

Secondly, according to (bisim-left) we must have:
$\mathcal{R}(\mathtt{lt}(\mathtt{zipl}(\mathtt{evenl}(t), \mathtt{oddl}(t))), \mathtt{lt}(t))$. This follows from:

$$\begin{aligned} \mathtt{lt}(\mathtt{zipl}(\mathtt{evenl}(t), \mathtt{oddl}(t))) &= \mathtt{lt}(\mathtt{zipl}(\mathtt{n}(\mathtt{r}(t), \mathtt{oddl}(\mathtt{lt}(t)), \mathtt{rt}(t)), \mathtt{oddl}(t))) \\ &= \mathtt{lt}(\mathtt{n}(\mathtt{r}(t), \mathtt{zipl}(\mathtt{oddl}(t), \mathtt{oddl}(\mathtt{lt}(t))), \mathtt{rt}(t))) \\ &= \mathtt{zipl}(\mathtt{oddl}(t), \mathtt{oddl}(\mathtt{lt}(t))) \\ &= \mathtt{zipl}(\mathtt{evenl}(\mathtt{lt}(t)), \mathtt{oddl}(\mathtt{lt}(t))) \\ &=_{\mathcal{R}} \mathtt{lt}(t) \end{aligned}$$

And finally, according to (bisim-right) we must have:
$\mathcal{R}(\mathtt{rt}(\mathtt{zipl}(\mathtt{evenl}(t), \mathtt{oddl}(t))), \mathtt{rt}(t))$. This follows from:

$$\begin{aligned}\mathtt{rt}(\mathtt{zipl}(\mathtt{evenl}(t), \mathtt{oddl}(t))) &= \mathtt{rt}(\mathtt{zipl}(\mathtt{n}(\mathtt{r}(t), \mathtt{oddl}(\mathtt{lt}(t)), \mathtt{rt}(t)), \mathtt{oddl}(t))) \\ &= \mathtt{rt}(\mathtt{n}(\mathtt{r}(t), \mathtt{zipl}(\mathtt{oddl}(t), \mathtt{oddl}(\mathtt{lt}(t))), \mathtt{rt}(t))) \\ &= \mathtt{rt}(t)\end{aligned}$$

All requirements for $\mathcal{R}$ hold, so $\mathcal{R}$ is a bisimulation and therefore we have proven $\mathtt{zipl}(\mathtt{evenl}(t), \mathtt{oddl}(t)) = t$ for all $t \in T$.

3.3 Circular coinduction

Just like with bisimulation we make an extra rule to cover both the left and right subtree of a given node. Let E be a tree specification. This gives us the following rules to prove $E \models l = r$.

Theorem 3.7. *Let* $\mathtt{fr}$ *be a function of type* $t \to d$ *that is 'fresh', i.e.* $\mathtt{fr}$ *does not occur in* E, l, r*. For every tree model* M*, if* $M \models E$ *and the following three properties hold, we may conclude* $M \models l = r$*.*

$$\begin{aligned}&M \models \mathtt{r}(l) = \mathtt{r}(r) \\ &M \models \mathtt{fr}(l) = \mathtt{fr}(r) \Rightarrow M \models \mathtt{fr}(\mathtt{lt}(l)) = \mathtt{fr}(\mathtt{lt}(r)) \\ &M \models \mathtt{fr}(t_1) = \mathtt{fr}(r) \Rightarrow M \models \mathtt{fr}(\mathtt{rt}(l)) = \mathtt{fr}(\mathtt{rt}(r))\end{aligned}$$

Corollary 3.8. *If the following three rules hold, then* $E \vdash l = r$*.*

$$E \vdash \mathtt{r}(l) = \mathtt{r}(r) \qquad \text{(coind-root)}$$

$$E \cup (\mathtt{fr}(l) = \mathtt{fr}(r)) \vdash \mathtt{fr}(\mathtt{lt}(l)) = \mathtt{fr}(\mathtt{lt}(r)) \qquad \text{(coind-left)}$$

$$E \cup (\mathtt{fr}(l) = \mathtt{fr}(r)) \vdash \mathtt{fr}(\mathtt{rt}(l)) = \mathtt{fr}(\mathtt{rt}(r)) \qquad \text{(coind-right)}$$

Proof We can prove equality of trees t_1, t_2 as defined in Equation (tree-1) by using Corollary 3.8. Let M be an arbitrary tree model for which we have $M \models E$. We can prove this by induction on the path p in the tree.
For $p = \epsilon$ we have $t_1(\epsilon) = \mathtt{r}(t_1) = \mathtt{r}(t_2) = t_2(\epsilon)$ by (coind-root).
For $p \in \{0,1\}^*$ we have the induction hypothesis $t_1(p) = t_2(p)$. Now we're left to prove $t_1(0:p) = t_2(0:p)$ and $t_1(1:p) = t_2(1:p)$. Let us define $\mathtt{fr}(t) = t(p)$ for which the assumption $\mathtt{fr}(t_1) = \mathtt{fr}(t_2)$ in (coind-left) and (coind-right) becomes the induction hypothesis. Then we obtain:
$t_1(0:p) = \mathtt{lt}(t_1)(p) = \mathtt{fr}(\mathtt{lt}(t_1)) = \mathtt{fr}(\mathtt{lt}(t_2)) = \mathtt{lt}(t_2)(p) = t_2(0:s)$ and
$t_1(1:p) = \mathtt{rt}(t_1)(p) = \mathtt{fr}(\mathtt{rt}(t_1)) = \mathtt{fr}(\mathtt{rt}(t_2)) = \mathtt{rt}(t_2)(p) = t_2(1:p)$ by using (coind-left) and (coind-right) respectively.

Example 3.9. In this example we prove equality of two trees by circular coinduction and bisimulation to show the similarities with term rewriting. Let E be the following tree specification:

$$\begin{aligned}
\mathtt{alt} &= \mathtt{n}(0, \mathtt{n}(1, \mathtt{alt}, \mathtt{alt}), \mathtt{n}(1, \mathtt{alt}, \mathtt{alt})) \\
\mathtt{inv}(n(0, \tau_1, \tau_2)) &= \mathtt{n}(1, \mathtt{inv}(\tau_1), \mathtt{inv}(\tau_2)) \\
\mathtt{inv}(n(1, \tau_1, \tau_2)) &= \mathtt{n}(0, \mathtt{inv}(\tau_1), \mathtt{inv}(\tau_2))
\end{aligned}$$

To prove: $E \models \mathtt{lt}(\mathtt{alt}) = \mathtt{inv}(\mathtt{alt})$.

Circular coinduction We need to prove all rules in Corollary 3.8. By Equation (coind-root) we must prove $\mathtt{r}(\mathtt{lt}(\mathtt{alt})) = \mathtt{r}(\mathtt{inv}(\mathtt{alt}))$. This follows from :

$$\begin{aligned}
\mathtt{r}(\mathtt{lt}(\mathtt{alt})) &= \mathtt{r}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})) \\
&= 1 \\
&= \mathtt{r}(\mathtt{n}(1, \mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})), \mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})))) \\
&= \mathtt{r}(\mathtt{inv}(\mathtt{alt}))
\end{aligned}$$

By Equation (coind-left) we have the coinduction hypothesis
CH_1 : $\mathtt{fr}(\mathtt{lt}(\mathtt{alt})) = \mathtt{fr}(\mathtt{inv}(\mathtt{alt}))$. Now we must prove $\mathtt{fr}(\mathtt{lt}(\mathtt{lt}(\mathtt{alt}))) = \mathtt{fr}(\mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})))$. We derive:

$$\begin{aligned}
\mathtt{fr}(\mathtt{lt}(\mathtt{lt}(\mathtt{alt}))) &= \mathtt{fr}(\mathtt{lt}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt}))) \\
&= \mathtt{fr}(\mathtt{alt})
\end{aligned}$$

and

$$\begin{aligned}
\mathtt{fr}(\mathtt{lt}(\mathtt{inv}(\mathtt{alt}))) &= \mathtt{fr}(\mathtt{lt}(\mathtt{n}(1, \mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})), \mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt}))))) \\
&= \mathtt{fr}(\mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})))
\end{aligned}$$

Unfortunately we can't use CH_1 to prove equality here. From Equation (coind-right) we will get the same terms, because the `alt` tree is symmetrical. This means that now we need to prove:
$E \cup \mathrm{CH}_1 \models \mathtt{alt} = \mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt}))$.

Again, we start with (coind-root) by which we must prove $\mathtt{r}(\mathtt{alt}) = \mathtt{r}(\mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})))$. This follows from:

$$\begin{aligned}
\mathtt{r}(\mathtt{alt}) &= 0 \\
&= \mathtt{r}(\mathtt{n}(0, \mathtt{inv}(\mathtt{alt}), \mathtt{inv}(\mathtt{alt}))) \\
&= \mathtt{r}(\mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})))
\end{aligned}$$

Next, by (coind-left) we have the coinduction hypothesis
CH_2 : $\mathtt{fr}(\mathtt{alt}) = \mathtt{fr}(\mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})))$. Now we must prove $\mathtt{fr}(\mathtt{lt}(\mathtt{alt})) = \mathtt{fr}(\mathtt{lt}(\mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt}))))$. We derive:

$$\begin{aligned}\mathtt{fr}(\mathtt{lt}(\mathtt{alt})) &=_{\mathrm{CH1}} \mathtt{fr}(\mathtt{inv}(\mathtt{alt})) \\ &= \mathtt{fr}(\mathtt{lt}(\mathtt{n}(0, \mathtt{inv}(\mathtt{alt}), \mathtt{inv}(\mathtt{alt})))) \\ &= \mathtt{fr}(\mathtt{lt}(\mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt}))))\end{aligned}$$

This proves the second requirement. The third requirement (coind-right) will give a similar proof, because the `alt` tree is symmetrical.

Bisimulation Now we make a proof by defining a bisimulation $\mathcal{R}$ with $\{(\mathtt{lt}(\mathtt{alt}), \mathtt{inv}(\mathtt{alt}))\} \subseteq \mathcal{R}$.

The first requirement of bisimulation in Definition 3.5 is equal to the first requirement of circular coinduction, so the proof is the same. The second requirement (bisim-left) can be rewritten:

$$\begin{aligned}\mathtt{lt}(\mathtt{lt}(\mathtt{alt})) &= \mathtt{lt}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})) \\ &= \mathtt{alt}\end{aligned}$$

and

$$\begin{aligned}\mathtt{lt}(\mathtt{inv}(\mathtt{alt})) &= \mathtt{lt}(\mathtt{n}(1, \mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})), \mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})))) \\ &= \mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt}))\end{aligned}$$

We can't prove equality of these terms yet, so we add this relation to $\mathcal{R}$. The third requirement (bisim-right) will return the same pair, because the `alt` tree is symmetrical. This means that now we need to prove:
$\{(\mathtt{lt}(\mathtt{alt}), \mathtt{inv}(\mathtt{alt})), (\mathtt{alt}, \mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})))\} \subseteq \mathcal{R}$.

Again we skip the trivial (bisim-root) which has already been proven. For (bisim-left) we derive:

$$\begin{aligned}\mathtt{lt}(\mathtt{alt}) &=_{\mathcal{R}} \mathtt{inv}(\mathtt{alt}) \\ &= \mathtt{lt}(\mathtt{n}(0, \mathtt{inv}(\mathtt{alt}), \mathtt{inv}(\mathtt{alt}))) \\ &= \mathtt{lt}(\mathtt{inv}(\mathtt{n}(1, \mathtt{alt}, \mathtt{alt})))\end{aligned}$$

This proves the second requirement. The third requirement (bisim-right) will give a similar proof, because the `alt` tree is symmetrical.

3.4 Transforming trees to streams

We could compare two infinite binary trees by transforming them into streams first and then comparing those using our methods discussed before in Chapter 2. For this we make a special `ZIP` function that takes all

data elements from a tree and places those elements in a stream. We define the ZIP function by using the zip function we know from combining two streams.

$$\begin{aligned}\texttt{ZIP}(n(x,t_1,t_2)) &= x : \texttt{zip}(\texttt{ZIP}(t_1),\texttt{ZIP}(t_2))\\ \texttt{zip}(x:s_1,s_2) &= x : \texttt{zip}(s_2,s_1)\end{aligned}$$

Lemma 3.10. $t_1 = t_2 \Leftrightarrow \texttt{ZIP}(t_1) = \texttt{ZIP}(t_2)$

Proof ($\Rightarrow$) Now we prove by circular coinduction on streams that if two trees are equal, their ZIP streams must be equal as well.

The first requirement (coind-head) follows from:
$\texttt{hd}(\texttt{ZIP}(t)) = \texttt{hd}(\texttt{r}(t) : \texttt{zip}(\texttt{ZIP}(\texttt{lt}(t)),\texttt{ZIP}(\texttt{rt}(t)))) = \texttt{r}(t) = t(\epsilon)$. Since we know that $t_1 = t_2$, according to (tree-1) we know that $t_1(\epsilon) = t_2(\epsilon)$ and therefore $\texttt{hd}(\texttt{ZIP}(t_1)) = \texttt{hd}(\texttt{ZIP}(t_2))$.

For the second requirement (coind-tail) we need to use the extended rule with special contexts, Theorem 4.2 of [13]. We have the coinduction hypothesis
$\texttt{fr}(C[\texttt{ZIP}(t_1)]) = \texttt{fr}(C[\texttt{ZIP}(t_2)])$ in which C is a special context. From Section 2.4 we know that $\texttt{zip}(\Box, s_2)$ and $\texttt{zip}(s_1, \Box)$ are special contexts. Now we derive:

$$\begin{aligned}\texttt{fr}(\texttt{tl}(\texttt{ZIP}(t_1))) &= \texttt{fr}(\texttt{tl}(\texttt{r}(t_1) : \texttt{zip}(\texttt{ZIP}(\texttt{lt}(t_1)),\texttt{ZIP}(\texttt{rt}(t_1)))))\\ &= \texttt{fr}(\texttt{zip}(\texttt{ZIP}(\texttt{lt}(t_1)),\texttt{ZIP}(\texttt{rt}(t_1))))\\ &=_{\text{CH}} \texttt{fr}(\texttt{zip}(\texttt{ZIP}(\texttt{lt}(t_2)),\texttt{ZIP}(\texttt{rt}(t_1))))\\ &=_{\text{CH}} \texttt{fr}(\texttt{zip}(\texttt{ZIP}(\texttt{lt}(t_2)),\texttt{ZIP}(\texttt{rt}(t_2))))\\ &= \texttt{fr}(\texttt{tl}(\texttt{r}(t_2) : \texttt{zip}(\texttt{ZIP}(\texttt{lt}(t_2)),\texttt{ZIP}(\texttt{rt}(t_2)))))\\ &= \texttt{fr}(\texttt{tl}(\texttt{ZIP}(t_2)))\end{aligned}$$

Both circular coinduction requirements for streams hold, so we have $\texttt{ZIP}(t_1) = \texttt{ZIP}(t_2)$.

Proof ($\Leftarrow$) Unfortunately I do not have a proof for this side of the equation.

There are many ways to define the ZIP function. The most intuitive is to perform a breadth-first search on the tree and put all data elements in order in the stream. However, the definition is less straight-forward. The

following stream specification is based on [1], an implementation in Haskell.

$$\begin{aligned}
\texttt{ZIP}_2(t) &= \texttt{bfs}([t]) \\
\texttt{bfs}(s) &= \texttt{map}(\texttt{r}, s) : \texttt{bfs}(\texttt{map}(\texttt{children}, s)) \\
\texttt{children}(t) &= [\texttt{lt}(t)] : [\texttt{rt}(t)] \\
\texttt{map}(\texttt{f}, x : s) &= \texttt{f}(x) : \texttt{map}(\texttt{f}, s) \\
\texttt{map}(\texttt{f}, x) &= \texttt{f}(x)
\end{aligned}$$

We use the notation $[t]$ to make a list of one element t. `bfs` is of type $s \to s$ and concatenates two streams into one using a special : of type $s \times s \to s$. Here we first take the root element of every tree in s and then continue recursively with the children of every tree in s. `children` is of type $t \to s$ and concatenates the left and right subtree together. Finally, `map` is of type $\texttt{f} \times s \to s$ and makes a list of $\texttt{f}(x)$ for all elements x in s.

A good `ZIP` function must evaluate all elements in the tree. With $\texttt{ZIP}_2$ this is straight-forward, but does the first `ZIP` also take all elements from the tree and what is the order in which they are placed? At first sight it seems random, but we observe for each path p, q:

$$\begin{cases} p = 1^n 0q \to p' = 0^n 1q \\ p = 1^n \to p' = 0^{n+1} \end{cases}$$

If we have $(\texttt{ZIP}(t))_n = t(p)$, then $(\texttt{ZIP}(t))_{n+1} = t(p')$. Unfortunately we do not have a proof for this observation. What we do know is that the first equation is simple binary addition in reverse. Each new path raises the old path by one, in which the most-left number is the least significant bit. Therefore every path will be chosen once and all data elements in the tree will get a position in the `ZIP` stream.

If we assume $\texttt{ZIP}_1$ and $\texttt{ZIP}_2$ are both correct `ZIP` functions, then what makes a 'good' `ZIP` function? Although the order of the elements in the stream can be different, most 'good' `ZIP` functions will have one thing in common: they still traverse the tree by depth. They will not evaluate an element at a certain depth when they haven't evaluated every element in lower depths yet. For example a faulty `ZIP` function that would always first evaluate the left subtree would only evaluate one path in the three, because with infinite trees it will never evaluate the rest. However, the order of the elements doesn't matter. $\texttt{ZIP}_1$ and $\texttt{ZIP}_2$ both apply to Lemma 3.10, but $\texttt{ZIP}_1(t) \neq \texttt{ZIP}_2(t)$

Lemma 3.11. *A good* `ZIP` *function is defined by*
$\forall t \in T, \forall p \in \{0,1\}^*\ (t(p) = (\texttt{ZIP}(t))_{\texttt{f}(p)})$
with $\texttt{f}(p) \geq 2^{\texttt{len}(p)} - 1$ *and* $\texttt{f}(p) < 2^{\texttt{len}(p)+1} - 1$

Here we use a function $\texttt{f}$ that outputs an integer given a path and a function $\texttt{len}$ that returns the length of the path. One could think of a good $\texttt{ZIP}$ function for which the restrictions on $\texttt{f}(p)$ do not hold, but it would be hard to define such a function.

Example 3.12. Let us take the $\texttt{ZIP}_2$ function and prove correctness by Lemma 3.11. The first requirement is trivial: every element in the tree will get a certain position in the stream, because of the breadth-first search. For the second requirement let us consider path 00. We know this has index 3 in the $\texttt{ZIP}_2$ stream, because it is preceded by ϵ, 0 and 1. We notice that the requirement is fulfilled: $3 \geq 2^{\texttt{len}(00)} - 1 = 2^2 - 1 = 3$ and $3 < 2^3 - 1 = 7$. This was a left-most path with the lowest index at a certain depth, so all paths at that depth will pass the lower bound. For the right-most path with the highest index at a certain depth we can consider path 11 as an example, for which we know it has index 6 in the $\texttt{ZIP}_2$ stream. Again we find the requirements fulfilled: $6 \geq 3$ and $6 < 7$.

Chapter 4

Related Work

Streams have been defined by H. Zantema in [12, 13]. Their definition is unrestrictive and allows a great variety of streams.
Endrullis et al. give an interesting overview of stream properties in [3], highlighting productivity, complexity, equality and 'turtle graphics'. They give a partial order of degrees of streams, where a degree is an equivalence class of streams, ranging from eventually periodic streams to uncountable streams.

The notion of proving equality of streams by bisimulation was introduced by V. Capretta in [2]. If their algorithm terminates successfully or diverges, the defined relation is indeed a bisimulation and the stream equation has at most one solution. Their work relies on abstracting corresponding subterms. Bisimulation can be used in many contexts, one of the first examples is in coalgebra by B. Jacobs and J. Rutten [6]. A few other areas of interest are elaborated in [10] by D. Sangiorgi.

Circular coinduction has been studied extensively by J. Goguen, K. Lin and G. Roşu. It was introduced in [4] and elaborated in [5] to be able to prove conditional equations and making it more versatile by adding case analysis. D. Lucanu and G. Roşu released CIRC [7], a circular coinductive prover implemented as an extension of Maude. In [8] they extended CIRC to be able to find special contexts, a means to obtain a distinguished class of special hypotheses.

In the field of automated equation provers F. Staals has made an implementation in Coq [11], although not as powerful as CIRC. H. Zantema and J. Endrullis introduced Streambox, which is an improvement on CIRC. It does not restrict to a specific format of behavioral equations and it does not require termination. It has a simple criterion for checking for a class of special contexts and finally it adds a technique to prove stream equality by exploiting unicity. However, G. Roşu has shown in [9] that proving

equality of streams is a Π_2^0-complete problem, which includes both the recursively enumerable and the co-recursively enumerable classes. This means that there is no complete proof system for equality of streams.

Chapter 5

Conclusions

We have given a specification of streams and what it means for two streams to be equal. From there we introduced two new methods of proving equality: bisimulation and circular coinduction. We have compared their definition with induction and shown their workings with a few examples. We discussed similarities and differences in proving and took a quick look at special contexts, an extension to circular coinduction.

The second part of this paper introduced infinite binary trees and showed what made them different from streams. We introduced new definitions for bisimulation and circular coinduction to be able to compare infinite binary trees and clarified with examples. We ended with a new way of comparing trees by transforming them into streams first. There are many possible `ZIP` functions and we tried to give a definition on what makes a good `ZIP` function.

Acknowledgments Our thanks go out to H. Geuvers for his guidance and ideas for writing this paper. We also want to thank H. Zantema for his ideas on how to define infinite binary trees.

Bibliography

[1] S. Behrens. Haskell: Breadth-first tree traversal. `http://jjinux.blogspot.nl/2005/12/haskell-breadth-first-tree-traversal.html`, 2005.

[2] V. Capretta. Bisimulations generated from corecursive equations. In *Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics*, volume 265 of *Electric Notes in Theoretical Computer Science*, pages 245 – 258. Elsevier, 2010.

[3] J. Endrullis, C. Grabmayer, D. Hendriks, and J.W. Klop. Infinite streams. Written for the NWO BRICKS-project Infinity, 2009.

[4] J. Goguen, K. Lin, and G. Roşu. Circular coinductive rewriting. In *Proceedings, 15th International Conference on Automated Software Engineering (ASE'00)*. Institute of Electrical and Electronics Engineers Computer Society, 2000.

[5] J. Goguen, K. Lin, and G. Roşu. Conditional circular coinductive rewriting with case analysis. In *Recent trends in Algebraic Development Techniques (WADT02)*, volume 2755 of *Lecture Notes in Computer Science*, pages 216 – 232. Springer, 2003.

[6] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222 – 259, 1997.

[7] D. Lucanu and G. Roşu. CIRC: A circular coinductive prover. In *CALCO'07*, volume 4624 of *Lecture Notes in Computer Science*, pages 372 – 378. Springer, 2007.

[8] D. Lucanu and G. Roşu. Circular coinduction with special contexts. In *Proceedings of the 11th International Conference on Formal Engineering Methods (ICFEM'09)*, volume 5885 of *Lecture Notes in Computer Science*, pages 639 – 659. Springer, 2009.

[9] G. Roşu. Equality of streams is a Π_2^0-complete problem. In *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming (ICFP'06)*. ACM, 2006.

[10] D. Sangiorgi. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems*, 31 (4):111 – 151, 2009.

[11] F. Staals. Proving equality between streams. University of Technology Eindhoven, 2006.

[12] H. Zantema. Well-definedness of streams by transformation and termination. *Logical Methods in Computer Science*, 6(3), 2010.

[13] H. Zantema and J. Endrullis. Proving equality of streams automatically. In *Schmidt-Schauß, M. (ed.) 22nd International Conference on Rewriting Techniques and Applications (RTA'11)*, volume 10 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 393 – 408. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011.