Computing with Semigroup Congruences

Michael Torpey

5th September 2014

Abstract

In any branch of algebra, congruences are an important topic. An algebraic object's congruences describe its homomorphic images and quotients, and therefore they are an important part of the object's structure. However, in many branches of algebra, congruences are not studied directly, but the necessary theory is built up using other objects which are more readily understandable or which simplify the computation of important attributes. For example, group congruences are studied indirectly using normal subgroups, and ring congruences correspond to two-sided ideals.

In semigroup theory, in general, congruences are studied directly. However, certain categories of semigroups do have convenient properties which allow us to describe their congruences in other ways. For instance, a previous report by this author describes how to represent the congruences of 0-simple semigroups using linked triples.

In this report we consider several representations of semigroup congruences and how they can be used in computation: congruences on inverse semigroups may be described by *congruence pairs*, and certain calculations may be performed very quickly as a result; a congruence on an arbitrary semigroup may be described by a set of *generating pairs*, a very concise way of recording all its information; a finite 0-simple semigroup's congruences can be described by a linked triple, and we give a new way of finding a congruence's linked triple using its generating pairs; and we attempt to extend the use of linked triples to an arbitrary semigroup, using the principal factors of its $\mathcal{J}$-classes, which are all 0-simple.

The algorithms described are summarised in pseudo-code, and code is attached to this report for use in the GAP computational algebra system.

Contents

Chapter 1

Introduction

1.1 Motivation

In any branch of algebra, congruences are an important topic. By the various homomorphism theorems which have been found, an algebraic object's congruences are closely associated with its homomorphic images and its quotients. These properties describe the structure and symmetry of this object, and give a very good model of understanding what the object is and how it works.

Despite the importance of congruences, in many branches of algebra congruences are not studied directly. An object's quotients and homomorphic images are still discovered, but the necessary theory is built up using other, equivalent objects – either because these other objects are easier to understand than congruences *per se*, or because certain important calculations can be completed more efficiently than by studying sets of pairs.

For example, group theory has the concept of a congruence on a group, and this concept aptly describes the group's quotients and homomorphic images. However, for any congruence ρ there is a normal subgroup N whose cosets are precisely the congruence classes of ρ. Hence group congruences are studied "by proxy", and a rich theory is built up using only normal subgroups. [1, p.154]

Similarly, ring theory has the concept of a congruence on a ring. But for a congruence ρ on a ring R, there exists a two-sided ideal I whose residue classes are precisely the congruence classes of ρ. Hence ring congruences are also not studied directly, but their theory is composed by studying the ring's ideals. [1, p.154]

In semigroup theory, in general, congruences are studied directly. So far there has not been discovered a "better" object which can be used in place of semigroup congruences, in the way that normal subgroups are used in place of group congruences. However, certain categories of semigroups do have convenient properties which allow us to describe their congruences in other ways.

A *completely 0-simple* semigroup has its non-universal congruences in bijective correspondence with its *linked triples* (see Definition 3.2), and these linked triples have enough information not only to describe the congruence completely,[1, p.83-91][2, p.4-11] but to make calculations regarding the congruence much faster computationally. [2, p.29-30]

An inverse semigroup also has a useful characterisation of congruences: its

congruences are in bijective correspondence with its *congruence pairs* (see Definition 5.12), and these congruence pairs also completely specify the congruence, giving simpler (and therefore computationally faster) methods of calculating certain properties of the congruence. [1, p.157]

In this report, we examine some of these alternative ways of representing semigroup congruences, and investigate how they can be used to give improved computational methods. We present several entirely new algorithms, as well as some new lemmas and theorems which are used to justify them. Furthermore, we present GAP code which implements these algorithms. All this represents original research.

1.2 Contents

The first topic we discuss is *generating pairs*, in Chapter 2. Generating pairs are a way of describing semigroup congruences using a very small amount of space (computationally or otherwise). A small set of pairs $\mathbf{R}$ is stored, and this is used as a description of $\mathbf{R}^\sharp$, the smallest congruence containing $\mathbf{R}$. We show that this construction is well-defined, and present a new way to go about calculating the congruence from a set of generating pairs.

In Chapter 3 we describe, for a 0-simple semigroup, an entirely new algorithm for calculating a congruence's linked triple using only its generating pairs. We include any theory necessary to justify this algorithm, including lemmas and a theorem which also constitute entirely original research.

The *linked triples* representation we have described applies only to finite simple and 0-simple semigroups. In Chapter 4 we explore the possibilities of using this method to represent congruences on an arbitrary semigroup S, by considering in turn each of the $\mathcal{J}$-classes of S and describing the congruences on its (0-simple) principal factor. This has never been done before, and several approaches are considered, but no satisfactory theorem is found.

Finally, Chapter 5 contains an explanation of congruences on inverse semigroups. We state some general theory about inverse semigroups, and then explain the link between congruences and *congruence pairs* (see Definition 5.12). We give algorithms for verifying a congruence pair, determining whether a pair is in the congruence based on its congruence pair, and listing all the elements of a congruence class using the congruence pair.

1.3 Expected Readership

This report is written to be understood by readers at postgraduate or high undergraduate level, who have undertaken some study of semigroup theory. As far as possible, it is designed to be self-explanatory, requiring as little pre-existing knowledge of the field as possible. However, it does not intend to explain all of semigroup theory from zero knowledge, and so a reference book such as [1] might be useful to readers less familiar with the area.

Chapter 3 contains many references to a previous report [2], and while Section 3.1 summarises all the results necessary to understand the chapter, readers wanting a fuller understanding of the area might consult that report.

1.4 GAP Implementation

Throughout this report, we are concerned with the problem of *computing* with congruences. Various algorithms are given showing in a concrete way how these computations can be done. Inside the body of this report, these algorithms are presented by pseudo-code, but accompanying this report is an implementation of the algorithms in the *Semigroups* package [4] of the GAP computational algebra system [3].

The files attached to this report are the full source code of the *Semigroups* package, modified to include my implementations of the algorithms described in this report. These implementations can be found in the following locations, inside the `gap/` directory.

- Generating Pairs – `pairs-cong.gi` and `pairs-cong.gd`
- Inverse Semigroups – `inverse-cong.gi` and `inverse-cong.gd`
- Linked Triple from Generating Pairs – `AsRZMSCongruenceByLinkedTriple` function in `reesmat-cong.gi`

1.5 Basic Definitions

Here we give a few basic definitions which will be used in later chapters.

Definition 1.1. A **relation R** on a semigroup S is a subset of $S \times S$. If $(x, y) \in \mathbf{R}$, then we say that x is $\mathbf{R}$-related to y, and we may write this as $x\ \mathbf{R}\ y$.

Definition 1.2. A relation $\mathbf{R}$ on a semigroup S is called

- **reflexive** if $(x, x) \in \mathbf{R}$ for all $x \in S$,
- **symmetric** if $(x, y) \in \mathbf{R}$ implies $(y, x) \in \mathbf{R}$,
- **transitive** if $(x, y), (y, z) \in \mathbf{R}$ implies $(x, z) \in \mathbf{R}$, and
- an **equivalence** if it is reflexive, symmetric, and transitive.

An equivalence $\mathbf{E}$ partitions a semigroup into *equivalence classes*, such that $(x, y) \in \mathbf{E}$ if and only if x and y are in the same equivalence class. We denote the equivalence class of x by $\mathbf{E}_x$.

Definition 1.3. A relation $\mathbf{R}$ on a semigroup S is called

- **left-compatible** if $(ax, ay) \in \mathbf{R}$ for all $(x, y) \in \mathbf{R}$ and $a \in S$,
- **right-compatible** if $(xa, ya) \in \mathbf{R}$ for all $(x, y) \in \mathbf{R}$ and $a \in S$, and
- **compatible** if $(xz, yt) \in \mathbf{R}$ for all $(x, y), (z, t) \in \mathbf{R}$.

In fact, a relation is compatible if and only if it is left-compatible and right-compatible. [1, p.22-23]

Definition 1.4. A relation ρ on a semigroup S is called a **congruence** if it is reflexive, symmetric, transitive, and compatible.

Definition 1.5. For a semigroup S, the **trivial congruence** or *diagonal relation* Δ_S is the relation on S given by $\{(x, x) \mid x \in S\}$.

Chapter 2

Generating Pairs

For a semigroup S, a congruence ρ is a set of pairs contained in $S \times S$. Given certain pairs which are in ρ, the four axioms of a congruence (see Definition 1.4) imply certain other pairs which must be in ρ; for example, if we have the pair $(a, b) \in \rho$, then we must also have the pair $(b, a) \in \rho$ by symmetry. Similarly, if we have the pairs $(x, y), (y, z) \in \rho$, then we must also have the pair $(x, z) \in \rho$ by transitivity.

In fact, in general a congruence ρ with a great number of pairs may be reduced to a list of pairs $\mathbf{R} \subset \rho$ only a fraction of the size, from which all other pairs in ρ will follow using the definition of a congruence. We may use this fact to describe a congruence computationally, storing only the pairs in the set $\mathbf{R}$ – we call these the *generating pairs* – and thus vastly reducing the time and space complexity of various congruence algorithms. This motivates the following body of theory, which will allow us to describe an algorithm to calculate a congruence from it generating pairs.

2.1 Background Theory

Definition 2.1. Let $\mathbf{R}$ be a relation on a semigroup S. We define $\mathbf{R}^\sharp$ as the smallest congruence ρ on S such that $\mathbf{R} \subseteq \rho$.

To show that this is well-defined, we need to prove that there is one congruence containing $\mathbf{R}$ which is strictly smaller than all other such congruences. Consider any two congruences ρ and σ on S, such that $\mathbf{R} \subseteq \rho$ and $\mathbf{R} \subseteq \sigma$. Now consider the relation

$$\rho \cap \sigma = \{(x, y) \in S \times S \mid (x, y) \in \rho \text{ and } (x, y) \in \sigma\}.$$

Clearly $\mathbf{R} \subseteq \rho \cap \sigma$.

(R) Since (x, x) is in ρ and σ for all $x \in S$, $\rho \cap \sigma$ is reflexive.

(S) If $(x, y) \in \rho \cap \sigma$ then $(x, y) \in \rho$ and $(x, y) \in \sigma$. By the symmetry of ρ and σ therefore, $(y, x) \in \rho$ and $(y, x) \in \sigma$, and so $(y, x) \in \rho \cap \sigma$. Hence $\rho \cap \sigma$ is symmetric.

(T) Let $(x, y), (y, z) \in \rho \cap \sigma$. Similarly to the symmetric argument, we can use the transitivity of ρ and σ to show that $(x, z) \in \rho \cap \sigma$, so $\rho \cap \sigma$ is transitive.

(C) Let $(x,y),(z,t) \in \rho \cap \sigma$. Since ρ and σ are congruences, we know that (xy,zt) is in both ρ and σ, so $(xy,zt) \in \rho \cap \sigma$ and $\rho \cap \sigma$ is compatible.

Therefore $\rho \cap \sigma$ is a congruence.

If ρ and σ are distinct but of equal size, their intersection $\rho \cap \sigma$ is a congruence containing $\mathbf{R}$ which is strictly smaller than each of them. Hence there is precisely one smallest congruence containing $\mathbf{R}$, which we may call $\mathbf{R}^\sharp$.

This gives us a concise way of describing a congruence; however, this representation can only be useful if we have a good algorithm to calculate $\mathbf{R}^\sharp$ from $\mathbf{R}$, or indeed to determine whether a pair (a,b) is in $\mathbf{R}^\sharp$ given only $\mathbf{R}$.

We first need to establish a few definitions, in which $\mathbf{R}$ is a relation on a semigroup S. Let

$$\mathbf{R}^{-1} = \{(x,y) \in S \times S \mid (y,x) \in \mathbf{R}\}.$$

Next, let $\circ$ be the operation of concatenation, so that for two relations $\mathbf{R}_1$ and $\mathbf{R}_2$ on S,

$$\mathbf{R}_1 \circ \mathbf{R}_2 = \{(x,y) \in S \times S \mid \exists z \in S : (x,z) \in \mathbf{R}_1, (z,y) \in \mathbf{R}_2\},$$

and for $n \in \mathbb{N}$ let

$$\mathbf{R}^n = \underbrace{\mathbf{R} \circ \cdots \circ \mathbf{R}}_{n \text{ times}}.$$

Definition 2.2. The **transitive closure** $\mathbf{R}^\infty$ of a relation $\mathbf{R}$ is the relation given by

$$\mathbf{R}^\infty = \bigcup_{n \in \mathbb{N}} \mathbf{R}^n$$

Lemma 2.3. *If $\mathbf{R}$ is a reflexive relation, then $\mathbf{R}^\infty$ is the smallest transitive relation containing $\mathbf{R}$.*

Proof. Clearly $\mathbf{R} \subseteq \mathbf{R}^\infty$. First we show that $\mathbf{R}^\infty$ is transitive. Let $(x,y),(y,z) \in \mathbf{R}^\infty$, i.e. $(x,y) \in \mathbf{R}^m$ and $(y,z) \in \mathbf{R}^n$ for some $m,n \in \mathbb{N}$. Clearly,

$$(x,z) \in \mathbf{R}^m \circ \mathbf{R}^n = \mathbf{R}^{m+n},$$

and so $(x,z) \in \mathbf{R}^\infty$, so $\mathbf{R}^\infty$ is transitive.

Next we want to show that there is no transitive relation containing $\mathbf{R}$ which is *smaller* than $\mathbf{R}^\infty$. Let $\mathbf{T}$ be a transitive relation such that $\mathbf{R} \subseteq \mathbf{T}$. Since $\mathbf{T}$ is transitive, $\mathbf{T}^n = \mathbf{T}$ for all $n \in \mathbb{N}$. Since $\mathbf{R} \subseteq \mathbf{T}$, we also have that $\mathbf{R}^n \subseteq \mathbf{T}^n = \mathbf{T}$ for all $n \in \mathbb{N}$. Hence

$$\mathbf{R}^\infty = \bigcup_{n \in \mathbb{N}} \mathbf{R}^n \subseteq \mathbf{T},$$

so $\mathbf{R}^\infty$ is no larger than $\mathbf{T}$. [1, p.21] □

Now we are ready to make a definition which produces the equivalence relation generated by $\mathbf{R}$.

Definition 2.4. For a relation $\mathbf{R}$ on a semigroup S, we define $\mathbf{R}^e$ as the relation $\left(\mathbf{R} \cup \mathbf{R}^{-1} \cup \Delta_S\right)^\infty$.

Lemma 2.5. *For a relation $\mathbf{R}$ on a semigroup S, $\mathbf{R}^e$ is the smallest equivalence relation $\mathbf{E}$ on S such that $\mathbf{R} \subseteq \mathbf{E}$.*

Proof. Clearly $\mathbf{R} \subseteq \mathbf{R}^e$.

Similar to the proof of Lemma 2.3, we will prove that $\mathbf{R}^e$ is an equivalence relation, and then go on to prove that there is no smaller equivalence relation containing $\mathbf{R}$.

Let $\mathbf{Q} = \mathbf{R} \cup \mathbf{R}^{-1} \cup \Delta_S$, so that $\mathbf{R}^e = \mathbf{Q}^\infty$. Since Δ_S contains all the pairs necessary for reflexivity, we know that $\mathbf{Q}$ is reflexive, and therefore $\mathbf{Q}^\infty$ is reflexive and, by Lemma 2.3, transitive.

To show symmetry, observe that $(x, y) \in \mathbf{R}$ if and only if $(y, x) \in \mathbf{R}^{-1}$, and that $(x, y) \in \Delta_S$ if and only if $x = y$. $\mathbf{Q}$ is therefore certainly symmetric, and

$$\mathbf{Q}^n = (\mathbf{Q}^{-1})^n = (\mathbf{Q}^n)^{-1},$$

and so $\mathbf{Q}^n$ is symmetric.

Now let $(x, y) \in \mathbf{R}^e$. For some $n \in \mathbb{N}$, we have $(x, y) \in \mathbf{Q}^n$. By the symmetry of $\mathbf{Q}^n$,

$$(y, x) \in \mathbf{Q}^n \subseteq \mathbf{Q}^\infty = \mathbf{R}^e,$$

and so $\mathbf{R}^e$ is symmetric. Hence $\mathbf{R}^e$ is an equivalence.

Now to show that $\mathbf{R}^e$ is the *least* such equivalence, consider any equivalence $\mathbf{E}$ on S such that $\mathbf{R} \subseteq \mathbf{E}$. Since $\mathbf{E}$ is reflexive, we know that $\Delta_S \subseteq \mathbf{E}$, and since $\mathbf{E}$ is symmetric and contains $\mathbf{R}$, we know that $\mathbf{R}^{-1} \subseteq \mathbf{E}$. Hence

$$\mathbf{Q} = \mathbf{R} \cup \mathbf{R}^{-1} \cup \Delta_S \subseteq \mathbf{E}.$$

Finally, since $\mathbf{E}$ is transitive and contains $\mathbf{Q}$, we know from Lemma 2.3 that $\mathbf{Q}^\infty \subseteq \mathbf{E}$. Hence $\mathbf{R}^e$ is contained in $\mathbf{E}$, and so is no larger than any equivalence on S. □

Definition 2.6. For a relation $\mathbf{R}$ on a semigroup S, we define $\mathbf{R}^c$ as the relation $\{(xay, xby) \mid (a, b) \in \mathbf{R},\ x, y \in S^1\}$.

Lemma 2.7. *For a relation $\mathbf{R}$ on a semigroup S, $\mathbf{R}^c$ is the smallest compatible relation on S containing $\mathbf{R}$.*

Proof. $\mathbf{R}^c$ certainly contains $\mathbf{R}$ – all the elements of $\mathbf{R}$ are encountered in the case that $x = y = 1$.

Let us show first that $\mathbf{R}^c$ is compatible. Let $(u, v) \in \mathbf{R}^c$ and let $w \in S$. Now there must exist $a, b \in S$ and $x, y \in S^1$ such that $u = xay$, $v = xby$, and $(a, b) \in \mathbf{R}$. Hence $wu = wx \cdot a \cdot y$ and $wv = wx \cdot b \cdot y$, and $wx \in S^1$, so $(wu, wv) \in \mathbf{R}^c$ and $\mathbf{R}^c$ is left-compatible. Similarly, $uw = x \cdot a \cdot yw$ and $vw = x \cdot b \cdot yw$, and $yw \in S^1$, so $(uw, vw) \in \mathbf{R}^c$ and $\mathbf{R}^c$ is right-compatible.

Finally we need to show that there is no compatible relation smaller than $\mathbf{R}^c$ which contains $\mathbf{R}$. For this purpose, let $\mathbf{C}$ be a compatible relation on S such that $\mathbf{R} \subseteq \mathbf{C}$. Now for any $(a, b) \in \mathbf{R}$ and $x, y \in S^1$, we must have $(xay, xby) \in \mathbf{C}$ by the definition of compatibility. Every element of $\mathbf{R}^c$ has this form, hence $\mathbf{R}^c \subseteq \mathbf{C}$. [1, p.26] □

The smallest compatible relation $\mathbf{R}^c$ has some properties which will be useful later. These properties make up the following lemmas:

Lemma 2.8. *For a relation $\mathbf{R}$, $(\mathbf{R}^{-1})^c = (\mathbf{R}^c)^{-1}$*

Proof. Let $\mathbf{R}$ be a relation on a semigroup S. $\mathbf{R}^{-1} = \{(a,b) \mid (b,a) \in \mathbf{R}\}$, so

$$(\mathbf{R}^{-1})^c = \{(xay, xby) \mid x, y \in S^1, (b,a) \in \mathbf{R}\}.$$

The inverse of this last expression is

$$\{(xay, xby) \mid x, y \in S^1, (a,b) \in \mathbf{R}\},$$

which is equal to $\mathbf{R}^c$. Now $\left((\mathbf{R}^{-1})^c\right)^{-1} = \mathbf{R}^c$, which is equivalent to what we wanted. □

Lemma 2.9. *Let S be a semigroup. The relation Δ_S is equal to ${\Delta_S}^c$.*

Proof. Clearly $\Delta_S \subseteq {\Delta_S}^c$ (let $x = y = 1$).

Now let $(u,v) \in {\Delta_S}^c$. We must have $u = xay, v = xby$ for some $(a,b) \in \Delta_S$ and $x, y \in S^1$. If $(a,b) \in \Delta_S$ then $a = b$, so $u = xay = xby = v$, so $(u,v) \in \Delta_S$. □

Lemma 2.10. *Let $\mathbf{A}$ and $\mathbf{B}$ be relations on a semigroup S. If $\mathbf{A} \subseteq \mathbf{B}$, then $\mathbf{A}^c \subseteq \mathbf{B}^c$.*

Proof. Let $\mathbf{A} \subseteq \mathbf{B}$, and let (xay, xby) be an arbitrary element of $\mathbf{A}^c$ where $(a,b) \in \mathbf{A}$ and $x, y \in S^1$. Since $\mathbf{A} \subseteq \mathbf{B}$, we have that $(a,b) \in \mathbf{B}$, and hence also that $(xay, xby) \in \mathbf{B}^c$. □

Lemma 2.11. *If $\mathbf{R}$ is a compatible relation, then $\mathbf{R}^n$ is also compatible, for all $n \in \mathbb{N}$.*

Proof. Let $\mathbf{R}$ be a compatible relation on a semigroup S, and let $n \in \mathbb{N}$. Now let $(a,b) \in \mathbf{R}^n$, and $x \in S$. Hence there exist $c_1, c_2, \ldots, c_n, c_{n+1} \in S$ such that $a = c_1$, $b = c_{n+1}$, and

$$(c_1, c_2), (c_2, c_3), \ldots, (c_n, c_{n+1}) \in \mathbf{R}.$$

Since $\mathbf{R}$ is left-compatible,

$$(xc_1, xc_2), (xc_2, xc_3), \ldots, (xc_n, xc_{n+1}) \in \mathbf{R},$$

and since $\mathbf{R}$ is right-compatible,

$$(c_1x, c_2x), (c_2x, c_3x), \ldots, (c_nx, c_{n+1}x) \in \mathbf{R},$$

and therefore

$$(xa, xb) \in \mathbf{R}^n \quad \text{and} \quad (ax, bx) \in \mathbf{R}^n,$$

so $\mathbf{R}^n$ is compatible. [1, p.26] □

Now that we have these ways of describing the smallest *equivalence* relation and the smallest *compatible* relation containing a set of pairs, we are ready to describe the smallest *compatible equivalence* relation, or congruence, containing those pairs:

Theorem 2.12. *Let $\mathbf{R}$ be a relation on a semigroup S. Then $\mathbf{R}^\sharp$, the smallest congruence on S which contains $\mathbf{R}$, is equal to $(\mathbf{R}^c)^e$.*

Proof. Since $\mathbf{R}^c$ is a relation, it follows from Lemma 2.5 that $(\mathbf{R}^c)^e$ is an equivalence, and it certainly contains $\mathbf{R}$. To show that it is a congruence, we now only need to show that it is compatible:

By Definition 2.4, $(\mathbf{R}^c)^e = \mathbf{Q}^\infty$, where

$$\mathbf{Q} = \mathbf{R}^c \cup (\mathbf{R}^c)^{-1} \cup \Delta_S.$$

Lemma 2.8 gives us that $(\mathbf{R}^c)^{-1} = (\mathbf{R}^{-1})^c$, and Lemma 2.9 gives us that $\Delta_S = {\Delta_S}^c$, so

$$\mathbf{Q} = \mathbf{R}^c \cup (\mathbf{R}^{-1})^c \cup {\Delta_S}^c,$$

and finally applying Lemma 2.10 gives us

$$\mathbf{Q} = (\mathbf{R} \cup \mathbf{R}^{-1} \cup \Delta_S)^c.$$

Hence by Lemma 2.7, $\mathbf{Q}$ is a compatible relation.

Let $a \in S$ and let $(x, y) \in (\mathbf{R}^c)^e = \mathbf{Q}^\infty$. By Definition 2.2, $(x, y) \in \mathbf{Q}^n$ for some $n \in \mathbb{N}$, and by Lemma 2.11 we know that $\mathbf{Q}^n$ is compatible. Hence

$$(ax, ay), (xa, ya) \in \mathbf{Q}^n \subseteq \mathbf{Q}^\infty = (\mathbf{R}^c)^e,$$

and so $(\mathbf{R}^c)^e$ is a congruence.

All that remains is to show that there is no congruence containing $\mathbf{R}$ which is smaller than $(\mathbf{R}^c)^e$.

Let ρ be a congruence containing $\mathbf{R}$. Since ρ is compatible, $\rho^c = \rho$ by Lemma 2.7; and since $\mathbf{R} \subseteq \rho$, by Lemma 2.10, $\mathbf{R}^c \subseteq \rho^c$. So we have

$$\mathbf{R}^c \subseteq \rho.$$

Finally, since ρ is an equivalence containing $\mathbf{R}^c$, we know from Lemma 2.5 that $(\mathbf{R}^c)^e \subseteq \rho$, so $(\mathbf{R}^c)^e$ is the smallest congruence on S containing $\mathbf{R}$. [1, p.26-27] □

This theorem gives a strong mathematical description of the congruence generated by a set of pairs; in fact, it allows us to derive the following, even simpler description of the congruence, which will be of great use in computing facts about the congruence:

Definition 2.13. Let S be a semigroup, and let $\mathbf{R} \subseteq S \times S$. If $c, d \in S$ are such that

$$c = xay \quad \text{and} \quad d = xby$$

for some $x, y \in S^1$ and either (a, b) or (b, a) is in $\mathbf{R}$, then we say that c and d are connected by an **elementary R-transition**. [1, p.27]

Theorem 2.14. *Let S be a semigroup, let $\mathbf{R} \subseteq S \times S$, and let $a, b \in S$. Then $(a, b) \in \mathbf{R}^\sharp$ if and only if there exists $n \in \mathbb{N}$ and a sequence of elements*

$$a = z_1 \to z_2 \to \cdots \to z_n = b$$

such that z_i is connected to z_{i+1} by an elementary $\mathbf{R}$-transition for $1 \leq i < n$. [1, p.27]

Proof. From Theorem 2.12 we have that $\mathbf{R}^\sharp = (\mathbf{R}^c)^e$. From Definition 2.4, this is equal to

$$(\mathbf{R}^c \cup (\mathbf{R}^c)^{-1} \cup \Delta_S)^\infty.$$

Hence by Definition 2.2, a pair $(a, b) \in S \times S$ is in $\mathbf{R}^\sharp$ if and only if there exists some $n \in \mathbb{N}$ and a sequence $a = z_1 \to z_2 \to \cdots \to z_n = b$ such that

$$(z_i, z_{i+1}) \in \mathbf{R}^c \cup (\mathbf{R}^c)^{-1} \cup \Delta_S$$

for $1 \leq i < n$.

This sequence of elements corresponds closely to the sequence of elementary $\mathbf{R}$-transitions described in the theorem, the only exception being that we may remove links in the sequence where $(z_i, z_{i+1}) \in \Delta_S$, i.e. where $z_i = z_{i+1}$. In this case we of course still have the shorter sequence

$$\cdots \to z_{i-1} \to z_i \to z_{i+2} \to \ldots$$

Hence we now only need to show that $(c, d) \in \mathbf{R}^c \cup (\mathbf{R}^c)^{-1}$ if and only if c and d are connected by an elementary $\mathbf{R}$-transition:

($\Leftarrow$): Let c and d be connected by an elementary $\mathbf{R}$-transition. Hence there exist $x, y \in S^1$ such that $c = xay$ and $d = xby$, where either $(a, b) \in \mathbf{R}$ or $(b, a) \in \mathbf{R}$. In the case that $(a, b) \in \mathbf{R}$ we have $(c, d) \in \mathbf{R}^c$, and in the case that $(b, a) \in \mathbf{R}$ we have $(c, d) \in (\mathbf{R}^c)^{-1}$. In either case, we have $(c, d) \in \mathbf{R}^c \cup (\mathbf{R}^c)^{-1}$, as required.

($\Rightarrow$): Let $(c, d) \in \mathbf{R}^c \cup (\mathbf{R}^c)^{-1}$. If $(c, d) \in \mathbf{R}^c$ then there exist $x, y \in S^1$ such that $c = xay$ and $d = xby$, for $(a, b) \in \mathbf{R}$. If on the other hand $(c, d) \in (\mathbf{R}^c)^{-1}$ then there still exist $x, y \in S^1$ such that $c = xay$ and $d = xby$, but for $(b, a) \in \mathbf{R}$. In either case, this gives an elementary $\mathbf{R}$-transition from c to d. □

2.2 The Algorithm

Theorem 2.14 provides a useful description of the congruence generated by a set of pairs. In the context of computational mathematics, we may certainly want to describe a semigroup congruence and compute certain properties of it, such as its pairs, its classes, or its join or meet with another congruence. Here we present an algorithm to determine whether a pair is in a congruence, using only its generating pairs.

Let ρ be a congruence on a semigroup S. We begin with a *lookup table* L – this is a list of integers which has one entry for each element x in S. This integer is the number of ρ_x, the congruence class to which x belongs. At the start of the algorithm, each element is assumed to be in its own singleton congruence class, so the list has the form

$$[1, 2, 3, \ldots, |S|].$$

Note that here we are using square-bracket notation for a list, where for example $[1, 4, 1, 7]$ is a list of four elements and $L[n]$ refers to the nth entry in the list L.

As the algorithm progresses, at various times we will find that two elements (say x and y) coincide, and so we will combine the classes ρ_x and ρ_y. One way to do this would be to go through the whole list, finding every entry which matches ρ_y, and updating it to point to ρ_x. However, this operation has high

time complexity, and would cause any implementation of this algorithm to take a long time to complete, Instead, we use the *union-find* algorithm [5], which updates the lookup table L in a more efficient way, as follows:

Rather than treating L as a simple table in which elements are ρ-related if and only if they have the same number in their L entry, we treat an entry in the table as a "pointer" to another entry in the table. Only if an entry in the table points to itself do we treat it as the actual number of the congruence class. Hence we have a function, FIND, which takes an integer between 1 and $|S|$ (referring to the position of an element $x \in S$) and returns the number of the congruence class:

```
procedure FIND(L, n)
    if L[n] = n then
        return n
    else
        return FIND(L, L[n])
    end if
end procedure
```

Now we may view the operation of finding an element's class as traversing a tree from its leaf up to its root, and we can view the entire connected tree as the class itself. In order to combine two classes, therefore, we have the function UNION, which simply finds the roots of the two trees and changes the higher one to point to the lower:

```
procedure UNION(L, m, n)
    M := FIND(L, m)
    N := FIND(L, n)
    if M < N then
        L[N] := M
    else if N < M then
        L[M] := N
    end if
end procedure
```

These two functions now allow us to transform a simple list of the integers 1 to $|S|$ into a set of trees which completely describe the classes of ρ. Note that this union-find method has automatically removed the problem of transitivity, as well as those of reflexivity and symmetry: if we simply iterate through the pairs of a relation, say $\mathbf{Q}$, and we relate the element x to y, and then y to z, we have combined the classes ρ_x, ρ_y and ρ_z into a single class, and so we have added the pair (x, z) with no additional effort; similarly every element x is related to itself in ρ_x from the very beginning; and relating x to y is precisely the same as relating y to x. In other words, if we perform UNION on all the pairs of a relation $\mathbf{Q}$ one by one, we produce a table L which describes the equivalence $\mathbf{Q}^e$.

Since $\mathbf{R}^\sharp = (\mathbf{R}^c)^e$, it is clear that we now only need to input the pairs of the relation $\mathbf{R}^c$ into UNION, and we will finish with a table which describes all the information of $\mathbf{R}^\sharp$.

$$\mathbf{R}^c = \{(xay, xby) \mid (a, b) \in \mathbf{R},\ x, y \in S^1\},$$

so we need a way to find every pair (xay, xby) for each $(a, b) \in \mathbf{R}$. Let X be a set of generators for the semigroup S. So now each pair we want to find has the form

$$(x_m \cdots x_2 x_1 a y_1 y_2 \cdots y_n, x_m \cdots x_2 x_1 b y_1 y_2 \cdots y_n),$$

where all the $x_i, y_i \in X$ are generators of S. To accomplish this, we have a list of pairs to inspect, P, which starts off containing precisely the pairs of $\mathbf{R}$. In turn, we inspect each of these pairs (a, b), and apply each of the generators $x \in X$ to it, first on the left (giving us the pair (xa, xb)) and then on the right (giving us (ax, bx)). Each one of these, if it has not been encountered before, is itself added to the list P to have further generators added later. In this way every possible $(xay, xby) \in \mathbf{R}^c$ is encountered and UNION applied to it.

If we follow this algorithm, every element of $\mathbf{R}^c$ will be found and included in the union-find method. However, a small improvement can be made which takes advantage of union-find already accounting for reflexivity and symmetry: when a new pair (a, b) is found, it is not added to P if $a = b$, or if we have already found (b, a).

After all the pairs have been applied in this way, we are left with a table L which represents a tree structure describing how the elements of S are contained in their ρ-classes. We now "flatten" this table, by replacing each entry $L[n]$ with FIND$(L[n])$. Now we have a table where, as in the naïve example earlier, $(a, b) \in \rho$ if and only if $L[a] = L[b]$. Finally, and optionally, we may decide to "normalise" the numbers in the table: depending on the contents of each class, the congruence classes may be numbered $1, 4, 11, 19 \ldots$ where we would rather they were numbered $1, 2, 3, 4 \ldots$ This change can be performed with linear complexity by a single sweep through the table at the end.

```
Require: S = ⟨X⟩ is a semigroup, n = |S|, R ⊆ S × S
  procedure CongruenceByGeneratingPairs(R)
      L := [1, 2, ..., n]
      P := []
      for (a, b) ∈ R do
          Union(L, a, b)
          Add (a, b) to P
      end for
      i := 0
      while i < Length(P) do
          i := i + 1
          Let (a, b) = P[i]
          for x ∈ X do
              if ax ≠ bx and (ax, bx), (bx, ax) ∉ P then
                  Add (ax, bx) to P
                  Union(L, ax, bx)
```

```
            end if
            if xa ≠ xb and (xa, xb), (xb, xa) ∉ P then
                Add (xa, xb) to P
                UNION(L, xa, xb)
            end if
        end for
    end while
    ▷ Normalise the table
    L := NORMALISE(L)
    return L
end procedure
```

We have used a function called NORMALISE to convert the union-find table L to a normal lookup table with congruence class numbers $1, 2, 3, 4 \ldots$ This function is shown below:

```
Require: L is a list with size n
  procedure NORMALISE(L)
    H := []
    next := 1
    for i ∈ {1 ... n} do
        I := FIND(L, i)
        if H[j] = I for some j ∈ ℕ then
            L[i] := j
        else
            H[next] := I
            L[i] := next
            next := next + 1
        end if
    end for
    return L
  end procedure
```

The procedure as above returns a full lookup table which allows us to check whether two elements are in $\mathbf{R}^\sharp$. However, if we do not require a full description of the congruence, but only want to test whether a certain pair (a, b) is in the congruence, then we may add a test inside the main *while* loop to check whether $\rho_a = \rho_b$, and return true if so. In this way, a result can be given very early in the process, and a lot of time may be saved. However, a "false" result can only be returned for certain once the entire algorithm has completed.

2.3 Implementation

The previous section presented an abstract algorithm which finds $\mathbf{R}^\sharp$ from a relation $\mathbf{R}$. However, in actually implementing this algorithm on a computer, there are certain technical considerations which affect the time it takes to complete. A few of these considerations are discussed here.

Firstly, since the entries of the list L correspond to the elements of the semigroup S, it is necessary to have some way of listing the elements of S in a fixed order, so that a given element can be looked up quickly and its position in L determined. The GAP computational algebra system[3] has an `Elements` method which calculates a list of all the elements of S. This list should be stored with the congruence object, and then elements can be looked up in the list and given a consistent unique integer. Another improvement, if the semigroup is one with complicated multiplication, is the storing of left and right Cayley graphs in advance; since we only ever multiply an element by the generators of the semigroup, this provides a very fast way to find the necessary products without actually having to multiply elements and look up their position in the list.

Another source of work for a computer is the test, on finding a new pair, of whether that pair has already been discovered and added to P. As the algorithm progresses, P may grow very large, and to iterate through the whole list each time looking for a pair might take a long time to complete. The use of a hash table to store found pairs significantly improves the time complexity of this operation. When a pair (a, b) is found, (a, b) and (b, a) are looked up in the hash table, and if they do not exist then (a, b) is added to P and to the hash table. In fact, by this method, when pairs in P have been fully inspected and all generators applied on the left and right, they may then be discarded to free space, with the knowledge that they still exist in the hash table and so will not be processed again.

The final consideration is the use of the algorithm, as mentioned at the end of the last section, to test whether a *specific pair* is in $\mathbf{R}^\sharp$. If the pair (a, b) is encountered early in the algorithm, or if several pairs together with transitivity imply that (a, b) must be in the congruence, then we may simply return true and stop working. However, it is prudent to save the work that has been done so far, so that when another pair is looked up or the whole congruence evaluated, calculations need not be repeated. Thus, a special object should be defined in the implementation, hidden from the user, which stores information so far computed, and resumes the calculation if another test is called. Of course, if ever the calculation is completed and a full lookup table produced, then it may be used to look pairs up almost instantly, and P and the hash table may be discarded after all.

Chapter 3

Linked Triples

The category of *0-simple semigroups* has an important property which makes the study of their congruences much simpler computationally: the concept of **linked triples**. A full description of how these objects can be used to carry out fast computations with congruences is included in another report which I wrote, *Computing with Congruences on Finite 0-Simple Semigroups* [2]. A few of the important results from that report are included in the next section, omitting any proofs:

3.1 Basic Results

Recall the following characterisation of finite 0-simple semigroups:

Theorem 3.1. (The Rees Theorem) *Every completely 0-simple semigroup is isomorphic to a Rees 0-matrix semigroup $\mathcal{M}^0[G; I, \Lambda; P]$, where G is a group and P has no row or column which consists entirely of zeros. Conversely, every such Rees 0-matrix semigroup is completely 0-simple. [1, p.72-75] [2, p.4]*

Hence we can consider any finite 0-simple semigroup by considering its isomorphic Rees 0-matrix semigroup. Next we give the definition of a linked triple, and assert that such a semigroup's linked triples correspond bijectively with its non-universal congruences.

Definition 3.2. For a finite 0-simple Rees 0-matrix semigroup $\mathcal{M}^0[G; I, \Lambda; P]$, a **linked triple** is a triple

$$(N, \mathcal{S}, \mathcal{T})$$

consisting of a normal subgroup $N \trianglelefteq G$, an equivalence relation $\mathcal{S}$ on I and an equivalence relation $\mathcal{T}$ on Λ, such that the following are satisfied:

1. $\mathcal{S} \subseteq \varepsilon_I$, where $\varepsilon_I = \{(i,j) \in I \times I \,|\, \forall \lambda \in \Lambda : p_{\lambda i} = 0 \iff p_{\lambda j} = 0\}$,
2. $\mathcal{T} \subseteq \varepsilon_\Lambda$, where $\varepsilon_\Lambda = \{(\lambda,\mu) \in \Lambda \times \Lambda \,|\, \forall i \in I : p_{\lambda i} = 0 \iff p_{\mu i} = 0\}$,
3. For all $i, j \in I$ and $\lambda, \mu \in \Lambda$ such that $p_{\lambda i}, p_{\lambda j}, p_{\mu i}, p_{\mu j} \neq 0$ and either $(i,j) \in \mathcal{S}$ or $(\lambda,\mu) \in \mathcal{T}$, we have that $q_{\lambda\mu ij} \in N$, where

$$q_{\lambda\mu ij} = p_{\lambda i} p_{\mu i}^{-1} p_{\mu j} p_{\lambda j}^{-1}.$$

(where $q_{\lambda\mu ij} = p_{\lambda i}p_{\mu i}^{-1}p_{\mu j}p_{\lambda j}^{-1}$). [1, p.86] [2, p.4]

Theorem 3.3. *For a finite 0-simple Rees 0-matrix semigroup $\mathcal{M}^0[G; I, \Lambda; P]$, there exists a bijective mapping Γ between its non-universal congruences and its linked triples. [1, p.91] [2, p.5-9]*

We may of course want to know how to calculate a congruence's corresponding linked triple. That is, given a congruence ρ, we may want to calculate $\Gamma(\rho)$. Let us write such a linked triple as $(N_\rho, \mathcal{S}_\rho, \mathcal{T}_\rho)$.

Definition 3.4. We define the equivalence $\mathcal{S}_\rho \subseteq I \times I$ by the rule that $(i, j) \in \mathcal{S}_\rho$ if and only if

- $(i, j) \in \varepsilon_I$, and
- $(i, p_{\lambda i}^{-1}, \lambda)\ \rho\ (j, p_{\lambda j}^{-1}, \lambda)$

for all $\lambda \in \Lambda$ such that $p_{\lambda i} \neq 0$ (and hence by ε_I, $p_{\lambda j} \neq 0$). [1, p.84] [2, p.5]

Definition 3.5. We define the equivalence $\mathcal{T}_\rho \subseteq \Lambda \times \Lambda$ by the rule that $(\lambda, \mu) \in \mathcal{T}_\rho$ if and only if

- $(\lambda, \mu) \in \varepsilon_\Lambda$, and
- $(i, p_{\lambda i}^{-1}, \lambda)\ \rho\ (i, p_{\mu i}^{-1}, \mu)$

for all $i \in I$ such that $p_{\lambda i} \neq 0$ (and hence by ε_I, $p_{\mu i} \neq 0$). [1, p.84] [2, p.5]

Let the top row of P be called 1_Λ. Label as 1_I the first (furthest left) column such that $p_{1_\Lambda 1_I} \neq 0$. Finally, let 1_G be the identity of the group G. This allows for the following definition:

Definition 3.6. We define N_ρ as the normal subgroup of G given by

$$N_\rho = \{a \in G \mid (1_I, a, 1_\Lambda)\ \rho\ (1_I, 1_G, 1_\Lambda)\}.$$

[1, p.84] [2, p.5]

Finally, we state how to determine whether a pair is in ρ using its linked triple:

Theorem 3.7. *Let $\mathcal{M}^0[G; I, \Lambda; P]$ be a finite 0-simple Rees 0-matrix semigroup, and let ρ be a congruence with linked triple $(N, \mathcal{S}, \mathcal{T})$. Then ρ has the property that two non-zero elements (i, a, λ) and (j, b, μ) are ρ-related if and only if*

1. *$(i, j) \in \mathcal{S}$;*
2. *$(\lambda, \mu) \in \mathcal{T}$;*
3. *$(p_{\xi i}ap_{\lambda x})(p_{\xi j}bp_{\mu x})^{-1} \in N$ for some (and therefore all) $x \in I, \xi \in \Lambda$ such that $p_{\xi i}, p_{\xi j}, p_{\lambda x}, p_{\mu x} \neq 0$;*

and 0 is related only to itself. [1, p.87-88] [2, p.5,8]

3.2 Calculating Linked Triple from Generating Pairs

The previous report [2] briefly outlines, in Section 4.1, a method of calculating a congruence's linked triple. This simplistic algorithm relies on a "black box" which is able to answer whether a given pair is in the congruence, and it does little more than simply interpret Definitions 3.4, 3.5 and 3.6. The method which was implemented in GAP to accompany that report relied on a pre-existing test for whether a pair is in ρ, and had high complexity.

In this section we give a new and simpler method for finding the linked triple of a congruence, directly from its generating pairs, without enumerating its classes or finding large lists of additional pairs. This algorithm, and the theory in this section, constitute original research.

Let us first consider how to determine whether a congruence ρ on a finite 0-simple Rees 0-matrix semigroup S is *universal*, given a set of generating pairs $\mathbf{R}$. By the 0-simplicity of S, the element 0 is either in a class by itself, or is related to every element in the semigroup.[1, p.83] Hence $\rho = S \times S$ if and only if there exists some non-zero element (i, a, λ) such that $(i, a, \lambda)\ \rho\ 0$. There is a simple way to test for this using the pairs in $\mathbf{R}$, as shown in the following lemma:

Lemma 3.8. *Let $\mathbf{R}$ be a relation on the finite 0-simple Rees 0-matrix semigroup $S = \mathcal{M}^0[G; I, \Lambda; P]$. The congruence $\mathbf{R}^\sharp$ is equal to $S \times S$ if and only if one of the following is true:*

- *$\mathbf{R}$ contains a pair $(x, 0)$ or $(0, x)$ where $x \neq 0$,*
- *$(i, a, \lambda)\ \mathbf{R}\ (j, b, \mu)$, where $(i, j) \notin \varepsilon_I$ or $(\lambda, \mu) \notin \varepsilon_\Lambda$.*

Proof. ($\Rightarrow$): Assume that neither of the options in the list is true. That is, every pair in $\mathbf{R}$ is either $(0, 0)$ or contains non-zero elements (i, a, λ) and (j, b, μ), where $(i, j) \in \varepsilon_I$ and $(\lambda, \mu) \in \varepsilon_\Lambda$.

Recall that $\mathbf{R}^\sharp = (\mathbf{R}^c)^e$. For 0 to be equivalent to another element in $(\mathbf{R}^c)^e$, 0 must be related to another element in $\mathbf{R}^c$ (reflexivity gives us only $(0, 0)$, symmetry can only give $(x, 0)$ if we already have $(0, x)$, and transitivity cannot relate 0 to x unless 0 is already related to a non-zero element). Hence we require $(0, x)$ or $(x, 0)$ in $\mathbf{R}^c$.

If $(0, 0)$ is left- or right-multiplied by any element, it still gives $(0, 0)$, so we need to consider the non-zero pairs in $\mathbf{R}$. Since for all rows $\lambda \in \Lambda$, $p_{\lambda i} = 0$ if and only if $p_{\lambda j} = 0$, we can see that left-multiplying by any element will either give two non-zero elements or two zeros. And since for all columns $i \in I$, $p_{\lambda i} = 0$ if and only if $p_{\mu i} = 0$, right-multiplying by an element will similarly always give two non-zero elements or two zeros. Hence 0 is in a class on its own in $\mathbf{R}$, $\mathbf{R}^c$, and $\mathbf{R}^\sharp$, so $\mathbf{R}^\sharp \neq S \times S$.

($\Leftarrow$): Certainly if $\mathbf{R}$ contains a pair $(x, 0)$ or $(0, x)$, then $\mathbf{R}^\sharp$ also contains that pair, and by the 0-simplicity of S, $\mathbf{R}^\sharp = S \times S$.[1, p.83]

Now consider the case where $(i, a, \lambda)\ \mathbf{R}\ (j, b, \mu)$, but $(i, j) \notin \varepsilon_I$. Hence, without loss of generality, there exists a row $\nu \in \Lambda$ such that $p_{\nu i} = 0$ but $p_{\nu j} \neq 0$. Now we left-multiply by the element $(1, 1, \nu)$:

$$\begin{aligned} (1, 1, \nu)(i, a, \lambda) &= 0 \\ (1, 1, \nu)(j, b, \mu) &= (1, p_{\nu j} b, \mu) \end{aligned}$$

Hence $(1, p_{\nu j} b, \mu)\ \mathbf{R}^\sharp\ 0$, and so $\mathbf{R}^\sharp = S \times S$ as before. A similar case applies where $(\lambda, \mu) \notin \varepsilon_\Lambda$. □

So our algorithm now has a quick method for determining whether a congruence is universal, using only its generating pairs. If a pair is found which matches one of the conditions in the last lemma, the algorithm can immediately terminate, since the universal congruence has no linked triple.

In finding an algorithm to calculate a linked triple from a set of generating pairs we can simplify the problem by viewing it in a certain light: we need to find *some* linked triple which describes a congruence containing $\mathbf{R}$; we then need to ensure that of all such linked triples, we have found the one which matches the *smallest* congruence possible. In fact, the size of a congruence is related to the size of the components of its linked triple, as follows:

Lemma 3.9. *Let N_ρ and N_σ be normal subgroups of G with $N_\rho \leq N_\sigma$, let $\mathcal{S}_\rho \subseteq \mathcal{S}_\sigma \subseteq I \times I$, and let $\mathcal{T}_\rho \subseteq \mathcal{T}_\sigma \subseteq \Lambda \times \Lambda$ such that*

$$(N_\rho, \mathcal{S}_\rho, \mathcal{T}_\rho) \quad \textit{and} \quad (N_\sigma, \mathcal{S}_\sigma, \mathcal{T}_\sigma)$$

are linked triples, corresponding to congruences ρ and σ respectively, on a finite 0-simple Rees 0-matrix semigroup $\mathcal{M}^0[G; I, \Lambda; P]$. Then $\rho \subseteq \sigma$.

Proof. Firstly, observe that $(0, 0)$ is in both ρ and σ; and observe that 0 is not related to a non-zero element by either congruence.

Now, let $(i, a, \lambda)\ \rho\ (j, b, \mu)$. Hence by Theorem 3.7,

$$(i, j) \in \mathcal{S}_\rho \subseteq \mathcal{S}_\sigma,$$

$$(\lambda, \mu) \in \mathcal{T}_\rho \subseteq \mathcal{T}_\sigma,$$

$$(p_{\xi i} a p_{\lambda x})(p_{\xi j} b p_{\mu x})^{-1} \in N_\rho \leq N_\sigma,$$

for ξ and x as in the theorem. Hence, by the same theorem,

$$(i, a, \lambda)\ \sigma\ (j, b, \mu).$$

□

This lemma tells us that if we reduce the size of any of a linked triple's components, we reduce the size of the corresponding congruence. Hence, in effect, we are searching for the *smallest* linked triple which relates every pair in $\mathbf{R}$.

Definition 3.10. For a finite 0-simple Rees 0-matrix semigroup $\mathcal{M}^0[G; I, \Lambda; P]$, we define the relations $\mathbf{R}_I$ and $\mathbf{R}_\Lambda$ by

$$\mathbf{R}_I = \big\{(i, j) \in I \times I \;\big|\; (i, a, \lambda)\ \mathbf{R}\ (j, b, \mu) \text{ for some } a, b \in G,\ \lambda, \mu \in \Lambda\big\},$$

$$\mathbf{R}_\Lambda = \big\{(\lambda, \mu) \in \Lambda \times \Lambda \;\big|\; (i, a, \lambda)\ \mathbf{R}\ (j, b, \mu) \text{ for some } a, b \in G,\ i, j \in I\big\}.$$

Theorem 3.11. *Let S be a finite 0-simple semigroup $\mathcal{M}^0[G; I, \Lambda; P]$, and let $\mathbf{R} \subseteq S \times S$ generate a non-universal congruence $\mathbf{R}^\sharp$. Let $\mathcal{S} = (\mathbf{R}_I)^e$, let $\mathcal{T} =$*

$(\mathbf{R}_\Lambda)^e$, *and let* N *be the smallest normal subgroup of* G *containing the set* X, *where*

$$\begin{aligned} X = & \left\{(p_{\xi i}ap_{\lambda x})(p_{\xi j}bp_{\mu x})^{-1} \;\middle|\; (i,a,\lambda)\ \mathbf{R}\ (j,b,\mu);\ \xi,x \text{ as in Theorem 3.7}\right\} \\ \cup & \left\{q_{\lambda\mu ij} \;\middle|\; (i,j)\in\mathbf{R}_I,\ \lambda,\mu\in\Lambda \text{ such that } p_{\lambda i},p_{\mu i}\neq 0\right\} \\ \cup & \left\{q_{\lambda\mu ij} \;\middle|\; (\lambda,\mu)\in\mathbf{R}_\Lambda,\ i,j\in I \text{ such that } p_{\lambda i},p_{\lambda j}\neq 0\right\}. \end{aligned}$$

Then $(N,\mathcal{S},\mathcal{T})$ *is the linked triple corresponding to* $\mathbf{R}^\sharp$.

Proof. First we will show that $(N,\mathcal{S},\mathcal{T})$ is *linked*, in the sense of Definition 3.2. Then we will show that the congruence ρ defined by $(N,\mathcal{S},\mathcal{T})$ contains all the pairs in $\mathbf{R}$. Finally we will show that ρ is contained in every congruence that contains $\mathbf{R}$.

Let us show that $(N,\mathcal{S},\mathcal{T})$ is linked. By definition, $N \trianglelefteq G$, and by Lemma 2.5, $\mathcal{S}$ and $\mathcal{T}$ are equivalence relations. Now we need to satisfy the three conditions in Definition 3.2. By definition, $\mathbf{R}$ generates a non-universal congruence, and so by Lemma 3.8, $\mathbf{R}_I \subseteq \varepsilon_I$ and $\mathbf{R}_\Lambda \subseteq \varepsilon_\Lambda$. Since ε_I and ε_Λ are equivalences, this gives us that $\mathcal{S} \subseteq \varepsilon_I$ and $\mathcal{T} \subseteq \varepsilon_\Lambda$, so conditions (1) and (2) are fulfilled.

Condition (3) follows from the definition of X, the generating set of the normal subgroup N. For each pair $(i,j)\in\mathcal{S}$ and for each pair $(\lambda,\mu)\in\mathcal{T}$, $q_{\lambda\mu ij}$ must be included in N. Indeed, for the pairs in $\mathbf{R}_I$ and $\mathbf{R}_\Lambda$, they are included in X, and all the others follow from the symmetry and transitivity applied in moving from $\mathbf{R}_I$ to $(\mathbf{R}_I)^e$. Hence $(N,\mathcal{S},\mathcal{T})$ is a linked triple.

Let us now show that $\mathbf{R}\subseteq\rho$, where ρ is the congruence defined by $(N,\mathcal{S},\mathcal{T})$. $(0,0)$ is in any congruence on S, so $(0,0)$ is certainly in ρ whether it is in $\mathbf{R}$ or not. Since $\mathbf{R}$ generates a *non-universal* congruence, there is certainly no $(0,x)$ or $(x,0)$ in $\mathbf{R}$ with $x\neq 0$. So we only need to consider non-zero pairs, as follows:

Let $(i,a,\lambda)\ \mathbf{R}\ (j,b,\mu)$. Hence $(i,j)\in\mathbf{R}_I\subseteq\mathcal{S}$ and $(\lambda,\mu)\in\mathbf{R}_\Lambda\subseteq\mathcal{T}$. Let $x\in I,\xi\in\Lambda$ be the first row and column such that $p_{\xi i},p_{\lambda x}\neq 0$; by the definition of X,

$$(p_{\xi i}ap_{\lambda x})(p_{\xi j}bp_{\mu x})^{-1}\in X\subseteq N.$$

These three observations directly satisfy the three requirements in Theorem 3.7, so $(i,a,\lambda)\ \rho\ (j,b,\mu)$. Hence $\mathbf{R}\subseteq\rho$.

Finally, we need to show that ρ is the *smallest* congruence containing $\mathbf{R}$. Let σ be an arbitrary congruence on S containing $\mathbf{R}$. If $\sigma = S\times S$, then certainly $\rho\subseteq\sigma$. If however, σ is non-universal, then σ has its own linked triple $(N_\sigma,\mathcal{S}_\sigma,\mathcal{T}_\sigma)$.

Let $(i,j)\in\mathcal{S}$. Hence $(i,j)\in(\mathbf{R}_I)^e$, so there is some sequence of elements

$$i = k_1 \to k_2 \to \cdots \to k_n = j$$

such that (k_r,k_{r+1}) or (k_{r+1},k_r) is in $\mathbf{R}_I$ for every $r\in\{1\ldots n-1\}$. From each step of this sequence, we obtain the knowledge that, for some $a,b\in G$ and $\lambda,\mu\in\Lambda$,

$$(k_r,a,\lambda)\ \mathbf{R}\ (k_{r+1},b,\mu)\quad\text{or}\quad(k_{r+1},b,\mu)\ \mathbf{R}\ (k_r,a,\lambda),$$

and hence

$$(k_r, a, \lambda)\ \sigma\ (k_{r+1}, b, \mu).$$

This now gives us some information about σ's linked triple: by Theorem 3.7,

$$(k_1, k_2), (k_2, k_3), \ldots (k_{n-1}, k_n) \in \mathcal{S}_\sigma,$$

and so by transitivity, $(i, j) \in \mathcal{S}_\sigma$. A similar argument establishes that if $(\lambda, \mu) \in \mathcal{T}$ then $(\lambda, \mu) \in \mathcal{T}_\sigma$. Hence $\mathcal{S} \subseteq \mathcal{S}_\sigma$ and $\mathcal{T} \subseteq \mathcal{T}_\sigma$.

Now let $n \in X$. Since X is the union of three sets, there are three possible cases which we must consider for where n lies. The first is that

$$n = (p_{\xi i} a p_{\lambda x})(p_{\xi j} b p_{\mu x})^{-1},$$

where $(i, a, \lambda)\ \mathbf{R}\ (j, b, \mu)$, and where $\xi \in \Lambda$ and $x \in I$ are a row and column such that $p_{\xi i}$ and $p_{\lambda x}$ are both non-zero. Since (i, a, λ) and (j, b, μ) are $\mathbf{R}$-related, they must also be σ-related, and so part (3) of Theorem 3.7 requires that $(p_{\xi i} a p_{\lambda x})(p_{\xi j} b p_{\mu x})^{-1} \in N_\sigma$. So $n \in N_\sigma$.

The second case is that $n = q_{\lambda\mu ij}$, where $i, j \in I$ and $\lambda, \mu \in \Lambda$ are such that $(i, j) \in \mathbf{R}_I$ and $p_{\lambda i}, p_{\mu i} \neq 0$ (hence $p_{\lambda j}, p_{\mu j} \neq 0$). We recall that $\mathbf{R} \subseteq \sigma$ and therefore $(i, j) \in \mathbf{R}_I$ implies $(i, j) \in \mathcal{S}_\sigma$. Now from Definition 3.2, in order for $(N_\sigma, \mathcal{S}_\sigma, \mathcal{T}_\sigma)$ to be *linked*, we must have $q_{\lambda\mu ij} \in N_\sigma$, so $n \in N_\sigma$. The third case, where $n = q_{\lambda\mu ij}$ and $(\lambda, \mu) \in \mathbf{R}_\Lambda$, is similar.

Hence $X \subseteq N_\sigma$, and since N_σ is a normal subgroup of G, this gives us that $\langle\langle X \rangle\rangle = N \leq N_\sigma$.

So we finally have $N \leq N_\sigma$, $\mathcal{S} \subseteq \mathcal{S}_\sigma$, and $\mathcal{T} \subseteq \mathcal{T}_\sigma$. Hence, by Lemma 3.9, we have that $\rho \subseteq \sigma$, so that ρ is the smallest congruence on S containing $\mathbf{R}$. □

This last theorem now gives us enough information to derive our algorithm, which is shown in the pseudo-code below. Note that this algorithm includes one additional improvement when calculating the elements of X. We require N to contain all the elements $q_{\nu\xi ij}$ where $(i, j) \in \mathbf{R}_I$, for all $\nu, \xi \in \Lambda$ such that $p_{\nu i}, p_{\xi i} \neq 0$. However, we need not add all of these elements to X for them to be included in N. It is sufficient to choose one fixed ν such that $p_{\nu i} \neq 0$, and add $q_{\nu\xi ij}$ for all the other ξ such that $p_{\xi i} \neq 0$. The other elements follow using transitivity and the definition of a normal subgroup.

```
Require: M^0[G; I, Λ; P] is a finite 0-simple Rees 0-matrix semigroup
  procedure LinkedTriple(R)
      L_I = [1, 2 ... |I|]
      L_Λ = [1, 2 ... |Λ|]
      X := ∅
      for (x, y) ∈ R do
          ▷ Check for the universal congruence
          if x = y then
              Skip this pair
          else if x = 0 or y = 0 then
              return Universal Congruence (no linked triple)
          end if
          Let x = (i, a, λ)
```

```
        Let y = (j, b, μ)
        if (i, j) ∉ ε_I or (λ, μ) ∉ ε_Λ then
            return Universal Congruence (no linked triple)
        end if

        ▷ Combine row and column classes
        UNION(L_I, i, j)
        UNION(L_Λ, λ, μ)

        ▷ Add generators for normal subgroup
        Choose ν ∈ Λ such that p_νi ≠ 0
        Choose k ∈ I such that p_λk ≠ 0
        Add (p_νi a p_λk)(p_νj b p_μk)^{-1} to X

        for ξ ∈ Λ \ {ν} such that p_ξi ≠ 0 do
            Add q_νξij to X
        end for
        for x ∈ I \ {k} such that p_λx ≠ 0 do
            Add q_λμkx to X
        end for
    end for

    L_I := NORMALISE(L_I)
    L_Λ := NORMALISE(L_Λ)
    Let N = ⟨⟨X⟩⟩
    Let S be the equivalence with lookup table L_I
    Let T be the equivalence with lookup table L_Λ
    return (N, S, T)
end procedure
```

Here we have made use of the NORMALISE function from Section 2.2 to convert the tables L_I and L_Λ to simple lookup tables which describe $\mathcal{S}$ and $\mathcal{T}$.

Chapter 4

Congruences by $\mathcal{J}$-Classes

My last report [2], and Chapter 3 of this report, describe a simple, efficient way of characterising congruences on finite simple and 0-simple semigroups: by linked triples. In this chapter we make an original, but unsuccessful attempt to extend this to a characterisation of congruences on arbitrary finite semigroups.

4.1 The Idea

Let S be a finite semigroup. It is of course made up of a number of $\mathcal{D}$-classes, and since S is finite, the $\mathcal{D}$-classes of S are precisely the same as the $\mathcal{J}$-classes of S. Recall that $\mathcal{J}$ is the relation which relates x to y if and only if x and y generate the same two-sided ideal: $S^1xS^1 = S^1yS^1$. Recall also that the $\mathcal{J}$-classes of S have a partial order $\leq$, defined by the rule that $J_x \leq J_y$ if and only if $S^1xS^1 \subseteq S^1yS^1$.

Let J be a $\mathcal{J}$-class of S, and let $x, y \in J$. Hence $S^1xS^1 = S^1yS^1$. Now consider the product xy: since $xy \in S^1xS^1$, we have $S^1xyS^1 \subseteq S^1xS^1$. Hence either $xy \in J$ or $J_{xy} \leq J$. In other words, the product of two elements of a $\mathcal{J}$-class is always in either the same $\mathcal{J}$-class or a lower one. An abstract form of this multiplication is shown below, in a construction where all classes below J are boiled down to a single element, and all classes above or incomparable to J are discarded:

Definition 4.1. Let S be a semigroup. The **principal factor** of a $\mathcal{J}$-class J of S is the semigroup J^* consisting of the set $J \cup \{0\}$, with multiplication $*$ given by

$$s * t = \begin{cases} st & \text{if } s, t, st \in J, \\ 0 & \text{otherwise.} \end{cases}$$

The principal factor of a $\mathcal{J}$-class must always be a 0-simple semigroup, and so we can apply the linked triple methods from [2] to find all the congruences of J^* and compute with them in an efficient manner. Since we can do this for all the $\mathcal{J}$-classes of S, it would be desirable if we could combine the congruences of all the principal factors in some way to give a description of a congruence on S. If there are n $\mathcal{J}$-classes in S, then we are looking for a theorem which associates a congruence ρ on S with an n-tuple whose entries are all congruences on the principal factors of the $\mathcal{J}$-classes. If such a theorem could be found,

the simplification of 0-simple semigroup congruences to linked triples could be extended to all finite semigroups.

4.2 Attempts

In this section we show some attempts to find a theorem to connect the congruences on a semigroup S to combinations of congruences on the $\mathcal{J}$-classes.

Given a congruence ρ on S, for each $\mathcal{J}$-class J we want a congruence ρ_J on J^* which is in some sense a "restriction" of ρ to J. The first definition we could try is to relate elements of J if and only if they are related in ρ, and to relate 0 only to itself. That is,

$$\rho_J = \big(\rho \cap (J \times J)\big) \cup \{(0,0)\}.$$

This seems a natural definition to use. However, if we use this definition it turns out that in some cases ρ_J is not a congruence:

Example 4.2. Let S be the semigroup generated by the two transformations

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 3 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 3 \end{pmatrix},$$

let J be the highest $\mathcal{J}$-class, and let $\rho = S \times S$. Then ρ_J as defined above is not a congruence.

Proof. For ease of writing, let us omit the source of transformations and simply write the image of the transformation in square brackets, so in this case $S = \langle [2\ 3\ 3], [3\ 1\ 3] \rangle$. Now let $\rho = S \times S$, the universal congruence on S. S has five elements: [1 3 3],[2 3 3],[3 1 3], and [3 2 3] which belong in one $\mathcal{J}$-class J; and [3 3 3] which belongs in a lower $\mathcal{J}$-class. Now let $x = y = z = [2\ 3\ 3]$, and let $t = [3\ 2\ 3]$. Certainly $(x,y),(z,t) \in \rho$, and since $x, y, z, t \in J$, we also have $(x,y),(z,t) \in \rho_J$. However, we multiply the two pairs to give $yt = [2\ 3\ 3] \in J$ and $xz = [3\,3\,3] \notin J$. In the context of J^* therefore, $xz = 0$, and so $(xz, yt) \notin \rho_J$, which shows that ρ_J is not a congruence. □

Clearly if ρ_J is not a congruence on J^* then it is not useful to us. We therefore attempt another definition of ρ_J, hoping to find that it is a congruence and also has the properties we want.

The problem with our last definition was that it did not account for congruences which had an element in J related to an element in a lower $\mathcal{J}$-class. Our last definition of ρ_J always has 0 in a class on its own, but there are certainly congruences ρ which relate elements from different $\mathcal{J}$-classes.

Since J^* is a 0-simple semigroup, we can define all its congruences by linked triples, *except* the universal congruence $J^* \times J^*$. This universal congruence is the only one which relates 0 to any non-zero element, since we know that for a 0-simple semigroup 0 is related either to everything or only to itself. We did not take this possibility into account when defining ρ_J. Therefore, let us give a new definition which applies this concept:

$$\rho_J = \begin{cases} J^* \times J^* & \text{if } \exists\, a \in J, b \in S \setminus J : (a,b) \in \rho, \\ \big(\rho \cap (J \times J)\big) \cup \{(0,0)\} & \text{otherwise.} \end{cases}$$

Firstly, we show that by this new definition, ρ_J is always a congruence.

Example 4.3. Let S be a semigroup, let J be a $\mathcal{J}$-class in S, and let ρ be a congruence on S. Then ρ_J is a congruence.

Proof. First let us rephrase ρ_J to help with the proof. A pair (x, y) is in ρ_J if and only if at least one of the following is true:

1. $x, y \in J$ and $(x, y) \in \rho$,
2. $x = y = 0$,
3. there exist $a \in J$ and $b \in S \setminus J$ such that $(a, b) \in \rho$.

If $x \in J$ then $(x, x) \in \rho_J$ by rule 1, and $(0, 0) \in \rho_J$ by rule 2, so ρ_J is reflexive. If $(x, y) \in \rho_J$ by any of the three rules then $(y, x) \in \rho_J$ by the same rule, so ρ_J is also symmetric.

To show transitivity, let $(x, y), (y, z) \in \rho_J$. Each of (x, y) and (y, z) is related by one of the three rules above, so there are nine cases in total:

- (11): $x, y, z \in J$ and $(x, z) \in \rho$ by transitivity, so $(x, z) \in \rho_J$ by (1).
- (12, 21): This case is impossible, since it would imply that $y \in J$ and $y = 0$.
- (22): $x = y = z = 0$, so $(x, z) \in \rho_J$ by (2).
- (13, 23, 31, 32, 33): There exist $a \in J$ and $b \in S \setminus J$ such that $(a, b) \in \rho$, so $(x, z) \in \rho_J = J^* \times J^*$ by (3).

Hence $(x, z) \in \rho_J$, so ρ_J is transitive.

Finally, let $(x, y), (z, t) \in \rho_J$. The relation ρ_J is a congruence if and only if we can show that $(xz, yt) \in \rho_J$ (something we disproved in Example 4.2 for the previous definition of ρ_J). Again since each of (x, y) and (z, t) can be in ρ_J by any of three different rules, we have nine cases in total:

- (11): $x, y, z, t \in J$ and $(x, y), (z, t) \in \rho$. Since ρ is a congruence, we have $(xz, yt) \in \rho$. Now we have three cases: either xz and yt are both in J, in which case $(xz, yt) \in \rho_J$ by (1); or one of xz and yt is in J and the other is not, in which case (3) is satisfied and so $(xz, yt) \in \rho_J = J^* \times J^*$; or finally, $xz = yt = 0 \in J^*$, which satisfies (2) and so $(xz, yt) \in \rho_J$.
- (12): $x, y \in J$ and $z = t = 0$. Hence $xz = yt = 0$, so $(xz, yt) \in \rho_J$ by (2).
- (21): $x = y = 0$ and $z, t \in J$. Hence $xz = yt = 0$, so $(xz, yt) \in \rho_J$ by (2).
- (22): $x = y = z = t = 0$, so $xz = yt = 0$, so $(xz, yt) \in \rho_J$ by (2).
- (13, 23, 31, 32, 33): There exist $a \in J$ and $b \in S \setminus J$ such that $(a, b) \in \rho$, so $(xy, zt) \in \rho_J = J^* \times J^*$ by (3).

Hence $(xy, zt) \in \rho_J$, so ρ_J is a congruence. □

Now for any congruence ρ on a semigroup S we have a congruence ρ_J on the principal factor of each $\mathcal{J}$-class. In order to make this useful, we have introduced the rule that if any element $x \in J$ is ρ-related to an element outside J, then $(x, 0) \in \rho_J$ and so $\rho_J = J^* \times J^*$. This is necessary for ρ_J to be a congruence, but it is only a useful construction if we can rely on ρ_J being, in

some sense, a "restriction" of ρ – if there is a pair of elements $x, y \in J$ such that $(x, y) \in \rho_J$ then we must have $(x, y) \in \rho$. If this condition is not satisfied, then any attempt we make to describe ρ by its $\mathcal{J}$-class congruence ρ_J will be unhelpful, since information about which elements in J are related will have been lost.

In fact, ρ_J may not be a restriction of ρ, as seen in the following example:

Example 4.4. We shall now define a semigroup S with two $\mathcal{J}$-classes, and a congruence ρ, such that there is a $\mathcal{J}$-class J and two elements $x, y \in J$ such that $(x, y) \in \rho_J$ but $(x, y) \notin \rho$.

Let J and K be groups both isomorphic to the symmetric group S_3. If x is a permutation in S_3, then we denote its images in J and K by x_J and x_K respectively. For example, we have the element $(1\ 3\ 2)_J \in J$, $(2\ 3)_K \in K$, and the identity $e_K \in K$. Furthermore we define two maps: $\phi : J \cup K \to S_3$ simply maps x_J to x and x_K to x, for each $x \in S_3$; and $\phi_K : J \cup K \to K$ maps x_J to x_K and x_K to itself.

Now we define the semigroup S as the set $J \cup K$, together with multiplication

$$x * y = \begin{cases} xy & \text{if } x, y \in J, \\ (x\phi_K)(y\phi_K) & \text{otherwise,} \end{cases}$$

and we define a relation ρ on S by the rule that $(x, y) \in \rho$ if and only if $x\phi$ and $y\phi$ are in the same coset of A_3. Clearly ρ is an equivalence, since it splits S into precisely two classes. If $(x, y) \in \rho$ and $a \in S$, then $x\phi$ and $y\phi$ are in the same coset of A_3. Since A_3 is a normal subgroup of S_3, we know that xa and ya must be in the same coset, as well as ax and ay. Hence ρ is a congruence.

Clearly J and K are the two $\mathcal{J}$-classes of S. Consider the elements $(1\ 2)_J$ and $(1\ 2)_K$. Since these elements map by ϕ to the same element in S_3, they are certainly ρ-related. This means that we have $x \in J$ and $y \in S \setminus J$ such that $(x, y) \in \phi$, and so $\rho_J = J^* \times J^*$.

Now consider the elements $(1\ 2)_J$ and $(1\ 2\ 3)_J$: since $(1\ 2\ 3) \in A_3$ but $(1\ 2) \notin A_3$, $(1\ 2)_J$ is not ρ-related to $(1\ 2\ 3)_J$. But we already know that all elements of J are ρ_J-related to each other. Hence we have $x, y \in J$ such that $(x, y) \in \rho_J$ but $(x, y) \notin \rho$. So in this case, ρ_J is not a "restriction" of ρ.

We have now tried two different ways of extending the *linked triples* method from 0-simple semigroups to arbitrary semigroups via their $\mathcal{J}$-classes, but there does not seem to be an easy way of doing so. Perhaps with more investigation it would be possible to find a satisfactory way of doing this. For now, we turn our attention to another class of semigroups with interesting congruence theory: the inverse semigroups.

Chapter 5

Inverse Semigroups

Another category of semigroup with an interesting characterisation of congruences is the *inverse semigroups*, where each congruence has a *congruence pair* which can be calculated, and used to determine the pairs of the congruence more quickly. Let us start by detailing some background theory about inverse semigroups:

5.1 Background Theory

Definition 5.1. Let S be a semigroup and $x \in S$. An element $y \in S$ is the **inverse** of x if and only if

$$xyx = x \text{ and } y = yxy.$$

[6, p.11]

Definition 5.2. An **inverse semigroup** is a semigroup S such that every element $x \in S$ has a unique inverse $x^{-1} \in S$. [6, p.158]

This section will consider the idempotents of S. Recall that an idempotent of S is any element $e \in S$ such that $ee = e$.

Theorem 5.3. *Let x be an element of an inverse semigroup S. Then xx^{-1} is an idempotent.*

Proof. $(xx^{-1})(xx^{-1}) = (xx^{-1}x)x^{-1} = xx^{-1}$. □

Theorem 5.4. *In an inverse semigroup S, the set of idempotents E is a commutative subsemigroup of S.*

Proof. Let $e, f \in E$ be idempotents, so that $ee = e$ and $ff = f$. Their product ef is in S, and so has an inverse which we call z (hence, $(ef)z(ef) = (ef)$ and $z(ef)z = z$).

Now, consider the element fze:

$$(ef)(fze)(ef) = effzeef = efzef = ef,$$

$$(fze)(ef)(fze) = fzeeffze = f(zefz)e = fze,$$

so fze is an inverse of ef, and z is also an inverse of ef. But by the definition of an inverse semigroup, ef has only one inverse, so $z = fze$. This tells us that z is an idempotent, since

$$zz = (fze)(fze) = f(zefz)e = fze = z.$$

Now, since z is an idempotent, z must be an inverse of z (since $z(z)z = z$), and since ef is by definition the inverse of z, we have that $ef = z$, so ef is also an idempotent. Therefore, we have that $ef \in E$, so E is closed and is a subsemigroup of S.

We now only need to show that idempotents commute. For $e, f \in E$, both ef and fe are idempotents that are self-inverse. Now we can show that fe is the inverse of ef, since

$$(ef)(fe)(ef) = effeef = (ef)(ef) = ef,$$

$$(fe)(ef)(fe) = feeffe = (fe)(fe) = fe,$$

and since ef is the inverse of ef and fe, by uniqueness, we have

$$ef = fe.$$

Since e and f are arbitrary idempotents, we can see that idempotents commute.
[1, p.146] □

Now that we have defined an inverse semigroup and its idempotent semigroup, we can start thinking about its congruences. Unless stated otherwise, in this section S refers to an inverse semigroup and ρ refers to a congruence on S.

Definition 5.5. The **trace** of ρ is the restriction of ρ to the idempotents of S. That is,

$$\operatorname{tr}\rho = \rho \cap (E \times E).$$

[1, p.155]

Lemma 5.6. *The trace of ρ is a congruence on E.*

Proof. Clearly $\operatorname{tr}\rho \subseteq E \times E$. Since ρ is a congruence on S, $\operatorname{tr}\rho$ contains every pair (e, e) such that $e \in E$, and so $\operatorname{tr}\rho$ is reflexive.

To show symmetry, let $(e, f) \in \operatorname{tr}\rho$. Hence $(e, f) \in \rho$, and since ρ is symmetric we have $(f, e) \in \rho$. Since f and e are both in E, $(f, e) \in \rho \cap (E \times E)$, and so $\operatorname{tr}\rho$ is symmetric.

Now let $(e, f), (f, g) \in \operatorname{tr}\rho$. Hence $(e, f), (f, g) \in \rho$, and since ρ is transitive we must also have $(e, g) \in \rho$. Since e and g are both in E we have $(e, g) \in \operatorname{tr}\rho$, and so $\operatorname{tr}\rho$ is transitive.

Finally, let $(e, f) \in \operatorname{tr}\rho$ and $g \in E$. Since E is closed under multiplication, we have the four elements $ge, gf, eg, fg \in E$. Since ρ is compatible on S, we have $(ge, gf), (eg, fg) \in \rho$, and since these pairs lie in $E \times E$, we have $(ge, gf), (eg, fg) \in \operatorname{tr}\rho$. Hence $\operatorname{tr}\rho$ is compatible, and so it is a congruence on E. □

Lemma 5.7. *The trace of ρ is **normal** in S, which is to say that*

$$(a^{-1}ea, a^{-1}fa) \in \operatorname{tr}\rho$$

for every pair $(e, f) \in \operatorname{tr}\rho$ and every element $a \in S$.

Proof. First, observe that $(e, f) \in \rho$ and since ρ is compatible, that $(a^{-1}ea, a^{-1}fa) \in \rho$. Now we need only show that $a^{-1}ea$ and $a^{-1}fa$ are in E. We consider the following:

$$(a^{-1}ea)(a^{-1}ea) = a^{-1}e(aa^{-1})ea = a^{-1}(aa^{-1})eea = a^{-1}ea,$$

using the facts that idempotents commute, that aa^{-1} is an idempotent, and that $a^{-1}aa^{-1} = a^{-1}$. Hence $a^{-1}ea$ and similarly $a^{-1}fa$ are in E, so $(a^{-1}ea, a^{-1}fa) \in \operatorname{tr}\rho$. [1, p.155] □

A congruence's trace goes some way towards describing the congruence completely. The only other thing which is needed is its *kernel*, defined as follows:

Definition 5.8. The **kernel** of ρ is the union of all the ρ-classes of S which contain idempotents. That is,

$$\operatorname{Ker}\rho = \bigcup_{e \in E} \rho_e.$$

[1, p.155]

Like the trace, the kernel has certain characteristics which can be identified from its definition, such as the following lemma which uses the new definition of a *normal subsemigroup.*

Definition 5.9. A subsemigroup N of a semigroup S is called **normal** if it is *full* (contains all the idempotents of S) and *self-conjugate* ($a^{-1}xa \in N$ for all $x \in N, a \in S$). [1, p.155]

Lemma 5.10. *The kernel of ρ is a normal subsemigroup of S. [1, p.155]*

Proof. First we should show that $\operatorname{Ker}\rho$ is a subsemigroup of S. Let $x, y \in \operatorname{Ker}\rho$. By the definition of $\operatorname{Ker}\rho$, there exist $e, f \in E$ such that $(x, e), (y, f) \in \rho$. Since ρ is a congruence,

$$(xy, ef) \in \rho,$$

and so since $ef \in E$, we have $xy \in \operatorname{Ker}\rho$, so $\operatorname{Ker}\rho$ is closed, and is a subsemigroup.

Clearly $\operatorname{Ker}\rho$ contains every idempotent of S, so it is full. We now only need to show that it is self-conjugate. Let $x \in N$ and $a \in S$. There must be some $e \in E$ such that $(x, e) \in \rho$, and since ρ is compatible, $(a^{-1}xa, a^{-1}ea) \in \rho$ as well. Since idempotents commute, we have

$$(a^{-1}ea)(a^{-1}ea) = a^{-1}e(aa^{-1})ea = a^{-1}(aa^{-1})eea = (a^{-1}aa^{-1})(ee)a = a^{-1}ea,$$

so $a^{-1}ea$ is an idempotent. Hence $a^{-1}xa \in \operatorname{Ker}\rho$, so $\operatorname{Ker}\rho$ is normal. □

We will soon see that the pair $(\operatorname{Ker}\rho, \operatorname{tr}\rho)$ is all that is needed to completely describe the congruence ρ – all of ρ's pairs and classes can be determined from these two components. It would also be desirable for us to say that all such pairs (N, τ) (consisting of a normal subsemigroup and a normal congruence) describe congruences on S. This is nearly true, in that every congruence on S can be described by some pair of this description. However, the kernel and trace of ρ have two more properties which must be observed – properties without which (N, τ) cannot describe a congruence.

Lemma 5.11. *Let ρ be a congruence on an inverse semigroup S. For all $a \in S$ and $e \in E$,*

1. *If $ae \in \operatorname{Ker}\rho$ and $(e, a^{-1}a) \in \operatorname{tr}\rho$, then $a \in \operatorname{Ker}\rho$.*
2. *If $a \in \operatorname{Ker}\rho$, then $(aa^{-1}, a^{-1}a) \in \operatorname{tr}\rho$.*

Proof. (1): Let $a \in S$ and $e \in E$ such that $ae \in \operatorname{Ker}\rho$ and $(e, a^{-1}a) \in \operatorname{tr}\rho$. Then there exists some $f \in E$ such that $(ae, f) \in \rho$. Since $(e, a^{-1}a) \in \rho$ and ρ is a congruence,

$$(ae, aa^{-1}a) = (ae, a) \in \rho,$$

and since $(ae, f) \in \rho$, we have $(a, f) \in \rho$, so $a \in \operatorname{Ker}\rho$.

(2): Let $a \in \operatorname{Ker}\rho$. There must be some $e \in E$ such that $(a, e) \in \rho$, and taking inverses, we must have $a^{-1}\ \rho\ e^{-1} = e$. Hence aa^{-1} and $a^{-1}a$ are both ρ-related to ee, so by transitivity, $(aa^{-1}, a^{-1}a) \in \operatorname{tr}\rho$. [1, p.155-6] □

These last two properties now give us all we need to give a useful characterisation of congruences on inverse semigroups: an abstract description of a pair which forms the kernel and trace of a congruence.

Definition 5.12. For an inverse semigroup S with idempotent semigroup E, a **congruence pair** is a pair (N, τ) consisting of a normal subsemigroup N of S and a normal congruence τ on S, such that

1. If $ae \in N$ and $(e, a^{-1}a) \in \tau$, then $a \in N$
2. If $a \in N$, then $(aa^{-1}, a^{-1}a) \in \tau$

for all elements $a \in S$ and $e \in E$. [1, p.156]

Now we can state the result which bijectively identifies congruence pairs with congruences, and allows for the computational representation we present in the next section.

Theorem 5.13. *Let ρ be a congruence on an inverse semigroup S with idempotent semigroup E. $(\operatorname{Ker}\rho, \operatorname{tr}\rho)$ is a congruence pair, and conversely, every congruence pair (N, τ) defines a congruence*

$$\rho_{(N,\tau)} = \{(x, y) \in S \times S \mid (x^{-1}x, y^{-1}y) \in \tau, xy^{-1} \in N\}$$

whose kernel is equal to N and whose trace is equal to τ. Finally, $\rho_{(\operatorname{Ker}\rho, \operatorname{tr}\rho)} = \rho$.

Proof. Let ρ be a congruence on S. Lemmas 5.7, 5.10 and 5.11 directly give us that $(\operatorname{Ker}\rho, \operatorname{tr}\rho)$ is a congruence pair. We need only prove the converse, that every congruence pair defines a congruence.

Let (N, τ) be a congruence pair, and let

$$\rho = \rho_{(N,\tau)} = \{(x, y) \in S \times S \mid (x^{-1}x, y^{-1}y) \in \tau, xy^{-1} \in N\}$$

as in the theorem. We want to show that ρ is a congruence.

The full subsemigroup N contains every idempotent in S – for any element $x \in S$, $xx^{-1}xx^{-1} = xx^{-1}$, so xx^{-1} is an idempotent and so is in N. And since τ is reflexive, $(x^{-1}x, x^{-1}x) \in \tau$, so $(x, x) \in \rho$, and ρ is reflexive.

Now let $(x, y) \in \rho$. Hence $(x^{-1}x, y^{-1}y) \in \tau$, and since τ is symmetric, $(y^{-1}y, x^{-1}x) \in \tau$ also. We also have $xy^{-1} \in N$ and so, since N is an inverse semigroup,[1, p.155] $(xy^{-1})^{-1} = yx^{-1} \in N$. Hence $(y, x) \in \rho$ and ρ is symmetric.

Let $(x, y), (y, z) \in \rho$. Hence $(x^{-1}x, y^{-1}y), (y^{-1}y, z^{-1}z) \in \tau$, and since τ is transitive, $(x^{-1}x, z^{-1}z) \in \tau$. Also $xy^{-1}, yz^{-1} \in N$, so by closure $x(y^{-1}y)z^{-1} \in N$. Certainly $y^{-1}y$ is an idempotent, and as stated above, is τ-related to $x^{-1}x$. Hence

$$x(y^{-1}y)z^{-1} = x(y^{-1}y)(z^{-1}zz^{-1}) = x(z^{-1}z)(y^{-1}y)z^{-1}.$$

Let $e = z(y^{-1}y)z^{-1}$, so $xz^{-1}e \in N$. Since $(x^{-1}x, y^{-1}y) \in \tau$ and since τ is normal,

$$\big(z(x^{-1}x)z^{-1}, z(y^{-1}y)z^{-1}\big) \in \tau,$$

which is to say that $\big(e, (xz^{-1})^{-1}(xz^{-1})\big) \in \tau$. Hence by Definition 5.12 part 1, $xz^{-1} \in N$, and so $(x, z) \in \rho$ and ρ is transitive.

Finally to show that ρ is a congruence, we need to show that it is both left- and right-compatible. Let $(x, y) \in \rho$ and $a \in S$. We have $(x^{-1}x, y^{-1}y) \in \tau$, and so, using the fact that τ is normal:

$$(a^{-1}x^{-1}xa, a^{-1}y^{-1}ya) \in \tau,$$

which is to say that $\big((xa)^{-1}(xa), (ya)^{-1}(ya)\big) \in \tau$.

Now to examine the elements in N: since $(x, y) \in \rho$ we have $xy^{-1} \in N$; note also that since N is full, the idempotent $aa^{-1} \in N$, and since N is self-conjugate, $y(aa^{-1})y^{-1} \in N$ as well. Now,

$$(xa)(ya)^{-1} = xaa^{-1}y^{-1} = x(aa^{-1})(y^{-1}y)y^{-1} = \big(xy^{-1}\big)\big(y(aa^{-1})y^{-1}\big) \in N.$$

So $(xa)(ya)^{-1} \in N$, and therefore $(xa, ya) \in \rho$ as required, so ρ is left-compatible. By similar logic, $(ax, ay) \in \rho$ and ρ is right-compatible. Hence ρ is a congruence.

Next we want to show that $\operatorname{Ker}\rho = N$. First let $x \in \operatorname{Ker}\rho$, so there must be some $e \in E$ such that $(x, e) \in \rho$. Hence by the definition of ρ, $x^{-1}x \;\tau\; e^{-1}e = e$ and $xe^{-1} = xe \in N$. Now Definition 5.12 part (1) gives us $x \in N$. So $\operatorname{Ker}\rho \subseteq N$.

Now let $x \in N$, and let $e = x^{-1}x$. Hence $xe^{-1} = x \in N$, and

$$x^{-1}x = x^{-1}xx^{-1}x = e^{-1}e,$$

so certainly $(x^{-1}x, e^{-1}e) \in \tau$, and so $(x, e) \in \rho$, and since $e \in E$, we have $x \in \operatorname{Ker}\rho$. So $\operatorname{Ker}\rho = N$.

Now we consider the trace. Let $(e, f) \in \operatorname{tr}\rho$. Hence $e, f \in E$ and $(e, f) \in \rho$. From the definition of ρ, this means that $(e^{-1}e, f^{-1}f) \in \tau$. Since e and f are both idempotents, they are both self-inverse, and so $e^{-1}e = e$ and $f^{-1}f = f$. Hence $(e, f) \in \tau$, so $\operatorname{tr}\rho \subseteq \tau$.

Conversely, let $(e, f) \in \tau$. Since τ is a congruence on E, we have $e, f \in E$, so as before, $e^{-1}e = e$ and $f^{-1}f = f$. Hence $(e^{-1}e, f^{-1}f) \in \tau$. Also, $ef^{-1} = ef$, which is an idempotent and therefore $ef^{-1} \in N$. Therefore by the definition of ρ, $(e, f) \in \rho$, and since e and f are both in E, we have $(e, f) \in \operatorname{tr}\rho$, as required. So $\operatorname{tr}\rho = \tau$.

To complete the proof, we want to show that our original congruence ρ is the same as the congruence $\sigma = \rho_{(\operatorname{Ker}\rho, \operatorname{tr}\rho)}$ generated by the congruence pair formed from its kernel and trace.

Let $(x, y) \in \rho$, so by taking inverses we also have $(x^{-1}, y^{-1}) \in \rho$. Now since ρ is a congruence, we can multiply these to give $(x^{-1}x, y^{-1}y) \in \rho$ (and since these two elements are idempotents, $(x^{-1}x, y^{-1}y) \in \operatorname{tr}\rho$), and by right-multiplication, $(xy^{-1}, yy^{-1}) \in \rho$ – which gives us $xy^{-1} \in \operatorname{Ker}\rho$ since yy^{-1} is an idempotent. Hence $(x, y) \in \sigma$, so $\rho \subseteq \sigma$.

Now let $(x, y) \in \sigma$. By the definition of σ we have $(x^{-1}x, y^{-1}y) \in \operatorname{tr}\rho$ and $xy^{-1} \in \operatorname{Ker}\rho$. There must be some $e \in E$ such that $(xy^{-1}, e) \in \rho$. Since this e is an idempotent, we have $e = e^{-1}e$ and we clearly have $\left(e^{-1}e, (xy^{-1})^{-1}(xy^{-1})\right) \in \rho$, so by transitivity, $(xy^{-1}, yx^{-1}xy^{-1}) \in \rho$. We can apply this in the following chain of equalities and congruences:

$$x = xx^{-1}x \ \rho \ xy^{-1}y \ \rho \ yx^{-1}xy^{-1}y \ \rho \ yy^{-1}yy^{-1}y = yy^{-1}y = y$$

Hence $(x, y) \in \rho$ and so $\rho = \sigma$, and the identification of congruences to congruence pairs is complete. [1, p.157-8] □

5.2 Algorithms

Now that we have established a good base of theory regarding congruences on inverse semigroups, we can describe an efficient way of representing these congruences computationally.

5.2.1 Representing a congruence by its congruence pair

To store a semigroup congruence computationally, we have several methods which have already been described; we can store a list of all the pairs in the congruence (perhaps the least efficient way); we can store a set of sets of semigroup elements, where each set specifies one congruence class; and we can store a lookup table as described in Section 2.2. But for inverse semigroups, we can now describe a new way: storing the *congruence pair* – the kernel and trace – of the congruence.

The first thing we want when implementing this in a computational algebra package is the possibility for a user to specify a congruence by supplying its congruence pair. If a user gives a valid congruence pair (N, τ), we can simply store that pair, and by Theorem 5.13 all information about that congruence can be extracted when needed. However, we do need a method to determine whether (N, τ) is indeed a congruence pair, by checking it against Definition 5.12. To this end, we give the following algorithm.

Require: S and N are finite inverse semigroups, τ is a congruence
  procedure IsCongruencePair($S, (N, \tau)$)
    Let X be a generating set for S
    Let $\tau_1, \tau_2, \ldots \tau_n$ be the congruence classes of τ
    if N is not a subsemigroup of S **then**
      return false
    end if
    if NrIdempotents(N) $\neq |E|$ **then** ▷ *Is N full?*
      return false

```
    end if
    for a ∈ N do                                   ▷ Is N self-conjugate?
        for x ∈ X do
            if x^{-1}ax ∉ N then
                return false
            end if
        end for
    end for
    ▷ Check conditions (1) and (2) in Def 5.12
    for i ∈ {1...n} do
        for f ∈ τ_i do
            for a ∈ L_f do
                if a ∈ N then
                    if aa^{-1} ∉ τ_i then
                        return false
                    end if
                else
                    for e ∈ τ_i do
                        if ae ∈ N then
                            return false
                        end if
                    end for
                end if
            end for
        end for
    end for
    return true
end procedure
```

There are a few remarks which must be made to justify parts of the last algorithm. The first is that we make use of a function called NrIdempotents which we assume to exist. This simply takes a semigroup and returns the number of its idempotents. Since $N \subseteq S$, if N and S have the same number of idempotents ($|E|$) then N is full. The Semigroups package [4] of the GAP system [3] contains such a method, which is used in the attached implementation.

In checking that N is self-conjugate, we test whether $x^{-1}ax \in N$ for all $a \in N$, but only for x in the *generators* of S. This greatly reduces the amount of work done in most cases, and relies on the observation that if $x \in S$ is the product of generators $x_1 \cdot x_2 \cdots x_n$, then

$$x^{-1}ax = x_n^{-1} \cdots x_2^{-1} x_1^{-1} a x_1 x_2 \cdots x_n,$$

and so by observing that $x_1^{-1}ax_1 \in N$, we can repeat the logic to see that $x_2^{-1}(x_1^{-1}ax_1)x_2 \in N$ and so on. It would be desirable if we only had to conduct this test for a in the generators of N, but it is not clear that

$$x^{-1}ax, x^{-1}bx \in N \quad \Rightarrow \quad x^{-1}abx \in N,$$

so in this algorithm we are required to enumerate all the elements of N.

Verifying conditions (1) and (2) of Definition 5.12 could be quite expensive computationally, particularly condition (1) which implies testing three conditions for every combination of $a \in S$ and $e \in E$. To improve the time complexity, rather than going through every element $a \in S$, we instead observe that $a^{-1}a$ is an idempotent, and therefore we go through just the idempotents, a much smaller set, indexed by the congruence classes of τ. For each idempotent f, we need to identify the elements a such that $a^{-1}a = f$ – we use the following lemma:

Lemma 5.14. *Let $f \in E$ and $a \in S$. $f = a^{-1}a$ if and only if $(a, f) \in \mathcal{L}$.*

Proof. ($\Rightarrow$): Let $f = a^{-1}a$. Hence $a \cdot f = aa^{-1}a = a$ and $a^{-1} \cdot a = f$, so $(a, f) \in \mathcal{L}$.

($\Leftarrow$): Now let $(a, f) \in \mathcal{L}$. As above, we know that $(a, a^{-1}a) \in \mathcal{L}$, which means that f and $a^{-1}a$ are idempotents in the same $\mathcal{L}$-class of S. Since S is inverse, there is only one idempotent in each $\mathcal{L}$-class[1, p.145], and so $f = a^{-1}a$. □

Hence for each idempotent f, we simply look through its $\mathcal{L}$-class L_f and inspect each a we find, knowing that for each one $f = a^{-1}a$. Since we are going through the idempotents f in order of the τ-classes, we know immediately which τ-class $a^{-1}a$ is in. If a is found to be in N then by condition (2) we must have aa^{-1} in the same class.

If, however, a is found not to be in N, then we instead test condition (1), which may perhaps be best rewritten as

1. If $a \notin N$ and $(e, a^{-1}a) \in \tau$, then $ae \notin N$.

Since every $a \in S$ is in an $\mathcal{L}$-class, and since every $\mathcal{L}$-class has an idempotent, we will find every $a \notin N$ at this part of the loop, and since we have already identified τ_i which contains precisely the idempotents e such that $(e, a^{-1}a) \in \tau$, it makes sense to perform the test for condition (1) here. It is as simple as going through each $e \in \tau_i$ and returning false if for any one of them $ae \in N$.

By using the improvements above, we arrive at the algorithm which was given, which has the advantage of listing the elements of each τ-class just once, using it, and then moving onto the next τ-class. For this reason, a good implementation of the congruence τ itself might well be as a partition of E – a list of lists of elements, where each list describes a class. This is the approach taken in the attached GAP implementation.

5.2.2 Pair Inclusion

Now we are storing a congruence ρ on an inverse semigroup S as a congruence pair (N, τ), we need a way of performing some common operations, most notably determining whether a pair (x, y) is in ρ. This is actually quite simple, and in most cases computationally cheaper than by using the generating pairs method described in Chapter 2.

The method is derived straight from Theorem 5.13, which gives us

$$\rho = \{(x, y) \in S \times S \mid (x^{-1}x, y^{-1}y) \in \tau, xy^{-1} \in N\}.$$

The algorithm is therefore as follows:

```
procedure (x, y) ∈ ρ
    if x^{-1}x ∈ τ_{y^{-1}y} then
        if xy^{-1} ∈ N then
            return true
        end if
    end if
    return false
end procedure
```

5.2.3 Class Evaluation

One last important operation we might want to carry out is to evaluate a class – that is, given an element $x \in S$, to find a list of all the elements to which x is ρ-related. To accomplish this, we again use Lemma 5.14 to find every $y \in S$ such that $(x^{-1}x, y^{-1}y) \in \tau$): for each e in the τ class of $x^{-1}x$, find all the elements $y \in L_e$, elements that are $\mathcal{L}$-related to e.

The algorithm is given as follows:

```
procedure GetRelatedElements(ρ, x)
    ρ_x := ∅
    for e ∈ τ_{x^{-1}x} do
        for y ∈ L_e do
            if xy^{-1} ∈ N then
                Add y to ρ_x
            end if
        end for
    end for
    return ρ_x
end procedure
```

These algorithms have been implemented in the code of the attached file `inverse-cong.gi` – this is to be included in the Semigroups package [4] of the GAP system [3], along with simple functions to allow the manipulation of objects representing congruence classes and quotients.

Chapter 6

Evaluation

We have considered the theory of congruences for several different types of semigroup: we have used some existing theory from sources in the bibliography, as well as developing some new theory for ourselves in the form of new lemmas and theorems, and we have applied the sum of this knowledge to create several new algorithms. In that sense, a large part of this report represents original research which improves the complexity of several common problems in computational semigroup theory.

Indeed, attached to this report is a body of original GAP code which applies these algorithms so that they can actually be used, and full advantage can be taken of the improvements they represent. Their performance is tested in the next section.

6.1 Benchmarking

Several parts of the GAP code attached to this report were tested against existing implementations in the GAP library [3] or the Semigroups package [4], and the time taken to complete certain calculations was recorded. The results are presented here for comparison.

6.1.1 Generating Pairs

The algorithm described in Section 2.2 was implemented in the file `pairs-cong.gi` attached to this report. We constructed a semigroup, and a congruence by generating pairs, and we tested the `in` method (for determining whether a particular pair is in a congruence) for three different pairs (x, y) and recorded the time taken to return a result. Then, for each pair (x, y) we used an existing GAP implementation of semigroup congruences to test an equivalent condition (whether x is in the congruence class containing y) and again recorded the time taken.

The semigroup S chosen was a random transformation semigroup on 6 points with 6 generators. The precise semigroup can be specified to the program as follows.

```
gens := [ Transformation( [ 1, 3, 4, 1, 3, 5 ] ),
          Transformation( [ 2, 4, 6, 1, 6, 5 ] ),
          Transformation( [ 4, 1, 2, 6, 2, 1 ] ),
```

```
          Transformation( [ 4, 6, 4, 3, 3, 3 ] ),
          Transformation( [ 5, 1, 6, 1, 6, 3 ] ),
          Transformation( [ 5, 2, 5, 3, 5, 3 ] ) ];
s := Semigroup( gens );
```

The congruence ρ was specified by a set of two randomly-generated pairs; it can be specified as follows.

```
cgens := [ [ Transformation( [ 5, 5, 2, 4, 2, 4 ] ),
             Transformation( [ 1, 5, 4, 5, 4, 5 ] ) ],
           [ Transformation( [ 3, 3, 3, 6, 3, 3 ] ),
             Transformation( [ 1, 6, 6, 6, 6, 1 ] ) ] ];
cong := SemigroupCongruenceByGeneratingPairs( s, cgens );
```

Then random pairs (x, y) were created and passed to the two different functions. The times taken to complete are shown in the following table. Note that x and y are transformations, given in the table as the image which should be passed to GAP to specify them. The "Old" column shows the time taken in milliseconds to determine whether $(x, y) \in \rho$ using the previous implementation in the GAP library, and "New" shows that of the implementation attached to this report. Each test was performed 20 times, and the figure shown is the average.

x	y	Time (ms)	
		Old	New
`[ 3, 4, 6, 3, 6, 4 ]`	`[ 1, 5, 1, 3, 1, 5 ]`	5261	1996
`[ 5, 3, 3, 5, 3, 4 ]`	`[ 3, 2, 2, 2, 2, 3 ]`	9767	2668
`[ 6, 1, 1, 4, 1, 6 ]`	`[ 6, 3, 5, 6, 5, 3 ]`	9331	3633

It should be noted that the semigroup and congruence were re-initialised in GAP immediately before each test, and so a full enumeration was started before each one. In a single session without this re-initialisation, the information calculated so far about the congruence would be stored, and successive calls to test whether a pair is in the congruence would be faster as a result. In any case, it can be seen that the new algorithm performs slightly faster than the old.

We should also recall that while the old algorithm carries out explicit multiplications, the new one uses left and right Cayley graphs which must be calculated in advance. Similarly, a full list of elements must be produced so that we can look up elements as integers. In a real-life situation, it is quite possible that these objects may have been calculated before the user executes any congruence-related commands, and if so, it would have been stored for later use. In any case, it is a once-only overhead, and if several different pairs are tested then it is only on the very first call that these objects need to be calculated.

The amount of time it takes, for the semigroup S above, to calculate the left and right Cayley graphs and a list of elements, is on average 1309 ms (over 100 tests). We might therefore subtract 1309 from each entry in the "New" column above, if we were to assume that these objects had already been calculated. This adjustment is a far stronger claim for this new method over the existing one.

6.1.2 Linked Triple from Generating Pairs

Section 3.2 gives an algorithm for calculating the linked triple of a finite 0-simple semigroup congruence, using the generating pairs of the congruence. A naïve algorithm for this task was included in the code written to accompany [2], now a part of [4], and an implementation of the new algorithm has now replaced it in the file `reesmat-cong.gi` attached to this report. Here we constructed a finite 0-simple semigroup, and specified a congruence on it by generating pairs. Then we used the old and new algorithms to find the congruence's linked triple.

Let G be the permutation semigroup $\langle (3\ 6), (4\ 6) \rangle$ (isomorphic to S_3) and define P as a 13×27 matrix containing pairs from G^0 (due to its size, this matrix is omitted here). This allows us to form the matrix $S = \mathcal{M}^0[G; I, \Lambda; P]$.

Now we let ρ be a congruence generated by the pairs

$$\big((3, (4\ 6), 5), (17, (4\ 6), 5)\big),$$

$$\big((3, (3\ 4), 7), (17, (3\ 4), 7)\big).$$

This semigroup and congruence were chosen using random functions in GAP; S is 0-simple as required.

We used the two different algorithms to compute the linked triple of this congruence, and we performed each test multiple times. The old algorithm from [4], which we ran three times, took over 5 minutes to complete in each case (5m19s, 5m19s, 5m24s respectively). The new algorithm was run 1000 times, and always completed in less than one tenth of a second. On average, it took 4 milliseconds.

We carried out one more test using a different semigroup. Let $G = \langle (3\ 4) \rangle$ (a permutation group isomorphic to C_2), and let P be the matrix specified to GAP using the following code.

```
p := [
  [ 0, (3,4), 0, 0, (3,4), 0, (), 0, (3,4), 0, 0, (3,4), () ],
  [ 0, 0, (3,4), 0, 0, (3,4), (), (), (3,4), 0, (), 0, 0 ],
  [ 0, (3,4), (), (), (3,4), 0, 0, (3,4), 0, (), 0, 0, 0 ],
  [ 0, (3,4), 0, (), (3,4), (3,4), (), 0, (3,4), (), (), 0, 0 ],
  [ (), 0, (3,4), 0, (), (3,4), (3,4), 0, 0, (), 0, 0, (3,4) ],
  [ 0, (), (3,4), 0, (), (3,4), 0, (), 0, 0, (), (), (3,4) ],
  [ (3,4), (), (), 0, 0, (), 0, 0, (), (3,4), 0, (), 0 ],
  [ (), (3,4), 0, 0, 0, 0, (3,4), (3,4), 0, (), (3,4), (3,4), 0 ],
  [ 0, 0, 0, (), 0, (3,4), 0, 0, 0, (), (), (), (3,4) ],
  [ (), 0, (3,4), (3,4), (), 0, (3,4), 0, 0, 0, (3,4), (3,4), 0 ] ];
```

Let $S = \mathcal{M}^0[G; I, \Lambda; P]$ as before, and let ρ be the congruence with the single generating pair

$$\big((2, (3\ 4), 1), (2, e, 1)\big).$$

We calculated the linked triple again, several times with each method. The old method was run twenty times, and took around 89 seconds to complete (88.812s on average). The new method was run 1000 times, and again completed every time in under one tenth of a second. The average time was 4 milliseconds.

This is clearly a huge performance improvement, and the new code will soon replace the old code in [4].

6.1.3 Inverse Semigroup Congruences

Finally, we carried out tests on the method in Section 5.2.2, which determines for an inverse semigroup S and a congruence ρ on S, whether or not a pair (x, y) is in S.

We constructed a semigroup, and specified a congruence using generating pairs. Then we calculated its congruence pair using methods in the attached file `inverse-cong.gi`, and created an "inverse semigroup congruence by congruence pair" object. Three random pairs (x, y) were then created, and for each pair, we used the `in` method from `inverse-cong.gi` to determine whether (x, y) was in the new congruence object (the "new" method). Then, as in Section 6.1.1, we used the function in the GAP library to determine, using the generating pairs representation, whether x was in the congruence class of y (the "old" method). The time taken to complete each of these tests was recorded in milliseconds. The test parameters are as follows.

The semigroup S chosen was a random inverse semigroup on 6 points with 6 generators. The precise semigroup can be specified to the program as follows.

```
gens := [
  PartialPerm( [ 1, 2, 3, 4 ], [ 4, 1, 2, 6 ] ),
  PartialPerm( [ 1, 2, 4 ], [ 4, 6, 3 ] ),
  PartialPerm( [ 1, 2, 4 ], [ 5, 2, 3 ] ),
  PartialPerm( [ 1, 2, 3, 6 ], [ 1, 3, 4, 5 ] ),
  PartialPerm( [ 1, 2, 3, 4, 6 ], [ 2, 4, 6, 1, 5 ] ),
  PartialPerm( [ 1, 2, 3, 6 ], [ 5, 1, 6, 3 ] ) ];
s := InverseSemigroup( gens );
```

The congruence ρ was specified by a set of two randomly-generated pairs, which can be specified as follows.

```
cgens := [
  [ PartialPerm( [ 4 ], [ 1 ] ), PartialPerm( [ 2 ], [ 6 ] ) ],
  [ PartialPerm( [ 5 ], [ 5 ] ), PartialPerm( [ 1,2,4 ], [ 1,6,4 ] ) ] ];
cong := SemigroupCongruenceByGeneratingPairs( s, cgens );
```

Then three random pairs (x, y) were created and passed to the two different functions. The pairs which were created, and the times taken, are recorded in the following table. Each figure is the average of 1000 tests, and each element x or y is given as a list of the two parameters passed to `PartialPerm` to construct it.

x	y	Time (ms)	
		Old	New
`( [1,2,4], [4,3,2] )`	`( [1,2,4], [2,4,3] )`	163	0
`( [4], [1] )`	`( [6], [3] )`	22	0
`( [1,2,4,6], [5,1,2,4] )`	`( [], [] )`	9337	0

Clearly, once we have calculated the congruence pair of an inverse semigroup congruence, it is a great deal faster to look up pairs using the congruence pair representation than to look them up using the existing method. However, to compute the congruence pair to begin with is costly. The congruence pair of the

semigroup S which we have been working with takes an average of 9567 ms to compute (over 20 tests). Clearly, if we only wanted to look up one pair, it would be faster to use the existing implementation and exhaustively enumerate pairs. However, if we want to look up several pairs, we soon find that the congruence pair implementation requires less runtime in total. Furthermore, the possibility exists for a user to specify a congruence originally by a congruence pair without using a set of generating pairs, in which case this overhead is not required.

6.2 Further Work

The work presented in this report is open-ended and could be extended in various ways. In Chapter 4 we attempted to find a way to break down a congruence on an arbitrary semigroup into smaller pieces by somehow describing how that congruence acts on that semigroup's $\mathcal{J}$-classes. No method was found, and several natural courses of enquiry were shown to be fruitless. However, it might still be possible to decompose S into 0-simple semigroups in some other way. There might still be some definition of ρ_J which would have the properties we need.

Chapter 5 could also be extended. We give a characterisation of inverse semigroups and show how we can compute with congruences by storing congruence pairs. One extension might be to find a good way of describing the classes of that congruence (analogous to "class triples" on 0-simple semigroups in [2, p.20-23]). Another good extension would be to find some fast way of enumerating *all* the congruences of an inverse semigroup. Finally, it is clearly useful to be able to calculate the congruence pair (the trace and kernel) of a congruence – an algorithm, analogous to the one described in Section 3.2, which finds the trace and kernel of a congruence directly from its generating pairs, could be much faster than the basic algorithm we implemented in `inverse-cong.gi`, which we derived straight from the definitions.

Bibliography

[1] Howie, J.M., *Fundamentals of Semigroup Theory*, Oxford Science Publications, 1995.

[2] Torpey, M.C., *Computing with Congruences on Finite 0-Simple Semigroups*, MT5991 Report, University of St Andrews, 2014.

[3] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.7.5*; 2014, (`http://www.gap-system.org`).

[4] Mitchell, J.D., *Semigroups - GAP package, Version 2.1*, 2014.

[5] Fiorio, C. & Gustedt, J., *Memory Management for Union-Find Algorithms*, Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science, Lübeck, Germany, 1997, 69 (electronic).

[6] Petrich, M., *Introduction to Semigroups*, Charles E. Merrill Publishing Co., 1973.