On Representability of Ordered Convoluted Monoids

Jaš Šemrl[1]

MEng Computer Science

Supervisor: Professor Robin Hirsch

Submission date: 29 April 2019

Abstract

Algebras of binary relations play an important role and have a wide range of applications in computer science. This is why numerous approaches have been proposed to axiomatise abstract algebraic structures that would be isomorphic to those algebras. Two notable examples are the Kleene Algebras and Tarski's Relation Algebras. However, the axiomatisation presented by the latter is not complete. Furtheremore, the class itself is not finitely axiomatisable. This is why we focus our search to subsignatures that are weaker in terms of abstracting the relation calculus, but still provide a wide range of applications.

We explore the nature of algebras with the subsignature $(\leq, 1', \smile, ;)$ and how they can be represented as algebras of relations. We begin by showing that the basic axiomatisation of the ordered convoluted monoids is insufficient to ensure representability and as such needs to be extended. We then provide a recursive definition of an infinite set of sound axioms that ensure representability. We hypothesise that this is not equivalent to any finite set of axioms.

Therefore, we focus our search to representable classes. We prove a novel result stating that the class of representable algebras of total relations is finitely axiomatisable for this signature. Furthermore, finite algebras will be finitely representable with the upper bound on the size being exponential in terms of the number of elements. We also look at a special subclass of these algebras that follow the Pseudo-Triangle Law and show they can be represented with a linear upper bound on the size. We also propose algorithms for constructing these representations.

We implement algorithmic construction of these representations using Python and analyse the time complexity of generating these. We also use first-order logic counter-example finder Mace to illustrate our argument supporting the conjecture that the signature is not finitely axiomatisable.

Contents

Chapter 1

Introduction

Binary relations have a great ability to illustrate various concepts in logic and reasoning. This is why they have found a number of applications in computer science in all fields ranging from databases [3] to artificial intelligence [7]. This is why it is important to research the theoretical side of the logic of relations.

A binary relation over a set is defined as a collection of ordered pairs of two elements in that set. For example, we can define the relation *parent* over the set of all people in the world as all ordered pairs of two people where the first person in the pair is the parent of the second. Note that it is important for this to be ordered as, for example, it may be the case that $(John, Jane) \in parent$ because John is Jane's parent, however $(Jane, John) \notin parent$ as Jane is not John's parent.

Moreover, we can define some operations in the relation calculus, such as the converse and composition. These can be easily illustrated using family relations. For example, the converse of $father$ is *child* and the composition of *parent* and *sister* is *aunt*. What one can see is that the composition is associative, but not commutative - sister's father is not father's sister.

Now, we can define algebras of relations. These are sets with relations as their elements that are closed under pre-defined operations. Those can be from relation calculus as well as set operations like the union, intersection, set difference and complement. We may also demand certain constants like the identity relation. In the example of family relations, this can be thought of as the $self$ relation which is the set of all pairs $(person_1, person_2)$ where $person_1 = person_2$.

Concrete algebras of relations are very easy to understand and work with. However, we are interested in the abstraction of these in the style of Boolean Algebras being the abstraction of unary relations. Boolean algebras contain the universal and empty set whilst being closed under complement union and intersection. And it turns out that there exists a finite number of axioms you can assign to this structure to ensure that there exists an isomoporphic algebra of unary relations.

In the 1940s Tarski proposes an extension of Boolean algebras to binary relations [14]. This algebra would also contain the identity relation and be closed under composition and converse. However, unlike the Boolean Algebras, Tarski could not find a finite set of axioms that would guarantee such an algebraic structure to be isomorphic to an algebra of relations - we call this isomorphism a representation. Furthermore, it turns out that there is no such finite set of axioms [10].

We therefore focus on algebras corresponding to different subsignatures and consider if there exists a finite number of axioms to ensure a representation for all such algebras. These may not have the full expression of the Tarski's relation algebras, but still found a variety of applications from regular expression equivalence [12] to program validation and termination [5, 6].

Here we explore the nature of representability of algebras with a signature that includes the identity element and the order predicate, along with the converse and composition. We show that the basic axiomatisation of this algebraic structure (called an ordered convoluted monoid) is not sufficient to guarantee representability and explore the axiomatisations that do.

We propose a recursive definition of an infinite set of axioms that are complete and provide an intuitive argument with counterexamples that there likely does not exist a finite set equivalent to it. We then outline some representation classes, defined by certain properties (e.g. totality) that are not true for all algebras of relations. We show that these are not only representable, but also finitely representable (with an exponential upper bound). We also provide construction algorithms for these representations.

In the second chapter of the report we introduce the preliminary knowledge for the issue alongside the historical perspective on the issue. In the third chapter we define ordered convoluted monoids, their basic axiomatisation and show why it is not complete. Then we provide a complete recursive axiomatisation in the fourth chapter which brings us to the fifth chapter where we discuss what classes are finitely axiomatisable. We also provide code and explanations of our practical implementations of the discussed theory in the appendices.

Chapter 2

Preliminaries

2.1 Introduction to Algebras of Binary Relations

We first look at some key operations in the calculus of binary relations (from now on, we will refer to them as relations)

DEFINITION 1 (Converse) *Let R be a relation over the set X. We define its converse $\breve{R}$ as*

$$\breve{R} = \{(y, x) | (x, y) \in R\}$$

DEFINITION 2 (Composition) *Let R, S both be relations over X. We define the composition of S and T (denoted $S;T$) as*

$$R;S = \{(x, z) | \exists y : (x, y) \in R \wedge (y, z) \in S\}$$

To gain some intuition on these definitions, we can illustrate them with examples of family trees. For example, if we define a relation $Parent$ such that $(a, b) \in Parent$ if a is a parent of b, the converse of $Parent$ will be a relation $Child = Pa\breve{r}ent$ such that $(a, b) \in Child$ if a is a child of b. Another example is the $Cousin$ relation, of which the converse is simply $Cousin$. That relation is said to be self-converse.

Continuing with the example of such relations, $Aunt$ can be a composition of two relations $Parent$ and $Sister$. Or $Cousin$ can be a composition of $Parent$, $Sibling$ and $Child$. Note that the order of operands is important.

This leads us to some properties of these operations:

PROPOSITION 3 *For any set X it is true that $\forall R \subseteq X \times X : \breve{\breve{R}} = R$*

PROOF:

Since $\breve{R} = \{(y, x) | (x, y) \in R\}$, it follows that $\breve{\breve{R}} = \{(x, y) | (x, y) \in R\} = R$ $\square$

PROPOSITION 4 *For any set X it is true that* $\forall R, S \subseteq X \times X : R \subseteq S \rightarrow \breve{R} \subseteq \breve{S}$

PROOF:

If $(x, y) \in \breve{R}$ then $(y, x) \in R$. By $R \subseteq S$, it follows that $(y, x) \in S$ if $(y, x) \in R$. If $(y, x) \in S$ then $(x, y) \in \breve{S}$. Therefore, $(x, y) \in \breve{R} \rightarrow (x, y) \in \breve{S}$ provided $R \subseteq S$. Or in other words, $R \subseteq S \rightarrow \breve{R} \subseteq \breve{S}$ $\square$

PROPOSITION 5 ; *is associative*

PROOF:

For any set X, take any $R, S, T \subseteq X \times X$. We firstly look at the definition of $R; (S; T)$. We know that

$$S; T = \{(z, y) | \exists w \in X : (z, w) \in S \wedge (w, y) \in T\}$$

Therefore

$$R; (S; T) = \{(x, y) | \exists z \in X : (x, z) \in R \wedge (\exists w \in X : (z, w) \in S \wedge (w, y) \in T)\}$$

Or simply

$$R; (S; T) = \{(x, y) | \exists z, w \in X : (x, z) \in S \wedge (z, w) \in S \wedge (w, y) \in T\}$$

Similarly

$$R; S = \{(x, w) | \exists z \in X : (x, z) \in R \wedge (z, w) \in S\}$$

and hence

$$(R; S); T = \{(x, y) | \exists w \in X : (\exists z \in X : (x, z) \in R \wedge (z, w) \in S) \wedge (w, y) \in T\}$$

Or simply

$$R; (S; T) = \{(x, y) | \exists z, w \in X : (x, z) \in S \wedge (z, w) \in S \wedge (w, y) \in T\}$$

Therefore, we can see that for all $R, S, T \subseteq X \times X$, it holds that $R; (S; T) = (R; S); T$ or put simply ; is associative. $\square$

PROPOSITION 6 *For any set X it is true that* $\forall R, S, T \subseteq X \times X : R \subseteq S \rightarrow R; T \subseteq S; T$

PROOF:

Let R, S, T be any three relations over set X. Given $(x, y) \in R; T$ then there exists some z such that $(x, z) \in R \wedge (z, y) \in T$. Since $R \subseteq S$, it is true that $(x, z) \in R \rightarrow (x, z) \in S$. If $(x, z) \in S \wedge (z, y) \in T$ this means $(x, y) \in S; T$. Therefore $\forall R, S, T \subseteq X \times X : R \subseteq S \rightarrow R; T \subseteq S; T$. $\square$

REMARK 7 *Propositions 4 and 6 can also be referred to as the monotonicity of* $\smile$ *and* ; *over* $\subseteq$ *respectively.*

We now want to discuss Algebras of binary relations. We begin by defining a slightly non-standard relaxed definition of an algebra

DEFINITION 8 (Algebraic Structure) *An algebraic structure over an underlying set* A *is a set of constants, predicates and operations of finite arity on* A.

DEFINITION 9 (Algebra) *An underlying set* A, *together with an algebraic structure over it is called an Algebra.*

DEFINITION 10 (Signature) *A signature* S *of an algebraic structure is a finite tuple of predicates, constants and operations, specified by this algebraic structure.*

DEFINITION 11 (Class of Algebras) *A class of algebras* $K(S)$ *is a class of algebras that have the same signature* S *and obey the same axioms.*

REMARK 12 *Note that the above definitions allow us to define predicates and constants. A more standard definition of algebraic structures only allows us to define operations.*

Examples of classes of algebras include boolean algebras, groups, rings and fields. The last two are examples of two classes of algebras with the same signature $S = (0, 1, -, ^{-1}, +, \cdot)$, but different axioms.

Elements of an algebra may also be relations. The constants, predicates and operations can then be described in terms of relation calculus and even set theory. In this case, we say that this is an algebra of binary relations.

DEFINITION 13 (Class of Algebras of Binary Relations) *Let* S *be a signature of predicates, constants and operations, defined for binary relations. A class of binary relations* $R(S)$ *is a class of all algebras* A *with the signature* S, *such that for each* $A \in R(S)$, *there exists some (possibly infinite) set* X *such that all* $a \in A$ *are binary relations over* X.

Our goal is to provide a class of abstract algebras $K(S)$ that would illustrate the expressiveness of the relation calculus whilst ensuring that each algebra in it would have a guaranteed isomorphic algebra of binary relations in $R(S)$. We define some tools for this task.

DEFINITION 14 (Representation) *A representation of a an algebra* A *(denoted* θ*) is a mapping defined as* $\theta : A \to \mathcal{P}(X \times X)$ *for which it holds:*

1. θ *is faithful (i.e.* $\forall a, b \in A : a^\theta = b^\theta$ *iff* $a = b$*)*
2. θ *correctly represents all constants, predicates and operations. For example, if converse is included in the signature* $\forall a \in A : (\breve{a})^\theta = (\breve{a^\theta}) = \{(y, x) | (x, y) \in a^\theta\}$

We illustrate this with a simple example. Let B be a boolean algebra with two elements (for the definition of boolean algebra see section 2.2). We propose an algebra of relations over the set $X = \{x\}$ with an underlying set $\{\emptyset, \{(x, x)\}\}$ such that $0^\theta = \emptyset$ and $1^\theta = \{(x, x)\}$ and $\cdot$ is represented by $\cap$ and $+$ is represented by $\cup$ and $-$ is represented by ^{c}. This is a representation of B. We check this in the following steps:

1. We first check faithfulness. I.e. for all $a \not\leq b$, there exists (x, y) such that $(x, y) \in a^\theta$ but not b^θ. This algebra only has one such pair: $1 \not\leq 0$. And $(x, x) \in 1^\theta$ but $(x, x) \notin 0^\theta$.

2. $1, 0$ are correctly represented as it is indeed true that $0^\theta = \emptyset$ and $1^\theta = U = \{(x, x)\}$

3. $-$ is correctly represented: $-0^\theta = \emptyset^c = U = 1^\theta$ and $-1^\theta = U^c = \emptyset = 0^\theta$

4. $+, \cdot$ are correctly represented as $0^\theta \cap 1^\theta = \emptyset \cap U = \emptyset = (0 \cdot 1)^\theta$ and similarly for all other pairs. We also check $0^\theta \cup 1^\theta = \emptyset \cup U = U = (0 + 1)^\theta$ and similarly for all other pairs.

Now that we have defined what a representation is, we can define the following

DEFINITION 15 (Representability) *A class of algebras $K(S)$ is representable over a class of algebras of relations $R(S)$ iff all algebras $A \in K$ has a representation.*

DEFINITION 16 (Finite Representability) *A representable class of algebras $K(S)$ is finitely representable over a class of algebras of relations $R(S)$ iff for all finite algebras $A \in K(S)$, there exists $A^\theta \in R(S)$, defined over a finite set X.*

Now, the representability is a good tool to reason about algebras. However, we are also interested in the reasoning about the signatures. This is why we define

DEFINITION 17 (Finite Axiomatisability) *A signature S is finitely axiomatisable for a class of algebras of relations $R(S)$ if there exists a finite number of first order sound axioms that will (together with the signature) form a representable class of algebras $K(S)$.*

By a first order axiom we mean a first order logic formula that may use constants, predicates and operations (functions) in the signature. We define soundness as follows

DEFINITION 18 (Soundness) *An axiom is sound for a class of algebras of relations $R(S)$ if it holds for any algebra of relations in the class.*

REMARK 19 *When we speak about the soundnes/finite axiomatisibility/representability over the classes of algebras of relations, we mean soundnes/finite axiomatisibility/representability over some set of algebras of relations. However, in the most general case (i.e. the set of all algebras of relations with a given signature), we simply say that the axiom is sound, a signature is finitely axiomatisable or an algebraic structure is (finitely) representable.*

2.2 Related Work and Research Done in Relation Algebras

The exploration of algebras of relations began by Boole [1] with a successful axiomatisation of a signature that faithfully represents and outlines the properties of the unary relation calculus. We nowadays refer to these as Boolean Algebras. Their modern definition is as follows:

DEFINITION 20 *A boolean algebra B is an instance of an algebraic structure with a signature $(0, 1, -, +, \cdot)$ that obeys the following axioms:*

1. *$+, -$ are associative and commutative*

2. $a \cdot b = -(-a + -b)$

3. $a \cdot (b + c) = a \cdot b + a \cdot c$

4. $a + 0 = a$

5. $a + 1 = 1$

6. $a + -a = 1$

7. $-0 = 1$

8. $-(-a) = a$

$\forall a, b, c \in B$ *[8]*

The simplest example of such a structure is a two-element boolean algebra $0, 1$ where the $+, \cdot$ are defined as follows

$+$	0	1	$\cdot$	0	1
0	0	1	0	0	0
1	1	1	1	0	1

This class of algebras is representable over any class of unary relations. Since the axioms for this can be expressed as a finite set of first order formulas, the signature is finitely axiomatisable for algebras of unary relations. It turns out it is also finitely axiomatisable for algebras of binary relations. However, the algebraic structure does not encapsulate the expressiveness of the binary relation calculus.

This inability to express the calculus of binary relations was the motivation for DeMorgan to propose the study of algebraic properties of binary relations [4] which eventually led to the Tarski's exploration in axiomatising the class of representable relation algebras **RRA** in the 1940s.

In [14], he provides a set of axioms for the signature $(0, 1, 1', -, +, \cdot, \smile, ;)$ that are sound in terms of calculus of relations.

DEFINITION 21 (Relation Algebra) *A Relation Algebra $\mathcal{A}$ is an instance of the algebraic structure with the signature $(0, 1, 1', -, +, \cdot, \smile, ;)$ and the following axioms*

1. $(0, 1, -, +, \cdot)$ *is a Boolean Algebra*

2. $;$ *is associative*

3. $1'$ *is the identity element for* $;$

4. $\breve{\breve{a}} = a$ *and* $(a\breve{;}b) = \breve{b};\breve{a}$

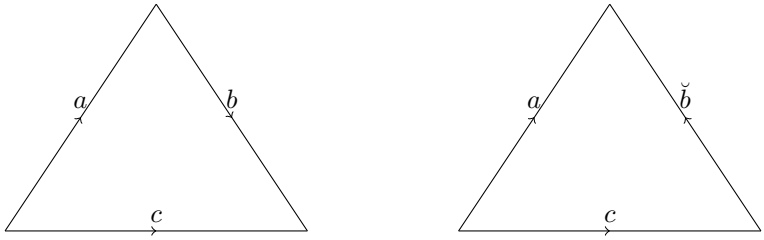
5. $a;1 = 1$ *or* $1;(-a) = 1$

6. $(a + b)^{\smile} = \breve{a} + \breve{b}$ *and* $a;(b + c) = a;b + a;c$

7. $(a;b) \cdot c = 0 \leftrightarrow (c;\breve{b}) \cdot a = 0$

$\forall a, b, c \in \mathcal{A}$

These axioms are all sound and are intuitive, with the exception of (7). This is the so-called Percian Triangle Law (discovered by DeMorgan [4]). Whilst we will not prove its soundness for binary relations, we will illustrate it using the two diagrams.



What we can see is that if there is an overlap between $a; b$ and c, there must be an overlap between $c; \breve{b}$ and a and vice versa.

Whist the Tarski signature is great for demonstrating the expressiveness of the relation calculus, the axioms provided did not guarantee a representation. Furthermore, the signature itself has been shown not to be finitely axiomatisable, for example in [10].

Many subsignatures have followed, including some finitely axiomartisable like those of ordered domain algebras [11]. The signature of an ordered monoid $(\leq, 1', ;)$ has been shown non-finitely axiomatisable [9]. Similarly, there is no finite axiomatisation for the signature of an ordered convoluted semigroup $(\leq, \smile, ;)$ [2]. The finite axiomatisability of $(\leq, 1', \smile, ;)$ remains an open question [9].

Chapter 3

Representing Ordered Convoluted Monoids

3.1 Introduction to Ordered Convoluted Monoids

We now look at the ordered convoluted monoids. We define them as

DEFINITION 22 (Ordered Convoluted Monoid) *An ordered convoluted monoid is an algebra with the signature* $(1', \leq, \smile, ;)$ *where*

1. $(1', ;)$ *is a monoid*
2. $\leq$ *is a partial order*
3. $\smile$ *is a converse*
4. *The converse of composition law applies*
5. $;, \smile$ *are monotone over* $\leq$

We unwrap this definition by looking at each of these properties. Firstly, we define a monoid to be

DEFINITION 23 (Monoid) *A monoid is an algebra with the signature* $(1, *)$ *where* $*$ *is an associative operation and* 1 *is an identity element (i.e.* $\forall x : 1 * x = x * 1 = x$*)*

This falls exactly between the semigroup (that does not have an identity element) and a group (which has an inverse).

PROPOSITION 24 *The axioms of the monoid are sound for binary relations.*

PROOF:

We have shown in proof of Proposition 5 that ; is indeed associative. This is why we only need to show that $\forall R \in X \times X : 1'; R = R; 1' = R$. We know that

$$1'; R = \{(x, y) | \exists z : (x, z) \in 1' \wedge (z, y) \in R\}$$

Since $(x, z) \in 1'$ holds iff $x = z$

$$1'; R = \{(z, y) | (z, y) \in R\} = R$$

We can similarly show this for $R; 1'$. Hence, $1'$ is indeed an identity element. Therefore, the axioms of the monoid are sound for binary relations. □

We now continue to define a partial order. In terms of calculus of relations, $\leq$ is represented as $\subseteq$. We axiomatise it as follows

DEFINITION 25 (Partial Order) *A binary predicate $\leq$ is a partial order if it is*

1. *reflexive, i.e.* $\forall a : a \leq a$
2. *transitive, i.e.* $\forall a, b, c : (a \leq b \wedge b \leq c) \rightarrow a \leq c$
3. *anti-symmetric, i.e.* $\forall a, b : (a \leq b \wedge b \leq a) \rightarrow a = b$

PROPOSITION 26 *$\subseteq$ is a partial order.*

PROOF:

We check all three properties

1. $\forall R : R \subseteq R$ as $R = R$
2. we show that $\forall R, S, T : R \subseteq S \wedge S \subseteq T \rightarrow R \subseteq T$

   we can see that given the left hand side of the implication $\forall a : a \in R \rightarrow a \in S \wedge a \in S \rightarrow a \in T$. Therefore $\forall a : a \in R \rightarrow a \in T$ given $R \subseteq S \wedge S \subseteq T$. Therefore, $R \subseteq T$ given $R \subseteq S \wedge S \subseteq T$.
3. we show that $\forall R, S : R \subseteq S \wedge S \subseteq R \rightarrow R = S$

   Given the left hand side $\forall a : a \in R \rightarrow a \in S \wedge a \in R \leftarrow a \in S$ or simply $\forall a : a \in R \leftrightarrow a \in S$ This means, given left hand side, $R = S$

As $\subseteq$ follows all three axioms of a partial order, it is a partial order. □

The introduction of the partial order allows us to define the upwardly closed sets. These will come in useful in a lot of future proofs.

DEFINITION 27 (Upwardly Closed Set) *Let U be a set of elements from a convoluted ordered monoid. U is an upwardly closed set, if there exists such a set A that defines U as follows*

$$U = \{u | a \leq u, a \in A\}$$

REMARK 28 *We denote an upwardly closed set U as $A^{\uparrow}$, where A is the defining set of U. If A consists of only one element, call it a, we may denote U as $a^{\uparrow}$. This is also called a principal upwardly closed set.*

The final definition we would like to add is the set of all upwardly closed sets $\Upsilon(\mathcal{O})$

DEFINITION 29 ($\Upsilon(\mathcal{O})$) *We define the set of all upwardly closed sets of an algebra $\mathcal{O}$ as*

$$\Upsilon(\mathcal{O}) = \{A^{\uparrow} | A \subseteq \mathcal{O}\}$$

We continue by looking at the converse. This is not the same as the inverse in groups where all $a \in G$ have an a^{-1} such that $a; a^{-1} = 1'$. We will show that it is impossible to generally define an inverse in terms of relation calculus. This is why we define the converse as a weaker unary operation.

PROPOSITION 30 *It is not true that one can generally define an inverse a^{-1} for every relation a over X so that $a; a^{-1} = 1'$ where $1'$ denotes the identity relation over X with respect to composition ;.*

PROOF:

We show this by counterexample to the claim that there exists an inverse for all elements. Consider the zero relation $\emptyset$. One can see that $\forall R \in \mathcal{P}(\mathcal{X} \times \mathcal{X}) : \emptyset; R = \emptyset$. Since $\emptyset$ is clearly not the identity relations $1'$ $\square$

Therefore, we consider the converse. In Section 2.1, we discuss how we can define the converse in terms of relation algebras. Now, we provide some axioms for a convoluted monoid. It is true for all a, b

1. $\breve{\breve{a}} = a$ (the converse law)
2. $\breve{1'} = 1'$
3. $\breve{a;b} = \breve{b}; \breve{a}$ (the converse of composition law)
4. $a \leq b \rightarrow \breve{a} \leq \breve{b}$ (monotonicity of converse over partial order)

Similarly to the monotonicity of $\smile$ over $\leq$, we define the monotonicity of ; over $\leq$ as

$$\forall a, b, c : a \leq b \rightarrow a; c \leq b; c$$

We have also shown in Propositions 6, 4 that all of these axioms are sound for binary relations. Therefore, it follows that the axioms of ordered convoluted monoids are indeed all sound. This means that all algebras of relations with the signature of the ordered convoluted monoid are in fact ordered convoluted monoids. The more interesting question that we discuss now, is whether this axiomatisation is complete. For this, we firstly need to define some tools.

3.2 Representations Of Ordered Convoluted Monoids

We now discuss what the representation of an ordered convoluted monoid (and other algebraic structures with the same signature) should look like.

3.2.1 The Class Of Representable Ordered Convoluted Monoids

We begin by formalising the class of representable ordered convoluted monoids **ROCM**.

DEFINITION 31 (ROCM) *Let $S = (\leq, 1', \smile, ;)$ and $R(S)$ the class of all algebras of relations with the signature S. The class of representable ordered convoluted monoids* **ROCM** $\subseteq K(S)$ *is defined as the set of all monoids that are representable over $R(S)$.*

Put simply, an ordered convoluted monoid is representable if it is isomorphic to some algebra of relations over some set X that possesses the identity element and is closed under composition and converse.

3.2.2 Representations as Graphs

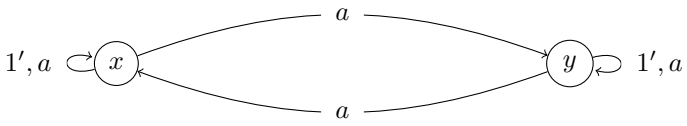
We can think of these representations as labelled directed graphs over the set X.

DEFINITION 32 (Representation Graph) *Representation graph $G = (X, E, \lambda)$ is a labeled directed graph for some instance $\mathcal{O}$ of $R(S)$ with the set of vertices X, the set of edges $E \subseteq X \times X$ and the labelling function $\lambda : E \to \mathcal{P}(\mathcal{O}) \setminus \{\emptyset\}$ where:*

1. $E = \{(x, y) \in X \times X | \exists R \in \mathcal{O} : (x, y) \in R\}$
2. $\lambda(x, y) = \{R | (x, y) \in R\}$

This is best illustrated by an example. Consider a monoid $\{1', a\}$ with $1' \leq a, a; a = a$ and $\breve{a} = a$. This is represented over a set $X = \{x, y\}$ where $a = \{(x, x), (x, y), (y, x), (y, y)\}$. It can be checked that this is indeed a representation.

This representation has a representation graph



These graphs will have certain properties, which we will discuss now.

LEMMA 33 *If $a \not\leq b$ then there must exist an edge (x, y) such that $a \in \lambda(x, y)$ but $b \notin \lambda(x, y)$*

PROOF:

If $a \not\leq b$ then $a^\theta \not\subseteq b^\theta$. This means that $\neg\forall(x, y) \in X \times X : (x, y) \in a^\theta \to (x, y) \in b^\theta$. Or in other words, there exists a pair $(x, y) \in a^\theta$ such that $(x, y) \notin b^\theta$. This means that in the representation graph, there will exist an edge (x, y) such that $a \in \lambda(x, y)$ but $b \notin \lambda(x, y)$ provided $a \not\leq b$. □

LEMMA 34 *The identity element will only be in the label of an edge (x, y) iff $x = y$.*

PROOF:

Since $1'^\theta = \{(x, x) | x \in X\}$, it clearly follows that if $1'$ is in the label $x = y$. Since 1^θ includes (x, x) for all $x \in X$, it is clear that if $x = y$, $1' \in \lambda(x, y)$ □

LEMMA 35 *If $a \in \lambda(x,y)$ where $(x,y) \in X \times X$, then $\breve{a} \in \lambda(y,x)$.*

PROOF:

By definition $\breve{a}^\theta = \{(y,x) | (x,y) \in a^\theta\}$ and therefore, if $a \in \lambda(x,y)$ it must holds that $\breve{a} \in \lambda(y,x)$. $\square$

LEMMA 36 *If $a \in \lambda(x,y)$ where $(x,y) \in X \times X$, then all elements of $a^\uparrow$ are also in (x,y).*

PROOF:

Clearly, if $a \leq a'$ then $(x,y) \in a^\theta \rightarrow (x,y) \in a'^\theta$. Therefore, if $a \in \lambda(x,y)$, it is also true that $a' \in \lambda(x,y)$ for all a' satisfying $a \leq a'$ $\square$

LEMMA 37 *If $a = b;c$ and there exist some edges $(x,y),(y,z)$ such that $b \in \lambda(x,y)$ and $c \in \lambda(y,z)$, it follows that $a \in \lambda(x,z)$*

PROOF:

By definition, $a^\theta = b^\theta ; c^\theta$ contains all pairs (x,z) where $(x,y) \in b^\theta$ and $(y,z) \in c^\theta$. Therefore, $a \in \lambda(x,z)$. $\square$

LEMMA 38 *If $a \leq b;c$ and $a \in \lambda(x,z)$, then there exists a vertex y such that $(x,y) \in b^\theta$ and $(y,z) \in c^\theta$.*

PROOF:

If $a \leq b;c$ then if $(x,z) \in a^\theta$, so is $(x,z) \in (b;c)^\theta$ which is defined as a set of all (x,z) where there exists a y such that $(x,y) \in b^\theta$ and $(y,z) \in c^\theta$. Therefore, in the representation graph, there will exist a vertex y such that $(x,y) \in b^\theta$ and $(y,z) \in c^\theta$, given $a \in \lambda(x,z)$ and $a \leq b;c$ $\square$

3.3 Representation by Games

Now, we would like to check if an ordered convoluted monoid $\mathcal{O}$ is representable. One of the possible strategies for this is by slightly modified representation games as shown in [10]. We define a two-player game played by $\forall$ (we will call him Adam) and $\exists$ (we will call her Eve).

3.3.1 Rules of The Representation Game

The game $\Gamma_n(\mathcal{O})$ is played for n moves. After each move, $\exists$ will be asked to return a graph $G = (V, \lambda)$ where

$$\lambda : V \times V \rightarrow \mathcal{P}(\mathcal{O}) \times \mathcal{P}(\mathcal{O})$$

We say that λ assigns two labels to each pair of vertices (x,y)

$$\lambda(x,y) = (\lambda_\top(x,y), \lambda_\bot(x,y))$$

Where

$$\lambda_\top : V \times V \to \mathcal{P}(\mathcal{O})$$

$$\lambda_\bot : V \times V \to \mathcal{P}(\mathcal{O})$$

Where $\lambda_\top$ is the set of elements of $\mathcal{O}$ assigned to the edge by $\exists$ and $\lambda_\bot$ is the set of forbidden elements of $\mathcal{O}$. In order for $\forall$ to win, he has to request various modifications to the graph G through n moves which will leave $\exists$ with no choice but to assign an element of $\lambda_\bot(x, y)$ to $\lambda_\top(x, y)$. If he fails to do so within the n moves, $\exists$ wins.

To put it more formally, $\exists$ wins if and only if after n moves the network is consistent. We say the network is consistent if and only if:

$$\forall (x, y) \in V \times V : \lambda_\top(x, y) \cap \lambda_\bot(x, y) = \emptyset$$

Initialisation Move The game begins by $\forall$ picking a pair $a, b \in \mathcal{O}$ such that $a \not\leq b$. Now, $\exists$ has two choices. She can decide to initialise G as

$$\begin{aligned} V &= \{x\} \\ \lambda_\top(x, x) &= \{1', a, \breve{a}\} \\ \lambda_\bot(x, x) &= \{b, \breve{b}\} \end{aligned}$$

or alternatively as

$$\begin{aligned} V &= \{x, y\} \\ \lambda_\top(x, y) &= \{a\} \qquad & \lambda_\top(y, x) &= \{\breve{a}\} \\ \lambda_\bot(x, y) &= \{b, 1'\} \qquad & \lambda_\bot(y, x) &= \{\breve{b}, 1'\} \\ \lambda_\top(x, x) = \lambda_\top(y, y) &= \{1'\} \qquad & \lambda_\bot(x, x) = \lambda_\bot(y, y) &= \emptyset \end{aligned}$$

After the initialisation, n moves take place. They can be one of the following:

Order Move $\forall$ may pick $x, y \in V$ and some $a \in \lambda(x, y)_\top$ along with some a^+ for which it holds that $a \leq a^+$. If $a^+ \in \lambda(x, y)_\top$, $\exists$ is not required to do anything. Otherwise, she must add a^+ to $\lambda(x, y)_\top$ and $\breve{(}a^+)$ to $\lambda(y, x)_\top$

Witness Move $\forall$ may pick a pair $x, y \in V$ and some $a; b \in \lambda_\top(x, y)$ and demand to see a witness, i.e. such a vertex z such that $a \in \lambda_\top(x, z)$ and $b \in \lambda_\top(z, y)$. If such a vertex z already exists in V, $\exists$ may do nothing. Otherwise, she may either pick $z \in V$ and add a to $\lambda_\top(x, z)$, b to $\lambda_\top(z, y)$, $\breve{a}$ to $\lambda_\top(z, x)$ and $\breve{b}$ to $\lambda_\top(y, z)$.

Alternatively, add a new vertex, let's call it $w \notin V$ and add w to V. Then she sets $\lambda_\top(w, w) = \{1'\}$ and $\lambda_\bot(w, w) = \emptyset$ Then, for all vertices $v \in V \setminus \{w\}$ set $\lambda_\bot(v, w) = \lambda_\bot(w, v) = \{1'\}$. Then,

for all $v \in V \setminus \{w, x, y\}$ set $\lambda_\top(v, w) = \lambda_\top(w, v) = \emptyset$. Finally, she must also add a to $\lambda_\top(x, w)$, b to $\lambda_\top(w, y)$, $\breve{a}$ to $\lambda_\top(w, x)$ and $\breve{b}$ to $\lambda_\top(y, w)$.

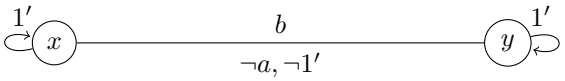
Composition Move $\forall$ may pick $x, y, z \in V$ and some $a \in \lambda_\top(x, y)$ and $b \in \lambda_\top(y, z)$. Now, if $a; b \in \lambda_\top(x, z)$, $\exists$ may do nothing. Otherwise, she must add $a; b$ to $\lambda_\top(x, z)$ and $\breve{b}; \breve{a}$ to $\lambda_\top(z, x)$.

REMARK 39 *When $\forall$ requests ordering, witness or composition moves that do not require $\exists$ to do anything, we call these moves redundant.*

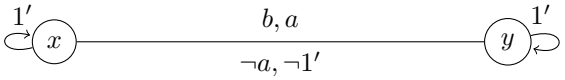
We will illustrate an example run of this game on the ordered convoluted monoid $\mathcal{O}$ with self-converse elements $\{1', a, b\}$. $\leq$ consists of $1' \leq b$, $a \leq b$ and all reflexive pairs. Since the elements are self converse, we will omit the arrows. The composition (of non-identity elements) is defined as

;	a	b
a	1'	b
b	b	b

In the initialisation move, $\forall$ picks a starting pair $b \not\leq a$. And $\exists$ responds with, where (x, y) is the edge we set $\lambda_\top$ to $\{b\}$ and $\lambda_\perp$ to $\{a, 1'\}$. We use the $\neg$ symbol to denote the elements of $\lambda_\perp$.

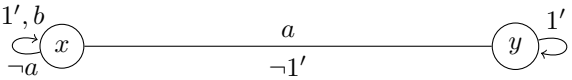


For the first move, $\forall$ says that $1' \leq a; a$, and requests a witness over (x, x). $\exists$ cannot find such a witness, but chooses not add any new vertices and chooses to add a to $\lambda_\top(x, y)$ and $\lambda_\top(y, x)$ and G now looks like this

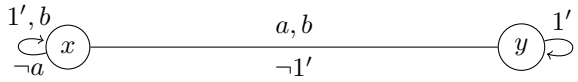


We see that a is both in $\lambda_\top(x, y)$ and $\lambda_\perp(x, y)$. Therefore, $\exists$ lost the game. However, this game could have gone differently. For example, $\exists$ could chose to go for a single vertex representation after $\forall$ demanding $b \not\leq a$ resulting in

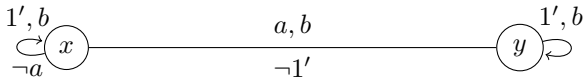
Now, $\forall$ would request to see the witness $1' \leq a; a$ over (x, x), resulting in $\exists$ adding a vertex y



Now, $\forall$ performs an order move and requests b to be added to $\lambda_\top(x, y)$ since $a \leq b$ and $a \in \lambda_\top(x, y)$.



He performs another another order move as requests b to be added to $\lambda_\top(y,y)$ since $1' \leq b$ and $1' \in \lambda_\top(y,y)$. This results in



What we see here, is that $\exists$ has actually produced a graph on which all the requests $\forall$ can make are redundant. This is because this graph is a representation. This is indicative of a theorem presented in Section 3.3.2.

3.3.2 Eve's Winning Strategy Theorem

What we have seen in the previous section is that the resulting graph of the representation game (if we use $\lambda_\top$ in place of λ) can actually be a representation graph. However, what we show in this section is even stronger. We look at the following theorem

THEOREM 40 (Hirsch and Hodkinson) *An ordered convoluted monoid $\mathcal{O}$ has a representation if and only if $\exists$ has a winning strategy for the representation game $\Gamma_n(\mathcal{O})$ for all $n \in \mathbb{N}$*

PROOF:

We first show that if there is a representation, there exists a winning strategy. Assume there exists a representation with a representation graph $G^* = (V^*, E^*, \lambda^*)$. We devise $\exists$'s winning strategy as follows. If the game is played over $G = (V, \lambda)$, we introduce an injective mapping $f : V \to V^*$. And ensure that at any move, for any $x, y \in G : \lambda_\top(x,y) \subseteq \lambda^*(f(x), f(y))$ and $\lambda_\bot(x,y) \cap \lambda^*(f(x), f(y)) = \emptyset$. Thus, it will also hold $\lambda_\bot(x,y) \cap \lambda_\top(x,y) = \emptyset$ and the network will remain consistent throughout the game. We now show by induction that these properties hold throughout the game.

Base Case At initialisation (move 0), $\forall$ demands $a \not\leq b$, $\exists$ picks an edge with the label a, but not b, let's call it $(x^*, y^*) \in E^*$. Such an edge is guaranteed to exist by Lemma 33, noting it might be the case that $x^* = y^*$.

1. If $x^* = y^*$, then $\exists$ adds a single vertex x and sets $f(x) = x^*$. We know that $a, \breve{a} \in \lambda(x^*, x^*)$ and $b, \breve{b} \notin \lambda^*(x^*, x^*)$ by Lemma 35. Also, by Lemma 34 we know $1' \in \lambda(x^*, x^*)$. Therefore, $\lambda_\top(x,y) \subseteq \lambda^*(f(x), f(y))$ and $\lambda_\bot(x,y) \cap \lambda^*(f(x), f(y)) = \emptyset$.
2. If $y^* \neq x^*$ then she adds two vertices x, y and set $f(x) = x^*$ and $f(y) = y^*$. Again, by Lemma 35, we know We know that $\breve{a} \in \lambda^*(y^*, x^*)$ and $\breve{b} \notin \lambda^*(y^*, x^*)$. by Lemma 34 we know $1'$ is in $\lambda^*(x^*, x^*)$, $\lambda^*(y^*, y^*)$, but not in $\lambda^*(x^*, y^*)$ or $\lambda(y^*, x^*)$. Hence, we conclude $\lambda_\top(x,y) \subseteq \lambda(f(x), f(y))$ and $\lambda_\bot(x,y) \cap \lambda^*(f(x), f(y)) = \emptyset$.

Induction now, we check that given for all $v, w \in V$ that $\lambda_\top(v,w) \subseteq \lambda^*(f(v), f(w))$ and $\lambda_\bot(v,w) \cap \lambda^*(f(v), f(w)) = \emptyset$ up to some n-th move (where $n = 0, 1, 2, ...$), we will preserve this property for the $n+1$-st move.

1. First, we check the order move. $\forall$ picked some pair of vertices x, y and some element $a \in \lambda_\top(x,y)$ and some a^+ such that $a \leq a^+$. We know by Lemma 36 that given $a \in \lambda^*(f(x), f(y))$, so it $a^+ \in \lambda^*(f(x), f(y))$ and by Lemma 35 so is $a^+ \in \lambda^*(f(y), f(x))$. Therefore, it still holds that for all $v, w \in V$ that $\lambda_\top(v,w) \subseteq \lambda^*(f(v), f(w))$ and $\lambda_\bot(v,w) \cap \lambda^*(f(v), f(w)) = \emptyset$
2. When $\forall$ requests a witness $(x,y),(y,z)$ with labels including a, b respectively over some other edge (x,z) with $a; b \in \lambda_\top(x,z)$, $\exists$ finds such a $y^* \in V^*$ that $a \in \lambda^*(f(x), y^*)$ and $b \in \lambda^*(y^*, f(z))$. By Lemma 38, it is guaranteed to exist. If there is no $y \in V$ such that $f(y) = y^*$, she must add a new vertex y to V and set $f(y) = y^*$. We know by Lemma 34 that all of the identity labels that will be added, will preserve the property that for any $w, v \in V$ that $\lambda_\top(w,v) \subseteq \lambda^*(f(w), f(v))$ and $\lambda_\bot(w,v) \cap \lambda^*(f(w), f(v)) = \emptyset$. We also know by Lemma 35 that adding $a, \breve{a}, b$ and $\breve{b}$ to $\lambda_\top(x,y), \lambda_\top(y,x), \lambda_\top(y,z)$ and $\lambda_\top(z,y)$ respectively will preserve that property.
3. Finally, we check the composition moves. If $\forall$ requests an edge (x,z) should have $a; b \in \lambda_\top(x,z)$, because some pair of some y such that $a \in \lambda_\top(x,y)$ and $b \in \lambda_\top(y,z)$, she adds it. By Lemma 37, we know that $\lambda^*(f(x), f(z))$ will also contain $a; b$. Therefore, we have shown that the property of any $w, v \in V$ having $\lambda_\top(w,v) \subseteq \lambda^*(f(w), f(v))$ and $\lambda_\bot(w,v) \cap \lambda^*(f(w), f(v)) = \emptyset$ is preserved.

Since $\exists$ can establish the property of any $w, v \in V$ having $\lambda_\top(w,v) \subseteq \lambda^*(f(w), f(v))$ and $\lambda_\bot(w,v) \cap \lambda^*(f(w), f(v)) = \emptyset$ on the initialisation move and preserve it through all other moves, she will consequently keep the network consistent. Therefore, she has a winning strategy for $\Gamma_n(\mathcal{O})$ for all $n \in \mathbb{N}$ if $\mathcal{O}$ is representable.

Now we give an outline of proof that if $\exists$ has a winning strategy for any $\Gamma_n(\mathcal{O})$, $\mathcal{O}$ itself is representable. We only show this for countable algebras $\mathcal{O}$, although the theorem does generalise to uncountable monoids as well - see [10]. We know that $\exists$ has a winning strategy for any Γ_n. For every pair $a \not\leq b$, we consider a play where $\forall$ starts with the initial request of $a \not\leq b$. We know that after n-th round, there arises a set of possible moves, which we enumerate as

$$\mu_{n,0}, \quad \mu_{n,1}, \quad \mu_{n,2}, \quad \ldots$$

We know that, since there network G after the n-th move is finite and the $\mathcal{O}$ is countable, this set is also countable. Therefore, he may schedule his moves in the following way. For $n = 0, 1, 2, 3, ...$ perform the moves $\mu_{i,n-i}$ for $i = 0, 1, 2, ..., n$. To visualise this:

$$\begin{array}{cccc} \mu_{0,0}, & \mu_{0,1}, & \mu_{0,2}, & \ldots \\ & \mu_{1,0}, & \mu_{1,1}, & \ldots \\ & & \mu_{2,0}, & \ldots \\ & & & \ldots \end{array}$$

Schedule the moves as follows:

$$\mu_{0,0},\ \mu_{0,1},\ \mu_{1,0},\ \mu_{0,2},\ \mu_{1,1},\ \mu_{2,0},\ \ldots$$

What we see is that every move will be requested at some point during the game. Another observation we make is that $\exists$ really only has a finite number of choices to make at every move. At composition and order moves, she has no choice at all and with the witness move, she can only choose between the finite number of existing vertices along with the option of adding a new vertex. Now, we consider a game tree where the root vertex is the G returned by $\exists$ after the initialisation move and each vertex's child vertices represent a possible response to the next request made by $\forall$ with the above scheduling. This tree, as we observed, has finite branching, but is itself infinite (as $\exists$ has a winning strategy). Therefore, by König Tree Lemma, it will have an infinite branch.

What we notice is that in its limit, this branch is a graph $G^{a \not\leq b} = (V, \lambda)$ where $\lambda(x, y) = (\lambda_\top(x, y), \lambda_\bot(x, y))$ for all $x, y \in V$. This G has some properties:

1. $\lambda_\top$ represents $\leq$ correctly, as every possible order move is requested at some point
2. $\lambda_\top$ represents $\smile$ correctly, as every time some a is added to some $\lambda_\top(x, y)$, she adds $\breve{a}$ to $\lambda_\top(y, x)$
3. $\lambda_\top$ represents $1'$ correctly, as $1'$ is added to $\lambda_\top(x, x)$ for all $x \in V$ and for all $x \neq y \in V : 1' \in \lambda_\bot(x, y)$ and $\lambda_\bot(x, y) \cap \lambda_\top(x, y) = \emptyset$.
4. $\lambda_\top$ represents ; correctly as all composition and witness moves will eventually be performed by $\forall$
5. There exists a pair $x, y \in V$ such that $a \in \lambda_\top(x, y)$ but $b \notin \lambda_\top(x, y)$

Now, we take a disjoint union of all such graphs

$$(V^*, \lambda^*) = \bigcup_{a \not\leq b \in \mathcal{O}} G^{a \not\leq b}$$

And define G^* as:

$$G^* = (V^*, \lambda^*_\top)$$

We see that all $\leq, 1', \smile, ;$ are still represented correctly, however, more importantly, we also see that

$$\forall a \not\leq b : \exists x, y \in V^* : a \in \lambda^*_\top(x, y) \wedge b \notin \lambda^*_\top(x, y)$$

In other words, G^* is also faithful. Therefore, we can only conclude that G^* is a

representation graph. This means that given $\exists$ has a winning strategy for any $\Gamma_n(\mathcal{O})$ for $n = 0, 1, 2, 3, ...$, $\mathcal{O}$ will have a representation. $\square$

3.4 Representability of Ordered Convoluted Monoids

We now have all the tools to consider an example of an unrepresentable ordered convoluted monoid. We will discuss in Chapter 4 what the axioms that it does not follow are and why will this guarantee it is not representable. We also discuss in Appendix A how to use an automated first order logic theorem prover and counterexample finder to obtain it. But for now, we will use it to show

THEOREM 41 *The axiomatisation of convoluted ordered monoids is not sufficient for the signature* $(1', \leq, \smile, ;)$ *to guarantee representability.*

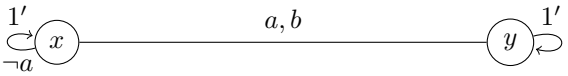
PROOF:

Consider the ordered convoluted monoid with self-converse elements $\{1', a, b\}$, the $\leq$ including only the reflexive pairs and composition defined as

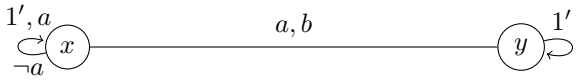
;	1'	a	b
1'	1'	a	b
a	a	b	1'
b	b	1'	a

It can be easily checked that it obeys the axioms of an ordered convoluted monoid. Now consider the game played over this. $\forall$ picks $1' \not\leq a$. $\exists$ can only pick a single vertex representation of $1'$ as $1' \leq 1'$. Since all elements are self-converse, we omit arrows.

$\forall$ now requests a witness $1' \leq a; b$ over (x, x). Since $\exists$ cannot add these two labels to (x, x) as she would lose the game, she has to add y resulting in



However, since $b \in \lambda_\top(x, y)$ and $b \in \lambda_\top(y, x)$ $\forall$ can now request the composition move on (x, x). Since $b; b = a$, $\exists$ must add a to $\lambda_\top(x, x)$. This results in



This network is not consistent and $\exists$ loses the game. What we can also see is that she didn't have really a choice in any of the moves. Therefore, we have performed an exhaustive inspection which revealed that $\exists$ cannot have a winning strategy. And by Theorem 40, this means that this ordered convoluted monoid has no representation. Therefore, we have found a counterexample demonstrating that the axioms of a convoluted ordered monoid are not sufficient to guarantee representations. $\square$

Chapter 4

Recursive Axiomatisation of The Signature

4.1 Non-Finite Axiomatisability Conjecture

What we can observe from Section 3.4 is that there exist axioms that are sound, but not covered by the axiomatisation of ordered convoluted monoids. We currently hypothesise that the signature itself is not finitely axiomatisable. We begin our argument with the following observation

PROPOSITION 42 $\forall R \subseteq X \times X : R \subseteq R; \breve{R}; R$

PROOF:

Let R be a relation over X. As ; is associative (See Proposition 5), we can express $R; \breve{R}; R$ as

$$R; \breve{R}; R = \{(x, y) | \exists z, w \in X : (x, z) \in R \wedge (z, w) \in \breve{R} \wedge (w, y) \in R\}$$

or simply

$$R; \breve{R}; R = \{(x, y) | \exists z, w \in X : (x, z), (w, z), (w, y) \in R\}$$

Now, take any $(x, y) \in R$. Let $z = y$ and $w = x$. We can see that $(x, z), (w, z), (w, y) \in R$. Therefore, there exist such a pair z, w that guarantees (x, y) to also be in $R; \breve{R}; R$. From this it follows that $R \subseteq R; \breve{R}; R$ for any $R \subseteq X \times X$. $\square$

From this, it follows that we should add $\forall a : a \leq a; \breve{a}; a$ to the axiomatisation. And we can see, in the counterexample used in Proof of Theorem 41, $a; \breve{a}; a = b; a = 1'$ and $a \not\leq 1'$.

Now, it turns out that this is also not a sufficient axiomatisation. Consider the following property

PROPOSITION 43 $\forall R, S, T \subseteq X \times X : R \subseteq S; T \rightarrow R \leq S; \breve{S}; R$

PROOF:

Let R, S, T be relations over X. We define $S; \breve{S}; R$ as

$$S; \breve{S}; R = \{(x, y) | \exists z, w \in X : (x, z) \in S \wedge (z, w) \in \breve{S} \wedge (w, y) \in R\}$$

or equivalently

$$S; \breve{S}; R = \{(x, y) | \exists z, w \in X : (x, z), (w, z) \in S \wedge (w, y) \in R\}$$

Now consider any $(x, y) \in R$. Given $R \subseteq S; T$, we know from Lemma 38 that there exists a z such that $(x, z) \in S$ and $(z, y) \in S$. Now, let $w = x$ and we see that $(w, y) \in R$ and $(w, z) \in S$. Therefore, we see that there exist such z, w for all $(x, y) \in R$ that guarantees $(x, y) \in S; \breve{S}; R$. Therefore $R \subseteq S; \breve{S}; R$ given $R \subseteq S; T$. □

Now, the question is if the axiom $\forall a, b, c : c \leq a; b \rightarrow c \leq a; \breve{a}; c$ can be implied from the axioms we have considered so far. The answer turns out to be no. We consider the ordered convoluted monoid with three self-converse elements $\{1', a, b\}$. $\leq$ includes the reflexive pairs, along with the $1' \leq b$ and $a \leq b$ and the composition is defined as

;	1'	a	b
1'	1'	a	b
a	a	a	b
b	b	b	b

One can check that this is indeed a convoluted ordered monoid that also follows the axiom of $\forall a : a \leq a; \breve{a}; a$. However, $1' \leq a; b$, but $1' \not\leq a; \breve{a}; 1' = a; a; 1' = a$. Therefore, we found a counterexample to the claim that $\forall a, b, c : c \leq a; b \rightarrow c \leq a; \breve{a}; c$ can be implied from the existing axiomatisation.

Now, we could show that this algebra will not have a representation by games, but we can simply imply it from the following.

LEMMA 44 *If an algebra does not satisfy a sound axiom, it does not have a representation.*

PROOF:

Assume that there exists a representation of this algebra. Since the representation is isomorphic to the algebra, it is true that the axiom does not hold for the representation. However, the axiom is sound, therefore it holds for any algebra of relations. We have reached a contradiction and our assumption is wrong. Therefore, such an algebra does not have a representation.

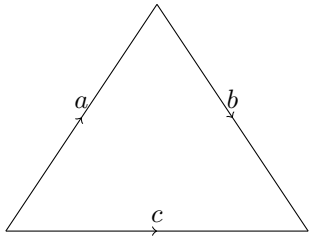
□

What we have seen so far are some examples of axioms that are not in the axiomatisation of ordered convoluted monoids, but are necessary to guarantee representations. More details on how to generate these counterexamples presented using Mace [13] can be found in Appendix A.

We continue our search for further axioms with a simple observation. $\forall a : a \leq a; \breve{a}; a$ follows from $\forall a, b, c : c \leq a; b \rightarrow c \leq a; \breve{a}; c$ as $a \leq a; 1'$ from which it follows that $a \leq a; \breve{a}; a$. Similarly,

one can deduce that $\forall a, b, c : c \leq a; b \to c \leq c; \breve{b}; b$ as $c \leq a; b \to \breve{c} \leq \breve{b}; \breve{a}$ by monotonicity and converse of composition. And from that it follows that $\breve{c} \leq \breve{b}; b; \breve{c}$ and hence $c \leq c; \breve{b}; b$.

We visualise this as



We have now added the additional axiom (let's call it $\sigma_a = \forall a, b, c : c \leq a; b \to c \leq a; \breve{a}; c$) to our axiomatisation. However, we hypothesise that the following axiom, let's call it σ_b, cannot be shown from this new axiomatisation

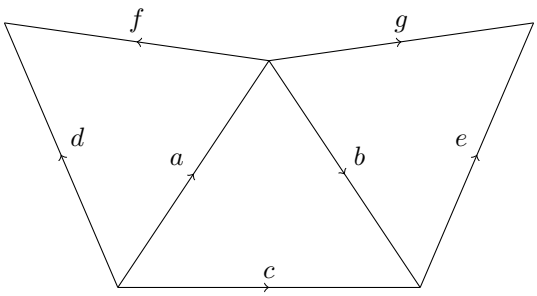
$$\forall a, b, c, d, e, f, g :$$

$$c \leq a; b \quad \wedge \quad a \leq d; \breve{f} \quad \wedge \quad b \leq g; \breve{e}$$

$$\downarrow$$

$$c \leq a; g; \breve{g}; f; \breve{f}; b \quad \wedge \quad c \leq a; g; \breve{g}; f; \breve{f}; \breve{a}; c$$

Again, this is best visualised as follows.



Although, we believe that there exists a counter-example of an algebra that would follow the basic axiomatisation and σ_a, but not σ_b, the counter example searcher did not find one, as the size of it is likely to be a lot higher than what is feasible to compute in terms of time.

Now, we know that σ_a is sound for algebras of binary relations, but we will not provide a proof at the moment. We will rather show it for the more general case in Proposition 45.

What we will hypothesise further is that σ_a, σ_b and the axioms from the basic axiomatisation will not be enough to infer the following (we will call it σ_c)

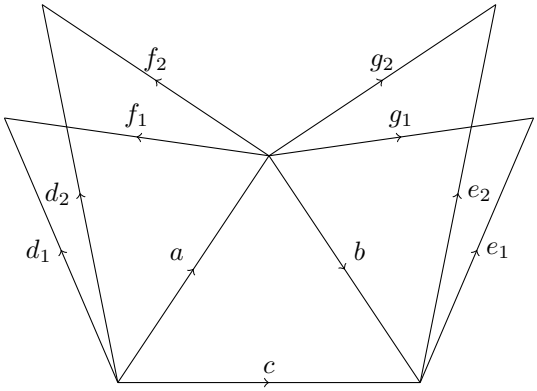
$$\forall a, b, c, d_1, d_2 e_1, e_2, f_1, f_2, g_1, g_2 :$$

$$c \leq a; b \quad \wedge \quad a \leq d_1; \breve{f}_1 \quad \wedge \quad b \leq g_1; \breve{e}_1 \quad \wedge \quad a \leq d_1; \breve{f}_1 \quad \wedge \quad b \leq g_1; \breve{e}_1$$

$$\downarrow$$

$$c \leq a; g_1; \breve{g}_1; f_1; \breve{f}_1; g_2; \breve{g}_2; f_2; \breve{f}_2; b \quad \wedge \quad c \leq a; g_1; \breve{g}_1; f_1; \breve{f}_1; g_2; \breve{g}_2; f_2; \breve{f}_2; \breve{a}; c$$

This is visualised as



We now have an intuition on the generalised formula $\sigma(n)$ which is defined as follows:

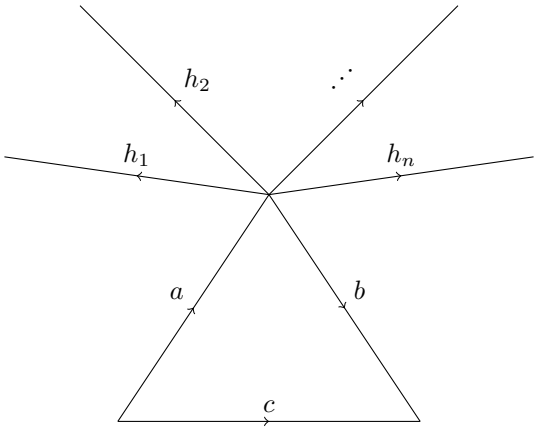
$$\sigma(n) = \forall a, b, c, h_1, h_2, ..., h_n :$$

$$c \leq a; b \quad \wedge \quad \bigwedge_{i=1}^{n} \Big(a \leq a; h_i; \breve{h_i} \quad \vee \quad b \leq h_i; \breve{h_i}; b \Big)$$

$$\downarrow$$

$$c \leq a; h_1; \breve{h_1}; h_2; \breve{h_2}; ...; h_n; \breve{h_n}; b \quad \wedge \quad c \leq a; h_1; \breve{h_1}; h_2; \breve{h_2}; ...; h_n; \breve{h_n}; \breve{a}; c$$

We visualise it as



To gain some further intuition, we can see, how we can infer σ_c from $\sigma_d(n)$. Take any $a, b, c, d_1, d_2 e_1, e_2, f_1, f_2, g_1, g_2$ such that:

$$c \leq a; b \quad \wedge \quad a \leq d_1; \breve{f_1} \quad \wedge \quad b \leq g_1; \breve{e_1} \quad \wedge \quad a \leq d_1; \breve{f_1} \quad \wedge \quad b \leq g_1; \breve{e_1}$$

Let $h_1 = f_1, h_2 = g_1, h_3 = f_f, h_4 = g_2$. We know that $a \leq a; h_1; \breve{h_1}, a \leq a; h_3; \breve{h_3}$ and $b \leq h_2; \breve{h_2}; b, b \leq h_2; \breve{h_2}; b$. From this, we can infer from $\sigma(4)$ that

$$c \leq a; g_1; \breve{g_1}; f_1; \breve{f_1}; g_2; \breve{g_2}; f_2; \breve{f_2}; b \quad \wedge \quad c \leq a; g_1; \breve{g_1}; f_1; \breve{f_1}; g_2; \breve{g_2}; f_2; \breve{f_2}; \breve{a}; c$$

Similarly, we can infer σ_a from $\sigma_d(0)$ and σ_b from $\sigma_d(2)$. However, we conjecture, one cannot infer $\sigma_d(n+1)$ from $\sigma_d(n)$. However, what we do know about $\sigma_d(n)$ is the following

PROPOSITION 45 *For any $n \in \mathbb{N} \cup \{0\}$, $\sigma(n)$ is sound for binary relations.*

PROOF:

We know that in a representation over X of some $\mathcal{O}$. Take any three elements $a, b, c \in \mathcal{O}$ such that $c \leq a; b$, we know by Lemma 38 that for any $x, y \in X$ given $(x, y) \in c^\theta$, there exists z such that $(x, z) \in a^\theta$ and $(z, y) \in b^\theta$. Now, given that all h_i for $i = 1, 2, 3, ..., n$ it holds that

$$a \leq a; h_i; \breve{h_i} \quad \vee \quad b \leq h_i; \breve{h_i}; b$$

we can see again by Lemma 38 that there exists such a w_i such that $(z, w_i) \in h_i^\theta$ along with $(w_i, x) \in \breve{h_i}; \breve{a}$ or $(w_i, y) \in \breve{h_i}; b$. Furthermore, we can see that by Lemma 35, $(w_i, z) \in \breve{h_i}^\theta$. Now, we can see by Lemma 37 since $(w_i, z) \in \breve{h_i}^\theta$ and $(z, w_i) \in h_i^\theta$, we know that $(z, z) \in (h_i; \breve{h_i})^\theta$.

Moreover from Lemma 37 it also then follows that

$$(z, z) \in (h_1; \breve{h_1}; h_1; \breve{h_1}; ...; h_n; \breve{h_n})^\theta$$

and hence since $(x, z) \in a^\theta$ and $(z, y) \in b^\theta$

$$(x, y) \in (a; h_1; \breve{h_1}; h_1; \breve{h_1}; ...; h_n; \breve{h_n}; b)^\theta$$

given $(x, y) \in c^\theta$. This means that the axiom is indeed sound over binary relations. □

What we have observed in this section is an infinite set of first order formulas $\sigma_d(n)$ that are all sound for these, however, we conjecture that $\sigma_d(n) \nvdash \sigma_d(n+1)$. Furthermore, this leads us to believe that the class itself might not be finitely axiomatisable.

4.2 Recursive Axiomatisation of The Signature

Now that we have the intuition on why the signature is hard to axiomatise, we want to define an infinite set of sentences ϕ_n which will imply that an ordered convoluted monoid $\mathcal{O}$ will be representable if and only if $\mathcal{O} \models \phi_n$ for all $n \in \mathbb{N}$. To do that, we will first need to define some notation for network graphs.

Recall that the game $\Gamma_n(\mathcal{O})$ is played over a complete graph with $G = (V, \lambda)$ where

$$\lambda : V \times V \to \mathcal{P}(\mathcal{O}) \times \mathcal{P}(\mathcal{O})$$

and for all $x, y \in V$:

$$\lambda(x, y) = (\lambda_\top(x, y), \lambda_\bot(x, y))$$

where

$$\lambda_\top : V \times V \to \mathcal{P}(\mathcal{O})$$

$$\lambda_\perp : V \times V \to \mathcal{P}(\mathcal{O})$$

The network is said to be consistent if and only if for all $x, y \in V$:

$$\lambda_\top(x,y) \cap \lambda_\perp(x,y) = \emptyset$$

We will introduce some operations and constants for these graphs

DEFINITION 46 (Label Addition) *Given a network graph $G = (V, (\lambda_\top, \lambda_\perp))$, for some $x, y \in V$ and $a \in \mathcal{O}$ to, we denote $G^\top[x,y,a]$ to be the graph*

$$G^\top[x,y,a] = (V, (\lambda'_\top, \lambda_\perp))$$

where for all $v, w \in V$ such that $(v,w) \notin \{(x,y),(y,x)\}$

$$\lambda'_\top(v,w) = \lambda_\top(v,w)$$

and

$$\lambda'_\top(x,y) = \lambda_\top(x,y) \cup \{a\} \qquad \lambda'_\top(y,x) = \lambda_\top(y,x) \cup \{\breve{a}\}$$

Similarly, we define for some $x, y \in V$ and $a \in \mathcal{O}$ to, we denote $G^\perp[x,y,a]$ to be the graph $G^\perp[x,y,a] = (V, (\lambda_\top, \lambda'_\perp))$ where for all $v, w \in V$ such that $(v,w) \notin \{(x,y),(y,x)\}$ we define $\lambda'_\perp(v,w) = \lambda_\perp(v,w)$ and

$$\lambda'_\perp(x,y) = \lambda_\perp(x,y) \cup \{a\} \qquad \lambda'_\perp(y,x) = \lambda_\perp(y,x) \cup \{\breve{a}\}$$

DEFINITION 47 (Vertex Addition) *Given a network graph $G = (V, (\lambda_\top, \lambda_\perp))$, for some $x, y \in V$ and $a, b \in \mathcal{O}$, we define $G^+[x,y,a,b]$ to be the graph*

$$G^+[x,y,a,b] = (V \cup \{z\}, (\lambda'_\top, \lambda'_\perp))$$

where $z \notin V$ and, for all $x, y \in V$:

$$\lambda'_\top(x,y) = \lambda_\top(x,y) \qquad \lambda'_\perp(x,y) = \lambda_\perp(x,y)$$

and for all $w \in V \setminus \{x,y\}$:

$$\lambda'_\top(w,z) = \lambda'_\top(z,w) = \emptyset \qquad \lambda'_\perp(w,z) = \lambda'_\perp(z,w) = \{1'\}$$

and finally

$$\begin{aligned}
\lambda'_\top(x,z) &= \{a\} & \lambda'_\perp(x,z) &= \lambda'_\perp(z,x) = \{1'\} \\
\lambda'_\top(z,x) &= \{\breve{a}\} & \lambda'_\perp(y,z) &= \lambda'_\perp(z,y) = \{1'\} \\
\lambda'_\top(z,y) &= \{b\} & \lambda'_\top(z,z) &= \{1'\} \\
\lambda'_\top(y,z) &= \{\breve{b}\} & \lambda'_\perp(z,z) &= \emptyset
\end{aligned}$$

DEFINITION 48 ($G_1^{\not\leq}$) *Given some $a, b \in \mathcal{O}$, we define $G_1^{\not\leq}[a,b]$ as*

$$G_1^{\not\leq}[a,b] = (\{x\}, (\lambda_\top, \lambda_\bot))$$

where

$$\lambda_\top(x,x) = \{1', a, \breve{a}\}$$
$$\lambda_\bot(x,x) = \{b, \breve{b}\}$$

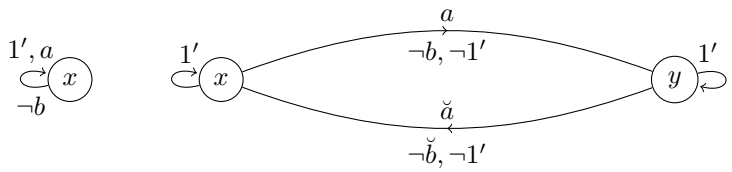
DEFINITION 49 ($G_2^{\not\leq}$) *Given some $a, b \in \mathcal{O}$, we define $G_2^{\not\leq}[a,b]$ as*

$$G_2^{\not\leq}[a,b] = (\{x,y\}, (\lambda_\top, \lambda_\bot))$$

where

$$\lambda_\top(x,y) = \{a\} \qquad \lambda_\top(y,x) = \{\breve{a}\}$$
$$\lambda_\bot(x,y) = \{b, 1'\} \qquad \lambda_\bot(y,x) = \{\breve{b}, 1'\}$$
$$\lambda_\top(x,x) = \lambda_\top(y,y) = \{1'\} \qquad \lambda_\bot(x,x) = \lambda_\bot(y,y) = \emptyset$$

We visualise $G_1^{\not\leq}[a,b]$ (left) and $G_2^{\not\leq}[a,b]$ (right)



We now have all the tools to define a first order formula $\sigma_n(G)$ for some $G = (V, (\lambda_\top, \lambda_\bot))$ where $n = 0, 1, 2, ...$

$$\sigma_0(G) = \bigwedge_{x,y \in V} \bigwedge_{a \in \lambda_\top(x,y)} \bigwedge_{b \in \lambda_\bot(x,y)} \neg(a = b)$$

and

$$\begin{aligned}\sigma_{n+1}(G) = &\bigwedge_{x,y \in V} \bigwedge_{a \in \lambda_\top(x,y)} \forall b : (a \leq b \to \sigma_n(G^\top[x,y,b])) \qquad \wedge \\ &\bigwedge_{x,y,z \in V} \bigwedge_{a \in \lambda_\top(x,y)} \bigwedge_{b \in \lambda_\top(y,z)} \sigma_n(G^\top[x,z,a;b]) \qquad \wedge \\ &\bigwedge_{x,y \in V} \bigwedge_{c \in \lambda_\top(x,y)} \forall a \forall b : \left(c = a;b \to \left(\bigvee_{z \in V} \sigma_n\left(\left(G^\top[x,z,a]\right)^\top[z,y,b]\right) \vee \sigma_n\left(G^+[x,y,a,b]\right)\right)\right)\end{aligned}$$

LEMMA 50 *In a representation game $\Gamma(\mathcal{O})$, if after some move, $\exists$ returns a network graph G, she will have a winning strategy for the next n moves if and only if*

$$\mathcal{O}, G \models \sigma_n(G)$$

PROOF:

We show this by induction over n.

Base Case First, we show that she has a winning strategy for 0 more moves if and only if

$$\mathcal{O}, G \models \sigma_0(G)$$

We know $\exists$ has a winning strategy for 0 more moves if and only if the network $G = (V, (\lambda_\top, \lambda_\bot))$ is consistent, i.e.

$$\forall x, y \in V : \lambda_\top(x, y) \cap \lambda_\bot(x, y) = \emptyset$$

which is true if and only if

$$\bigwedge_{x,y \in V} \bigwedge_{a \in \lambda_\top(x,y)} \bigwedge_{b \in \lambda_\bot(x,y)} \neg(a = b)$$

i.e.

$$\mathcal{O}, G \models \sigma_0(G)$$

Induction Now, given that for any G' $\exists$ will have a winning strategy for n more moves iff $\mathcal{O}, G \models \sigma_n(G)$, it will also be true that for any G $\exists$ will have a winning strategy for $n+1$ more moves iff

$$\mathcal{O}, G \models \sigma_{n+1}(G)$$

We know $\exists$ will have a winning strategy for $n+1$ more moves if and only if she will have a winning strategy after responding to any move adam may make:

- Order Moves:

  We know from the rules of the game that $\exists$ has a winning strategy for $n+1$ moves when the first move is an order move on some graph $G = (V, (\lambda_\top, \lambda_\bot))$ if and only if for any two $x, y \in V$ in the graph and any $a \in \lambda_\top(x, y)$ it is true for all $b \in \mathcal{O}$ such that $a \leq b$ that $\exists$ will have a winning strategy for $G^\top[x, y, b]$. And since, by induction hypothesis, she will have a winning strategy for n more moves on $G^\top[x, y, b]$ if and only if $\mathcal{O}, G^\top[x, y, b] \models \sigma_n(G^\top[x, y, b])$, we can see that she will have a winning strategy for $n+1$ moves if faced with an order move if and only if

$$\mathcal{O}, G \models \bigwedge_{x,y \in V} \bigwedge_{a \in \lambda_\top(x,y)} \forall b : (a \leq b \rightarrow \sigma_n(G^\top[x, y, b]))$$

- Composition Moves:

  rules of the game dictate that $\exists$ has a winning strategy for $n+1$ given the first move is a composition move on some graph $G = (V, (\lambda_\top, \lambda_\bot))$ if and only if for any three $x, y, z \in V$ in the graph and any $a \in \lambda_\top(x, y)$ and any $b \in \lambda_\top(y, z)$, she will have a winning strategy for n moves after adding any $a; b$ to $\lambda_\top(x, z)$. By induction hypothesis this is true if and only if $\mathcal{O}, G^\top[x, y, a; b] \models \sigma_n(G^\top[x, y, a; b])$. Therefore, $\exists$ has a winning strategy for $n+1$ moves when faced with any composition move if and only if

$$\mathcal{O}, G \models \bigwedge_{x,y,z \in V} \bigwedge_{a \in \lambda_\top(x,y)} \bigwedge_{b \in \lambda_\top(y,z)} \sigma_n(G^\top[x, z, c])$$

- Witness Moves:

  It follows from the rules of the game, $\exists$ will have a winning strategy for $n+1$ moves given the first move is the witness move on some graph $G = (V, (\lambda_\top, \lambda_\bot))$ if and only if for any pair of vertices $x, y \in V$ and every $c \in \lambda_\top(x, y)$ and every pair $a, b \in \mathcal{O}$ such that $c = a; b$ at least one of the following will be true:

  1. She will be able to find some vertex z and retain a winning strategy for n moves after adding a to $\lambda_\top(x, z)$ and b to $\lambda_\top(z, y)$. By induction hypothesis, she will retain the winning strategy iff $\mathcal{O}, G \models \sigma_n\left(\left(G^\top[x, z, a]\right)^\top[z, y, b]\right)$
  2. She will be able add a new vertex w to V and add $\lambda_\top(x, w)$ and b to $\lambda_\top(z, w)$ alongside other appropriate labels and continue to have a winning strategy for n more moves. By the induction hypothesis, this will hold iff $\mathcal{O}, G \models \sigma_n\left(G^+[x, y, a, b]\right)$

  Therefore, she will have a winning strategy for $n+1$ moves, provided the first move is a witness move iff

$$\mathcal{O}, G \models \bigwedge_{x,y \in V} \bigwedge_{c \in \lambda_\top(x,y)} \forall a \forall b : \left(c = a; b \rightarrow \left(\bigvee_{z \in V} \sigma_n\left(\left(G^\top[x, z, a]\right)^\top[z, y, b]\right) \vee \sigma_n\left(G^+[x, y, a, b]\right)\right)\right)$$

As the first move can be any of the three, $\exists$ has a winning strategy (given that for any G' $\exists$ has a winning strategy for n more moves given $\mathcal{O}, G' \models \sigma_n(G')$) for $n+1$ more moves on any G if and only iff $\mathcal{O}, G$ semantically entail the disjunction of these three clauses, which turns out to be σ_{n+1}.

Therefore, we have shown by induction that given $\exists$ has returned some network G, she will have a winning strategy for n more moves if and only if

$$\mathcal{O}, G \models \sigma_n(G)$$

where $n = 0, 1, 2, ...$ $\square$

Now, consider the following:

DEFINITION 51 (ϕ_n) *Let ϕ_n be a first order sentence*

$$\phi_n = \forall a \forall b : \Big(\neg(a \leq b) \rightarrow (\sigma_n(G_1^{\not\leq}[a,b]) \vee \sigma_n(G_2^{\not\leq}[a,b]))\Big)$$

THEOREM 52 *An ordered convoluted monoid $\mathcal{O}$ is representable if and only if*

$$\mathcal{O} \models \{\phi_n | n \in \mathbb{N}\}$$

PROOF:

When playing $\Gamma_n(\mathcal{O})$, $\forall$ may request at the initialisation any pair $a \not\leq b \in \mathcal{O}$. By Lemma 50, $\exists$ will have a winning strategy for any game of length n played on some network G if and only if $\sigma_n(G)$. Since $\exists$ has a choice of returning $G_1^{\not\leq}[a,b]$ or $G_2^{\not\leq}[a,b]$ in the first move, she must only have a winning strategy for at least one. Therefore, she will have a winning strategy for n moves given any requested $a \not\leq b$ if and only if

$$\mathcal{O} \models \forall a \forall b : \Big(\neg(a \leq b) \rightarrow (\sigma_n(G_1^{\not\leq}[a,b]) \vee \sigma_n(G_2^{\not\leq}[a,b]))\Big)$$

By Theorem 40, $\mathcal{O}$ will have a representation if and only if for any $n \in \mathbb{N}$. Therefore, we conclude that $\mathcal{O}$ is representable if and only if

$$\mathcal{O} \models \{\phi_n | n \in \mathbb{N}\}$$

□

Chapter 5

Finitely Axiomatisable Classes

We have seen an argument for our conjecture that the signature $(\leq, 1', \smile, ;)$ is not finitely axiomatisable. However, we will show some subclasses of algebras of relations which can be finitely axiomatised for this signature.

5.1 Total Relations

First we will look at total relations over a set X. A relation R is total if and only if $\forall x \in X : \exists y \in X : (x, y) \in R$. This proves to have an interesting consequence

LEMMA 53 *The axiom $\forall a : 1' \leq a; \breve{a}$ is sound for algebras of total relations.*

PROOF:

We know that for any $x \in X$, there exists a y such that $(x, y) \in R$. Therefore, therefore, we also know that $(y, x) \in \breve{R}$. Hence $\forall x \in X : (x, x) \in R; \breve{R}$. This means that every element of $1'$ is also in any $R; \breve{R}$ where R is a total relation. $\square$

We now consider the axiomatisation of algebras of total relations in the context of the signature of ordered convoluted monoids. We will show the following

THEOREM 54 *The class of representable algebras with the signature $(\leq, 1', \smile, ;)$ is finitely axiomatisable for total relations.*

PROOF:

To prove this theorem, we first propose an axiomatisation:

- $(1', ;)$ is a monoid
- $\leq$ is a partial order
- $\smile$ is a converse, i.e. $\forall a : a = \breve{\breve{a}}$, $\breve{1'} = 1'$ and $\forall a, b : (a; b)^{\smile} = \breve{b}; \breve{a}$
- $\smile, ;$ are monotone over $\leq$
- $\forall a : 1' \leq a; \breve{a}$

We see that this is the basic axiomatisation of the Ordered Convoluted Monoids along with this new totality axiom. We have already shown these are all sound in the context of total relations.

We will show that all such Ordered Convoluted Monoids of Total Relations $\mathcal{O}$ are representable over the set of all upwardly closed sets $\Upsilon(\mathcal{O})$. We define $\theta : \mathcal{O} \to \mathcal{P}(\Upsilon(\mathcal{O}) \times \Upsilon(\mathcal{O}))$ such that

$$\forall a \in \mathcal{O} : a^\theta = \{(S,T) : S; a \subseteq T \wedge T; \breve{a} \subseteq S\}$$

We will now show that this is indeed a representation. To do that, we must check that it is faithful and that all the operations and predicates are represented correctly. We do that through Lemmata 55-59. $\square$

LEMMA 55 *The mapping θ is faithful.*

PROOF:

We need to show that $\forall a, b \in \mathcal{O}, a \not\leq b$ it holds that $a^\theta \not\subseteq b^\theta$. Take the element $((1')^\uparrow, a^\uparrow) \in \Upsilon(\mathcal{O}) \times \Upsilon(\mathcal{O})$. Since $\forall c : 1' \leq c \to 1'; a \leq c; a \to a \leq c; a \to c; a \in a^\uparrow$ it holds that $(1')^\uparrow; a \subseteq a^\uparrow$. Since $1' \leq a; \breve{a}$ and $\forall a' : a \leq a' \to a; \breve{a} \leq a'; \breve{a} \to 1' \leq a'; \breve{a}$ it is true that $a^\uparrow; \breve{a} \subseteq (1')^\uparrow$. Therefore, $((1')^\uparrow, a^\uparrow) \in a^\theta$. On the other hand $a \not\leq 1'; b$ therefore, $(1')^\uparrow; b \not\subseteq a^\uparrow$ and $((1')^\uparrow, a^\uparrow) \notin b^\theta$. $\square$

LEMMA 56 $\leq$ *is represented correctly.*

PROOF:

Take any upwardly closed set S. It is true by monotonicity that $a \leq b$ implies $\forall s \in S : s; a \leq s; b$ and therefore the upwardly closed set $(S; a)^\uparrow$ will contain all elements of form $s; b$, i.e. all elements of $(S; b)$ and consequently $(S; b)^\uparrow$. Hence, if $a \leq b$ then $\forall S : (S; b)^\uparrow \subseteq (S; a)^\uparrow \subseteq T$ and similarly $(T; \breve{b})^\uparrow \subseteq S$. Therefore, if $(S,T) \in a$, then $(S,T) \in b$ or $a^\theta \subseteq b^\theta$.

$\square$

LEMMA 57 $1'$ *is represented correctly.*

PROOF:

Here, we need to show that $(S,T) \in 1'^\theta \leftrightarrow S = T$. $\leftarrow$ is trivial, so we focus on $\to$. Since if $(S,T) \in 1'^\theta$, it follows $S; 1' \subseteq T$ and $T; \breve{1'} = T; 1' \subseteq S$ it is true that $S \subseteq T$ and $T \subseteq S$, or stated simply $S = T$. $\square$

LEMMA 58 $\smile$ *is represented correctly.*

PROOF:

Take any $(S,T) \in a^\theta$ for any a. Since $T; \breve{a} \subseteq S$ and $S; a = S; \breve{\breve{a}} \in T$ it follows that $(T,S) \in \breve{a}^\theta$. $\square$

LEMMA 59 ; *is represented correctly.*

PROOF:

For any pair a, b take any $(S,T) \in a^\theta$ and any $(T,V) \in b^\theta$. Since $S;a \subseteq T$ and $T;b \subseteq T$ it is true (by associativity of ;) that $S;(a;b) \subseteq V$. Similarly, since $V;\breve{b} \subseteq T$ and $T;\breve{a} \subseteq S$ it is true that $V;\breve{b};\breve{a} \subseteq S$. And since $(a,b)^\smile = \breve{b};\breve{a}$, it holds that $V;(a,b)^\smile \subseteq S$. Therefore if $(S,T) \in a^\theta$ and $(T,V) \in b^\theta$ then $(S,V) \in (a;b)^\theta$.

Conversely, take any $(S,V) \in (a;b)^\theta$. Let $T = (S;a)^\uparrow \cup (V;\breve{b})^\uparrow$. Clearly $S;a \subseteq T$ and $T;\breve{b} \subseteq S$. We know that $S;a;b \subseteq V$ and since by $1' \leq b;\breve{b}$ it is true $\forall v \in V$ that $v;b;\breve{b} \in V$ or put simply $V;b;\breve{b} \subseteq V$, we know that $(T,V) \in b^\theta$. We can similarly show $(S,V) \in a^\theta$. It is also clear that a union of two upwardly closed sets is also a closed set, therefore for any $(S,V) \in (a;b)^\theta$, there exists an upwardly closed set T such that $(S,T) \in a^\theta$ and $(T,V) \in b^\theta$. □

Now that we have shown that the class of total relations is finitely axiomatisable, we can observe another neat property

COROLLARY 60 *All ordered convoluted monoids of total relations are finitely representable.*

PROOF:

We know that an ordered convoluted monoid of total relations $\mathcal{O}$ has a representation over the set $\Upsilon(\mathcal{O})$. By definition, $\Upsilon(\mathcal{O}) \subseteq \mathcal{P}(\mathcal{O})$ for which it holds that $|\mathcal{P}(\mathcal{O})| = 2^{|\mathcal{O}|}$. Therefore, $|\Upsilon(\mathcal{O})| \leq 2^{|\mathcal{O}|}$ which is finite since $|\mathcal{O}|$ is finite. □

Furthermore, we can conclude

COROLLARY 61 *An ordered convoluted monoid of total relations $\mathcal{O}$ is guaranteed a representation of size $2^{|\mathcal{O}|}$ or smaller.*

Moreover, we can see

COROLLARY 62 *The decision problem of whether a finite algebra $\mathcal{O}$ is an ordered convoluted monoid of total relations is in P.*

PROOF:

We show that by looking at the Turing Machine described by the following pseudo code

Algorithm 1: Decide Representable Total

Input: $\mathcal{O}$
Simulate Decide Monoid $\langle \mathcal{O} \rangle$
Simulate Decide Order $\langle \mathcal{O} \rangle$
Simulate Decide Converse $\langle \mathcal{O} \rangle$
Simulate Decide Monotone $\langle \mathcal{O} \rangle$
Simulate Decide Totality $\langle \mathcal{O} \rangle$
Accept

where

Algorithm 2: Decide Monoid

Input: $\mathcal{O}$

for $a \in \mathcal{O}$ **do**
  if $\neg(a = 1'; a \wedge a = a; 1')$ **then**
    Reject

for $(a, b, c) \in \mathcal{O}^3$ **do**
  if $\neg((a; b); c = a; (b; c))$ **then**
    Reject

Accept

Algorithm 3: Decide Order

Input: $\mathcal{O}$

for $a \in \mathcal{O}$ **do**
  if $\neg(a \leq a)$ **then**
    Reject

if $\neg(\breve{1}' = 1')$ **then**
  Reject

for $(a, b) \in \mathcal{O}^2$ **do**
  if $a \leq b \wedge b \leq a$ **then**
    if $\neg(a = b)$ **then**
      Reject

for $(a, b, c) \in \mathcal{O}^3$ **do**
  if $a \leq b \wedge b \leq c$ **then**
    if $\neg(a \leq c)$ **then**
      Reject

Accept

Algorithm 4: Decide Converse

Input: $\mathcal{O}$

if $\neg(\breve{1}' = 1')$ **then**
  Reject

for $a \in \mathcal{O}$ **do**
  if $\neg(a = \breve{\breve{a}})$ **then**
    Reject

for $(a, b) \in \mathcal{O}^2$ **do**
  if $\neg((a; b)^{\smile} = \breve{b}; \breve{a})$ **then**
    Reject

Accept

Algorithm 5: Decide Monotone

Input: $\mathcal{O}$

for $(a, b) \in \mathcal{O}^2$ **do**
  if $a \leq b$ **then**
    if $\neg(\breve{a} \leq \breve{b})$ **then**
      Reject

for $(a, b, c) \in \mathcal{O}^3$ **do**
  if $a \leq b$ **then**
    if $\neg(a; c \leq b; c)$ **then**
      Reject

Accept

Algorithm 6: Decide Totality

Input: $\mathcal{O}$

for $a \in \mathcal{O}$ **do**
  if $\neg(1' \leq a; \breve{a})$ **then**
    Reject

Accept

We can see that this Turing Machine decides the problem. We can also see that Decide Totality has a time complexity of $O(n)$, Decide Converse $O(n^2)$ and Decide Monoid, Order and Monotone $O(n^3)$. When combined into Decide Representable Total, this gives us a cubic complexity. Therefore, there exists a Turing Machine that decides the problem in polynomial time, i.e. the decision problem is in P. □

Finally, we can see that this enables us to algorithmically construct representations. We see that

COROLLARY 63 *A representation of an Ordered Convoluted Monoid of Total Relations $\mathcal{O}$ can be constructed in exponential time with respect to $n = |\mathcal{O}|$.*

PROOF:

We do that by simply demonstrating this with the algorithm istelf:

Algorithm 7: Construct Representation Total

Input: $\mathcal{O}$
$V \leftarrow$ Construct $\Upsilon(\mathcal{O})$
$E \leftarrow \Upsilon(\mathcal{O}) \times \Upsilon(\mathcal{O})$
for $(S, T) \in E$ **do**
  $\lambda(S, T) \leftarrow \emptyset$
  for $a \in \mathcal{O}$ **do**
    if $S; a \subseteq T \wedge T; \breve{a} \subseteq S$ **then**
      $\lambda(e) \leftarrow \lambda(e) \cup a$
return $G \leftarrow (V, E, \lambda)$

We see that according to Theorem 54, this will result in a representation graph. Furthermore, given $n = |\mathcal{O}|$, we can see that constructing the vertices and the edges can be done in worst case ($|\Upsilon(\mathcal{O})| = 2^n$) in $O(2^{2n})$. Furthermore, the outer loop has at most 2^{2n} iterations, the inner one has at most n operations and the checking of the condition $S; a \subseteq T \wedge T; \breve{a} \subseteq S$ takes at most n time. Therefore, the time for the second part of the algorithm is bounded by $O(2^{2n}n)$, which is exponential time with respect to n. □

This means that we have an upper bound on the size of the smallest representation of any Ordered Convoluted Monoid of Total Relations $\mathcal{O}$ and the upper bound on the complexity of both the representability decision problem and the algorithmic construction of the representation. We implement these algorithm using Python, see Appendix B.

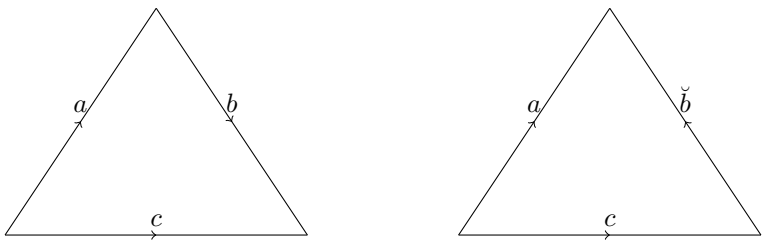
5.2 Pseudo Triangle Law

Another interesting class of Algebras of Relations we can look at is the algebras that follow the Pseudo-Triangle law. We name it that as it seems to resemble the Percian Triangle Law. However, it does not follow from it. Although these are just a special case of algebras of total relations, we examine them as they have very small representations.

We define the Pseudo-Triangle Law as follows

$$\forall a, b, c : c \leq a; b \rightarrow a \leq c; \breve{b} \tag{5.1}$$

We can visualise this in the same way as we did the Percian Triangle Law:



Although 5.1 might seem sound due to its resemblance to the Triangle Law, we observe

PROPOSITION 64 *For every non-empty set X, it is true that*

$$\exists R, S, T \in \mathcal{P}(X \times X) : R \subseteq S;T \wedge S \nsubseteq R;\breve{T}$$

PROOF:

Take any $S, T \in \mathcal{P}(X \times X) \setminus \{\emptyset\}$. We know that $\emptyset \subseteq S;T$. However, we also know that $\emptyset;\breve{T} = \emptyset$ and hence $S \nsubseteq \emptyset;\breve{T}$. □

What we also know is that ordered convoluted monoids that obey (5.1) are definitely representable. This simply follows from

PROPOSITION 65 *An algebra with the signature $(\leq, 1', \smile, ;)$ that follows the basic axiomatisation of OCM and obeys (5.1) also follows the axiomatisation of algebras of total relations.*

PROOF:

We know that any algebra that follows this axiomatisation will obey all the axioms of OCM. Hence, we only need to show that $\forall a : 1' \leq a;\breve{a}$.

This is simple. Since we know that $\forall a : a \leq 1';a$. Due to the pseudo triangle law, it is then also true that $\forall a : 1' \leq a;\breve{a}$. □

It trivially follows that

COROLLARY 66 *The class of representable algebras with the signature $(\leq, 1', \smile, ;)$ is finitely axiomatisable Algebras of Relations that obey (5.1).*

COROLLARY 67 *Ordered convoluted monoids that obey (5.1) are finitely representable.*

However, the bounds on the size and the time complexity of construction we have observed for algebras of total relations are a lot higher than the bound we are about to observe for algebras that follow the pseudo triangle law.

We begin by observing the order predicate in these structures.

PROPOSITION 68 *In algebras that follow the pseudo triangle law it holds that*

$$\forall a, a' : a \leq a' \rightarrow a = a'$$

PROOF:

Take any element a and any element a' such that $a \leq a'$. Clearly, $a \leq a';1'$. Furthermore, from the pseudo triangle law, it also holds that $a' \leq a;\breve{1'} = a$. Hence, any for two elements $a \leq a'$ it also holds that $a' \leq a$. Hence $a = a'$. □

This means that the order predicate in any algebra that follows this axiomatisation is necessarily the equality predicate. This means that all ordered convoluted monoids that follow the pseudo triangle law are in fact just convoluted monoids as the definition of the order predicate can be omitted altogether.

What interestingly follows from Propositions 65 and 68 is

COROLLARY 69 *In ordered convoluted monoids that follow the Pseudo Triangle Law, the converse operation is an inverse.*

PROOF:

By Proposition 65, $\forall a : 1' \leq a; \breve{a}$ and by Proposition 68 it also holds that $\forall a : a; \breve{a} \leq 1'$. Hence

$$\forall a : a; \breve{a} = 1'$$

or put simply, $\smile$ operation is an inverse. □

This means that every ordered convoluted monoid that follows the pseudo triangle law is in fact a group. Interestingly, this implies that since Cayley shows the linear bound on the size of representations of groups, it follows

THEOREM 70 *Every ordrered convoluted monoid $\mathcal{O}, |\mathcal{O}| > 1$ that follows the pseudo-triangle law can be represented over the set of its elements.*

PROOF:

We define the representation θ over $\mathcal{O}$ as follows

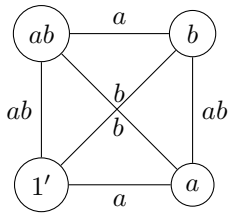
$$\forall a \in \mathcal{O} : a^\theta = \{(b, b; a) | b \in \mathcal{O}\}$$

We check through Lemmata 71 to 73 that his is indeed a representation. □

We illustrate this with an example θ of such an algebra (isomorphic to Klein Four-Group) with four elements $1', a, b, ab$, all of which are self-converse and the composition of non-identity is defined as follows:

;	a	ab	b
a	1'	b	ab
ab	b	1'	a
b	ab	a	1'

This algebra has a representation which is visualised as follows. We omit the arrows, as the elements are self-converse.



LEMMA 71 *θ is faithful.*

PROOF:

Take any two elements $a \not\leq b$. We know that $(1', a) \in a^\theta$ as $1'; a = a$. However, $1'; b \neq a$ and hence $(1', a) \notin b'^\theta$. □

LEMMA 72 *$\leq, 1', \smile$ are correctly represented by θ*

PROOF:

$\leq$ follows from Proposition 68. Now, for any a, b, if $(a, b) \in 1^\theta$, it follows $a = 1; b$ or simply $a = b$. And finally, for any a, b, c if $(b, c) \in a^\theta$ it follows that $c = b; a$. Since by Corollary 69 $a; \breve{a} = 1'$, it is clear that $(c, b) = (b; a, b; a; \breve{a}) \in \breve{a}^\theta$. $\square$

LEMMA 73 *; is correctly represented by θ*

PROOF:

Firstly, we show that for any a, b, c, d, e, it holds that given $(c, d) \in a^\theta$ and $(d, e) \in b^\theta$, so is $(c, e) \in (a; b)^\theta$. Since we know $d = c; a$ and $e = d; b$, it follows that $e = c; (a; b)$ and hence, clearly $(d, e) \in b^\theta$.

Now we show that given a, b, c, e where $(d, e) \in (a; b)^\theta$, there exists a d such that $(c, d) \in a^\theta$ and $(d, e) \in b^\theta$. $\square$

This means that the representations of such algebras are very small. But furthermore, we can reason about the complexity of related problems

COROLLARY 74 *The decision problem of whether or not an algebra with a signature $(\leq, 1', \smile, ;)$ is an ordered convoluted monoid that obeys (5.1) is in P.*

PROOF:

Since, we conclude from Proposition 68 and Corollary 69, an algebra with a signature $(\leq, 1', \smile, ;)$ is an ordered convoluted monoid that obeys (5.1) if and only if it is also a group. Therefore, the Turing Machine described by the following pseudo code will decide the problem

Algorithm 8: Decide Group

```
Input: 𝒪
for a ∈ 𝒪 do
    if ¬(a = 1'; a ∧ a = a; 1') then
        Reject
    if ¬(a; ă = 1') then
        Reject
for (a, b) ∈ 𝒪² do
    if a ≤ b then
        if ¬(a = b) then
            Reject
for (a, b, c) ∈ 𝒪³ do
    if ¬((a; b); c = a; (b; c)) then
        Reject
Accept
```

We can see that the first loop takes n, the second n^2 and the third n^3 iterations of constant time. Therefore, the complexity of it is $O(n^3)$, which means that the decision problem is in P. □

COROLLARY 75 *Ordered convoluted monoids that follow the Pseudo Triangle Law of size n have representations which can be constructed in $O(n^2)$ time.*

PROOF:

Theorem 70 enables us to use the following algorithm for the construction of a representation:

Algorithm 9: Construct Representation Group

Input: $\mathcal{O}$
$V \leftarrow \mathcal{O}$
$E \leftarrow \mathcal{O} \times \mathcal{O}$
for $(a, b) \in E$ **do**
  $\lambda(a, b) = \{\breve{a}; b\}$

Clearly, the loop takes n^2 repetitions of constant time, which means that the representation can indeed be constructed in $O(n^2)$ time. □

We implement the two algorithms from the above proofs in Appendix B.

Chapter 6

Conclusion and Future Work

What we have shown is that the basic axiomatisation of the signature is not sufficient to guarantee representations as algebras of binary relations. Therefore, in order to axiomatise the signature in a complete manner, we needed to look further.

We have presented an intuitive argument as to why we currently hypothesise, there is no finite set of axioms that would be complete and sound for binary relations. However, further work is necessary to either confirm or reject this hypothesis and this problem of finite axiomatisability remains open [9].

However, we have presented a recursive axiomatisation and shown that it is both sound and complete. We believe that this could be simplified further and potentially even used to prove whether the decision problem of representability of finite structures with the signature $(\leq, 1', \smile, ;)$ over binary relations is decidable, as raised in [9].

We have also demonstrated that the class of representable algebras with the signature $(\leq, 1', \smile, ;)$ is finitely axiomatisable for total relations. To our knowledge, this is a novel result. Furthermore, the finite structures are represented on a finite bases and the decision problem of representability of finite algebras with the signature $(\leq, 1', \smile, ;)$ over total relations is in P as we proved. We also show an interesting result of how if a monoid obeys the so-called pseudo-triangle law implies that it will be not only an ordered convoluted monoid of total relations, but also a group and that its representation will be linear in size.

We developed a tool in python that decides if an algebra is in one of the classes that are known to be representable. Furthermore, the tool generates a representation for this algebra. This demonstrates the implementability of the results shown in the project and provides a useful tool for the researchers in the area.

Bibliography

[1] G. Boole. The calculus of logic. *Cambridge and Dublin Mathematical Journal*, 3(1848):183–198, 1848.

[2] D. Bredikhin. An abstract characterization of some classes of algebras of binary relations. *Algebra and number theory*, (2):3–19, 1977.

[3] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[4] A. De Morgan. *Syllabus of a proposed system of logic*. Walton and Maberly, 1860.

[5] J. Desharnais, P. Jipsen, and G. Struth. Domain and antidomain semigroups. In *International Conference on Relational Methods in Computer Science*, pages 73–87. Springer, 2009.

[6] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, 2011.

[7] I. Düntsch, H. Wang, and S. McCloskey. Relations algebras in qualitative spatial reasoning. *Fundamenta Informaticae*, 39(3):229–248, 1999.

[8] S. R. Givant. *Introduction to Boolean algebras*. Undergraduate texts in mathematics. Y. Springer, New York, 2009.

[9] R. Hirsch. The class of representable ordered monoids has a recursively enumerable, universal axiomatisation but it is not finitely axiomatisable. *Logic Journal of the IGPL*, 13(2):159–171, 2005.

[10] R. Hirsch and I. Hodkinson. *Relation algebras by games*, volume 147. Elsevier, 2002.

[11] R. Hirsch and S. Mikulás. Ordered domain algebras. *Journal of Applied Logic*, 11(3):266–271, 2013.

[12] A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *Journal of Automated Reasoning*, 49(1):95–106, 2012.

[13] W. McCune. Prover9 and mace4, 2005.

[14] A. Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941.

Appendix A

Identifying Unrepresentable Monoids Using Mace

We use Mace [13] to find algebras that follow the basic axiomatisation of Ordered convoluted monoids that do not follow certain axioms that are sound for binary relations. Therefore, these are not representable, as we have shown in previous chapters.

Mace works as follows. It takes as input a number of first order formulas that the resulting structure must follow (the assumptions) and a number of first order formulae that we would like to show a counter example of (the goals). Alternatively, one can use Prover9 [13] as an automated theorem proved to show how goals can be inferred from the assumptions.

In our first example, we use the following code to find an algebras that follows the basic axiomatisation of ordered convoluted monoids, but not the following axiom

$$\forall a : a \leq a; \breve{a}; a$$

```
formulas(assumptions).
  all a all b all c ((a<b & b<c) -> a<c).
  all a all b ((a<b & b<a) <-> a=b).
  all a (I*a = a).
  con(I) = I.
  all a con(con(a)) = a.
  all a all b con(a*b)=con(b)*con(a).
  all a all b all c (a*(b*c) = (a*b)*c).
  all a all b (a<b -> con(a)<con(b)).
  all a all b all c (a<b -> ((a*c)<(b*c))).
end_of_list.

formulas(goals).
  all a (a < ((a*con(a))*a)).
end_of_list.
```

Due to symbol limitations enforced by the tool, we used `<` to denote $\leq$, `I` to denote $1'$, `con` to denote $\smile$ and `*` to denote ;. This returns an output file, which contains computational information, as well as the following two snippets

```
============================== CLAUSES FOR SEARCH ====================
formulas(mace4_clauses).
-(x < y) | -(y < z) | x < z.
-(x < y) | -(y < x) | y = x.
x < y | y != x.
x < y | x != y.
I * x = x.
con(I) = I.
con(con(x)) = x.
con(x * y) = con(y) * con(x).
(x * y) * z = x * (y * z).
-(x < y) | con(x) < con(y).
-(x < y) | x * z < y * z.
-(c1 < (c1 * con(c1)) * c1).
end_of_list.
============================== end of clauses for search =============
```

and

```
============================== MODEL =================================
interpretation( 3, [number=1, seconds=0], [

        function(I, [ 0 ]),

        function(c1, [ 1 ]),

        function(con(_), [ 0, 1, 2 ]),

        function(*(_,_), [
                            0, 1, 2,
                            1, 2, 0,
                            2, 0, 1 ]),

        relation(<(_,_), [
                            1, 0, 0,
                            0, 1, 0,
                            0, 0, 1 ])
]).
============================== end of model ==========================
```

This illustrates the algebra with three elements $\{0, 1, 2\}$ that are all self-converse as illustrated in `function(con(_), [ 0, 1, 2 ])` and have the ordering predicate and the composition func-

tion defined by the two Cayley tables in the output file. But it also provides the counter example with the valuation of `c1` is set to `1`

```
-(c1 < (c1 * con(c1)) * c1).
```

Similarly, we use the following code to find an example of an algebra that follows the basic axiomatisation, the $\forall a : a \leq a; \breve{a}; a$ but not the

$$\forall a, b, c : c \leq a; b \rightarrow c \leq a; \breve{a}; c$$

```
formulas(assumptions).
  all a all b all c ((a<b & b<c) -> a<c).
  all a all b ((a<b & b<a) -> a=b).
  all a (I*a = a).
  con(I) = I.
  all a con(con(a)) = a.
  all a all b con(a*b)=con(b)*con(a).
  all a all b all c (a*(b*c) = (a*b)*c).
  all a all b (a<b -> con(a)<con(b)).
  all a all b all c (a<b -> ((a*c)<(b*c))).
  all a (a < ((a*con(a))*a)).
end_of_list.

formulas(goals).
  all a all b all c (c<(a*b) -> c<((a*con(a))*c)).
end_of_list.
```

This produces the following clauses for search:

```
============================== CLAUSES FOR SEARCH ====================
formulas(mace4_clauses).
-(x < y) | -(y < z) | x < z.
-(x < y) | -(y < x) | y = x.
I * x = x.
con(I) = I.
con(con(x)) = x.
con(x * y) = con(y) * con(x).
(x * y) * z = x * (y * z).
-(x < y) | con(x) < con(y).
-(x < y) | x * z < y * z.
x < (x * con(x)) * x.
c3 < c1 * c2.
-(c3 < (c1 * con(c1)) * c3).
end_of_list.
============================== end of clauses for search =============
```

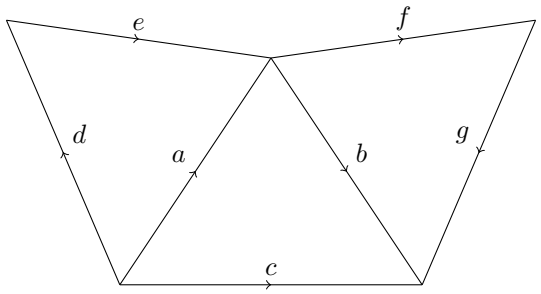
And the following counter example:

```
============================== MODEL ================================

interpretation( 3, [number=1, seconds=0], [

        function(I, [ 0 ]),

        function(c1, [ 1 ]),

        function(c2, [ 2 ]),

        function(c3, [ 0 ]),

        function(con(_), [ 0, 1, 2 ]),

        function(*(_,_), [
                            0, 1, 2,
                            1, 1, 2,
                            2, 2, 2 ]),

        relation(<(_,_), [
                            1, 0, 1,
                            0, 1, 1,
                            0, 0, 1 ])
]).

============================== end of model ==========================
```

Finally, we have wanted to find an example of an algebra that follows the basic axiomatisation of OCM along with $\forall a, b, c : c \leq a; b \to c \leq a; \breve{a}; c$ but not

$$\forall a, b, c, d, e, f, g : \Big((c \leq a; b \wedge a \leq d; e \wedge b \leq f; g) \to c \leq a; b; f; \breve{f}; \breve{e}; e; b\Big)$$

This is visualised as



We used the following input file:

```
formulas(assumptions).
  all a all b all c ((a<b & b<c) -> a<c).
  all a all b ((a<b & b<a) -> a=b).
  all a (I*a = a).
  con(I) = I.
  all a con(con(a)) = a.
  all a all b con(a*b)=con(b)*con(a).
  all a all b all c (a*(b*c) = (a*b)*c).
  all a all b (a<b -> con(a)<con(b)).
  all a all b all c (a<b -> ((a*c)<(b*c))).
  all a all b all c (c<(a*b) -> c<((a*con(a))*c)).
end_of_list.

formulas(goals).
  all a all b all c all d all e all f all g
    ((c<a*b & a<d*e & b<f*g) -> (c<(a*((f*con(f))*(con(e)*e)))*b)).
end_of_list.
```

However, Mace could not find a counter example small enough as the computation becomes impractical when it is searching for models larger than 4 or 5.

Appendix B

Python Implementations of Construction Algorithms

We implement the algorithms that decide whether the input algebra is an ordered convoluted monoid, if it is an ordered convoluted monoid and if it is an ordered convoluted monoid that obeys the pseudo-triangle law (i.e. a group) or an orderered convoluted monoids of total relations. It then generates the representation. The repository containing the code can be fount at `https://github.com/jasko1212si/RepresentationGenerator`

The input it takes is a `.json` file of that specifies the:

- `numOfElements`: size of the algebra, i.e. the list of elements of the algebra will be generated as `[0,1,...,numOfElements-1]` and the element `0` will be the identity element
- `conv`: the list signifying the converse function, i.e. the converse of the element `i` is stored at `conv[i]`
- `comp`: the cayley table of the composition operation, i.e. the composition of the elements `i` and `j` is stored at `comp[i][j]`
- `ord`: the list of ordered pairs `[a,b]` such that $a \leq b$, which will form the ordering predicate after transitive closure and the inclusion of the reflexive pairs

An example is shown below

```
{
  "numOfElements":2,
  "conv":[0,1],
  "comp":[
            [0,1],
            [1,1]
          ],
  "ord":[[0,1]]
}
```

We implement a program that decides if an algebra is a group or a total ordered convoluted monoid and constructs a representation and saves the adjacency matrix of the representation graph into `representation.json`, and exits otherwise.

File `RepresentOCM.py`

```
import sys
import json
import checkProperties
import construct
import constructOrder

try:
    filename = sys.argv[1]
    fp = open(filename)
    o = json.load(fp)
except:
    print("Could not open file, Exiting")
    sys.exit()


numOfElements = o['numOfElements']
conv = o['conv']
po = o['ord']
comp = o['comp']
representation = []

try:
    checkProperties.checkClosure(numOfElements, conv, comp, po)
    checkProperties.checkId(numOfElements, comp)
except Exception as error:
    print(error)
    sys.exit()

partialOrder = constructOrder.transitiveClosure(numOfElements, po)

try:
    checkProperties.checkAssoc(numOfElements, comp)
except Exception as error:
    print(error)
    sys.exit()

if checkProperties.checkGroup(numOfElements, conv, comp, partialOrder):
    print("Algebra is a Group, Constructing")
    representation = construct.constructGroup(numOfElements, conv, comp)
else:
    try:
```

```
        checkProperties.checkConv(numOfElements, conv, comp)
        checkProperties.checkOrder(numOfElements, partialOrder)
        checkProperties.checkMonotone(numOfElements, conv, comp, partialOrder)
    except Exception as error:
        print(error)
        sys.exit()

    if checkProperties.checkTotality(numOfElements, conv, comp, partialOrder):
        print("Algebra is a Total Ordered Convoluted Monoid, Constructing")
        representation = construct.constructTotal(numOfElements,
                                                  conv, comp, partialOrder)

if representation:
    fp = open('representation.json', 'w')
    json.dump(representation, fp)
else:
    print("Could not Determine Representability")
```

File construct.py

```
#enumerate powerset\{}
def powerSet(size, numOfElements):
    if (size == 1):
        return[[i] for i in range(numOfElements)]
    ret = []
    smaller = powerSet(size-1, numOfElements)
    ret += smaller
    for i in range(numOfElements):
        temp = []
        for s in smaller:
            if i > s[len(s)-1]:
                t = s[:]
                t.append(i)
                temp.append(t)
        ret += temp
    return ret

def constructUpwardlyClosed(numOfElements, partialOrder):
    power = powerSet(numOfElements, numOfElements)

    upwardlyClosed = []

    for p in power:
        include = True
        for a in p:
            for b in partialOrder[a]:
                if b not in p:
```

```
                    include = False
                    break
            if not include:
                break
        if include:
            upwardlyClosed.append(p)

    return upwardlyClosed

def constructTotal(numOfElements, conv, comp, partialOrder):
    upwardlyClosed = constructUpwardlyClosed(numOfElements, partialOrder)

    rep = [[[] for i in range(len(upwardlyClosed))]
                for i in range(len(upwardlyClosed))]

    for i in range(len(upwardlyClosed)):
        for j in range(len(upwardlyClosed)):
            for a in range(numOfElements):
                aInij = True
                for s in upwardlyClosed[i]:
                    temp = comp[s][a]
                    if temp not in upwardlyClosed[j]:
                        aInij = False
                        break
                if not aInij:
                    continue
                for t in upwardlyClosed[j]:
                    temp = comp[t][conv[a]]
                    if temp not in upwardlyClosed[i]:
                        aInij = False
                        break
                if aInij:
                    rep[i][j].append(a)

    return rep

def constructGroup(numOfElements, conv, comp):
    return [[comp[conv[a]][b]] for b in range(numOfElements)]
                                    for a in range(numOfElements)]
```

File constructOrder.py

```
def transitiveClosure(numOfElements, po):
    #construct partial order
    partialOrder = [[i] for i in range(numOfElements)]

    for pair in po:
```

```
            partialOrder[pair[0]].append(pair[1])

    #assure transitivity
    added = True
    while added:
        added = False
        for a in range(numOfElements):
            for b in partialOrder[a]:
                if a != b:
                    for c in partialOrder[b]:
                        if c not in partialOrder[a]:
                            added = True
                            partialOrder[a].append(c)

    return partialOrder
```

File checkProperties.py

```
# Check if the structure's predicates and functions are closed
def checkClosure(numOfElements, conv, comp, po):

    if numOfElements != len(conv):
        raise Exception("Converse length %d, should be %d, Exiting" %
                                                (len(conv), numOfElements))

    for i in range(numOfElements):
        if i not in conv:
            raise Exception("%d does not have a conv, Exiting" % i)

    if len(comp) != numOfElements:
        raise Exception("Bad composition matrix dimension, Exiting")

    for row in comp:
        if len(row) != numOfElements:
            raise Exception("Bad composition matrix dimension, Exiting")

        for elem in row:
            if elem < 0 or elem >= numOfElements:
                raise Exception("Composition not closed, Exiting")

    for pair in po:
        if len(pair) != 2:
            raise Exception("Bad patrial order dimension, Exiting")
        for elem in pair:
            if elem < 0 or elem >= numOfElements:
                raise Exception("Order contains illegal elements, Exiting")
```

```
# Check if the structure is associative
def checkAssoc(numOfElements, comp):
    for a in range(numOfElements):
        for b in range(numOfElements):
            for c in range(numOfElements):
                if comp[comp[a][b]][c] != comp[a][comp[b][c]]:
                    raise Exception("%d;(%d;%d) != (%d;%d);%d, Exiting"
                                                                %(a,b,c,a,b,c))

# Check the Identity
def checkId(numOfElements, comp):
    for a in range(numOfElements):
        if comp[0][a] != a:
            raise Exception("1';%d != %d, Exiting"%(a,a))


# Check if the algebra is a group (provided associativity and identity)
def checkGroup(numOfElements, conv, comp, partialOrder):
    for i in range(numOfElements):
        #check converse is inverse
        if comp[i][conv[i]] != 0:
            return False

        #check only one element (i) is in the set of elements greater than i
        if len(partialOrder[i]) != 1:
            return False
    return True

# Check the converse axioms are obeyed
def checkConv(numOfElements, conv, comp):
    if conv[0] != 0:
        raise Exception("conv of identity not identity, Exiting")

    for i in range(numOfElements):
        if conv[conv[i]] != i:
            raise Exception("conv of conv %d does not equal %d, Exiting"
                                                              % (i,i))

    for a in range(numOfElements):
        for b in range(numOfElements):
            if conv[comp[a][b]] != comp[conv[b]][conv[a]]:
                raise Exception("(%d;%d)~ != %d~;%d~, Exiting" %(a,b,b,a))

# Check anti-reflexivity of the Order
def checkOrder(numOfElements, partialOrder):
    for a in range(numOfElements):
        for b in range(numOfElements):
```

```
            if a!=b and a in partialOrder[b] and b in partialOrder[a]:
                raise Exception("%d < %d and %d < %d, but %d != %d, Exiting"
                                                        % (a,b,b,a,a,b))

def checkMonotone(numOfElements, conv, comp, partialOrder):
    #check monotonicity of conv
    for a in range(numOfElements):
        for b in range(numOfElements):
            if a in partialOrder[b] and conv[a] not in partialOrder[b]:
                raise Exception("%d<%d, but not %d~<%d"%(b,a))

    #check monotonicity of ;
    for a in range(numOfElements):
        for b in range(numOfElements):
            if a in partialOrder[b]:
                for c in range(numOfElements):
                    if comp[a][c] not in partialOrder[comp[b][b]]:
                        raise Exception("%d<%d, but not %d;%d<%d;%d, Exiting"
                                                        % (b,a,b,c,a,c))


# check for all a: 1 < a;conv(a)
def checkTotality(numOfElements, conv, comp, partialOrder):
    for a in range(numOfElements):
        if comp[a][conv[a]] not in partialOrder[0]:
            return False
    return True
```