ALGEBRAIC STRUCTURES IN PROOF ASSISTANT SYSTEMS

ALGEBRAIC STRUCTURES IN PROOF ASSISTANT SYSTEMS

BY

AKSHOBHYA KATTE MADHUSUDANA, B.Eng.

A REPORT

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTERS OF SCIENCE

Masters of Science (2023) McMaster University

(Department of Computing and Software) Hamilton, Ontario, Canada

TITLE:	Algebraic Structures in Proof Assistant Systems
AUTHOR:	Akshobhya Katte Madhusudana B.Eng. (Computer Science and Engineering), Bangalore University, Bangalore, India
SUPERVISOR:	Dr. Jacques Carette
NUMBER OF PAGES:	xii, 54

Lay Abstract

A lay abstract of not more 150 words must be included explaining the key goals and contributions of the thesis in lay terms that is accessible to the general public.

Abstract

Abstract here (no more than 300 words)

Your Dedication

Optional second line

Acknowledgements

Acknowledgements go here.

Contents

List of Figures

List of Tables

Notation, Definitions, and Abbreviations

Notation

$A \leq B$	A is less than or equal to B

Definitions

Challenge	With respect to video games, a challenge is a set of goals presented to the player that they are tasks with completing; challenges can test a variety of player skills, including accuracy, logical reasoning, and creative problem solving

Abbreviations

AI	Artificial intelligence

Declaration of Academic Achievement

The student will declare his/her research contribution and, as appropriate, those of colleagues or other contributors to the contents of the thesis.

Chapter 1

Introduction

Every thesis needs an introductory chapter

While you're here, you need to go into `definitions.tex` to set all the information needed for the front matter (e.g. title, author) and page header/footer.

You will also find the School of Graduate Studies' preparation guide (August 2021) for theses and reports. I would give it a quick read so you know what's expected.

Chapter 2

Algebraic Structures in Proof Assistant Systems - Survey

The proof assistant systems are computer software that helps to derive formal proofs with a joint effort of computers and humans. Computer proof assistants are used to formalize theories, and extend them by logical reasoning and defining properties. Saqib Nawaz et al. (2019) These systems are used to perform mathematics on computers. The proof assistant systems are widely used in defining and proving rather than doing numerical computations. The automated theorem proving is different from proof assistants in that they have less expressivity and make it almost impossible to define a generic mathematical theory. In Michael and Florian (2021). the author discusses several concerns on considering the proofs that are exclusively verified using a computer to be considered as a valid mathematical proof. However, the proofs written using the proof assistant systems are widely accepted among mathematicians and computer scientists.

The strength of any system is directly dependent on the availability of standard libraries

for those systems. The standard libraries are expected to provide resources to ease the use of such systems. In Jacques Carette and Sharoda (2019), the author discusses the difficulties in building such large libraries. One such problem is to structurally derive algebraic structures from one another in the hierarchy without explicitly defining axioms that become redundant. The author also proposes a solution to make use of the interrelationship in mathematics and thus reduce the efforts in building the library.

We consider the four more commonly used proof assistant systems that are all dependently typed, higher order programming languages that supports (atleast partially) proof by reflection. Proof by reflection is a technique where the system allows to derive proofs by systematic reasoning methods.

Agda 2 is a proof assistant system where proofs are expressed in a functional programming style. The Agda standard library aims to provide tools to ease the effort of writing proofs and also programs. The current version of the Agda standard library (1.7) is fully supported for the changes and developments in Agda. It provides clear documentation for installation, contribution, and style guide for the standard library.

Idris is developed as a functional programming language but is also used as a proof assistant system. The proofs are alike with coq and the type systems in Idris is uniform with agda. Idris 2 is a self-hosted programming language that combines linear-type-system. In this article, Idris 2 and Idris in used interchangeably and refer to Idris 2. Currently, there are no official package managers for Idris 2. However, several versions are under development.

Coq Paulin-Mohring (2012) is a theorem proving system that is written in the Ocaml programming language. It was first released in 1989 and is one of the most widely used proof assistant systems to define mathematical definitions, theory and to write proofs. The mathematical components library (1.12.0) includes various topics from data structures to algebra. In this article, we consider the mathematical component repository (mathcomp) that contains formalized mathematical theories. Mahboubi and Tassi (2021) The latest available release of mathcomp library is 1.12.0. The mathcomp library was started with the Four Colour Theorem to support formal proof of the odd order theorem.

Lean mathlib Community (2020) is an open-source project by Microsoft Research. Lean is a proof assistant system written in C++. The last official version of Lean was 3.4.2 and is now supported by the lean community. Lean 4 is the latest version of Lean and is a complete rewrite of previous versions of Lean. The mathlib mathlib Community (2020) library for lean 3 has the most coverage of algebra compared to the other 3 proof assistant systems discussed in the paper. The mathlib library of Lean is also maintained by the lean community for community versions of lean. It was developed on a small library that was in lean. It contained definitions of natural numbers, integers, and lists and had some coverage over algebra hierarchy. The latest version of mathlib has over 2794 definitions of algebra [3].

The main aim of this paper is to provide documentation for the algebraic coverage in various proof assistant systems. The most commonly used proof assistant systems are Agda, Idris2, Coq, and Lean. In this article, the latest available versions are considered i.e., Agda standard library 1.7, Idris 2.0,The Mathematical Components Library 1.13.0,

and The Lean mathematical library.

2.1 Experimental setup

It is not time efficient to manually look for the definitions in a large library. The source code of the standard libraries of Agda, Idris, Coq and lean are publicly available. We created a web crawler that extracts the code from the source code webpage and built a regular expression that is unique to each system to extract definitions. Thus a part of the process of building the table 1. was automated. Since the standard libraries are open source projects, it is difficult to maintain uniformity in the code. For example the definition might start with comment in the same line or structure parameters might be written in a new line. All this makes it difficult to correctly build the regular expression and will necessitate the task of verifying the results manually to some extent.

The rest of the article is structured as follows. Section 2 discuss about the algebraic structure definitions and its coverage in the proof assistant systems, while the section 3 covers the morphism definitions in those systems. The properties and solvers coverage in presented in section 4. The last section of the paper is the conclusion and discussion.

2.2 Algebraci Structures

The Agda standard library provides definitions with bundled versions of several algebraic structures. Algebra hierarchy is followed in defining structures Michael and Florian (2021). As an example, a semigroup is derived from magma and a monoid from semigroup.

```
record IsMagma (· : Op₂ A) : Set (a ⊔ ℓ) where
  field
    isEquivalence : IsEquivalence _≈_
    ·-cong        : Congruent₂ ·

  open IsEquivalence isEquivalence public

record IsSemigroup (· : Op₂ A) : Set (a ⊔ ℓ) where
  field
    isMagma : IsMagma ·
    assoc   : Associative ·

  open IsMagma isMagma public
```

The same follows for the bundle definitions of respective structures. Since the current version of the library has a limited number of structures, there might arise a problem of extending the hierarchy as described in [2]. One exemption for this hierarchical definition is the definition of a lattice. A lattice is defined independently in the standard library to overcome the redundant idempotent fields. A lattice structure that is defined in terms of join and meet semilattice is being added as a biased structure. The 1.7 version of the agda standard library has the definitions of structures with the respective bundle versions of Magma, Commutative Magma, to Ring, CommutativeRing, and Boolean Algebra. Another definition of most of the above algebraic structures is provided as a direct product of two other instances of algebraic structures. However, from semigroups, the structures are defined in terms of relevant categories. The structures also include

respective bundle definitions. A module is an abelian group with the ring of scalars. The ring of scalars has an identity element. The agda standard library defines left, right, and bi semimodules and modules. A similar hierarchical approach as other algebraic structures is followed in defining modules. As an example, a module is defined using bimodules and bimodules using bi-semimodules. An alternative definition of modules is given in "Algebra.Module.Structure.Biased". The structures have respective bundled versions.

In Idris 2, there is a considerable overlap of abstract algebra and category theory. The library defines various algebraic structures that include semigroup, monoid, group, abelian-group, semiring, and ring. It follows a hierarchical approach in defining structures similar to that in agda. For example, a semigroup is defined as a set with a binary operation that is associative and a monoid is defined in terms of semigroup with an identity element. Idris addresses identity as a neutral element.

```
interface Semigroup t where
  (<+>) : t -> t -> t
  semigroupOpIsAssociative : (l, c, r : t) -> l <+> (c <+> r) = (l <+> c) <+> r

interface Semigroup t => Monoid t where
  neutral : t
  monoidNeutralIsNeutralL : (l : t) -> l <+> neutral = l
  monoidNeutralIsNeutralR : (r : t) -> neutral <+> r = r
```

The algebra structures design hierarchy of the mathcomp library is inspired by the Packing mathematical structures. The ssralg file defines most of the basic algebraic

structures with their type, packers, and canonical properties. The hierarchy extends from Zmodule, rings to ring morphisms. The countalg file extends ssralg file to define countable types.

The mathlib extends the algebra hierarchy from semigroup to ordered fields. The library defines instances of free magma, free semigroup, free Abelian group, etc. An example of semigroup structure definition in the library is given below:

```
structure semigroup (G : Type u) :
  Type u
  mul : G → G → G
  mul_assoc : forall (a b c : G), (a * b) * c = a * b * c
```

Other instances of semigroups are derived from the definition of semigroup structure such as commutative semigroup, left and right cancellative semgroup. Similar definitions are extended from monoid and other structures.

Table 2.1: Algebraic structures in proof assistant systems

Algebraic Structure	**Agda**	**Coq**	**Idris**	**Lean**
Magma	✓	-	-	-
Commutative Magma	✓	-	-	-
Selective Magma	✓	-	-	-
Continued on next page				

Table 2.1 – continued from previous page

Algebraic Structure	Agda	Coq	Idris	Lean
IdempotentMagma	✓	-	-	-
AlternativeMagma	✓	-	-	-
FlexibleMagma	✓	-	-	-
MedialMagma	✓	-	-	-
SemiMedialMagma	✓	-	-	-
Semigroup	✓	✓	✓	✓
Band	✓	-	-	-
Commutative Semigroup	✓	-	-	✓
Semilattice	✓	-	-	✓
Unital magma	✓	-	-	-
Monoid	✓	✓	✓	✓
Commutative monoid	✓	✓	-	✓
Idempotent commutative monoid	✓	-	-	-
Bounded Semilattice	✓	-	-	-
Bounded Meetsemilattice	✓	-	-	-
Bounded Joinsemilattice	✓	-	-	-
Invertible Magma	✓	-	-	-
IsInvertible UnitalMagma	✓	-	-	-
Quasigroup	✓	-	-	-
Loop	✓	-	-	-

Continued on next page

Table 2.1 – continued from previous page

Algebraic Structure	Agda	Coq	Idris	Lean
Moufang Loop	✓	-	-	-
Left Bol Loop	✓	-	-	-
Middle Bol Loop	✓	-	-	-
Right Bol Loop	✓	-	-	-
NilpotentGroup	-	-	-	✓
CyclicGroup	-	-	-	✓
SubGroup	-	-	-	✓
Group	✓	✓	✓	✓
Abelian group	✓	-	✓	✓
Lattice	✓	-	-	✓
Distributive lattice	✓	-	-	-
Near semiring	✓	-	-	-
Semiringwithout one	✓	-	-	-
Idempotent Semiring	✓	-	-	-
Commutative semiring without one	✓	-	-	-
Semiring without annihilating zero	✓	-	-	-
Semiring	✓	✓	-	✓
Commutative semiring	✓	-	-	✓
Non associative ring	✓	-	-	-
Nearring	✓	-	-	-

Continued on next page

Table 2.1 – continued from previous page

Algebraic Structure	Agda	Coq	Idris	Lean
Quasiring	✓	-	-	-
Local ring	-	-	-	✓
Noetherian ring	-	-	-	✓
Ordered ring	-	-	-	✓
Cancellative commutative semiring	✓	-	-	-
Sub ring	-	-	-	✓
Ring	✓	✓	✓	✓
Unit Ring	✓	✓	✓	-
Commutative Unit ring	-	✓	-	-
Commutative ring	✓	✓	-	✓
Integral Domain	-	✓	-	-
LieAlgebra	-	-	-	✓
LieRing module	-	-	-	✓
Lie module	-	-	-	✓
Boolean algebra	✓	-	-	-
Preleft semimodule	✓	-	-	-
Left semimodule	✓	-	-	-
Preright semimodule	✓	-	-	-
right semimodule	✓	-	-	-
Bi semimodule	✓	-	-	-

Continued on next page

Table 2.1 – continued from previous page

Algebraic Structure	Agda	Coq	Idris	Lean
Semimodule	✓	-	-	-
Left module	✓	✓	-	-
Right module	✓	-	-	-
Bi module	✓	-	-	-
Module	✓	✓	-	✓
Field	-	✓	✓	✓
Decidable Field	-	✓	-	-
Closed field	-	✓	-	-
Algebra	-	✓	-	-
Unit algebra	-	✓	-	✓
Lalgebra	-	✓	-	-
Commutative unit algebra	-	✓	-	-
Commutative algebra	-	✓	-	-
NumDomain	-	✓	-	-
Normed Zmodule	-	✓	-	-
Num field	-	✓	-	-
Real domain	-	✓	-	-
Real field	-	✓	-	-
Real closed field	-	✓	-	-
Vector space	-	✓	-	-

Continued on next page

Table 2.1 – continued from previous page

Algebraic Structure	Agda	Coq	Idris	Lean
Zmodule Quotients type	-	✓	-	-
Ring Quotient type	-	✓	-	-
Unit rint quotient type	-	✓	-	-
Additive group	-	✓	-	-
characteristic zero	-	-	-	✓
Domain	-	-	-	✓
Chain Complex	-	-	-	✓
Kleene Algebra	✓	-	-	-
IsHeytingCommutativeRing	✓	-	-	-
IsHeytingField	✓	-	-	-

2.3 Morphism

One of the benefits of the Agda standard library is that it provides morphisms for the structures defined in the library. A raw bundle instance is defined in Algebra. Bundles and the morphisms for those raw structures are provided. For example, raw magma is used in magma morphisms. The library defines homomorphism, monomorphism, and isomorphism for those structures. The library also provides the composition of morphisms between algebraic structures. The morphism definitions for Magma, Monoid, Group, NearSemiring, Semiring, Ring, Lattice are available in the standard library. An example of magma morphisms as defined in the standard library is as follows.

```
record IsMagmaHomomorphism (⟦_⟧ : A → B) : Set (a ⊔ ℓ₁ ⊔ ℓ₂) where
  field
    isRelHomomorphism : IsRelHomomorphism _≈₁_ _≈₂_ ⟦_⟧
    homo              : Homomorphic₂ ⟦_⟧  _·_ _∘_

  open IsRelHomomorphism isRelHomomorphism public
    renaming (cong to ⟦⟧-cong)
```

Similar definitions for monomorphism and isomorphism are included in agda standard library.

The morphism definitions in the Idris library define morphisms in category theory. A group homomorphism is a structure-preserving function between two groups and is defined as follows :

```
interface (Group a, Group b) => GroupHomomorphism a b where
  to : a -> b

  toGroup : (x, y : a) ->  to (x <+> y) = (to x) <+> (to y)
```

The group theory directory defines groups, group morphisms, subgroups, cyclic, nilpotent groups, and isomorphism theorems. There is no group homomorphism instead, it is defined with proofs for map-one and map-mul for monoid homomorphism. The definition of monoid homomorphism is give below:

```
structure monoid_hom (M : Type*) (N : Type*) [mul_one_class M] [mul_one_class N]
  extends one_hom M N, mul_hom M N
```

The mathlib library extends monoid and groups to define rings and ring morphisms. Bundled structure is used to define ring morphisms.

2.4 Properties

The Agda standard library provides constructs of modules such as a bi-product construct and tensor unit using two R-modules. The library also includes the relation between function properties with basal setoid and sets for propositional equalities. The library includes ring, monoid solvers for equations of the same. However, these solvers are under construction and not optimized for performance.

The coq library has rings and field tactics to achieve algebraic manipulations in some of the algebraic structures. The library also includes specialized tactics such as interval and gappa to work with real numbers and floating point nubmers. Paulin-Mohring (2012)

The Idris library defines properties or laws of algebraic structures. The unique-Inverse defines that the inverses of monoids are unique. Other laws on groups include self-squaring i.e., identity element of a group is self-squaring, inverse elements of a group satisfy the commutative property, laws of double negation. It also defines squareId-Commutative i.e., a group is abelian if every square in a group is neutral, inverseNeutralIsNeutral, and other properties of an algebraic group. The Latin-square-property is defined as for any two elements a and b, ax =b and ya = b exists. Other algebraic properties for groups are y = z if x + y = x + z, y = z if y + x = z + x, ab = 0 -> a = b-1, and ab = 0 -> a-1 = b. An example of a definition is shown below.

```
public export
neutralProductInverseL : Group ty => (a, b : ty) ->
  a <+> b = neutral {ty} -> inverse a = b
neutralProductInverseL a b prf =
  cancelLeft a (inverse a) b $
    trans (groupInverseIsInverseL a) $ sym prf
```

The library also includes laws on homomorphism that homomorphism over group preserves identity and inverses. Some laws on ring structures are also included in the library such as x0 = 0, (-x)y = -(xy), x(-y) = -(xy), (-x)(-y) = xy, (-1)x = -x, and x(-1) = -x. The algebraic coverage of Idris 2 is limited and is under development. There are no official definitions for solvers or higher structures such as modules, fields, or vector space. The Idris 2 is comparatively new and is under continuous development to strengthen the language and also as a mechanical reasoning system.

The mathlib library of Lean 3 includes algebra over rings such as associative algebra over a commutative ring, Lie algebra, Clifford algebra, etc. Lie algebra is defined as a module satisfying Jacobi identity. Without scalar multiplication, a lie algebra is a lie ring. The library extends rings to define fields and division ring covering many aspects of fields such as the existence of closure for a field, Galois correspondence, rupture field, and others.

Chapter 3

Theory Of Quasigroup and Loop in Agda

Applications of non associative algebras are explored in various fields of study. For example, Einstein's formula of addition of velocities gives a loop structure, Quasigroups of various orders are used in field of cryptography, Lie algebra is used in differential geometry. Proof assistant systems such as Agda are helpful in verifying some of the properties of these structures. They are interactive software that help to derive complex mathematical proofs. In this chapter we formalize two important non associative algebras - quasigroup, loop structure. A Quasigroup (Q, ·, /, \) is a type (2,2,2) algebra for which the binary operations \and / are defined such that division is always possible. A loop is a quasigroup with identity. We explore morphisms and direct product for these structures and derive proofs for some of the properties of these structures.

3.1 Definition

A set that has a binary operation is called Magma. In this case a Magma is total and should not be confused with groupoid that need not be total. Let Q be a non empty set. Left and division is defined with identities.

$$y = x \cdot (x \backslash y) \tag{3.1.1}$$

$$y = x \backslash (x \cdot y) \tag{3.1.2}$$

$$y = (y / x) \cdot x \tag{3.1.3}$$

$$y = (y \cdot x) / x \tag{3.1.4}$$

The Agda definition is given below

```
LeftDivides^l  : Op₂ A → Op₂ A → Set _
LeftDivides^l  _·_  _\\_ = ∀ x y → (x · (x \\ y)) ≈ y


LeftDivides^r : Op₂ A → Op₂ A → Set _
LeftDivides^r _·_ _\\_ = ∀ x y → (x \\ (x · y)) ≈ y


RightDivides^l  : Op₂ A → Op₂ A → Set _
RightDivides^l  _·_ _//_ = ∀ x y → ((y // x) · x) ≈ y


RightDivides^r : Op₂ A → Op₂ A → Set _
RightDivides^r _·_ _//_ = ∀ x y → ((y · x) // x) ≈ y
```

We can combine the left and right division as follows

```
LeftDivides : Op₂ A → Op₂ A → Set _
LeftDivides · \\ = (LeftDivides^l · \\) × (LeftDivides^r · \\)

RightDivides : Op₂ A → Op₂ A → Set _
RightDivides · // = (RightDivides^l · //) × (RightDivides^r · //)
```

Note that we use // and \\instead of / and \respectively to overcome the conflict with overloaded or escape characters.

The Quasigroup structure can be structurally derived from Magma in Agda as

```
record IsQuasigroup (· \\ // : Op₂ A) : Set (a ⊔ ℓ) where
  field
    isMagma       : IsMagma ·
    \\-cong       : Congruent₂ \\
    //-cong       : Congruent₂ //
    leftDivides   : LeftDivides · \\
    rightDivides  : RightDivides · //

  open IsMagma isMagma public
```

A loop is a quasigroup that has identity element.

$$x \cdot e = e \cdot x = x \tag{3.1.5}$$

```
LeftIdentity : A → Op₂ A → Set _
LeftIdentity e _·_ = ∀ x → (e · x) ≈ x

RightIdentity : A → Op₂ A → Set _
RightIdentity e _·_ = ∀ x → (x · e) ≈ x

Identity : A → Op₂ A → Set _
Identity e · = (LeftIdentity e ·) × (RightIdentity e ·)
```

Loop structure can be structurally derived from quasigroup.

```
record IsLoop (· \\ // : Op₂ A) (ϵ : A) : Set (a ⊔ ℓ) where
  field
    isQuasigroup : IsQuasigroup · \\ //
    identity     : Identity ϵ ·

  open IsQuasigroup isQuasigroup public
```

A loop is called a right bol loop if it satisfies the identity (Equation 3.1.6)

$$((z \cdot x) \cdot y) \cdot x = z \cdot ((x \cdot y) \cdot x) \tag{3.1.6}$$

A loop is called a left bol loop if it satisfies the identity (Equation 3.1.7)

$$x \cdot (y \cdot (x \cdot z)) = (x \cdot (y \cdot x)) \cdot z \tag{3.1.7}$$

A loop is called middle bol loop if it satisfies the identity (Equation 3.1.8)

$$(z \cdot x) \cdot (y \cdot z) = z \cdot ((x \cdot y) \cdot z) \tag{3.1.8}$$

A left-right bol loop is called a moufang loop if it satisfies identity (Equation 3.1.9)

$$(z \cdot x) \cdot (y \cdot z) = z \cdot ((x \cdot y) \cdot z) \tag{3.1.9}$$

```
LeftBol : Op₂ A → Set _
LeftBol _·_ = ∀ x y z → (x · (y · (x · z))) ≈ ((x · (y · x)) · z )

RightBol : Op₂ A → Set _
RightBol _·_ = ∀ x y z → (((z · x) · y) · x) ≈ (z · ((x · y) · x))

MiddleBol : Op₂ A → Op₂ A  → Op₂ A  → Set _
MiddleBol _·_ _\\_ _//_ = ∀ x y z → (x · ((y · z) \\ x)) ≈ ((x // z) · (y \\ x))

Identical : Op₂  A → Set _
Identical _·_ = ∀ x y z → ((z · x) · (y · z)) ≈ (z · ((x · y) · z))
```

3.2 Morphism

A structure preserving map f between two structures of same type is called morphism or homomorphism. That is f : A → B and · is an operation on the structure then homomorphism is defined as f(x · y) = f(x) · f(y). A homomorphism that is injective is called monomorphism. If the structures are identical that is they are more than just similar

in structure then we can compare the structures with isomorphism. A homomorphism that is bijective is called isomorphism. The quasigroup homomorphism preserves both left and right division operations.

```
record IsQuasigroupHomomorphism (⟦_⟧ : A → B) : Set (a ⊔ ℓ₁ ⊔ ℓ₂) where
  field
    isRelHomomorphism : IsRelHomomorphism _≈₁_ _≈₂_ ⟦_⟧
    ·-homo            : Homomorphic₂ ⟦_⟧ _·₁_ _·₂_
    \\-homo           : Homomorphic₂ ⟦_⟧ _\\₁_ _\\₂_
    //-homo           : Homomorphic₂ ⟦_⟧ _//₁_ _//₂_

  open IsRelHomomorphism isRelHomomorphism public
    renaming (cong to ⟦⟧-cong)

record IsQuasigroupMonomorphism (⟦_⟧ : A → B) : Set (a ⊔ ℓ₁ ⊔ ℓ₂) where
  field
    isQuasigroupHomomorphism : IsQuasigroupHomomorphism ⟦_⟧
    injective                : Injective ⟦_⟧

  open IsQuasigroupHomomorphism isQuasigroupHomomorphism public

record IsQuasigroupIsomorphism (⟦_⟧ : A → B) : Set (a ⊔ b ⊔ ℓ₁ ⊔ ℓ₂) where
   field
     isQuasigroupMonomorphism : IsQuasigroupMonomorphism ⟦_⟧
     surjective               : Surjective ⟦_⟧

   open IsQuasigroupMonomorphism isQuasigroupMonomorphism public
```

The loop morphism preserves left and right divisions along with the identity element

```
record IsLoopHomomorphism (⟦_⟧ : A → B) : Set (a ⊔ ℓ₁ ⊔ ℓ₂) where
  field
    isQuasigroupHomomorphism : IsQuasigroupHomomorphism ⟦_⟧
    ε-homo                   : Homomorphic₀ ⟦_⟧ ε₁ ε₂

  open IsQuasigroupHomomorphism isQuasigroupHomomorphism public

record IsLoopMonomorphism (⟦_⟧ : A → B) : Set (a ⊔ ℓ₁ ⊔ ℓ₂) where
  field
    isLoopHomomorphism   : IsLoopHomomorphism ⟦_⟧
    injective            : Injective ⟦_⟧

  open IsLoopHomomorphism isLoopHomomorphism public

record IsLoopIsomorphism (⟦_⟧ : A → B) : Set (a ⊔ b ⊔ ℓ₁ ⊔ ℓ₂) where
  field
    isLoopMonomorphism   : IsLoopMonomorphism ⟦_⟧
    surjective           : Surjective ⟦_⟧

    open IsLoopMonomorphism isLoopMonomorphism public
```

3.3 Morphism composition

If f is a morphism such that f : a → b and g is a morphism on same structure such that g : b → c then composition of morphism can be defined as g ∘ f : a → c.

```
isQuasigroupHomomorphism : IsQuasigroupHomomorphism Q₁ Q₂ f
  → IsQuasigroupHomomorphism Q₂ Q₃ g
  → IsQuasigroupHomomorphism Q₁ Q₃ (g ∘ f)
isQuasigroupHomomorphism f-homo g-homo = record
  { isRelHomomorphism = isRelHomomorphism
              F.isRelHomomorphism
              G.isRelHomomorphism
  ; ·-homo    = λ x y → ≈₃-trans
              (G.⟦⟧-cong ( F.·-homo x y ))
              ( G.·-homo (f x) (f y) )
  ; \\-homo     = λ x y → ≈₃-trans
              (G.⟦⟧-cong ( F.\\-homo x y ))
              ( G.\\-homo (f x) (f y) )
  ; //-homo     = λ x y → ≈₃-trans
              (G.⟦⟧-cong ( F.//-homo x y ))
              ( G.//-homo (f x) (f y) )
  } where module F = IsQuasigroupHomomorphism f-homo;
          module G = IsQuasigroupHomomorphism g-homo
```

```
isLoopHomomorphism : IsLoopHomomorphism L₁ L₂ f
  → IsLoopHomomorphism L₂ L₃ g → IsLoopHomomorphism L₁ L₃ (g ∘ f)
isLoopHomomorphism f-homo g-homo = record
  { isQuasigroupHomomorphism = isQuasigroupHomomorphism ≈₃-trans
                  F.isQuasigroupHomomorphism G.isQuasigroupHomomorphism
  ; ϵ-homo              = ≈₃-trans (G.⟦⟧-cong F.ϵ-homo) G.ϵ-homo
  } where module F = IsLoopHomomorphism f-homo;
          module G = IsLoopHomomorphism g-homo
```

Monomorphism and isomorphism compositions constructs for quasigroup and loop are defined similar to homomorphism and can be found in agda standard library.

3.4 DirectProduct

The direct product M × N of two quasigroups M and N is defined as a pair (m,n) where m ∈ M and n ∈ N. The direct product construct of left (right/middle) bol loop and moufang loop can be found in agda standard library and can be derived from loop structure.

```
quasigroup : Quasigroup a ℓ₁ → Quasigroup b ℓ₂
                → Quasigroup (a ⊔ b) (ℓ₁ ⊔ ℓ₂)
quasigroup M N = record
  { _\\_  = zip M._\\_ N._\\_
  ; _//_    = zip M._//_ N._//_
  ; isQuasigroup = record
    { isMagma = Magma.isMagma (magma M.magma N.magma)
    ; \\-cong = zip M.\\-cong N.\\-cong
    ; //-cong = zip M.//-cong N.//-cong
    ; leftDivides = (λ x y → M.leftDivides¹ ,
                 N.leftDivides¹ <*> x <*> y),
                  (λ x y → M.leftDivides² ,
                N.leftDivides² <*> x <*> y)
    ; rightDivides = (λ x y → M.rightDivides¹ ,
                 N.rightDivides¹ <*> x <*> y),
                    (λ x y → M.rightDivides² ,
                 N.rightDivides² <*> x <*> y)
    }
  } where module M = Quasigroup M; module N = Quasigroup N
```

```
loop : Loop a ℓ₁ → Loop b ℓ₂ → Loop (a ⊔ b) (ℓ₁ ⊔ ℓ₂)
loop M N = record
  { ε = M.ε , N.ε
  ; isLoop = record
    { isQuasigroup = Quasigroup.isQuasigroup
                 (quasigroup M.quasigroup N.quasigroup)
    ; identity = (M.identityˡ , N.identityˡ <*>_),
                     (M.identityʳ , N.identityʳ <*>_)
    }
  } where module M = Loop M; module N = Loop N
```

3.5 Properties

In this section we prove some of the properties of quasigroups and loops in Agda.

3.5.1 Properties of Quasigroup

Let (Q, ·, /, \) be a quasigroup then

a) Q is cancellative A quasigroup is left cancellative if x · y = x · z then y = z and a quasigroup is right cancellative if y · x = z · x then y = z. A quasigroup is cancellative if it is both left and right cancellative.

b) $\forall x, y, z \in Q$: x · y = z then y = x \z

c) $\forall x, y, z \in Q$: x · y = z then x = z / y

Proof:

```
a)
cancel^l : LeftCancellative _·_
cancel^l x {y} {z} eq = begin
  y             ≈⟨ sym( leftDivides^r  x y) ⟩
  x \\ (x · y)  ≈⟨ \\-cong^l eq ⟩
  x \\ (x · z)  ≈⟨ leftDivides^r  x z ⟩
  z             ■

cancel^r : RightCancellative _·_
cancel^r {x} y z eq = begin
  y             ≈⟨ sym( rightDivides^r  x y) ⟩
  (y · x) // x  ≈⟨ //-cong^r  eq ⟩
  (z · x) // x  ≈⟨ rightDivides^r  x z ⟩
  z             ■

cancel : Cancellative _·_
cancel = cancel^l , cancel^r

b)
y≈x\z : ∀ x y z → x · y ≈ z → y ≈ x \\  z
y≈x\z x y z eq = begin
  y             ≈⟨ sym (leftDivides^r x y) ⟩
  x \\  (x · y) ≈⟨ \\ -cong^l eq ⟩
  x \\  z       ■
```

```
c)
x≈z/y : ∀ x y z → x · y ≈ z → x ≈ z // y
x≈z/y x y z eq = begin
  x            ≈⟨ sym (rightDivides^r y x) ⟩
  (x · y) // y ≈⟨ //-cong^r eq ⟩
  z // y       ■
```

3.5.2 Properties of Loop

Properties of division operation holds for a loop.

Let (L, ·, /, \) be a Loop with identity x · e = x then the following properties holds

a)$\forall x \in L$: x / x = e

b)$\forall x \in L$: x \x = e

c)$\forall x \in L$: e \x = x

d)$\forall x \in L$: x / e = x

Proof:

```
a)
x//x≈ϵ : ∀ x → x // x ≈ ϵ
x//x≈ϵ x = begin
  x // x       ≈⟨ //-cong^r (sym (identity^l x)) ⟩
  (ϵ · x) // x ≈⟨ rightDivides^r x ϵ ⟩
  ϵ            ■
```

```
b)
x\\x≈ϵ : ∀ x → x \\ x ≈ ϵ
x\\x≈ϵ x = begin
  x \\ x      ≈⟨ \\-cong^l (sym (identity^r x )) ⟩
  x \\ (x · ϵ) ≈⟨ leftDivides^r x ϵ ⟩
  ϵ            ■

c)
ϵ\\x≈x : ∀ x → ϵ \\ x ≈ x
ϵ\\x≈x x = begin
  ϵ \\ x       ≈⟨ sym (identity^l (ϵ \\ x)) ⟩
  ϵ · (ϵ \\ x) ≈⟨ leftDivides^l ϵ x ⟩
  x            ■

d)
x//ϵ≈x : ∀ x → x // ϵ ≈ x
x//ϵ≈x x = begin
 x // ϵ       ≈⟨ sym (identity^r (x // ϵ)) ⟩
 (x // ϵ) · ϵ  ≈⟨ rightDivides^l ϵ x ⟩
 x             ■
```

Let (M, ·, /, \) be a middle bol loop then the following identities holds.

a)$\forall x y z \in M$: x · ((y · x) \x) = y \x

b)$\forall x y z \in M$: x · ((x · z) \x) = x // z

c)$\forall x y z \in M$: x · (z \x) ≈ (x / z) · x

d)$\forall xyz \in M$: (x / (y · z)) · x ≈ (x / z) · (y \x)

e)$\forall xyz \in M$: (x / (y · x)) · x ≈ y \x

f)$\forall xyz \in M$: (x / (x · z)) · x ≈ x / z

Proof:

a)

```
xyx\x≈y\\x : ∀ x y → x · ((y · x) \\ x) ≈ y \\ x
xyx\x≈y\\x x y = begin
  x · ((y · x) \\ x)   ≈⟨ middleBol x y x ⟩
  (x // x) · (y \\ x) ≈⟨ ·-cong^r (x//x≈ε x) ⟩
  ε · (y \\ x)        ≈⟨ identity^l ((y \\ x)) ⟩
  y \\ x              ■
```

b)

```
xxz\\x≈x//z : ∀ x z → x · ((x · z) \\ x) ≈ x // z
xxz\\x≈x//z x z = begin
  x · ((x · z) \\ x)  ≈⟨ middleBol x x z ⟩
  (x // z) · (x \\ x) ≈⟨ ·-cong^l (x\\x≈ε x) ⟩
  (x // z) · ε        ≈⟨ identity^r ((x // z)) ⟩
  x // z              ■
```

```
c)
xz\\x≈x//zx : ∀ x z → x · (z \\ x) ≈ (x // z) · x
xz\\x≈x//zx x z = begin
  x · (z \\ x)      ≈⟨ ·-cong^l (\\-cong^r( sym (identity^l z))) ⟩
  x · ((ϵ · z) \\ x) ≈⟨ middleBol x ϵ z ⟩
  x // z · (ϵ \\ x) ≈⟨ ·-cong^l (ϵ\\x≈x x) ⟩
  x // z · x        ■

d)
x//yzx≈x//zy\\x : ∀ x y z → (x // (y · z)) · x ≈ (x // z) · (y \\ x)
x//yzx≈x//zy\\x x y z = begin
 (x // (y · z)) · x  ≈⟨ sym (xz\\x≈x//zx x ((y · z))) ⟩
 x · ((y · z) \\ x)  ≈⟨ middleBol x y z ⟩
 (x // z) · (y \\ x) ■

e)
x//yxx≈y\\x : ∀ x y → (x // (y · x)) · x ≈ y \\ x
x//yxx≈y\\x x y = begin
  (x // (y · x)) · x  ≈⟨ x//yzx≈x//zy\\x  x y x ⟩
  (x // x) · (y \\ x) ≈⟨ ·-cong^r(x//x≈ϵ x) ⟩
  ϵ · (y \\ x)        ≈⟨ identity^l ((y \\ x)) ⟩
  y \\ x              ■
```

```
f)
x//xzx≈x//z : ∀ x z → (x // (x · z)) · x ≈ x // z
x//xzx≈x//z x z = begin
  (x // (x · z)) · x  ≈⟨ x//yzx≈x//zy\\x x x z ⟩
  (x // z) · (x \\ x) ≈⟨ ·-cong^l (x\\x≈ϵ x) ⟩
  (x // z) · ϵ        ≈⟨ identity^r (x // z) ⟩
  x // z              ■
```

3.5.3 Properties of Moufang Loop

Let (M, ·, /, \) be a moufang loop then the following identities holds.

a) Moufang loop is alternative.

A moufang loop is left alternative if it satisfies $(x \cdot x) \cdot y = x \cdot (x \cdot y)$, a moufang loop is right alternative if it satisfies $x \cdot (y \cdot y) = (x \cdot y) \cdot y$ and if a moufang loop alternative if it is both left and right alternative.

b) Moufang loop is flexible

A Moufant loop is flexible if it satisfies flexible identity $(x \cdot y) \cdot x = x \cdot (y \cdot x)$

c) $\forall x y z \in M$: $z \cdot (x \cdot (z \cdot y)) = ((z \cdot x) \cdot z) \cdot y$

d) $\forall x y z \in M$: $x \cdot (z \cdot (y \cdot z)) = ((x \cdot z) \cdot y) \cdot z$

e) $\forall x y z \in M$: $z \cdot ((x \cdot y) \cdot z) = (z \cdot (x \cdot y)) \cdot z$

Proof:

```
a)
alternative¹ : LeftAlternative _·_
alternative¹ x y = begin
  (x · x) · y       ≈⟨ ·-cong^r (·-cong^l (sym (identity¹ x))) ⟩
  (x · (ϵ · x)) · y ≈⟨ sym (leftBol x ϵ y) ⟩
  x · (ϵ · (x · y)) ≈⟨ ·-cong^l (identity¹ ((x · y))) ⟩
  x · (x · y)       ■

alternative^r : RightAlternative _·_
alternative^r x y = begin
  x · (y · y)         ≈⟨ ·-cong^l (·-cong^r(sym (identity^r y))) ⟩
  x · ((y · ϵ) · y)   ≈⟨ sym (rightBol y ϵ x) ⟩
  ((x · y) · ϵ ) · y  ≈⟨ ·-cong^r (identity^r ((x · y))) ⟩
  (x · y) · y         ■

alternative : Alternative _·_
alternative = alternative¹ , alternative^r

b)
flex : Flexible _·_
flex x y = begin
  (x · y) · x       ≈⟨ ·-cong^l (sym (identity¹ x)) ⟩
  (x · y) · (ϵ · x) ≈⟨ identical y ϵ x ⟩
  x · ((y · ϵ) · x) ≈⟨ ·-cong^l (·-cong^r (identity^r y)) ⟩
  x · (y · x)       ■
```

```
c)
z·xzy≈zxz·y : ∀ x y z → (z · (x · (z · y))) ≈ (((z · x) · z) · y)
z·xzy≈zxz·y x y z = sym (begin
  ((z · x) · z) · y ≈⟨ (·-cong^r  (flex z x )) ⟩
  (z · (x · z)) · y ≈⟨ sym (leftBol z x y) ⟩
  z · (x · (z · y)) ■)

d)
x·zyz≈xzy·z : ∀ x y z → (x · (z · (y · z))) ≈ (((x · z) · y) · z)
x·zyz≈xzy·z x y z = begin
  x · (z · (y · z))  ≈⟨ (·-cong^l  (sym (flex z y ))) ⟩
  x · ((z · y) ·  z) ≈⟨ sym (rightBol z y x) ⟩
  ((x · z) · y) · z  ■

e)
z·xyz≈zxy·z : ∀ x y z → (z · ((x · y) · z)) ≈ ((z · (x · y)) · z)
z·xyz≈zxy·z x y z = sym (flex z (x · y))
```

Chapter 4

Theory of Semigroup and Ring in Agda

Theory of Semigroup and Ring in Agda

Chapter 5

Theory of Kleene Algebra in Agda

Theory of Kleene Algebra in Agda

Chapter 6

Problem in Programming Algebra

Algebraic structures show variations in syntax and semantics depending on the system or language in which they are defined. Each systems discussed in chapter 1 have their own style of defining structures in the standard libraries. For example, in Coq Ring is defined without multiplicative identity. However, in Agda, Ring has multiplicative identity and Rng is defined as RingWithoutOne that has no multiplicative identity. This ambiguity in naming is also seen in literature. Another example is same structure having multiple definitions like Quasigroups. Quasigroups can be defined as type(2) algebra with latin square property or as type(2,2,2) with left and right division operators. Both the definitions are equivalent but they are structurally different. This chapter identifies and classifies five important problems that arises when defining algebraic structures in proof assistant systems.

6.1 Ambiguity in naming

Ambiguity arises when something can be interpreted in more than one way. The example of quasigroup having more than one definition can give rise to a scenario of making an incorrect interpretation of the algebraic structure when it is not clearly stated. In abstract algebra and algebraic structure these scenarios can be more common. This can be attributed to lack of naming convention that is followed in naming algebraic structures and it's properties. For example Ring and Rng. Some mathematicians define Ring as an algebraic structure that is an abelian group under addition and a monoid under multiplication. This definition is also be named explicitly as ring with unit or ring with identity. Rng is defined as an algebraic structure that is an abelian group under addition and a semigroup under multiplication. The same structure is also defined as ring without identity. However, this definitions are often interchanged i.e., some mathematicians define ring as ring without identity that is multiplication has no identity or is a semigroup. This ambiguity is some time attributed to the language of origin of the algebraic structures. In this case rng is used in French where as ring in english. These confusions can be seen in literature and in online blogs where it is difficult to imply the definition of intent when they are not explicitly defined.

In Agda, ring is defined as an algebraic structure with two binary operations + and * where + is an abelian group and * is a monoid. The binary operation * distributes over + that is multiplication distributes over addition and it has a zero.

```
record IsRing (+ * : Op₂ A) (-_ : Op₁ A) (0# 1# : A) : Set (a ⊔ ℓ) where
  field
    +-isAbelianGroup : IsAbelianGroup + 0# -_
    *-cong           : Congruent₂ *
    *-assoc          : Associative *
    *-identity       : Identity 1# *
    distrib          : * DistributesOver +
    zero             : Zero 0# *

  open IsAbelianGroup +-isAbelianGroup public
```

Rng is defined as ring wihthout one where one is assumend to be multiplication identity.

```
record IsRingWithoutOne (+ * : Op₂ A) (-_ : Op₁ A) (0# : A) : Set (a ⊔ ℓ) where
  field
    +-isAbelianGroup : IsAbelianGroup + 0# -_
    *-cong           : Congruent₂ *
    *-assoc          : Associative *
    distrib          : * DistributesOver +
    zero             : Zero 0# *
```

Another example of ambiguity is Nearring. In some papers, Nearring is defined as a structure where addition is a group and multiplication is a monoid. But some mathematicians use the definition where multiplication is a semigroup. The same confusion also arises in defining semiring and rig structures. Wikipedia states that the term rig originated as a joke that it is similar to rng that is missing alphabet n and i to represent

the identity does not exist for these structures. In Agda rig is defined as semiring without one where one is represents the multiplicative identity.

For axioms of structures, the names are usually invented when defining the structure. As an example when defining Kleene Algebra in Agda, starExpansive and starDestructive names were invented (inspired from what is used in literature). Due to lack of common practice many names can be coined for the same axiom.

```
record IsKleeneAlgebra (+ * : Op₂ A) ( ⁻* : Op₁ A)
                                  (0# 1# : A) : Set (a ⊔ ℓ) where
  field
    isIdempotentSemiring    : IsIdempotentSemiring + * 0# 1#
    starExpansion              : StarLeftExpansion 1# + * ⁻*
    starDestructive            : StarRightExpansion+ * ⁻*
  open IsIdempotentSemiring isIdempotentSemiring public
```

6.2 Equivalent but structurally different

Quasigroup structure is an example that can be defined in two ways. A type (2) Quasigroup can be defined as a set Q and binary operation · can be defined as that is a magma and satisfies latin square property. Quasigroup of type (2,2,2) is a structure with three binary operations, a magma for which division is always possible. Latin square property states that for each a , b in set Q there exists unique elements x , y in Q such that the following property is satisfied (Wikipedia contributors, 2022)

$$a \cdot x = b$$

$$y \cdot a = b$$

Another definition of quasigroup is given as type (2,2,2) algebra in which for a set Q and binary operations $\cdot$, \ , / quasigroup should satisfy the below identities that implies left division and right division.

$$y = x \cdot (x \setminus y)$$

$$y = x \setminus (x \cdot y)$$

$$y = (y / x) \cdot x$$

$$y = (y \cdot x) / x$$

In Agda standard library the quasigroup is defined as type (2,2,2) algebra given below.

```
record IsQuasigroup (· \ \ // : Op₂ A) : Set (a ⊔ ℓ) where
  field
    isMagma        : IsMagma ·
    \ \-cong        : Congruent₂ \ \
    //-cong        : Congruent₂ //
    leftDivides    : LeftDivides · \\
    rightDivides   : RightDivides · //


  open IsMagma isMagma public
```

A quasigroup with signature (2) and a quasigroup with signature (2,2,2) are equivalent but are structurally different. In the algebra hierarchy, a Loop is an algebraic structure that is a quasigroup with identity. It can be observed the same problem persists through the hierarchy. If a loop is defined with a quasigroup that is type (2,2,2) algebra

then it a loop structure of type (2) will be forced to be defined with sub-optimal name. One plausible solution to this problem is to define the structures in different modules and import restrict them when using. This problem of not being able to overload names for structures also affects when defining types of quasigroup or loops such as bol loop and moufang loop.

Since quasigroup is defined in terms of division operation, loop is also defined as a type (2,2,2) algebra in Agda. The definition of loop structure in Agda is given below.

```
record IsLoop (· \ \ // : Op₂ A) (ε : A) : Set (a ⊔ ℓ) where
  field
    isQuasigroup : IsQuasigroup · \ \ //
    identity     : Identity ε ·

  open IsQuasigroup isQuasigroup public
```

6.3 Redundant field in structural inheritance

Redundancy arises when there is duplication of the same field. In programming redundant of code is considered a bad practice and is usually avoided by modularising and creating functions that perform similar tasks. In algebraic structures, redundant fields can be introduced in structures that are defined in terms of two or more structures. For example semiring can be as commutative monoid under addition and a monoid under multiplication and multiplication distributes over addition. In Agda, both monoid and commutative monoid have an instance of equivalence relation. If semiring is defined in

terms of commutative monoid and monoid then this definition of the semiring will have a redundant equivalence field. This redundancy can also be seen in other structures like ring, lattice, module, etc., To remove this redundant field in Agda the structure except the first is opened and expressed in terms of independent axioms that they satisfy. For example, semiring without identity or rig structure in Agda is defined as

```
record IsSemiringWithoutOne (+ * : Op₂ A) (0# : A) : Set (a ⊔ ℓ) where
  field
    +-isCommutativeMonoid : IsCommutativeMonoid + 0#
    *-cong                : Congruent₂ *
    *-assoc               : Associative *
    distrib               : * DistributesOver +
    zero                  : Zero 0# *

  open IsCommutativeMonoid +-isCommutativeMonoid public
```

From the above definition it is evident that an instance of semigroup should be constructed and is not directly available when using semiring without one structure. To overcome this problem an instance is created in the definition as follows along with near semiring structure.

```
  *-isMagma : IsMagma *
  *-isMagma = record
    { isEquivalence = isEquivalence
    ; ·-cong        = *-cong
```

```
    }

  *-isSemigroup : IsSemigroup *
  *-isSemigroup = record
    { isMagma = *-isMagma
    ; assoc   = *-assoc
    }

  isNearSemiring : IsNearSemiring + * 0#
  isNearSemiring = record
    { +-isMonoid    = +-isMonoid
    ; *-cong        = *-cong
    ; *-assoc       = *-assoc
    ; distrib^r     = proj_2 distrib
    ; zero^l        = zero^l
    }
```

The above technique will effectively remove the redundant equivalence relation but it also fails to express the structure in terms of two or more structures that is commonly used in literature and in other systems. Agda 2.0 removed redundancy by unfolding the structure. This solution should make sure that the structure clearly exports the unfolded structure whose properties can be imported when required.

6.4 Identical structures

In abstract algebra when formalising algebraic structures from the hierarchy, same algebraic structure can be derived from two or more structures. One such example is Nearring. Nearring is an algebraic structure with two binary operations addition and multiplication. Near ring is a group under addition and is a monoid under multiplication and multiplication right distributes over addition. In this case near-ring is defined using two algebraic structures group and monoid. Other definition of near-ring can be derived using the structure quasiring. Quasiring is an algebraic structure in which addition is a monoid, multiplication is a monoid and multiplication distributes over addition. Using this definition of quasiring, near-ring can be defined as a quasiring which has additive inverse.

In Agda nearring is defined in terms of quasiring with additive inverse

```
record IsNearring (+ * : Op₂ A) (0# 1# : A) (_⁻¹ : Op₁ A) : Set (a ⊔ ℓ) where
  field
    isQuasiring : IsQuasiring + * 0# 1#
    +-inverse   : Inverse 0# _⁻¹ +
    ⁻¹-cong     : Congruent₁ _⁻¹

  open IsQuasiring isQuasiring public
```

Note that in some literature, near-ring is defined in which multiplication is a semigroup that is without identity. This can be attributed to the problem with ambiguity. It can be analysed that having two different definitions for same structure is not a good practice. If near-ring is defined using quasiring then it should also give an instance of additive group without having it to construct it when using the above formalisation. This solution might

solve the problem at first but in practice this becomes tedious and can go to a point at which this can be impractical especially when formalising structures at higher level in the algebra hierarchy.

6.5 Equivalent structures

Consider the example of idempotent-commutative-monoid and bounded semilattice. It can be observed that both are essentially same structure. In this case it could be redundant to define two different structures from different hierarchy. Instead in Agda, aliasing is used. Idempotent-commutative-monoid is defined and an aliasing for bounded semilattice is given.

```
record IsIdempotentCommutativeMonoid (· : Op₂ A) (ϵ : A) : Set (a ⊔ ℓ) where
  field
    isCommutativeMonoid : IsCommutativeMonoid · ϵ
    idem                : Idempotent ·

  open IsCommutativeMonoid isCommutativeMonoid public

IsBoundedSemilattice = IsIdempotentCommutativeMonoid
module IsBoundedSemilattice · ϵ (L : IsBoundedSemilattice · ϵ) where

  open IsIdempotentCommutativeMonoid L public
```

Note that some mathematicians argue that bounded semilattice and idempotent commutative monoid are not structurally the same structures but are isomorphic to

each other. We do not consider this argument in the scope of this thesis.

Chapter 7

Conclusion

Every thesis also needs a concluding chapter

Appendix A

Your Appendix

Your appendix goes here.

Appendix B

Long Tables

This appendix demonstrates the use of a long table that spans multiple pages.

Col A	Col B	Col C	Col D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D

Continued on the next page

Continued from previous page

Col A	**Col B**	**Col C**	**Col D**
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D
A	B	C	D

Bibliography

Russell O'Connor Jacques Carette and Yasmine Sharoda. 2019. Building on the Diamonds between Theories: Theory Presentation Combinators. *arXiv preprint arXiv:1812.08079* (2019).

Assia Mahboubi and Enrico Tassi. 2021. *Mathematical Components*. Zenodo. https://doi.org/10.5281/zenodo.4457887

The mathlib Community. 2020. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New Orleans, LA, USA) *(CPP 2020)*. Association for Computing Machinery, New York, NY, USA, 367–381. https://doi.org/10.1145/3372885.3373824

Carette Jacques Farmer William M. Kohlhase Michael and Rabe Florian. 2021. Big Math and the One-Brain Barrier: The Tetrapod Model of Mathematical Knowledge. *Math Intelligencer* 43 (2021), 78–87. https://doi.org/doi.org/10.1007/s00283-020-10006-0

Christine Paulin-Mohring. 2012. *Introduction to the Coq Proof-Assistant for Practical Software Verification*. Springer Berlin Heidelberg, Berlin, Heidelberg, 45–95. https://doi.org/10.1007/978-3-642-35746-6_3

M. Saqib Nawaz, Moin Malik, Yi Li, Meng Sun, and M. Ikram Ullah Lali. 2019. A Survey on Theorem Provers in Formal Methods. *arXiv e-prints*, Article arXiv:1912.03028 (Dec. 2019), arXiv:1912.03028 pages. arXiv:1912.03028 [cs.SE]

Wikipedia contributors. 2022. Quasigroup — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/w/index.php?title=Quasigroup&oldid=1123964335` [Online; accessed 7-January-2023].