SAARLAND UNIVERSITY

FACULTY OF NATURAL SCIENCES AND TECHNOLOGY I

DEPARTMENT OF COMPUTER SCIENCE

BACHELOR'S THESIS

A Syntactic Theory of Finitary Sets

Author:
Denis MÜLLER

Advisor:
Steven SCHÄFER

Supervisor:
Prof. Dr. Gert Smolka

Reviewers:
Prof. Dr. Gert Smolka
Prof. Dr. Holger Hermanns

Submitted: 21.07.2015

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,______________________ ______________________________
(Datum/Date) (Unterschrift/Signature)

Acknowledgements

First of all, I would like to thank Prof. Dr. Smolka for offering me this thesis, as well as for his highly enlightening lectures on constructive type theory, Coq and set theory.

Furthermore, I am very grateful to my Advisor Steven Schäfer for the many great ideas he came up with, for guiding me back onto the right track when I lost myself in the midst of proof terms and for his helping me out with his enormous technical prowess when I encountered problems with Coq.

I especially appreciated the sparkling talks of many highly motivated students of the Programming Systems Lab and the very productive working atmosphere caused thereby.

Last but not least, a big thank you to Vanessa for not only proofreading my thesis but also for her considerable patience and extensive soothing when Coq did not want to work the way I wanted it to work.

Contents

1 Introduction 6

I Hereditarily finite sets 10

2 Basics 10

3 Equivalence 11

4 Decidability 14
4.1 Comparing trees 14
4.2 Sorting trees 16

5 Elements and subsets 18
5.1 Element Characterisation 18
5.2 Subset Characterisation 19

6 Bisimulation 20

II Finitary sets 21

7 Basics 21

8 Bisimulation 23

9 ZF Axioms 24
9.1 Empty 25
9.2 Adjunction 26
9.3 Pairing 27
9.4 Union 27
9.5 Separation 28
9.6 Replacement 28
9.7 Power 28

10 Transitive closure 29

11 Constructing a set type 33
11.1 Graphs over $\mathbb{N}$ 34
11.2 Conversion functions 37
11.3 Quotient types 39
11.4 $\mathcal{N}$ 40

12 Future Work 42

III Appendix 43

13 File Structure 43

14 References 44

1 Introduction

This thesis aims to give models for hereditarily finite sets and finitary sets based on constructive type theory. Constructive type theory is a logic which differs from classical mathematics in two main points:

- Whereas in the former, types are ubiquitous, the latter has set theory at its foundation. Many constructs in classical mathematics thus rely - more or less explicitly - on sets, and their counterparts in constructive type theory are more often than not just (inductive) types.

  The natural numbers provide a simple example for this: they are defined as sets in classical mathematics, but are represented by an inductive type in constructive type theory ($n : \mathbb{N} ::= O \mid S\ n$).

  Furthermore, there is the so-called Curry-Howard isomorphism [1], which states that statements can also be represented by types. The elements of such a type then represent the proofs of the respective statement. As a consequence, checking the correctness of a proof becomes equivalent to type-checking the proof term, which facilitates automated proof checking.

- As the name implies, constructive type theory only allows constructive proofs. This is not the case for classical mathematics, where one usually relies on an assumption called excluded middle (XM for short, sometimes referred to as "tertium non datur"), which states that for every proposition P, either P or its negation holds. This principle justifies a technique called proof by contradiction, which is often used to prove existence of certain things. This technique, however, is intrinsically non-constructive. As a consequence, sometimes the problem arises that we know some element of a type has a property, but we do not have an explicit element satisfying that property. In constructive type theory, on the other hand, every existential proof comes with a so-called witness, i.e. an explicit element that satisfies the property, hence avoiding the aforementioned problem. Therefore, constructive type theory does not have the assumption XM.

The entire development related to this thesis is formalized in the proof assistant Coq, which is one of many proof assistants based on constructive type theory (more specifically the calculus of inductive definitions) that make use of the Curry-Howard isomorphism to check the correctness of proofs. The aforementioned assumption XM is independent in Coq, which means that we can either assume XM or its negation, and the resulting logic remains consistent.

As mentioned before, classical mathematics is based on set theory. Various axiomatizations of sets have been proposed, the standard choice of today being the so-called ZF-theory due to Zermelo and Fraenkel [2] [3] [4] , which is comprised of an element relation $\in$ and the following axioms:

- The axiom of extensionality characterizes the equality of sets:
  $\forall M\, N.\, M = N \iff (\forall x.\, x \in M \iff x \in N)$
- The axiom of existence guarantees the existence of a unique empty set denoted by $\emptyset$ such that $\forall M. M \notin \emptyset$.
- The axiom of pairing guarantees for any M, N the existence of a unique set that we denote by {M, N} such that $\forall M\, N\, N'.\, N' \in \{M, N\} \iff N' = M \vee N' = N$.

As a consequence, this also gives us the existence of a singleton set containing only a single set M, for any set M:

$\{M\} := \{M, M\}$ such that $x \in \{M\} \iff x = M$.

Sets obtained by the axiom of pairing are called unordered pairs, since the order of M and N is irrelevant (due to the axiom of extensionality).

- The axiom of union guarantees the existence of a unique set denoted by $\bigcup N$ for any set N such that $\forall M\, N.\, M \in \bigcup N \iff \exists N' \in N. M \in N'$.

  There is also the notion of a binary union operator $\cup$, which can be defined in terms of $\bigcup$ as follows:

  $M \cup N := \bigcup\{M, N\}$.

  It is easy to see that $x \in M \cup N \iff x \in M \vee x \in N$.

  Furthermore, there is an operation called adjunction which will be important later on.

  Its definition is $M; N := M \cup \{N\}$. Note that $x \in M; N \iff x \in M \vee x = N$.

- The axiom of power guarantees, for any set M, the existence of a unique set, the so-called power set, that we denote by $\mathcal{P}(M)$

  such that $\forall N.\, N \in \mathcal{P}(M) \iff N \subseteq M$.

- The axiom of separation guarantees the existence of a unique set that we denote by

  $\{x \in M \mid P\ x\}$ for any set M and predicate P

  such that $y \in \{x \in M \mid P\ x\} \iff y \in M \wedge P\ y$. While predicates are usually modelled by sets in classical mathematics, we model predicates by the intrinsic functions of constructive type theory. In particular, such a predicate P has type Set $\rightarrow$ Prop. Prop is the universe of predicates, which means that any statement has type Prop. Note that, in order to avoid inconsistencies, P has to be an extensional predicate. This will be discussed in the second part of the thesis.

- The axiom of replacement guarantees the existence of a unique set denoted by

  $\{f\ x | x \in M\}$ for any set M and function f. As with the axiom of separation, functions are also sets in classical mathematics. However, since in this thesis, we study sets from the perspective of constructive type theory, we use its native functions. In particular, the type of f for this axiom has to be f : Set $\rightarrow$ Set. There is another, more general, version of this axiom that uses relations instead of functions. If the given relation is total, we basically have the same case as here. However, if we are only given a partial relation, the result will be a subset of the set which results from our version of replacement. Due to the axiom of separation, however, this system does not lose any expressiveness, as we can combine the axioms of replacement and separation to model replacement based on relations. On the other hand, if we were to use the stronger version of replacement, the axiom of separation would become redundant.

- The axiom of infinity guarantees the existence of an infinite set X such that

  $\emptyset \in X \wedge \forall y.\, y \in X \implies y; y \in X$.

  Note that $y;y = y \cup \{y\}$. Without this axiom, we are not able to construct sets with infinitely many elements. Often, X is restricted to be minimal, in which case it coincides with the von Neumann ordinal [5] ω, which is basically the set of all natural numbers $\mathbb{N}$.

- The axiom of regularity (also called axiom of foundation) is different from the previous axioms in that it does not give us the possibility to construct new sets, but limits the sets we can potentially build. It states that every set is well-founded, i.e. every descending chain of elements is finite. In other words, every set only has finitely many successors with respect to the membership relation.

The need for a consistent axiomatization of set theory arose due to Cantor's naive set theory, where sets are defined as any imaginable collection of elements, being inconsistent: Consider the set $R := \{x | x \notin x\}$. It is easy to see that $R \in R \iff R \notin R$. This inconsistency of Cantor's naive set theory is known as Russel's paradox [6].
Furthermore, there is the axiom of choice, whose usual formulation is as follows: For any set X whose elements are non-empty, there is a choice function $\mathrm{f} : \mathrm{X} \rightarrow \bigcup X$ such that $\forall y \in X.\, f y \in y$. While the axiom of choice seems intuitive and entirely reasonable, it has some unexpected and counter-intuitive consequences, such as the Banach–Tarski paradox [7] and Zermelo's well-ordering theorem [8][9], which states that any set with a choice function has a well-ordering. Therefore, the axiom of choice has caused a lot of controversy. It should be noted that the axiom of choice is independent of the remaining ZF-axioms. Hence, it can either be consistently assumed or omitted. Furthermore, if one omits the axiom of infinity, it is easy to construct a choice function, since only finite sets remain. The aforementioned controversy regarding this axiom persists only in the presence of the axiom of infinity. When assuming the axiom of choice in addition to the remaining ZF-axioms, one reaches a set theory commonly abbreviated ZFC (Zermelo-Fraenkel + choice).

There is an operation on sets whose usual construction relies on the axiom of infinity, namely the transitive closure of a set M, which basically gives us a set that contains all sets that can be reached via a descending chain of elements of M. The aforementioned construction goes as follows:
$tc\, M := \bigcup\limits_{n \in \mathbb{N}} (\bigcup^n M)$, where $\bigcup^n$ denotes iterating $\bigcup$ n times.

A special class of well-founded sets are the so-called hereditarily finite sets. These are sets with only finitely many elements, all of which are in turn hereditarily finite.
One application of hereditarily finite sets is Robin Millner's calculus of communicating systems (CSS)[10], which is one of many process calculi that can be used to model concurrent behaviour. Hereditarily finite sets can be used to model the fragment of CCS consisting of only one action, the empty process, a binary + operator on processes and action prefixes, as shown by Abramsky [11].

Another kind of sets is non-well-founded sets. These may contradict the axiom of regularity, i.e. they are not necessarily well-founded. The simplest example of such a set is the set $\Omega = \{\Omega\}$. It is obvious that Ω is not well-founded. However, non-well-founded sets also include all well-founded sets. Non-well-founded sets have been thoroughly invastigated by many people, and the most fundamental work on them is due to Peter Aczel. In his book [12], he axiomatizes non-well-founded sets and gives examples of applications of non-well-founded sets. In lieu of the axiom of regularity, there is the so-called Anti-Foundation Axiom (AFA) for non-well-founded sets, which states that every accessible pointed graph (apg) corresponds to a unique set. An accessible pointed graph is a directed graph with a root vertex that can reach every other vertex. We will use a similar representation for graphs, but do not require accessibility, i.e. we allow vertices that are unreachable from the root, but these are basically irrelevant for our purposes and interpretation.

We use the term finitary to denote a special class of non-well-founded sets. This class is very similar to hereditarily finite sets in that every finitary set has only finitely many elements, each of which are in turn finitary. However, as opposed to hereditarily finite sets, finitary sets need not be well-founded. Ω is the prime example of a finitary set.

Continuing on the connection of sets to CCS, if we also allow recursive processes in the fragment of CCS considered above, we end up with a fragment of CCS that can be modelled by finitary sets.
The relation between (general) non-well-founded set and full CCS has been studied before, e.g. by Aczel [12] and Baldamus [13].

In the first part of the thesis, we give a model for hereditarily finite sets based on binary trees. We will consider a couple of possible equivalence relation on these trees, prove their equivalence and show how to decide tree equivalence.

In the second part of the thesis, we construct a premodel for finitary sets based on graphs and give constructions for the axioms of existence, pairing, union, separation, replacement, power and extensionality modulo bisimulation. Since finitary sets are non-well-founded and only have finitely many elements, the axioms of regularity and infinity do not make sense.
Furthermore, we also give a construction for the transitive closure modulo bisimulation, whose standard definition is not possible in our model, since we do not have the axiom of infinity.
We show that the bisimulation we define is decidable and - based on a mapping between graphs and natural numbers - give a construction for the quotient type $\mathbb{G}\ /_{\approx}$, which we define as our type for finitary sets. Furthermore, we additionaly have Aczel's AFA, which states that for every apg, there exists a unique non well-founded set. The respective construction is trivial in our case, since every graph belongs to a unique equivalence class. The constructions for the axioms discussed above and the transitive closure then hold in their original form for our type of finitary sets. In addition, we also give a choice function on finitary sets. Since we do not have the axiom of infinity, it is intuitively clear that this works. Our characterization of the choice function γ we give looks slightly different than the axiom of choice above: We have that $\forall M \neq \emptyset.\ \gamma M \in M$. However, this formulation is equivalent to the usual one given above, although in our opinion much simpler.

Concerning conventions for this thesis, we should note the following two points:

- Whenever a free variable occurs somewhere, it is implicitly universally quantified.
- Since we study sets based on constructive type theory, we will in general prefer inductive characterizations and definitions over the first-order characterizations and definitions encountered in classical set theory.

Related Work

The initial interest for this thesis arose owing to the Bachelor Thesis of Kathrin Stark [14], which - among others - featured a formalization of a restricted subset of CCS without recursion in the proof assistant Coq. Incidentally, if one adds the restriction that there is only a single action, the fragment of CCS considered by Kathrin is exactly the same as one that can be modelled by hereditarily finite sets.

Dominik Kirst's Bachelor thesis [15] provides a substantial formalization of the classical ZF theory implemented in the proof assistant Coq.
Alexandre Miquel's slides on inconsistent type systems [16] contain a representation of sets as pointed graphs. He models extensional equality of sets by bisimilarity and membership as shifted bisimilarity, which is the same idea that we use in the second part of the thesis.

Contribution

We seem to be the first to give a formalization of a constructive model for finitary sets in Coq. In contrast to a lot of other work, the model we give does not assume any ZF axioms, but has explicit constructions for all ZF axioms that make sense for finitary sets, as well as for Aczel's AFA.
We also present a model for hereditarily finite sets based on binary trees and give different characterization of equivalence on these trees, and although we do not give explicit constructions for most ZF axioms in this case, the constructions used in the second part can be generalized and applied to our binary tree model, which would give us a model for hereditarily finite sets that admit all ZF axioms except infinity.

Part I
Hereditarily finite sets

2 Basics

In this first part of the thesis, we will give a model of hereditary finite (HF) sets based on constructive type theory. One possible characterization of HF sets is the following inductive predicate:

$$\frac{}{\text{HF } \emptyset} \qquad \frac{\text{HF M} \qquad \text{HF N}}{\text{HF (M ; N)}}$$

We will model these sets using binary trees.

Definition 2.1. s, t : $\mathbb{T}$ ::= $\dot{\emptyset}$ | s . t

The constructor s.t is right associative, i.e. s.t.u = s.(t.u). We use the following semantics for binary trees:

- $[\![\dot{\emptyset}]\!] = \emptyset$
- $[\![s.t]\!] = \{[\![s]\!]\} \cup [\![t]\!]$

Recall that the adjunction has the following property : M;N = M ∪ {N}. It is easy to see that the semantics we use for binary trees give us a model of hereditarily finite sets (although the order of the arguments is reversed).
We can get all child trees of a given tree using the $\mathcal{L}$ function, which collects all left children of elements along the right spine of the tree (which are, according to our semantics for binary trees, exactly the trees which model the children of the set modelled by the original tree). This is shown in figure 1, which depicts the tree s.t.u.$\dot{\emptyset}$. The children of this tree are s, t and u.

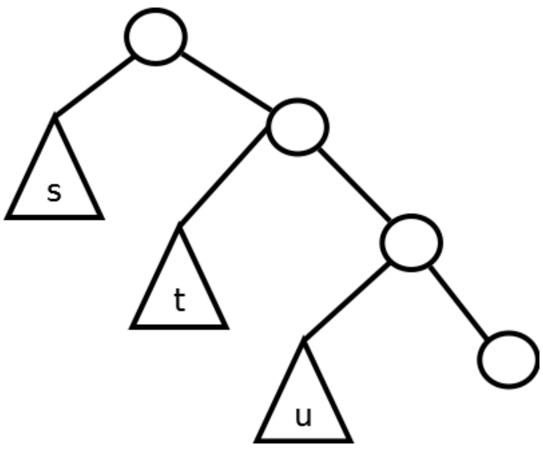

Figure 1: interpretation of tree structure

Definition 2.2. $\mathcal{L} : \mathbb{T} \to [\mathbb{T}]$
$\mathcal{L}\,\dot{\emptyset} := []$
$\mathcal{L}\,(s.t) := s :: (\mathcal{L}\ t)$

We can, of course, invert this operation.

Definition 2.3. $\mathcal{T} : [\mathbb{T}] \to \mathbb{T}$
$\mathcal{T}\,[] := \dot{\emptyset}$
$\mathcal{T}\ (t :: ts) := t.(\mathcal{T}\ ts)$

We can also append two trees:

Definition 2.4. append : $\mathbb{T} \to \mathbb{T} \to \mathbb{T}$
append $\dot{\emptyset}$ t := t
append (s.s') t := s.(append s' t)

We write s @ t instead of append s t.
Note that $\mathcal{L}(s@t) = \mathcal{L}(s) + \mathcal{L}(t)$ and $\mathcal{T}(l +\!\!+ l') = \mathcal{T}(l)\ @\ \mathcal{T}(l')$.
These functions will be very useful later on when we discuss characterizations of tree equivalence.

3 Equivalence

There are multiple ways to define equivalence on our model of hereditarily finite sets. In fact, we will see a few possibilities and prove their equivalence. Note that one usually defines equivalence based on an element relation. However, defining an element relation based on set equivalence is just as easy, since $M \in N \iff N \equiv \{M\} \cup N$.

There are two crucial properties that a relation which is intended to model set equivalence has to satisfy:

- Duplicate elements can be ignored

- The ordering of the elements is irrelevant

In fact, these two properties are already sufficient. Consider the least congruence satisfying these rules:

Definition 3.1. $\equiv : \mathbb{T} \to \mathbb{T} \to Prop$

$$\frac{}{s.s.t \equiv s.t} \qquad \frac{}{s.t.u \equiv t.s.u} \qquad \frac{}{\dot{\emptyset} \equiv \dot{\emptyset}}$$

$$\frac{s \equiv s' \qquad t \equiv t'}{s.t \equiv s'.t'} \qquad \frac{s \equiv t \qquad t \equiv u}{s \equiv u} \qquad \frac{\dot{\emptyset} \neq s \equiv t \neq \dot{\emptyset}}{s \equiv t}$$

The first two rules (Deletion and Swap, respectively) realise the properties discussed above. Technically speaking, the least congruence satisfying the two aforementioned properties is exactly the one given here, except for the symmetry rule, where we restricted both elements to be different from the empty tree. The rule we use, however, gives us slightly shorter proofs for certain lemmas in the Coq development. Moreover, it is easy to see that an unrestricted symmetry rule is admissible for this equivalence relation:

Lemma 3.2. $\forall s\, t.\, s \equiv t \implies t \equiv s$

Proof. Induction on $s \equiv t$. All cases are trivial. □

Reflexivity can also easily be established by induction. Note that $\equiv$ is obviously a congruence relation.
Furthermore, note that a tree is equivalent to $\dot{\emptyset}$ iff it is equal to $\dot{\emptyset}$:

Lemma 3.3. $\forall s.\, s \equiv \dot{\emptyset} \implies s = \dot{\emptyset}$

Proof. By induction on $s \equiv \dot{\emptyset}$. Due to the restriction in the symmetry rule, all cases are trivial. □

Of course, the same holds for $\dot{\emptyset} \equiv s$, due to symmetry.

Note that equivalent trees have equivalent children:

Definition 3.4. $s \precsim t := \forall s' \in \mathcal{L}(s).\, \exists t' \in \mathcal{L}(t).\, s' \equiv t'$.

Lemma 3.5. $\forall s\, t.\, s \equiv t \implies s \precsim t \land t \precsim s$

Proof. Induction on $s \equiv t$.

- If s = $\dot{\emptyset}$ = t, the claim follows, since $\dot{\emptyset} \precsim \dot{\emptyset}$ holds vacuously.
- If s = s'.s'.t' and t = s'.t' (or vice-versa), we can see that $\forall u.\, u \in \mathcal{L}(s) \iff u \in \mathcal{L}(t)$.
- If s = s'.t'.u' and t = t'.s'.u' (or vice-versa), $\forall u.\, u \in \mathcal{L}(s) \iff u \in \mathcal{L}(t)$ again holds.
- For the symmetry rule, the claim follows by IH. This is the reason why we have to prove $s \equiv t \implies s \precsim t \land t \precsim s$ instead of simply $s \equiv t \implies s \precsim t$ - in the latter case, the IH would not be strong enough.
- In the transitivity case, the claim again follows by IH and due to transitivity of $\equiv$.

- In the remaining case, we have $s \equiv s'$, $t \equiv t'$ and want to prove $(s.t) \precsim (s'.t')$ (the other direction is analogous). Let $u \in \mathcal{L}(s.t) = s :: \mathcal{L}(t)$. If u = s, the claim follows, since $s \equiv s' \in \mathcal{L}(s'.t')$. If $u \in \mathcal{L}(t)$, the claim follows by IH and due to transitivity of $\equiv$.

□

The other direction will be discussed later and is considerably harder to prove without further characterizations of $\equiv$.
Let us now consider a second equivalence relation.

Definition 3.6. $\rhd : \mathbb{T} \to \mathbb{T} \to Prop$

$$\frac{}{s.s.t \rhd s.t} \qquad \frac{}{s.t.u \rhd t.s.u} \qquad \frac{s \rhd s'}{s.t \rhd s'.t} \qquad \frac{t \rhd t'}{s.t \rhd s.t'}$$

We call $\rhd$ the *step relation*. This relation basically captures the behaviour of the Deletion and Swap rules.

Definition 3.7. $\equiv_\rhd : \mathbb{T} \to \mathbb{T} \to Prop$

$$\frac{}{s \equiv_\rhd s} \qquad \frac{s \rhd t \qquad t \equiv_\rhd u}{s \equiv_\rhd u} \qquad \frac{t \rhd s \qquad t \equiv_\rhd u}{s \equiv_\rhd u}$$

While the step relation handles the domain specific behaviour, $\equiv_\rhd$ adds reflexivity and admits transitivity and symmetry. The reason for having a second characterization is that in some cases, induction on $\equiv_\rhd$ is much easier than induction on $\equiv$.
We will soon see that $\equiv$ and $\equiv_\rhd$ are in fact equivalent.

Lemma 3.8. $\forall s\, t.\, s \rhd t \implies s \equiv_\rhd t.$

Proof. Consider the proof tree for this lemma:

$$\frac{s \rhd t \qquad \frac{}{t \equiv_\rhd t}}{s \equiv_\rhd t}$$

□

Lemma 3.9. $\equiv_\rhd$ admits transitivity, i.e.
$\forall s\, t\, u.\, s \equiv_\rhd t \implies t \equiv_\rhd u \implies s \equiv_\rhd u.$

Proof. By induction on $s \equiv_\rhd t$. The reflexivity case is trivial.
The case for the 2nd and 3rd rule are analogous.
Consider the second case. Let $s \rhd t$, $t \equiv_\rhd t' \equiv_\rhd u$. Due to the IH, t $\equiv_\rhd$ u. □

Lemma 3.10. $\equiv_\rhd$ admits symmetry, i.e.
$\forall s\, t.\, s \equiv_\rhd t \implies t \equiv_\rhd s.$

Proof. By induction on s $\equiv_\rhd$ t. The reflexivity case is again trivial.
In the second case, s $\rhd$ t and t $\equiv_\rhd$ u. Since s $\rhd$ t, we know that s $\equiv_\rhd$ t, hence t $\equiv_\rhd$ s. The claim follows by transitivity of $\equiv_\rhd$.
By the same reasoning, the claim follows in the third case. □

We will now show the equivalence of the two relations. Note that if s $\rhd$ t, then s $\equiv$ t, as can be proven easily by induction.

Lemma 3.11. $\forall s\, t.\, s \equiv t \iff s \equiv_{\triangleright} t.$

Proof.
"⇒": Induction on $s \equiv t$. The Deletion and Swap rules are handled by the step relation, whereas the cases for symmetry, transitivity have already been shown, since these rules are admissible for $\equiv_{\triangleright}$ and $\dot{\emptyset} \equiv_{\triangleright} \dot{\emptyset}$ follows due to reflexivity.
In the remaining case, $s.t \equiv s'.t'$ where $s \equiv s'$, $t \equiv t'$, the claim follows due to IH and transitivity of $\equiv_{\triangleright}$.
"⇐": Induction on $s \equiv_{\triangleright} t$. Reflexivity is admissible for $\equiv$.
Let s $\triangleright$ t and t $\equiv_{\triangleright}$ u. By IH, $t \equiv u$. Note that s $\equiv$ t, since s $\triangleright$ t. The claim follows due to transitivity of $\equiv$.
The remaining case, t $\triangleright$ s and t $\equiv_{\triangleright}$ u, is analogous to the second case.

□

This lemma, together with 3.5 immediately gives us the following lemma:

Lemma 3.12. $\forall s\, t.\, s \equiv_{\triangleright} t \implies$
$(\forall s' \in \mathcal{L}(s)\, \exists t' \in \mathcal{L}(t)\, .\, s' \equiv_{\triangleright} t') \wedge (\forall t' \in \mathcal{L}(t)\, \exists s' \in \mathcal{L}(s)\, .\, t' \equiv_{\triangleright} s')$.

4 Decidability

Now that we have seen some characterizations of tree equivalence, the question of how to decide whether two given trees are equivalent arises. We will do so by sorting the trees such that

- two sorted trees are equivalent iff they are equal
- Every tree is equivalent to its sorted form

Note that equality on trees is decidable, since trees are a simple inductive type.

4.1 Comparing trees

Definition 4.1. Order Order ::= LT | EQ | GT

Definition 4.2. cmp : $\mathbb{T} \to \mathbb{T} \to Order$
cmp $\dot{\emptyset}$ $\dot{\emptyset}$ = EQ
cmp $\dot{\emptyset}$ _ = LT
cmp _ $\dot{\emptyset}$ = GT
cmp (s.t) (s'.t') = $\begin{cases} cmp\ t\, t' & cmp\ s\, s' = EQ \\ x & cmp\ s\, s' = x \neq EQ \end{cases}$

Lemma 4.3. $\forall s\, t.\, cmp\ s\, t = EQ \implies s = t.$

Proof. By induction on s. The base case is trivial.
If t is empty, cmp (s.s') t $\neq$ EQ.
Otherwise, t = t.t' and cmp (s.s') (t.t') = EQ.
If cmp s t = EQ, then s = t by IH for s. In this case, s.s' = t.t' follows by the IH for s', since cmp s' t' = cmp (s.s') (t.t') = EQ.
If cmp s t $\neq$ EQ, then cmp (s.s') (t.t') $\neq$ EQ, yielding a contradiction. ↯

□

Similarly, cmp s s = EQ can be easily proven by induction.
We need three additional properties of cmp that any reasonable compare function should satisfy:

- cmp s t = LT $\implies$ cmp t s = GT
- cmp s t = GT $\implies$ cmp t s = LT
- cmp s t = LT $\implies$ cmp t u = LT $\implies$ cmp s u = LT

Lemma 4.4. $\forall s\, t$. cmp s t = LT $\implies$ cmp t s = GT.

Proof. We prove the equivalent $\forall t\, s$. cmp s t = LT $\implies$ cmp t s = GT by induction on t.
In the base case, cmp s $\dot{\emptyset}$ = LT yields a contradiction.
In the inductive case, we have cmp s $(t_1.t_2)$ = LT. If s = $\dot{\emptyset}$, cmp $(t_1.t_2)$ $\dot{\emptyset}$ = GT follows from the definition of cmp. Otherwise, s = $s_1.s_2$ and cmp $(s_1.s_2)$ $(t_1.t_2)$ = LT.
Case analysis on cmp $t_1\, s_1$.

- If cmp t_1 s_1 = GT, cmp $(t_1.t_2)$ $(s_1.s_2)$ = GT follows immediately.
- If cmp t_1 s_1 = EQ, we know that $t_1 = s_1$. Since cmp $(s_1.s_2)$ $(s_1.t_2)$ = LT, we know that cmp s_2 t_2 = LT, and the claim follows by IH for t_2.
- In the last case, cmp t_1 s_1 = LT. Case analysis on cmp s_1 t_1.
  - cmp s_1 t_1 = GT contradicts cmp $(s_1.s_2)$ $(t_1.t_2)$ = LT.
  - If cmp s_1 t_1 = EQ, we again have $s_1 = t_1$, contradicting cmp t_1 s_1 = LT.
  - In the last remaining case, cmp s_1 t_1 = LT, it follows by IH for t_1 that cmp t_1 s_1 = GT, contradicting cmp t_1 s_1 = LT.

□

The proof for the second lemma is analogous to the one we have just seen.

Lemma 4.5. $\forall s\, t\, u$. cmp s t = LT $\implies$ cmp t u = LT $\implies$ cmp t u = LT.

Proof. By induction on s.
The base case is trivial.
In the inductive case, s = $s_1.s_2$.
Likewise, t = $t_1.t_2$ and u = $u_1.u_2$, since neither t nor u can be $\dot{\emptyset}$, since $\forall v$. cmp v $\dot{\emptyset} \neq$ LT, as can be easily proven by induction.
Case analysis on cmp s_1 t_1.

- cmp s_1 t_1 = GT contradicts cmp s t = LT.
- If cmp s_1 t_1 = EQ, $s_1 = t_1$ and cmp s_2 t_2 = LT. In a further case analysis on cmp t_1 u_1, all cases are easy.
- In the remaining case, cmp s_1 t_1 = LT. Case analysis on cmp t_1 u_1.
  - cmp t_1 u_1 = GT contradicts cmp t u = LT.
  - If cmp t_1 u_1, cmp s_1 u_1 = LT follows by IH for s_1, implying cmp s u = LT.
  - If cmp t_1 u_1 = EQ, we know that $t_1 = u_1$, hence cmp s u = LT follows, since cmp s_1 u_1 = LT.

□

4.2 Sorting trees

Now that we have a sensible way to compare trees, we can sort them. For simplicity, we use insertion sort.

Definition 4.6. insert : $\mathbb{T} \to \mathbb{T} \to \mathbb{T}$
insert s $\dot{\emptyset}$ = s .: $\dot{\emptyset}$

$$\text{insert s } (t_1.t_2) = \begin{cases} s.t_1.t_2 & cmp\ s\ t_1 = LT \\ t_1.t_2 & cmp\ s\ t_1 = EQ \\ t_1.(insert\ s\ t_2) & cmp\ s\ t_1 = GT \end{cases}$$

Definition 4.7. sort : $\mathbb{T} \to \mathbb{T}$
sort $\dot{\emptyset}$ = $\dot{\emptyset}$
sort $(s_1.s_2)$ = insert (sort s_1) (sort s_2)

Note that we sort the trees strictly, i.e. there will be no duplicates in a sorted tree.
We first prove that $\equiv_{\triangleright}$ is invariant under sorting.

Lemma 4.8. $\forall s\, t.$ (s . t) $\equiv_{\triangleright}$ (insert s t)

Proof. By induction on t.
In the base case, insert s $\dot{\emptyset}$ = s.$\dot{\emptyset}$, hence the claim follows due to reflexivity of $\equiv_{\triangleright}$.
In the inductive case, t = $t_1.t_2$ and we want to show s.$t_1.t_2 \equiv_{\triangleright}$ insert s $(t_1.t_2)$. Case analysis on cmp s t_1.

- If cmp s t_1 = LT, then insert s $(t_1.t_2)$ = s.$t_1.t_2$.
- If cmp s t_1 = EQ, then s = t_1 and insert s $(t_1.t_2) = t_1.t_2$. Since $t_1.t_1.t_2 \triangleright t_1.t_2$, the claim follows.
- If cmp s t_1 = GT, then insert s $(t_1.t_2) = t_1$. (insert s t_2). Note that $s.t_1.t_2 \equiv_{\triangleright} t_1.s.t_2$. By IH for t_2, insert s $t_2 \equiv_{\triangleright}$ s.t_2 and the claim follows due to transitivity of $\equiv_{\triangleright}$.

□

Lemma 4.9. $\forall s.$ s $\equiv_{\triangleright}$ (sort s)

Proof. By induction on s. The base case is trivial.
In the inductive case, we have s = $s_1.s_2$ and $s_1 \equiv_{\triangleright}$ (sort s_1), $s_2 \equiv_{\triangleright}$ (sort s_2) due to IH. The claim follows due to transitivity and
$s_1.s_2 \equiv_{\triangleright}$ (sort s_1).$s_2 \equiv_{\triangleright}$ (sort s_1). (sort s_2) $\equiv_{\triangleright}$ insert (sort s_1) (sort s_2), where the last step was proven in the previous lemma. □

As mentioned at the beginning of this section, we want to prove that two sorted trees are equivalent iff they are equal. The direction from right to left follows from the reflexivity of $\equiv_{\triangleright}$. For the other direction, we first prove that the Swap and Deletion rules play nicely with sorting.

Lemma 4.10. $\forall s\, t.$ sort (s.s.t) = sort (s.t)

Proof. We prove the equivalent $\forall y\, x.$ insert x (insert x y) = insert x y by induction on y.
In the base case, insert x (x.$\dot{\emptyset}$) = x.$\dot{\emptyset}$, since cmp x x = EQ.
In the inductive case, we want to show that insert x (insert x $(y_1 . y_2)$) = insert x $(y_1 . y_2)$. Case analysis on cmp x y_1.

- If cmp x y_1 = LT, then insert x ($y_1.y_2$) = x.$y_1.y_2$. The claim follows, since cmp x x = EQ.
- If cmp x y_1 = EQ, then x = y_1 and insert x ($y_1.y_2$) = $y_1.y_2$ and the claim follows since cmp y_1 y_1 = EQ.
- If cmp x y_1 = GT, then insert x ($y_1.y_2$) = y_1.(insert x y_2). By IH for y_2, we know that insert x (insert x y_2) = insert x y_2), which proves the claim.

□

Lemma 4.11. $\forall s\, t\, u$. sort (s.t.u) = sort (t.s.u)

Proof. By induction on u. The base case is easy. In the inductive case, we want to show that sort (s.t.$u_1.u_2$) = sort (t.s.$u_1.u_2$). If cmp s t = EQ or cmp t u_1 = EQ or cmp s u_1 = EQ, we know that the respective trees are equal and that the duplicates will be removed, hence these cases are easy. Otherwise, w.l.o.g., cmp s t = LT.
If cmp t u_1 = LT, then cmp s u_1 = LT and sort (t.s.$u_1.u_2$) = s.t.$u_1.u_2$ = sort (s.t.$u_1.u_2$). Otherwise, cmp t u_1 = GT and there are two cases for cmp s u_1 (barring the EQ case):

- cmp s u_1 = LT. In this case, sort (t.s.$u_1.u_2$) = s.u_1.t.u_2 = sort (s.t.$u_1.u_2$).
- cmp s u_1 = GT. In this case, sort (s.t.$u_1.u_2$) = u_1. (insert s (insert t u_2)) and sort (t.s.$u_1.u_2$) = u_1.insert t (insert s (u_2)) and the claim follows by the IH for u_2.

□

These two lemmas give us the following property:

Lemma 4.12. $\forall s\, t.\, s \rhd t \implies s = t$.

Proof. By induction on s $\rhd$t. The base cases are handled by the two lemmas above, and the inductive cases follow by transitivity. □

Finally, we can prove our goal:

Lemma 4.13. $\forall s\, t. s \equiv_\rhd t \implies sort\ s = sort\ t$

Proof. By induction on s $\equiv_\rhd$ t. The reflexivity case is trivial. Otherwise, w.l.o.g., s $\rhd$t and t $\equiv_\rhd$ u. Then sort t = sort u by IH and sort s = sort t follows due to the previous lemma. □

Since we can decide if s = t for any s,t : $\mathbb{T}$, we can decide whether s $\equiv_\rhd$ t by checking if sort s = sort t.
Note that this also gives us a simple proof for the idempotency of sort:

Lemma 4.14. $\forall s$. sort (sort s) = sort s.

Proof. It suffices to show that sort s $\equiv_\rhd$ s, which follows from the symmetry of $\equiv_\rhd$ and the fact that s $\equiv_\rhd$ sort s. □

5 Elements and subsets

As mentioned before, set equivalence is usually defined via an element relation. In our case, however, it is more convenient to take the opposite route and start from an equivalence relation, in our case $\equiv_\triangleright$.

Definition 5.1. $\dot\in : \mathbb{T} \to \mathbb{T} \to$ Prop
s $\dot\in$ t := t $\equiv_\triangleright$ s.t

The definition of s $\dot\in$ t states that s is an element of t if adding s to t does not change its equivalence. In set terms, if M = {N} ∪ M, we can see that N has to be an element of M.
We can use the same idea to define a subset relation. While s.t adds s to t's children, we can use s@t to add all elements of $\mathcal{L}(s)$ to t's children.

Definition 5.2. $\dot\subseteq$: $Tree \to \mathbb{T} \to Prop$
s $\dot\subseteq$ t := t $\equiv_\triangleright$ s@t.

In this section, we will see alternative characterizations of $\dot\in$ and $\dot\subseteq$ and prove that $s \,\dot\subseteq\, t \implies t \,\dot\subseteq\, s \implies s \equiv_\triangleright t$. This so-called double inclusion principle is commonly used in mathematics to prove that two sets are equal.

5.1 Element Characterisation

One property that immediately follows from its definition is that $\dot\in$ is invariant under tree equivalence:

- $\forall s\, s'\, t.\, s \,\dot\in\, t \implies s' \equiv_\triangleright s \implies s' \,\dot\in\, t$
- $\forall s\, t\, t'.\, s \,\dot\in\, t \implies t \equiv_\triangleright t' \implies s \,\dot\in\, t'$

Recall that the $\mathcal{L}$ function gives us a list of all child trees, which can be considered its elements (modulo tree equivalence).

Lemma 5.3. $\forall s\, t.\, s \in \mathcal{L}(t) \implies s \,\dot\in\, t.$

Proof. By induction on t. The base case is trivial.
In the inductive case, t = $t_1.t_2$ and $s = t_1 \vee s \in \mathcal{L}(t)_2$.
If $s = t_1$, then s $\dot\in$ $s.t_2$ due to the Deletion rule.
If $s \in \mathcal{L}(t)_2$, by IH for t_2, we know that s $\dot\in$ t_2, i.e. $t_2 \equiv_\triangleright s.t_2$.
Hence, $t_1.t_2 \equiv_\triangleright t_1.s.t_2 \equiv_\triangleright s.t_1.t_2$. □

Lemma 5.4. $\forall s\, t.\, s \,\dot\in\, t \iff \exists t' \in \mathcal{L}(t)\,.\, s \equiv_\triangleright t'.$

Proof.
"⇒": Let s $\dot\in$ t. We know that t $\equiv_\triangleright$ s.t. Hence, lemma 3.12 gives us that there has to be $t' \in \mathcal{L}(t)$ such that $s \equiv_\triangleright t'$.
"⇐": This time, the proof goes by induction on $s \equiv_\triangleright t'$.

- If s = t', we know that s $\dot\in$ t by the previous lemma.
- Let s $\triangleright$ s', s' $\equiv_\triangleright$ t' and t' $\in \mathcal{L}(t)$. By IH, we know that s' $\dot\in$ t, i.e. t $\equiv_\triangleright$ s'.t.
  Therefore, t $\equiv_\triangleright$ s.t follows, since s $\equiv_\triangleright$ s' and $\dot\in$ is invariant under $\equiv_\triangleright$.
- The last case is analogous to the second case.

□

5.2 Subset Characterisation

The definition of subset for sets is $N \subseteq M := \forall x \in N.\, x \in M$.
We can also characterize $\dot\subseteq$ this way:

Lemma 5.5. $\forall s\, t.\, s \dot\subseteq t \iff \forall s' \dot\in s.\, s' \dot\in t$.

Proof.
"$\Rightarrow$:" Let s' $\dot\in$ s. We know that t $\equiv_\triangleright$ s@t. Hence, by lemma 3.5, t $\precsim$ s@t and s@t $\precsim$ t.
Since s' $\dot\in$ s, $\exists x \in \mathcal{L}(s)\,.s' \equiv_\triangleright x$ follows by lemma 5.4. We want to show s' $\dot\in$ t, so it suffices to show $\exists y \in \mathcal{L}(t)\,.s' \equiv_\triangleright y$. Since s@t $\precsim$ t, and x $\in \mathcal{L}(s)$, hence $x \in \mathcal{L}(s@t)$, the claim follows.
"$\Leftarrow$:" Induction on s. The base case is trivial.
In the inductive case, we want to show that t $\equiv_\triangleright$ s_1.(s_2@t), assuming that $\forall x.x \dot\in s_1.s_2 \implies x \dot\in t$.
First of all, note that s_2 $\dot\subseteq$ t, due to IH for s_2: If we have s' $\dot\in s_2$, then by lemma 5.4, we know that $\exists x \in \mathcal{L}(s_2)\,.s' \equiv_\triangleright x$, hence also s' $\dot\in s_1.s_2$, when our assumption implies that s' $\dot\in$ t.
Furthermore, it is easy to see that s_1 $\dot\in$ t, since $s_1 \in \mathcal{L}(s_1.s_2)$.
Due to transitivity of $\equiv_\triangleright$, in order to show t $\equiv_\triangleright$ s_1.(s_2@t), it suffices to prove t $\equiv_\triangleright$ s_1.t, which is equivalent to s_1 $\dot\in$ t. □

Similarly to $\dot\in$, $\dot\subseteq$ is also invariant under tree equivalence:

Lemma 5.6. $\forall s\, s'\, t.\, s \dot\subseteq t \implies s \equiv_\triangleright s' \implies s' \dot\subseteq t$

Proof. By lemma 3.12, we know that s $\precsim$ s' and s' $\precsim$ s. Let x $\dot\in$ s'. It suffices to show x $\dot\in$ t. Since s' $\precsim$ s and due to lemma 5.4, we know that x $\dot\in$ s, which gives us x $\dot\in$ t. □

Lemma 5.7. $\forall s\, t\, t'.\, s \dot\subseteq t \implies t \equiv_\triangleright t' \implies s \dot\subseteq t'$

Proof. Analogously to the proof above. □

As mentioned, before, the double-inclusion principle is a very common way to prove set equality in classic mathematics and is obviously based on the axiom of extensionality:
$\forall N\, M.\, N = M \iff N \subseteq M \wedge M \subseteq N$.
We will prove that $\equiv_\triangleright$ and $\dot\subseteq$ also behave this way. The direction from left to right is obvious, while the proof of the other direction is easy using the following lemma:

Lemma 5.8. $\forall s\, t.\, s@t \equiv_\triangleright t@s$.

Proof. By nested induction on s and t. The cases where s = $\dot\emptyset$ or t = $\dot\emptyset$ are trivial.
We want to show $s_1.(s_2@(t_1.t_2)) \equiv_\triangleright t_1.(t_2@(s_1.s_2))$.
Note that, by IH for t_2, $t_1.(t_2@(s_1.s_2))) \equiv_\triangleright t_1.s_1.(s_2@t_2)$ and $t_1.s_1.(s_2@t_2) \equiv_\triangleright t_1.s_1.(t_2@s_2)$ by IH for s_2.
Similarly, $s_1.(s_2@(t_1.t_2)) \equiv_\triangleright s_1.t_1.(t_2@s_2)$.
Due to transitivity, it suffices to show $s_1.t_1.(t_2@s_2) \equiv_\triangleright t_1.s_1.(t_2@s_2)$, which follows due to the Swap rule. □

Using this lemma, it is easy to prove double-inclusion:

Lemma 5.9. $\forall s\, t.s \dot\subseteq t \implies t \dot\subseteq s \implies s \equiv_\triangleright t$.

Proof. Let s $\dot\subseteq$ t and t $\dot\subseteq$ s. We know that s $\equiv_\triangleright$ t@s and t $\equiv_\triangleright$ s@t. Due to transitivity, it suffices to show that t@s $\equiv_\triangleright$ s@t, which follows by the previous lemma. □

With double-inclusion established, we can finally prove the other direction of lemma 3.5.

Lemma 5.10. $\forall s\, t.\, s \precsim t \implies s \dot\subseteq t$

Proof. Let s $\precsim$ t and u $\dot\in$ s. We show that u $\dot\in$ t. Note that lemma 5.4 gives us x $\in \mathcal{L}(s)$ such that x $\equiv_{\triangleright}$ u. since s $\precsim$ t, this means that there is y $\in \mathcal{L}(t)$ such that y $\equiv_{\triangleright}$ u, and the claim follows due to transitivity and by lemam 5.4. □

Lemma 5.11. $\forall s\, t.\, s \precsim t \implies t \precsim s \implies s \equiv t.$

Proof. Since s $\precsim$ t and t $\precsim$ s, we have s $\dot\subseteq$ t and t $\dot\subseteq$ s by the previous lemma, and s $\equiv_{\triangleright}$ t follows by the double inclusion principle. □

6 Bisimulation

So far, the equivalence relations we have looked at were based on the Deletion and Swap rules. Another possibility to define tree equivalence is bisimulation. The standard mathematical way to do this is by defining when a relation is a bisimulation. Two trees are then bisimilar if there is a bisimulation between them. We will take this approach in the second part of the thesis when considering non-well-founded sets. Another way of defining bisimulation, that is more natural for constructive type theory, is giving a coinductive definition:

Definition 6.1. $\approx : \mathbb{T} \to \mathbb{T} \to Prop$

$$\frac{\forall s' \in \mathcal{L}(s)\, \exists t' \in \mathcal{L}(t)\,.\, s' \approx t' \qquad \forall t' \in \mathcal{L}(t)\, \exists s' \in \mathcal{L}(s)\,.\, t' \approx s'}{s \approx t}$$

We use a double line in the inference rule to denote a coinductive definition. One way to think of coinductive proofs as opposed to inductive proofs is that coinductive proof trees may be infinite, whereas inductive proof trees have to be finite. However, our binary trees and all descending chains of $\mathcal{L}$-steps are also finite, which means that in the end, it does not make a difference whether the definition of $\approx$ on binary trees uses an inductive or coinductive rule. In this section, we will prove the equivalence of $\approx$ and $\equiv$.
In Coq, the definition uses "CoInductive" instead of "Inductive". The tactic "cofix" is the coinductive equivalent of "fix". One major difference between using induction and coinduction in Coq is that, while Coq generates induction lemmas automatically, it does not do so for coinductive definitions, which means any necessary coinduction principles have to be proven by hand, unless one decides to use third-party libraries.
We can easily prove that $\approx$ is reflexive and transitive, and that it admits both Deletion and Swap using induction on trees.
For transitivity, we need a coinduction lemma which basically relates our coinductive definition to the usual mathematical definition of bisimulation, namely that two trees are bisimilar if there exists a bisimulation between them:

Lemma 6.2. Given any relation R : Tree → Tree → Prop),
(∀ s t, R s t $\implies$
  (∀ s' ∈ L(s) ∃ t' ∈ L(t). R s' t' ∧
  (∀ t' ∈ L(t) ∃ s' ∈ L(s). R t' s'.)) $\implies$
∀ s t, R s t → s ≈ t.

Proof. The proof basically just constructs a recursive function. In Coq, instead of the usual "fix", "cofix" is required here.
The basic idea is that given R s t, we can easily show $\forall s' \in \mathcal{L}(s)\,\exists t' \in \mathcal{L}(t)\,.\, s' \approx t'$ (and $\forall t' \in \mathcal{L}(t)\,\exists s' \in \mathcal{L}(s)\,.\, t' \approx s'$ is analogous): due to the premise, R s t implies that ∀ s' ∈ L(s) ∃ t' ∈ L(t). R s' t', and by recursion, s' ≈ t'. ☐

Transitivity follows by using lemma 6.2 with R := (≈ ∘ ≈).
We have shown that all rules of ≡ are admissible for ≈, hence $\forall s\, t.\, s \equiv t \implies s \approx t$.
In order to show the other direction, we need a stronger induction lemma:

Lemma 6.3. $(\forall s.\,(\forall t \in \mathcal{L}(s)\,.\, P\,t) \implies P\,s) \implies \forall s.\, P\,s.$

Proof. Note that if $(\forall s.\,(\forall t \in \mathcal{L}(s)\,.\, P\,t) \implies P\,s)$, then $\forall v\, u.\, u \in \mathcal{L}(v) \implies Pu$. We can prove this by induction on v.
The base case is trivial, since u ∈ $\mathcal{L}(\dot{\emptyset})$ is a contradiction.
In the inductive case, v = $v_1.v_2$. Let $u \in \mathcal{L}(v_1.v_2)$, i.e. u = $v_1 \vee u \in \mathcal{L}(v_2)$.
If u = v_1, then the IH for v_1 gives us that $(\forall x \in \mathcal{L}(u)\,.\, P\,x)$, hence P u by our assumption.
Otherwise, u ∈ $\mathcal{L}(v_2)$, and P u follows due to IH for v_2.

The original claim now follows, since s ∈ $\mathcal{L}(s.s)$. ☐

In order to show $s \equiv t$, it suffices to show s $\dot{\subseteq}$ t ∧ t $\dot{\subseteq}$ s. Hence the following lemma completes the equivalence proof:

Lemma 6.4. $\forall s\, t.\, s \approx t \implies s \dot{\subseteq} t \wedge t \dot{\subseteq} s.$

Proof. By induction on s (using the stronger induction principle 6.3). Let s ≈ t. We show that s $\dot{\subseteq}$ t (t $\dot{\subseteq}$ s is analogous). Let x $\dot{\in}$ s. It suffices to show that x $\dot{\in}$ t. Since x $\dot{\in}$ s, we know that $\exists s' \in \mathcal{L}(s)\,.\, x \equiv_{\triangleright} s'$. Note that, because s ≈ t, we know that $\exists t' \in \mathcal{L}(t)\,.\, s' \approx t'$. By IH, s' $\dot{\subseteq}$ t' and t' $\dot{\subseteq}$ s', hence s ≡ t and the claim follows due to transitivity. ☐

Part II
Finitary sets

7 Basics

Non well-founded sets are sets that, in contrast to ZF sets, can have an infinite descending chain of elements. The smallest example of such a set is called Omega : Ω = {Ω}.
Peter Aczel defines non well-founded sets as sets that can be depicted by an accessible pointed graph (apg). Apgs are basically rooted, connected graphs.
For example, Omega can be represented apg in figure 2:

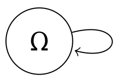

Figure 2: Ω

We use a very similar notion of graphs, but will not require connectivity. This makes our definitions and proofs simpler, which is especially noticeable in the Coq development. However, unreachable vertices are irrelevant for our representation. We represent finitary sets as rooted graphs up to bisimulation.

Definition 7.1. A graph is a 4-tuple (X, edgeRel, dom, root), where

- X is a type with decidable equality
- edgeRel : $X \to X \to \mathbb{B}$ is the transition relation
- dom : [X] is the domain of the graph
- root denotes the root of the graph

We use $\mathbb{G}$ to denote the type of graphs.

For any graph g, we use the notation root g to denote the root of g, etc. In Coq, this is accomplished using a record for the definition of graphs. We need the decidable equality on X in Coq, but for the sake of clarity, we neglect it in this text.

We also use the notation x : g as a shortcut for x : t g. While this is an intuitive notation, it only gives us information about the type of x and does not imply that x is also in the domain in g. The domain of the graph is the set of all its vertices. We always consider only the subgraph induced by the domain, which means we will usually use the following function instead of edgeRel.

Definition 7.2. E : $g \to g \to \mathbb{B}$
E x y := $\begin{cases} edgeRel\ x\, y & x \in dom(g) \land y \in dom(g) \\ false & otherwise \end{cases}$

We construct graphs as G edgeRel dom root, and leave the type of the graph, as well as the decider for equality on its type, implicit.
In the next section, will first introduce the notion of bisimulation for our representation of graphs and prove its decidability.
Then, we will define a membership relation on graphs and prove that our model supports all ZF axioms except for infinity and regularity modulo bisimulation. As mentioned in the introduction, we want to use graphs to model finitary sets, the axioms of regularity and infinity do not make sense for our purposes.
We will also show how the transitive closure of a graph can be computed, since the usual ZF construction for the transitive closure of a set requires the axiom of infinity.
Thereafter, we will actually construct a type for non well-founded sets, namely the quotient type $\mathbb{G}/_{\approx}$. Every non-well founded set is represented by a graph such that two non well-founded sets are equal iff the underlying graph is the same. We achieve this by giving a mapping between graphs and natural numbers and show that for any type X with a decidable equivalence relation R and a suitable mapping X $\leftrightarrow \mathbb{N}$, we can construct $^{X}/_{R}$. Note that the existence of such a quotient type is not clear in general, from the perspective of constructive type theory.
Finally, we show how to lift the definition of equality, the element relation and the constructions for the ZF axioms we are interested in to this type.

8 Bisimulation

Definition 8.1. We call a relation p : $g_1 \rightarrow g_2 \rightarrow \mathbb{B}$ a bisimulation for two graphs g_1, g_2 iff
$\forall x\, y.\, p\, x\, y = true \implies$
$\quad (\forall x'.\, E\, x\, x' = true \implies$
$\quad\quad \exists y'.\, E\, y\, y' = true \wedge p\, x'\, y' = true) \wedge$
$\quad (\forall y'.\, E\, y\, y' = true \implies$
$\quad\quad \exists x'.\, E\, x\, x' = true \wedge p\, x'\, y = true)$.
Two graphs g_1 and g_2 are bisimilar ($g_1 \approx g_2$) :=
$\quad \exists p : g_1 \rightarrow g_2 \rightarrow \mathbb{B}.\, bisim\, p \wedge p\, (root\, g_1)\, (root\, g_2) = true$

We use the notation dom' g := (root g) :: dom g. Though it is not often needed, $x \in dom'\, g$ is much more readable than $x = root\, g \vee x \in dom\, g$.
It is easy to see that our bisimulation is an equivalence relation:

- Reflexivity : $\forall g.\, g \approx g$
  - Consider p := $\lambda x\, y.\, x = y$.
- Symmetry : $\forall g_1\, g_2.\, g_1 \approx g_2 \implies g_2 \approx g_1$
  - Given a bisimulation r : $g_1 \rightarrow g_2 \rightarrow \mathbb{B}$
  - Consider p : $g_2 \rightarrow g_1 \rightarrow \mathbb{B} := \lambda y\, x.\, rxy$
- Transitivity : $\forall g_1\, g_2\, g_3.\, g_1 \approx g_2 \implies g_2 \approx g_3 \implies g_1 \approx g_3$
  - Given two bisimulations r : $g_1 \rightarrow g_2 \rightarrow \mathbb{B}$, q : $g_2 \rightarrow g_3 \rightarrow \mathbb{B}$
  - Consider p : $g_1 \rightarrow g_3 \rightarrow \mathbb{B}$:=
    $\lambda\, x\, z.\, \exists y.\, (y \in dom'\, g_2) \wedge r\, x\, y = true \wedge q\, y\, z = true$
- In each case, it is easy to verify that the given p actually is a bisimulation.

We will first show how to prove whether a given p : $g_1 \rightarrow g_2 \rightarrow \mathbb{B}$ is a bisimulation. Afterwards, we will show that for any two graphs g_1, g_2 we can decide whether $g_1 \approx g_2$ by trying all possible relations p : $g_1 \rightarrow g_2 \rightarrow \mathbb{B}$.

It is easy to see that for any p : $g_1 \rightarrow g_2 \rightarrow \mathbb{B}$ with
$\quad \forall x\, y. P\, x\, y = true \implies x \in dom' g_1 \wedge y \in dom' g_2$
it suffices to only consider x, x', y and y' that are in the respective domain, i.e.
p is a bisimulation for g_1, $g_2 \iff$
$\forall x \in dom'\, g_1. \forall y \in dom' g_2.\, p\, x\, y = true \implies$
$\quad (\forall x' \in dom' g_1.\, E\, x\, x' = true \implies$
$\quad\quad \exists y' \in dom'\, g_2.\, E\, y\, y' = true \wedge p\, x'\, y' = true) \wedge$
$\quad (\forall y' \in dom' g_2.\, E\, y\, y' = true \implies$
$\quad\quad \exists x' \in dom' g_1.\, E\, x\, x' = true \wedge p\, x'\, y = true)$.
This characterization makes the decidability proof much easier : If a given p behaves this way, then it is trivially decidable whether p is a bisimulation or not (in fact, this is so easy that Coq can automatically derive this proof), since at each point, we only have to consider a finite amount of vertices. Note that the root of a graph g need not be contained in its domain. Hence, we need to explicitly check for the root when we want to limit our relation to the "relevant" elements, which is we we use dom' in the characterization above and in the definition below.

We will see later on that any graph whose root is not contained in its domain represents the empty set.

Definition 8.2. $p\big|_{dom'} : \quad g_1 \to g_2 \to \mathbb{B}$
$p\big|_{dom'}$ x y := $\begin{cases} edgeRel\ x\, y & x \in dom'\, g_1 \wedge y \in dom'\, g_2 \\ false & otherwise \end{cases}$

We can use $p\big|_{dom'}$ to restrict p to the "relevant" elements. Note that for any bisimulation p, $p\big|_{dom'}$ is also a bisimulation.

Definition 8.3. Let X, Y be types and xs : [X], ys : [Y].
allFuns xs ys := $map(\lambda A.\lambda x\, y.(x, y) \in A)(\mathcal{P}(xs \times ys)$

Note that the $\mathcal{P}$ in the definition denotes the power list, not the power set, which is basically the same construction, but the order matters.
We can prove that $f \in allFuns\ xs\, ys$ for any f with $(\forall x\, y.\, f\, x\, y = true \implies x \in xs \wedge y \in ys)$, provided that equality on X and Y is decidable.
This assumes an axiom called functional extensionality, i.e. $\forall f\, g.\, f = g \iff \forall x.\, fx = fy$, which is not built into Coq. If we do not want to assume this axiom, we can proof a slightly weaker version, namely $\exists f' \in allFuns\, xs\, ys.\, \forall x\, y.\, f\, x\, y = f'\, x\, y$, but we feel that functional extensionality is a very intuitive notion, and is in fact the usual definition of function equality in classical mathematics.
To prove this claim, we need the additional lemma $\forall xs\, ys.\, ys \subseteq xs \implies \exists zs \in \mathcal{P}xs \wedge zs \equiv ys$. This is intuitive, since it also holds for power sets, and can be proven by induction on xs.
This implies that for any p, $p\big|_{dom'} \in allFuns\ (dom'\, g_1)\, (dom'\, g_2)$.

Theorem 8.4. $\forall g_1\, g_2, dec(g_1 \approx g_2)$

Proof. We know that for any bisimulation p : $\quad g_1 \to g_2 \to \mathbb{B}$,
$p\big|_{dom'} \in allFuns\ (dom'\, g_1)\, (dom'\, g_2)$. Since any function in $allFuns\, (dom'\, g_1)\, (dom'\, g_2)$ can only ever return true for arguments in dom' g_1 and dom' g_2, we can easily decide if $p\big|_{dom'}$ is a bisimulation.
It suffices to decide if there is a bisimulation p in $allFuns\, (dom'\, g_1)\, (dom'\, g_2)$ such that $p\, (rootg_1)\, (rootg_2) = true$. We can obviously decide that, since we only have finitely many functions to consider. Assume there is such a bisimulation in $allFuns\, (dom'\, g_1)\, (dom'\, g_2)$. Then we have a bisimulation for g_1, g_2 and are done.
Suppose there is no such bisimulation in $allFuns\, (dom'\, g_1)\, (dom'\, g_2)$. Then $g_1 \not\approx g_2$: Assume otherwise. Then we have a bisimulation p : $g_1 \to g_2 \to \mathbb{B}$. However, then $p\big|_{dom'} \in allFuns\, (dom'\, g_1)\, (dom'\, g_2)$ is a bisimulation ↯.

□

9 ZF Axioms

In this section, define an element relation based on the bisimulation of the previous section and give constructions for all ZF axioms except Regularity and Infinity.

Definition 9.1. subgraph (x : g) := G (edgeRel g) (dom g) x.

The subgraph for any given node x in a graph g can be obtained by just replacing the root with the node x. Note that x is not required to be reachable from the root. This makes the definitions in Coq easier, but we will never use subgraph on an unreachable node.

Definition 9.2. child_nodes, children
child_nodes g := filter (fun x ⇒ E (root g) x = true) (dom g)
children g := map subgraph child_nodes

The child_nodes of a graph g are all nodes reachable from the root in one step and the children are the respective subgraphs. Now we can define our element relation for graphs:

Definition 9.3. $g_1 \dot{\in} g_2 := \exists g_3 \in children\, g_2 .\, g_1 \approx g_3$.

Needless to say, our element relation is of course decidable and Coq can derive this automatically. Consider the definition inclusion and equivalence based on $\dot{\in}$:

Definition 9.4. $g_1 \dot{\subseteq} g_2 := \forall g .\, g \dot{\in} g_1 .implies, g \dot{\in} g_2$.

This relation is also decidable : Note that if we have some $g \dot{\in} g_1$, then there is a $g' \in children\, g_1$ such that $g \approx g'$, by the definition of $\dot{\in}$. Since the subgraph for g' is trivially an element of g_1, and due to the transitivity of $\approx$, it suffices to show that $\forall g \in children g_1 .\, g \dot{\in} g_2$ is decidable, which is easy, since there are only finitely many children.

Definition 9.5. $g_1 \equiv g_2 := g_1 \dot{\subseteq} g_2 \wedge g_2 \dot{\subseteq} g_1$

Two graphs g_1, g_2 are equivalent, i.e. $g_1 \equiv g_2$, iff they have the same elements. We will now give a proof of the first ZF axiom (modulo bisimulation) for graphs: extensionality.

Theorem 9.6. *Extensionality*
$\forall g_1\, g_2 .\, g_1 \approx g_2 \iff g_1 \equiv g_2$.

Proof.
"⇒" Let $g_1 \approx g_2$. Due to symmetry of $\approx$, it suffices to show $g_1 \dot{\subseteq} g_2$. Let $g \dot{\in} g_1$. By the definition of $\dot{\in}$, we have a vertex $x \in dom\, g_1$ such that $E(root\, g_1)x = true \wedge g \approx subgraph\, x$.
Because $g_1 \approx g_2$ and $E(root\, g_1)x = true$, we also have a vertex $y \in dom\, g_2$ such that
$E\,(root\, g_2)\, y = true \wedge p\, x\, y = true$, where p is the witness of $g_1 \approx g_2$.
Obviously, $subgraph\, y \in children\, g_2$. It remains to be shown that $g \approx subgraph\, y$. Since $g \approx subgraph\, x$ and due to the transitivity of $\approx$, it suffices to show that $subgraph\, x \approx subgraph\, y$.
It can easily be shown that the relation p is also a bisimulation for subgraph x and subgraph y.
"⇐" Let $g_1 \equiv g_2$ and p := $\lambda x\, y .\, subgraph\, x \equiv subgraph\, y$.
Obviously, p (root g_1) (root g_2) = true.
Consider $x, x' \in dom\, g_1$ such that $E\, x\, x' = true$ and $y \in dom\, g_2$.
It is clear that $subgraph\, x' \dot{\in} subgraph\, x$.
Since $subgraph\, x \equiv subgraph\, y$, there is some $y' \in dom\, g_2$ such that $E\, y\, y' = true \wedge subgraph\, x' \approx subgraph\, y'$.
Due to the direction already proven, we know that $subgraph\, x' \equiv subgraph\, y'$. □

Next, we will see how we can actually represent sets as graphs.

9.1 Empty

Definition 9.7. $\dot{\emptyset} := G(\lambda x\, y .\, false)[tt]tt$.

The empty set is represented by a graph with no transitions where Unit := tt is a type with exactly one element. Note that it is bisimilar to any graph whose root is not in its domain. Later on, we will see that there is a reason why we chose this definition instead of using an empty domain.
Note that has the same characterization as the empty set : $\nexists g.\, g \mathrel{\dot\in} \dot\emptyset$, which means that it represents our construction for the axiom of existence.

9.2 Adjunction

Next, we will give a construction for adjunction:

Definition 9.8. Adjunction
$g_1; g_2$:= G f ($\{None\} \cup \{Some\,(inl\,x) \mid x \in dom\,g_1\} \cup \{Some\,(inr\,y) \mid y \in dom\,g_2\}$) None, where f is defined as follows:
f None (Some (inl (root g_1)):= true
f None (Some (inr y)) := E (root g_2) y
f (Some (inl x)) (Some (inl y)) := E x y
f (Some(inr x)) (Some (inr y)) := E x y
f _ _ := false

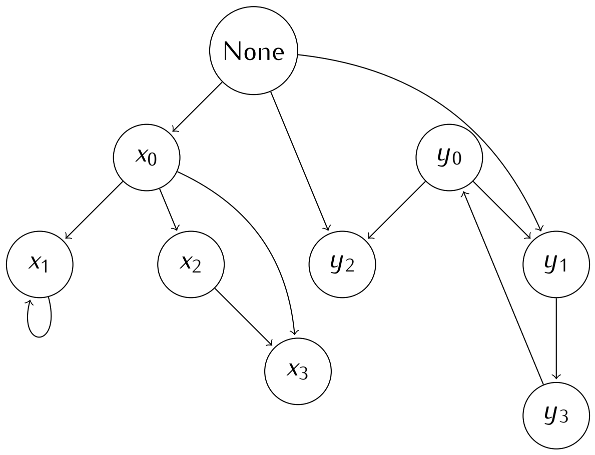

Figure 3: Graph for $g_1; g_2$

Note that in the definition of adjunction, we use the more readable set notation for replacement and binary union to define the domain of the new graph; these of course denote the respective map and append operations on lists.
Figure 3 shows an example for adjunction. The idea is to construct a new graph g whose children are the root of g_1 and all children of g_2. Note that the type of this new graph is option ($g_1 + g_2$). This gives us an additional vertex (None), which we use as the new root. Furthermore, we make sure that there is no edge from any vertex to the root or between vertices from the left and right part of the graph (i.e. those corresponding to g_1 and g_2, respectively).
It is easy to see that $\forall x\, y : g_1$. E (Some (inl x)) (Some (inl y) = true $\iff$ E x y = true. Analogously, $\forall x\, y : g_2$. E (Some (inr x)) (Some (inr y) = true $\iff$ E x y = true.
Since we have the same structure in the left part of the graph as in g_1, it is easy to see that for any x : g_1, subgraph x $\approx$ subgraph (Some (inl x)). Likewise, for any y : g_2, we have

subgraph y $\approx$ subgraph (Some (inr y)).

The characteristic property of Adjunction is : $\forall g.\, g \mathrel{\dot\in} g_1; g_2 \iff g \approx g_1 \vee g \mathrel{\dot\in} g_2$. It is easy to see that our construction fulfils this property:

Proof. If $g \mathrel{\dot\in} g_1; g_2$, then there is a vertex (Some z) such that E None (Some z) = true. In the case where z = Some (inl x), we know that subgraph (Some (inl x)) $\approx$ (subgraph x).
Furthermore, we know that in this case, x must be the root of g_1. By transitivity of $\approx$, we know that $g \approx g_1$.
In the other case, z = Some (inr y), we know that subgraph (Some (inr y)) $\approx$ subgraph y. Furthermore, we know that E (root g_2) y = true, hence $g \mathrel{\dot\in} g_2$.

If $g \approx g_1 \vee g \mathrel{\dot\in} g_2$, by the same reasoning as above, we know that $g \mathrel{\dot\in} g_1; g_2$.
□

We already know how to get the list of child graphs for any given graphs. Adjunction allows us to define the reverse operation : To build a graph that contains all graphs in a given list as its children.

Definition 9.9.
$\mathcal{L}\,[] = \dot\emptyset$
$\mathcal{L}(x :: xs)$ = x; $\mathcal{L}$ xs

It is easy to show that $\forall xs\, g.\;\; g \mathrel{\dot\in} \mathcal{L}\, xs \iff \exists g' \in xs.\, g \approx g'$.
Due to extensionality and this property of $\mathcal{L}(,)$ it is easy to show that $\forall g.\, g \approx \mathcal{L}(children\, g)$.
Using $\mathcal{L}$ and children, we can define the remaining ZF axioms easily by transforming our graphs to lists of graphs, transforming this list according to the requirements of the axiom at hand, and finally transforming the resulting list back to a graph.

9.3 Pairing

The axiom of pairing has the characteristic property $\forall g.\, g \mathrel{\dot\in} \{g_1, g_2\} \iff g \approx g_1 \vee g \approx g2$.

Definition 9.10.
$\{g_1, g_2\} := \mathcal{L}[g_1, g_2]$

The characteristic property follows directly from the properties of $\mathcal{L}$. Using pairing, we can define singleton in the usual way:

Definition 9.11. Singleton $\{g\} := \{g, g\}$.

Of course, $\forall g'.\, g' \mathrel{\dot\in} \{g\} \iff g \approx g'$ follows immediately from the characteristic property of UPair.

9.4 Union

The next ZF axiom we will consider is union.
Its characteristic property is: $\forall g_1\, g_2.\, g_1 \mathrel{\dot\in} \bigcup g_2 \iff \exists g \mathrel{\dot\in} g_2.\, g_1 \mathrel{\dot\in} g$. We will again do the main work on lists of graph.

Definition 9.12.
$\bigcup g := \mathcal{L}(flatten(map\, children(children\, g)))$
or, written in set notation:
$\bigcup g := \{g'' \mid \exists g' \in children\, g.\, g'' \in children\, g'\}$

The function flatten used in the definition flattens a list of lists :

Definition 9.13.
flatten [] = []
flatten (x :: xs) = x ++ flatten xs

Using induction, we can easily prove the following: $\forall xs, x \in flatten\, xs \iff \exists ys \in xs.\, x \in ys$. The characteristic property of Union follows from this property and the properties of $\mathcal{L}$ that we have already seen.

9.5 Separation

The next ZF axiom we will consider is separation. Its characteristic property is
$g_1 \dot\in \{g \dot\in g_2 | P\, g\} \iff \exists g \dot\in g_2 . P\, g \wedge g_1 \approx g$.

Definition 9.14. $\{g' \dot\in g | P\, g'\} := \mathcal{L}(filter P(children M))$

In order for this to work, there are two requirements for P :

- P needs to be decidable. This is required by filter.
- P needs to be extensional, that is $\forall g_1 \approx g_2.\, P\, g_1 \iff P\, g_2$. The reason for this is the following : If we have to graphs $g_1 \approx g_2$, then $\{g' \dot\in g_1 | P\, g'\}$ should be bisimilar to $\{g' \dot\in g_2 | P\, g'\}$. Suppose g_1 and g_2 are singletons with elements g'_1 and g'_2, respectively. If we have a given P that is not extensional and, say P g'_1, but $\neq$ P g'_2, then $\{g' \dot\in g_1 | P\, g'\} \approx \{g'_1\} \not\approx \emptyset \approx \{g' \dot\in g_2 | P\, g'\}$. We can avoid such cases by requiring that P be extensional.

9.6 Replacement

The next axiom, replacement, is similar to separation:
Its characteristic property is $g_1 \dot\in \{f\, g | g \dot\in g_2\} \iff \exists g \dot\in g_2 . g_1 \approx f\, g$ where f : $Graph \rightarrow Graph$ is an extensional function.
We can define Replacement as follows:

Definition 9.15. $\{f\, g | g \dot\in g'\} := \mathcal{L}(map\, f\, (children\, g'))$

Note that f is required to be extensional, i.e. $\forall g_1\, g_2.\, g_1 \approx g_2 \implies f\, g_1 \approx f\, g_2$, due to the same reasons why we require extensionality for Separation.

9.7 Power

The last ZF axiom we consider is power. In set theory, the power axiom gives us the existence of a power set for any given set. Its characteristic property is :
$g_1 \dot\in \mathcal{P}(g_2) \iff g_1 \dot\subseteq g_2$
We can use the power function on lists to do the hard work :

Definition 9.16. $\mathcal{P}(g) := \mathcal{L}(map\, \mathcal{L}(\mathcal{P}(children\, g)))$

The proof is not hard and requires only the following lemma in addition to what we have already seen: $filter\, P\, xs \in \mathcal{P}(xs)$, where xs is any list over any type X and p : $X \Rightarrow Prop$ is any decidable proposition.
The proof of this lemma is a simple induction on xs. This works because the elements of the lists in $\mathcal{P}(xs)$ appear in the same order as they do in xs.

10 Transitive closure

As stated in the introduction, the usual construction of the transitive closure of a set in ZF depends on the axiom of infinity. In this section, we will give a construction for the transitive closure of a graph.

Definition 10.1. transitive (g) := $\forall g'.\, g' \dot{\in} g \implies g' \dot{\subseteq} g$.

The transitive closure of a set is basically the set of all its successors with respect to the element relation. In our case, this is basically the set of all subgraphs whose root is reachable from the root of the original graph. Therefore, we will first show the decidability of reachability in a graph and then give a construction of tc.

We call a vertex y in a graph g reachable from another vertex x if there is a path xs of vertices such that the first vertex is y, the last vertex is x and every vertex in xs is adjacent to its neighbours in xs.

Definition 10.2. xs : $x \rightarrow^* y$

$$\frac{x \in dom\, g}{[x] : x \rightarrow^* x} \qquad \frac{Exx' = true \qquad xs : x' \rightarrow^* y}{(x :: xs) : x \rightarrow^* y}$$

We use the notation $x \rightarrow^* y$ to denote that y is reachable from x, i.e. $x \rightarrow^* y := \exists xs : x \rightarrow^* y$, and $xs : x \rightarrow^* y$ to denote that xs is a path from x to y. Note that $\forall xs : x \rightarrow^* y.\, xs \subseteq dom\, g$.
We want to decide whether $x \rightarrow^* y$. The idea is that any path from x to y without loops has at most $|dom\, g| - 1$ edges (the corresponding list of vertices contains all vertices $v \in dom\, g$). Therefore, it is sufficient to decide whether x reaches y in less than $|dom\, g|$ steps.

Definition 10.3. reach_in : $g \rightarrow g \rightarrow \mathbb{N} \rightarrow \mathbb{B}$
reach_in x y 0 := $x \in dom\, g \land x = y$

reach_in x y (S n) := $\begin{cases} true & reach\_in\, x\, y\, n = true \\ \exists x' \in dom\, g.\, E\, x\, x' = true \land reach\_in\, x'\, y\, n = true & otherwise \end{cases}$

reach_in x y n decides if x reaches y in less than n steps, i.e. using a path with at most n vertices.
There are a few properties of reach_in and $xs : x \rightarrow^* y$ we will need that are easy to prove by induction :

- $reach\_in\, x\, y\, (S\, n) = true \implies x = y \lor \exists z.\, E\, x\, z = true \land reach\_in\, z\, y\, n = true$
- $reach\_in\, x\, y\, n = true \implies m > n \implies reach\_in\, x\, y\, m = true$
- $xs : x \rightarrow^* y \implies reach\_in\, x\, y\, |xs| = true$
- $reach\_in\, x\, y\, n = true \implies x \rightarrow^* y$

Consider the following function ρ which remove all loops from a path:

Definition 10.4. $\rho : [g] \to [g]$
$\rho\,[] = []$

$$\rho\,(x :: xs) = \begin{cases} remove\_until\, x\,(\rho xs) & x \in \rho xs \\ x :: (\rho xs) & otherwise \end{cases}$$

ρ uses the following function remove_until which removes all elements appearing before a given x from a list:

Definition 10.5. remove_until $g \to [g] \to [g]$
remove_until x [] = []
remove_until x (x :: xs) = x :: xs
remove_until x (y :: xs) = remove_until x xs

It is obvious that for any x and xs, |remove_until x xs| <= $|xs|$.
Note that remove_until preserves paths:

Lemma 10.6. $\forall z\,(xs : x \to^* y).\, z \in xs \implies (remove\_until\, z\, xs) : z \to^* y.$

Proof. Induction on $xs : x \to^* y$.

- In the base case, xs = [x]. Since $z \in xs$, we know that xs = [z].

  Therefore, remove_until xs = [z] : $z \to^* z$

- In the inductive case, xs = (x :: xs'), xs' : $x' \to^* y$ and E x x' = true.

  If x = z, then (z :: xs') is a path from z to y.

  Otherwise, remove_until z (x :: xs') = remove_until z xs' and we know that $z \in xs'$, since $z \in (x :: xs')$ and $z \neq x$. Therefore, remove_until z xs' is a path from z to y by IH.

□

Using the previous lemma, it is easy to show that ρ preserves paths, i.e.
$\forall xs : x \to^* y.\, \rho\, xs : x \to^* y$.
Furthermore, for any xs, $\rho\, xs$ does not contain duplicates. It is easy to see that for any duplicate-free $xs \subseteq ys$, $|xs| \leq |ys|$. These two properties, along with the simple fact that $\rho\, xs \subseteq xs$, imply that $|\rho\, xs| <= |dom\, g|$, since for any $xs : x \to^* y, xs \subseteq dom\, g$.

Theorem 10.7. $\forall x\, y.\, x \to^* y \iff reach\_in\, x\, y\, |dom\, g| = true$

Proof.
"$\Leftarrow$" : We have already seen that this holds for any n, not just for $|dom\, g|$.
"$\Rightarrow$" : Assume we have a path $xs : x \to^* y$ and that reach_in x y $|dom\, g|$ = false.
We know that $\rho\, xs : x \to^* y$ and $|\rho\, xs| \leq |dom\, g|$.
This means that reach_in x y $|\rho\, xs|$ = true,
which in turn implies that reach_in x y $|dom\, g|$ = true,
since reach_in x y remains true for larger step size. $\lightning$

□

The theorem gives us an easy way to decide whether $x \to^* y$. We use $x \to^+ y$ to denote that x reaches y using at least one edge.

Definition 10.8. $x \rightarrow^+ y := \exists x'. E\,x\,x' = true \wedge x' \rightarrow^* y$.

We will need this definition later on when defining the transitive closure of a graph. Furthermore, $x \rightarrow^+ y$ is obviously decidable and $x \rightarrow^+ y \iff \exists xs : x \rightarrow^* y. |xs| > 1$.
Consider the following inductive definition of $\dot\in^n$:

Definition 10.9.

$$\frac{\text{g1} \approx \text{g2}}{\text{g1} \;\dot\in^0\; \text{g2}} \qquad \frac{\text{g1} \;\dot\in \text{g2} \qquad \text{g2} \;\dot\in^n\; \text{g3}}{\text{g1} \;\dot\in^{Sn}\; \text{g3}}$$

We can see that for any graph g_1, g_2 and r, x: g_2,
$r \rightarrow^* x \implies g_1 \approx$ subgraph x $\implies \exists n. g1 \dot\in^n$ (sugraph r).
Likewise, $r \rightarrow^+ x \implies g_1 \approx$ subgraph x $\implies \exists n > 0. g1 \dot\in^n (sugraph\, r)$.
The first property is easy to prove by induction on the path, while the second one is even easier to prove by giving a path of length > 1. Note that, in the special case where r = root g2, subgraph g_2 r = g_2.

We can also proof the other direction, that is
$g_1 \dot\in^n g_2 \implies root\, g_2 \in dom\, g_2 \implies \exists x. root\, g2 \rightarrow^* x \wedge g_1 \approx$ subgraph x and
$n > 0 \implies g1 \dot\in^n g2 \implies \exists x.(root\, g2) \rightarrow^+ x \wedge g_1 \approx$ subgraph x, respectively. This time, the proof is of course by induction on n. The restriction $g_2 \in$ dom g_2 is necessary: Consider the case where dom g_2 = []. As we have seen before, this means that $g_2 \approx \emptyset$. However, [root g2] is not a valid path in this case. In the second case, no such restriction is required, since, as we can see, $g_1 \dot\in^n g_2 \implies n > 0 \implies root\, g2 \in dom\, g2$.
Next, we give the definition of the transitive closure:

Definition 10.10. tc : Graph → Graph
tc g := G f (None :: (map Some (dom g))) (None) where
f (Some x) (Some y) := edgeRel x y
f None (Some y) := $(root\, g) \rightarrow^+ y$.
f _ _ := false

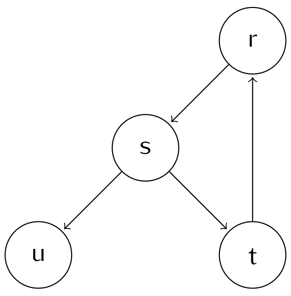

Figure 4: Example graph g

Consider the graph g shown in figure 4. Its transitive closure is shown in figure 5.
We will prove that $g_1 \dot\in tc\, g_2 \iff \exists n > 0. g_1 \dot\in^n g_2$.
The reason why we need $root\, g \rightarrow^+ y$ in the definition is because $x \in dom\, g \implies x \rightarrow^* x$ holds, as mentioned before. If we replaced $\rightarrow^*$ by $\rightarrow^+$ in the definition of tc, and $root\, g_2 \in dom\, g_2$, then $subgraph\,(root\, g_2) = g_2 \dot\in tc\, g_2$. However, if there is no other vertex v : g_2 that reaches the root again, $g_2 \dot\notin^n g_2$ for any $n > 0$.

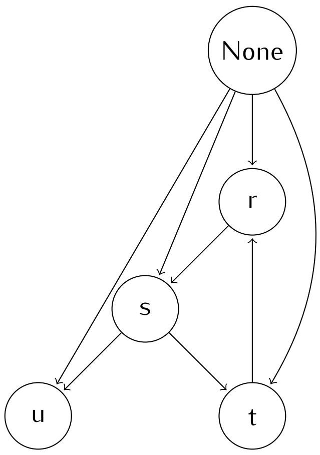

Figure 5: transitive closure of g

Another question that might arise is why we need to use option g as the type and not simply g, and use edgeRel (root g) x := root g $\to^+$ x in the definition of tc, i.e. leave all edges as they are and add edges from the root to every reachable vertex. While this might seem sensible at first, there is a problem, namely that the structure of the "old" graph is changed.
Consider the example graph g from figure 4. Using the faulty method just described would yield the following result on g:

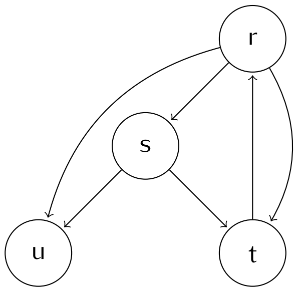

Figure 6: Wrong way of construction tc(g)

What should be the case in this scenario instead is that g $\dot{\in}^n$ tc g, but by changing the transitions from the root, we have "lost" the old graph. Our construction makes sure that there never is an edge from (Some x) to None, i.e. from some other vertex to the root of (tc g).
Note that tc g $\dot{\in}^n$ tc g ($n > 0$) is not a contradiction by itself, e.g. if g = Ω. This is because Omega is already its own transitive closure, i.e. $\Omega \approx tc\,\Omega$ and of course, $\Omega \,\dot{\in}\, \Omega$.
From the definition of tc, it follows that E (root (tc g)) (Some x) = true $\iff$ $(root\,g) \to^+ x$.
All other vertices have the same edges as before : E x y = E (Some x) (Some y). This gives us the property that $\forall x : g.$ (subgraph x) $\approx$ subgraph (Some x), where subgraph x is of course a subgraph of g, whereas subgraph (Some x) is a subgraph of tc g.

Lemma 10.11. $\forall g\, g'.\, g' \,\dot{\in}\, (tc\, g) \iff \exists n > 0.\, g' \,\dot{\in}^n g$

Proof.
"$\Rightarrow$": Let $g' \,\dot{\in}\, (tc\, g)$. It is obvious that there has to be some x : g2 such that

root g2 $\rightarrow^+$ x and g' $\approx$ subgraph x ($\approx$ subgraph (Some x)).
This means that there is some $n > 0$ such that $g' \dot{\in}^n$ subgraph x, by the equivalences regarding $\dot{\in}^n$ and $\rightarrow^+$ discussed above.
"$\Leftarrow$": Let $n > 0$ and $g' \dot{\in}^n g$. We know that there is some x such that
$root\, g \rightarrow^+ x \wedge g' \approx$ subgraph x.
Since $root\, g \rightarrow^+ x$, we know that E None (Some x) = true. Due to the transitivity of $\approx$ and subgraph (Some x) $\approx$ subgraph x, we know that $g' \dot{\in}\, tc\, g$.

□

Let us consider the ZF characterization of the transitive closure next. The transitive closure (tc M) of a set M is the least transitive superset of M. Using the characterization $g' \dot{\in}\, tc\, g \iff \exists n > 0.\, g' \dot{\in}^n g$, proving this property is fairly straightforward:

- tc g is transitive. Let g" $\dot{\in}$ g' $\dot{\in}$ tc g. We know that g' $\dot{\in}^n$ g for some $n > 0$. Hence, g" $\dot{\in}^{S\,n}$ g, which in turn implies g" $\dot{\in}$ tc g.
- g $\dot{\subseteq}$ tc g. Let g' $\dot{\in}$ g. By the definition of $\dot{\in}$, this gives us an x : g such that E (root g) x = true and g' $\approx$ subgraph g. [root g, x] : $root\, g \rightarrow^* x$ and $|root\, g, x| > 1$ gives us that root g $\rightarrow^+$ x, which means that E None (Some x) = true. g' $\dot{\in}$ tc g follows from the fact that subgraph x $\approx$ subgraph (Some x).
- $\forall g^*.\, transitive\, g^* \implies g \dot{\subseteq} g^* \implies tc\, g \dot{\subseteq} g^*$. Let g^* be transitive and $g \dot{\subseteq} g^*$.

  We show $\forall n > 0 \forall g'.\, g' \dot{\in}^n g \implies g' \dot{\in} g^*$ (which gives us that $tc\, g \dot{\subseteq} g^*$) by induction on n.

  The base case is trivial.

  In the inductive case, n = S n' and $g' \dot{\in}^{S\,n'} g$. If n' = 0, i.e. n = 1, $g' \dot{\in}^1 g \iff g' \dot{\in} g$, which immediately gives us that $g' \dot{\in} g^*$, since $g \dot{\subseteq} g^*$. If n' = S m, the inductive hypothesis holds for any h with $h \dot{\in}^{S\,m} g$, since (S m) > 0. Since $g' \dot{\in}^{S\,n'} g$, there is some graph h such that $g' \dot{\in} h$ and $h \dot{\in}^{S\,m} g$. By IH, we know that $h \dot{\in} g^*$. Since g^* is transitive, this means that $h \dot{\subseteq} g^*$, and in particular $g' \dot{\in} g^*$ follows due to $g' \dot{\in} h$.

11 Constructing a set type

We have seen how we can use graphs to implement all ZF axioms (except regularity and infinity). Our versions of the these axioms, however, differ from the usual ones in that they use bisimulation instead of actual equality.
We can give a model for the usual axioms by moving from graphs to the quotient type of graphs modulo bisimulation. To achieve this in constructive type theory, we will give conversion functions f : $\mathbb{G} \rightarrow \mathbb{N}$, $f^{-1} : \mathbb{N} \rightarrow \mathbb{G}$ such that

- $g_1 \approx g_2 \implies f\, g_1 = f\, g_2$
- $g \approx f^{-1}(f\, g)$

We will proceed as follows :

- Show that every graph g is bisimilar to some graph whose vertices are natural numbers
- Give a function to compute the list of all such graphs (modulo bisimulation) up to a certain domain size

- Use the indices of these lists to convert between graphs and natural numbers
- Show that for every type X with a decidable equivalence relation R and a suitable mapping between X and $\mathbb{N}$, we can construct ${}^{X}/_{R}$.

11.1 Graphs over $\mathbb{N}$

For any graph g, we can get a bisimilar graph over natural numbers. The idea is to use the index of each element in the domain instead of the element itself and map the transitions between two elements to transitions between their indices.

Definition 11.1. We call graph g is called well formed if it does not contain duplicates and its domain starts with the root, i.e. dupfree g $\wedge \exists xs.\, dom g = root\, g :: xs$.

The notion of a well formed graph will be useful later on, since each element in a well formed graph has exactly one index. Furthermore, we know that the root of any well formed graph will always have index 0. Note that $\dot{\emptyset}$ is well formed, which is why dom $\dot{\emptyset}$ was defined as [tt].

Lemma 11.2. reordering lemma
For every graph g, we can give a bisimilar graph that is well-formed.

Proof.

- If $root\, g \notin dom\, g$, then the $g \approx \dot{\emptyset}$
- If $root\, g \in dom\, g$, then we can reorder the elements of g and remove duplicates to obtain a bisimilar graph.

□

The proof of the second claim is not hard. It is even easier if we use the fact that every isomorphism gives us a bisimulation:

Definition 11.3. Let X, Y be types and $f : X \to Y$. Let xs and ys be lists of type X and Y, respectively. We call f an isomorphism on xs and ys if $\exists f'. (\forall x \in xs.\, f'(f\,x) = x) \wedge (\forall y \in ys.\, f(f'\,y) = y)$.

Definition 11.4. Let g_1 and g_2 be graphs and $f : g_1 \to g_2$. We call g_1 mutated according to f if :

- map f (dom g_1) $\equiv$ dom g_2 and
- f (root g_1) = root g_2 and
- $\forall x\, y : g_1.\, E\, x\, y = true \iff E\,(f x)\,(f y) = true$

Lemma 11.5. Let g_1, g_2 be graphs mutated according to some isomorphism $f : g_1 \to g_2$ on dom g_1 and dom g_2. Then $g_1 \approx g_2$.

Proof. Consider p := $\lambda x\, y.\, (f x) = y$.

- Let $x, x' \in dom\, g_1, E\, x\, x' = true, y \in dom\, g_2, p\, x\, y = true$. Consider y' := f x'. Obviously, p x' y' = true. To show that E y y' = true, note that p x y = true $\implies$ y = f x. E y y' follows from the definition of mutated.

- Let $x \in dom\, g_1, y, y' \in dom\, g_2, E\, y\, y' = true, p\, x\, y = true$. Consider x' := y y'. Obviously, p x' y' = true. since f, f' are inverse functions, we again have E x x' = true.
- p (root g_1) (root g_2) follows from the definition of mutated.

□

We will use this lemma to prove that every well formed graph g is bisimilar to a graph over ℕ by constructing an isomorphism between g and ℕ on the domains of the respective graphs. But first, we need to define a few functions to convert between elements and indices and prove their correctness.

Definition 11.6. nth : [X] → ℕ → option X
nth [] _ = None
nth (x :: xs) 0 = Some x
nth (x :: xs) (S n) = nth xs n

Definition 11.7. index : X → [X] → option ℕ
index _ [] = None
index y (y :: xs) = Some 0
index y (x :: xs) = $\begin{cases} Some(Sn) & index\, y\, xs = Some\, n \\ None & otherwise \end{cases}$

Note that both nth and index return None in case the index/element was not contained in the given list. The following properties of nth and index can be proven by induction on the argument list:

- nth xs n = Some x $\implies$ x ∈ xs $\land\, n < |xs|$
- $n < |xs| \implies \exists x.$ nth xs n = Some x
- x ∈ xs $\implies \exists n.$ index x xs = Some n
- index x xs = Some n $\implies$ x ∈ xs $\land\, n < |xs|$
- index x xs = Some n $\implies$ nth xs n = Some x
- dupfree xs $\implies$ nth xs n = Some x $\implies$ index x xs = Some n
- dupfree xs $\implies$ nth xs n = Some x $\implies$ nth xs m = Some x $\implies$ n = m

We can construct the list of all elements from 0 until n using the **range** function:

Definition 11.8. range : ℕ → [ℕ]
range 0 := []
range (S n) := range n ++ [n]

Now, we can, for any well-formed graph g, give a graph over ℕ that is bisimilar to g.

Definition 11.9. graph_to_nat g := G e (range $|dom\, g|$) 0,
where e is defined as follows:
$e\, n\, m = \begin{cases} E\, x\, y & nth\, (dom\, g)\, n = Some\, x, nth\, (dom\, g)\, m = Some\, y \\ false & otherwise \end{cases}$

graph_to_nat g relates any element x ∈ dom g to its index in xs. Note that in any well-formed graph g, root g has index 0. Furthermore, since, index x (dom g) denotes the index of x in (dom g), we know that the following properties hold:

- E x y = true $\implies$ index x (dom g) = Some n $\implies$ index y (dom g) = Some m $\implies$ E n m = true
- E n m = true $\implies$ nth (dom g) n = Some x $\implies$ nth (dom g) m = Some y $\implies$ E x y = true.

We can use the following function to convert an x : g to n : $\mathbb{N}$:

Definition 11.10. convert_index : g → $\mathbb{N}$

convert_index (x : g) := $\begin{cases} n & index\, x\, (dom\, g) = Some\, n \\ |dom\, g| & otherwise \end{cases}$

Note that for any x ∈ dom g, convert_index will always return x's index. If x ∉ dom g, we get an index that is clearly not in range |dom g| either. This, together with the previous points, gives us that

Lemma 11.11. $\forall x\, y$. E x y $\iff$ E (convert_index x) (convert_index y).

Theorem 11.12. *For any well-formed graph g, g ≈ graph_to_nat g.*

Proof. We give an isomorphism on dom g and range $|dom\, g|$. Consider f := convert_index. Its inverse function is:

$f'nm := \begin{cases} x & nth(dom\, g)n = Some\, x \\ root\, g & otherwise \end{cases}.$

Furthermore, f and f' are inverse functions:

- Consider x ∈ dom g. Since x ∈ dom g, we know that convert x gives us an index n such that

  $n < |dom\, g| \wedge nth(dom\, g)n = Some\, x.$

  Hence, f' n = x.
- Consider n ∈ range $|dom\, g|$. Since n ∈ range $|dom\, g| \implies n < |dom\, g|$,

  there is some x ∈ dom g such that index x (dom g) = Some n. Hence, f x = convert_index x = n.

We still need to show that g is mutated with f and with respect to graph_to_nat g.

- map convert_index (dom g) === range $|dom\, g|$ is obvious.
- f (root g) = 0 ∧ f' 0 = root g follows from the fact that g is well-formed.
- due to lemma 11.11, $\forall x\, y$. E x y $\iff$ E (convert_index x) (convert_index y).

□

Note that for every well-formed graph g, graph_to_nat g is also well-formed. For any graph g, by transitivity of ≈ and the reordering lemma, we can thus give a well-formed graph g' over $\mathbb{N}$ such that g ≈ g'.

11.2 Conversion functions

Consider the function find_first_index, which finds the first index in a list such that the element in the list at the specified position satisfies a given property p.

Definition 11.13. find_first_index
find_first_index p [] = None

$$\text{find\_first\_index p (x :: xs)} = \begin{cases} Some\,0 & p\ x \\ Some\,(S\,n) & find\_first\_index\,p\,xs = Some\,n \\ None & otherwise \end{cases}$$

find_first_index has a few nice properties that one can easily verify using induction on xs:

- find_first_index p xs = Some n $\implies$ ∃ x ∈ xs. nth xs n = Some x ∧ p x
- x ∈ xs. p x $\implies$ $\exists n$. find_first_index p xs = Some n
- prefix xs ys $\implies$ find_first_index p xs = Some n $\implies$ find_first_index p ys = Some n
- $(\forall x.\, p\, x \iff q\, x)$ $\implies$ find_first_index p xs = find_first_index q xs

prefix is defined as expected, i.e. prefix xs ys := $\exists zs$. xs ++ zs = ys.
The next function we will need is mapcat :

Definition 11.14. mapcat : (X → [Y]) → [X] → [Y]
mapcat f xs := flatten (map f xs)

Note the following properties of mapcat :

- $\forall y.\, y \in mapcat\, f\, xs \iff \exists x \in xs.\, y \in f\, x$
- mapcat preserves prefixes, i.e. $\forall xs\, ys.\, prefix\, xs\, ys \implies prefix(mapcat\, f\, xs)(mapcat\, f\, ys)$
- $(\forall x, |f\, x| \geq 1) \implies \forall xs. |mapcat\, f\, xs| \geq |xs|$

Consider the list of natural numbers from 1 up to and including some given n:
[1..n] := map (+ 1) (range n)
It is obvious that for any two n, m ∈ ℕ with $n <= m$, [1..n] is a prefix of [1..m].
The next step is to enumerate all well-founded graphs over ℕ with a given domain size and 0 as the root.

Definition 11.15. $\alpha\, n$:= map $(\lambda f.\, G\, f\, (range\, n)\, 0)$ (allFuns (range n) (range n))

We use the allFuns function to enumerate all possible transition functions between (range n) and (range n) and for every such function f, we include the graph with 0 as the root, (range n) as the domain and f as the transition function in the result. This gives us every possible graph over natural numbers that is well-founded, has 0 as its root, (range n) as its domain and no transitions to elements outside of its domain. Hence, the following properties of α should come as no surprise.

- n > 0 $\implies$ g ∈ α n $\implies$ well-formed g ∧ t g = ℕ
- n > 0 $\implies$ well-formed g $\implies$ | dom g | = n $\implies$ graph_to_nat g ∈ α n

Note that the second point uses the correctness lemma for allFuns, which in turn uses functional extensionality.
For n $\in \mathbb{N}$, we can enumerate all graphs g with $1 \leq |domg| \leq n$ (modulo bisimulation) as follows:

Definition 11.16. $\beta\, n$:= mapcat α [1..n]

Lemma 11.17. $\forall g$. graph_to_nat g $\in \beta(S|$dom g $|)$

Proof. Recall that $|dom\,(graph\_to\_nat\,g)| \leq S(|dom\,g|)$. Hence, it remains to show that $graph\_to\_nat\,g \in \alpha(|dom\,(graph\_to\_nat\,g)|)$, which follows from the properties of α. □

Another property of β is $\forall n\, m.\, n \leq m \implies prefix(\beta\, n)(\beta\, m)$, which immediately follows from the fact that mapcat preserves prefixes, since [1..n] is a prefix of [1..m].
Now we convert between graphs and $\mathbb{N}$

Definition 11.18. f, f^{-1}

$$f\,g := \begin{cases} n & find\_first\_index\,(\approx g)\,(\beta\,(S|domg|)) = Some\ n \\ 0 & otherwise \end{cases}$$

$$f^{-1}\,n := \begin{cases} g & nth(\beta n)(Sn) = Some\ g \\ \emptyset & otherwise \end{cases}$$

Note that in both definitions, we know that the second case can never occur:

- We know that graph_to_nat g $\in$ $\beta(S|$dom g $|)$, hence find find_first_index $(\approx g)\,(\beta\,(S|domg|))$ will never be None.
- Note that for any n > 0, $|\alpha n| > 0$. Therefore, we know that $|\beta(S\,n)| >= Sn > n$, which means that nth $|\beta(Sn)|$ n can never be None.

Theorem 11.19. $\forall g.\, g \approx f^{-1}(f\,g)$

Proof. We know that find_first_index $(\approx g)$ $(\beta(S|dom\,g|))$ = Some (f g), which implies that there is some g' such that (nth $(\beta(S|dom\,g|))$ (f g)) = Some g' $\wedge$ g' $\approx g$. Either $f\,g \leq |dom\,g|$ or $|dom\,g| <= f\,g$.

- Assume $f\,g \leq |dom\,g|$. We know that the second case in the definition of f^{-1} can never occur, hence there is some g' such that $nth(\beta(f\,g))(S(f\,g)) = Some\ g'$. Furthermore, β (S (f g)) is a prefix of $\beta(S|dom\,g|)$.

  Hence, nth $(\beta(S(f\,g)))$ (f g) = nth $(\beta(S|dom\,g|))$ (f g) = Some g' and we know that g' $\approx$ g.
- Assume $|dom\,g| \leq f\,g$. Similarly to above, $\beta(S|dom\,g|)$ is a prefix of $\beta(S(f\,g))$. Thus, $nth(\beta(S|dom\,g|)(f\,g)) = nth(\beta(S(f\,g)))|dom\,g| = Some\,g'$ and we know that g' $\approx$ g.

□

Theorem 11.20. *Uniqueness of f g*
$\forall g\, g'.\, g \approx g' \iff f\,g = f\,g'.$

Proof.
"$\Rightarrow$": Let g $\approx$ g'. W.l.o.g., $|dom\, g| <= |dom\, g'|$.
Hence, $[1..|dom\, g|]$ is a prefix of $[1..|dom\, g'|]$. Therefore, $\beta\,(S|dom\, g|)$ is a prefix of $\beta\,(S|dom\, g'|)$, since mapcat preserves prefixes.
Note that, since g $\approx$ g', $\forall g''.\,(g'' \approx g) \iff (g'' \approx g')$.
Therefore, find_first_index ($\approx\ g'$) ($\beta\,(S|domg'|)$)= find_first_index ($\approx\ g$) ($\beta\,(S|dom\, g'|)$)= find_first_index ($\approx g$) ($\beta\,(S|dom\, g|)$), because $\beta\,(S|dom\, g|)$ is a prefix of $\beta\,(S|dom\, g'|)$.
"$\Leftarrow$": Let $f\,g = f\,g'$ We know that $\forall g.\, find\_first\_index(\approx\ g)(\beta(S|dom\, g|)) = Some(f\,g) \wedge \exists g'.nth(\beta(S|dom\, g|))(f\,g) = Some\, g' \wedge g \approx g'$
W.l.o.g, $|dom\, g| <= |dom\, g'|$. The claim follows again, since $\beta\,(S|dom\, g|)$ is a prefix of $\beta\,(S|dom\, g'|)$.

□

11.3 Quotient types

The existence of quotient types in constructive type theory is not obvious. We give a construction for the quotient type $^X/_R$, where

- X is a type, and R : X → X → X is a decidable equivalence relation
- f : $X \to \mathbb{N}$ such that $\forall x \approx y.\, f\,x = f\ y$
- $f^{-1} : \mathbb{N} \to X$ such that $\forall x.\, R(f^{-1}(f\,x))\,x$

The idea is to use {n | $f(f^{-1}$ n) = n}, which is the sigma type of all $n \in \mathbb{N}$ paired with a proof that $f(f^{-1}$ n) = n, as the quotient type. We can think of sigma types as dependent pairs. Using this representation, the class of x : X is simply f x (together with a suitable proof that $f(f^{-1}(f\,x)) = f\,x$), whereas we can get a representative element of a class {n | A} by means of $f^{-1}n$.

Definition 11.21. $^X/_R$
$\eta\, n := f(f^{-1}n)$
$^X/_R := \{n \mid \eta\, n = n\}$

Lemma 11.22. $\forall x.\, f\,x = \eta\,(f\,x)$.

Proof. f x = η (f x) follows from the properties of f and f^{-1}. □

Definition 11.23.
repr : $^X/_R \to X$
repr (n, _) := $f^{-1}n$
norm : $X \to {}^X/_R$
norm x := (x, f_repr x)

Now that we have defined what the class of an element and the representative element of a class are, we want to show that two elements are related via R iff they belong to the same class, i.e. $\forall x\, y.\, R\, x\, y \iff norm\, x = norm\, y$, and that two classes are equal whenever their representative elements are related via R, i.e. $\forall a\, b.a = b \iff R(repr\, a)(repr\, b)$.
We will split the proofs in two parts, namely $\forall x\, y.\, R\, x\, y \iff repr\,(norm\, x) = repr\,(norm\, y)$ and $\forall x\, y.repr\,(norm\, x) = repr\,(norm\, y) \iff norm\, x = norm\, y$. The second part might seem trivial, but the problem is that not only do the first components have to be equal, but also the second ones, i.e. the proofs. While we could assume the axiom proof irrelevance (PI), i.e. all

proofs of a given type (i.e. statement) are equal, we can do without assuming this axiom.

Let us consider the first part now.

Lemma 11.24. $\forall x\, y.\, R\, x\, y \iff repr\,(norm\, x) = repr\,(norm\, y)$

Proof.
"$\Rightarrow$": $f^{-1}\, x = f^{-1}\, y$ follows from R x y.
"$\Leftarrow$:" Assume repr (norm x) = repr (norm y). This means that $f^{-1}(f\, x) = f^{-1}(f\, y)$. We want to show R x y. We know that R x $(f^{-1}(f\, x))$ and R y $(f^{-1}(f\, y))$. Due to symmetry and transitivity of R, R x y follows. □

While lemma 11.24 is the intuitively more interesting part, the second part, $\forall x\, y. repr\,(norm\, x) = repr\,(norm\, y) \iff norm\, x = norm\, y$, is rather technical. As already mentioned before, we basically want to prove the equality of two dependent pairs (f x, A) and (f y,B) where A is a proof of $\eta(f\, x) = \eta(f\, y)$ and B is a proof of $\eta(f\, x) = \eta(f\, y)$ and we know that f x = f y.
This works without assuming PI because the type $\mathbb{N}$ has unique identity proofs. In Coq-terms, this means that every proof of type n = m, where n and m are natural numbers, is equal to eq_refl.
We end the discussion about this second part with the remark that simply stating A = B in Coq does not work, since A and B are of incompatible types. However, by matching on the equality proof of type (f x = f y) first, we can state the desired lemma, which is easily provable in Coq. We refer the interested reader to the Coq source code of this thesis and the Homotopy Type Theory book [17] which discusses the fundamental ideas why there is structure in equality proofs in chapter 2.

These two properties together give us the desired properties, namely
$\forall x\, y.\, R\, x\, y \iff norm\ x = norm\ y$
and
$\forall a\, b.\, a = b \iff R(repr a)(repr b)$.

11.4 $\mathcal{N}$

Since bisimulation on graphs is a decidable equivalence relation and we have suitable conversion functions $\mathbb{G} \leftrightarrow \mathbb{N}$, we can construct the quotient type $\mathbb{G}\,/_{\approx}$, which is our type for finitary sets. Furthermore, we can lift the definition of membership and the constructions we have for the ZF axioms to this level. We will see that $\mathbb{G}\,/_{\approx}$ gives us a proper model for the ZF axioms that we consider.

Definition 11.25. $\mathcal{N} := \mathbb{G}\,/_{\approx}$.

Definition 11.26. $M \in N := (reprM)\,\dot{\in}\,(reprN)$.

Definition 11.27. $M \subseteq N := \forall x \in M.\, x \in N$.

The constructions for the ZF-axioms we consider simply convert from $\mathcal{N}$ to $\mathbb{G}$, use the constructions for graphs, and convert back to $\mathcal{N}$.

- Axiom of existence : $\emptyset := norm\,\dot{\emptyset}$
- Axiom of pairing : $\{M, N\} := norm\{repr\, M, repr\, N\}$

- Axiom of union : $\bigcup M := norm(\bigcup repr\, M)$
- Axiom of power : $\mathcal{P}M := norm(\mathcal{P}(reprM))$

For the axioms of separation and replacement, we need to do a little extra work to convert the given relation/function to the correct type.

Definition 11.28. Lifting functions and predicates from $\mathcal{N}$ to $\mathbb{G}$
$lift_p$ (p : $\mathcal{N} \to$ Prop) := $\lambda g.$ p (norm g)
$lift_f$ (f : $\mathcal{N} \to \mathcal{N}$) := $\lambda g.$ repr (f (norm g))

Note that both $lift_p$ p and $lift_f$ f are extensional, since norm g = norm g' for any g $\approx$ g'. Furthermore, $lift_p$ p is of course decidable, provided that p is decidable.

- Axiom of separation : $\{x \in M \mid p\,x\} := norm\{x \dot{\in} repr\, M \mid (lift_p\, p)\, x\}$
- Axiom of replacement : $\{f\,x \mid x \in M\} := norm\{(lift_f\, f)\, x \mid x \dot{\in} repr\, M\}$

Furthermore, it is easy to see that $M \subseteq N \iff repr\, M \dot{\subseteq} repr\, N$. This implies that the axiom of extensionality also holds for $\mathcal{N}$:
$M = N \iff M \subseteq N \wedge N \subseteq M$.
Of course, $M \in N, M \subseteq N$ and $M = N$ are decidable for $\mathcal{N}$, since we know how to decide the respective properties modulo $\approx$ on $\mathbb{G}$.
As mentioned in the introduction, Aczel's AFA is trivial for our representation: We have conversion functions (repr and norm) between $\mathbb{G}$ and $\mathcal{N}$ and know that bisimilarity on the $\mathbb{G}$ level means equality on the $\mathcal{N}$ level.
Furthermore, it is easy to construct a choice function γ on $\mathcal{N}$ such that $\forall M \neq \emptyset.\ \gamma M \in M$.

Definition 11.29. $\gamma : \mathcal{N} \to \mathcal{N}$

$$\gamma M := \begin{cases} \emptyset & child\_nodes\,(repr\, M) = [] \\ norm\,(subgraph\, x) & child\_nodes\,(repr\, M) = x :: xs \end{cases}$$

Lemma 11.30. $\forall M \neq \emptyset.\ \gamma M \in M$.

Proof. Note that child_nodes (repr M) = [] $\iff$ repr M $\approx \dot{\emptyset}$. In this case, we know that M = $\emptyset$, contradicting our assumption. Otherwise, (subgraph x) $\dot{\in}$ (repr M), hence (norm (subgraph x)) $\in$ M. ☐

12 Future Work

There are a few directions in which one could further extend the work done in this thesis. First of all, we can extend our binary tree model for hereditarily finite sets by generalizing the constructions used in the second part, since all they rely on is an adjunction function and a conversion from the type and hand (in our case, $\mathbb{G}$ or $\mathbb{T}$) and a list of that type, which basically represents the children of the given element (and vice-versa). These conversions have to be idempotent modulo the equivalence relation we consider, which in the case of $\mathbb{G}$ is $\approx$, and for $\mathbb{T}$ is $\equiv$. We have already seen that $\equiv$ plays nicely together with the conversion functions ($\mathcal{L}$ and $\mathcal{T}$). Hence, generalizing the construction for the above mentioned axioms gives us an easy way to prove that $\mathbb{T}$ actually adheres to the ZF-model of HF sets (modulo $\equiv$).

Furthermore, the mapping we have between $\mathbb{G}$ and $\mathbb{N}$ basically gives us an easy way to enumerate finitary sets. We could also construct an enumeration for $\mathbb{T}$. One way to do this is by using Ackermann's encoding for hereditarily finite sets as natural numbers [18], which has already been formalized in Coq by Chad Brown [19]: every number has a unique binary encoding. When considering any number n, if the i-th bit is set to one in n, i is considered an "element" of n. For instance, the number 9 has the binary encoding 1001, and hence has the elements 3 and 0. This means that the number 0 corresponds to $\emptyset$, 1 corresponds to $\{\emptyset\}$ and so on. Since we sort trees in ascending order, it would be easy to compute the number corresponding to a sorted set by structural recursion.

Moreover, we have constructed an explicit quotient type $\mathbb{G}/_{\approx}$ as our type for finitary sets. This construction depends on our mapping between $\mathbb{G}$ and $\mathbb{N}$. It is possible to construct the quotient type $\mathbb{T}/_{\equiv}$ as a model for hereditarily finite sets, and we do not need the enumerability of $\mathbb{T}$ for this: We can use $\mathbb{T}/_{\equiv} := \{t | sort\, t = t\}$ as the quotient type. Since we know that sorting equivalent trees yields identical trees, all that is left to show is that the equality proof in the second component of the sigma type is unique. This holds, since equality on $\mathbb{T}$ is decidable. Note that this would not have worked for $\mathbb{G}$, because equality on $\mathbb{G}$ is undecidable.

In the introduction, we have mentioned the relation between CCS and sets. One application for finitary sets is the fragment of CCS with an empty process, a binary + operator and action prefix (with only a single action) and recursion. Formalizing this in Coq represents another opportunity for future work.

Part III
Appendix

13 File Structure

The Coq development[1] is divided into two parts:

- Part 1 corresponds to the development regarding the binary trees and can be found in the folder called "binary-trees".
  1. Tree.v contains the basic definitions of trees, the equivalence relations $\equiv$ and $\equiv_{\triangleright}$ and their properties.
  2. TreeOrder.v contains the properties and definitions concerning the comparison and sorting of trees, as well as the decidability of $\equiv$.
  3. TreeSim.v contains the definitions and properties of $\dot{\in}$, $\dot{\subseteq}$ and the double inclusion principle.
  4. Finally, Bisim.v contains the coinductive definition of $\approx$, as well as the equivalence proof of $\equiv$ and $\approx$.
- Part 2 contains all the files that belong to the development concerning finitary sets and can be found in the folder called "nwf-sets".
  1. Graphs.v contains the definition of graphs, bisimulation on graphs and its decidability, as well as the proof that two isomorphic graphs are bisimilar.
  2. GraphAxioms.v contains the constructions and proofs for the ZF axioms on graphs.
  3. GraphNat.v contains everything concerning the connection between graphs and natural numbers, starting from the reordering and graph_to_nat up to the conversion functions between $\mathbb{G}$ and $\mathbb{N}$ and their correctness.
  4. Quotient.v contains the generic construction for the quotient type $X/_R$ for a given type X with a relation R assuming suitable conversion functions $X \leftrightarrow \mathbb{N}$.
  5. GraphReach.v contains the definitions and lemmas connected to the reachability in a graph and its decidability.
  6. GraphTC.v contains the construction and correctness proof of our construction for the transitive closure of a graph
  7. NSet.v gives the definition of $\mathcal{N}$ and results regarding $\mathcal{N}$: The constructions for the ZF axioms, the transitive closure and the Anti-Foundation Axiom.

Both parts also contain the base library [2] from the core lecture Introduction to Computational Logic which is taught at Saarland University.

[1] available under `http://www.ps.uni-saarland.de/~dmueller/bachelor.php`
[2] `https://www.ps.uni-saarland.de/courses/cl-ss14/script/Base.v`

14 References

References

[1] H. Curry, *To H.B. Curry : essays on combinatory logic, lambda calculus, and formalism.* London New York: Academic Press, 1980.

[2] E. Zermelo, "Untersuchungen über die Grundlagen der Mengenlehre," *Mathematische Annalen*, vol. 65, no. 2, pp. 261–281, 1908.

[3] A. Fraenkel, "Zu den Grundlagen der Cantor-Zermeloschen Mengenlehre," *Mathematische Annalen*, vol. 86, no. 3, pp. 230–237, 1922.

[4] E. Zermelo, "Über Grenzzahlen und Mengenbereiche: Neue Untersuchungen über die Grundlagen der Mengenlehre.," *Fundamenta Mathematicae*, vol. 16, pp. 29–47, 1908.

[5] J. von Neumann, "Zur Einführung der transfiniten Zahlen," *Acta Scientiarum Mathematicarum (Szeged)*, vol. 1, no. 4, pp. 199–208, 1923.

[6] B. Russel, "The principles of mathematics," *Mathematische Annalen*, vol. 1, 1903.

[7] S. Wagon, *The Banach-Tarski Paradox.* Cambridge University Press, 1985. Cambridge Books Online.

[8] E. Zermelo, "Beweis, daß jede Menge wohlgeordnet werden kann. (Aus einem an Herrn Hilbert gerichteten Briefe)," *Mathematische Annalen*, vol. 59, pp. 514–516, 1904.

[9] E. Zermelo, "Neuer Beweis für die Möglichkeit einer Wohlordnung.," *Mathematische Annalen*, vol. 65, pp. 107–128, 1908.

[10] R. Milner, *A Calculus of Communicating Systems.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1982.

[11] S. Abramsky, "A Cook's Tour of the Finitary Non-Well-Founded Sets," *CoRR*, vol. abs/1111.7148, 2011.

[12] P. Aczel, *Non-well-founded Sets.* No. 14 in Lecture Notes, Center for the Study of Language and Information, Stanford University, 1988.

[13] M. Baldamus, "A Non-well-founded Sets Semantics for Observation Congruence over Full CCS," tech. rep., 1994.

[14] K. Stark, "Quantitative Recursion-Free Process Axiomatization in Coq," bachelor's thesis, Saarland University, May 2014.

[15] D. Kirst, "Formalised Set Theory: Well-Orderings and the Axiom of Choice," bachelor's thesis, Saarland University, August 2014.

[16] A. Miquel, "Inconsistent Type Systems." `http://www.cse.chalmers.se/research/group/logic/TypesSS05/Extra/miquel_sl3.pdf`, August 2005. [Online; accessed 19-July-2015].

[17] T. Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics.* Institute for Advanced Study: `http://homotopytypetheory.org/book`, 2013.

[18] W. Ackermann, "Die Widerspruchsfreiheit der allgemeinen Mengenlehre," *Mathematische Annalen*, vol. 114, pp. 305–315, 1937.

[19] C. Brown, "Ackermann Encoding of Decidable Hereditarily Finite Sets." `https://www.ps.uni-saarland.de/settheory/HF/HF.v`. [Online; accessed 19-July-2015].