Universal Algebra in HoTT

Andreas Lynge
201710283

Supervised by
Bas Spitters

A thesis presented for the degree of
Bachelor of Mathematics

Department of Mathematics
Aarhus University
Denmark

Revised: 21st June 2019

Abstract

This text presents a universal algebra development in Coq for the Homotopy Type Theory (HoTT) library. Developments of universal algebra in type theory are commonly using setoids to model quotient sets. Setoids are best avoided because they complicate the implementation. This report shows that setoids are not needed in homotopy type theoretic universal algebra. The development in this report contains definitions of subalgebra, product algebra and quotient algebra. These definitions are verified for correctness using category theoretic techniques. Later they are used to prove the three isomorphism theorems, which can be seen as a milestone. A key theorem of the development shows that isomorphic algebras are in fact equal in HoTT. This is used to show that the precategory of algebras for a signature is a univalent category, and we obtain equalities from the isomorphism theorems.

Abstract - Danish

Denne tekst præsenterer en universel algebra implementering i Coq for Homotopy Type Theory (HoTT) biblioteket. Implementeringer af universel algebra i type-teori bruger ofte setoids til at modellere kvotientmængder. Setoids bør undgås fordi de komplicerer implementeringen. Denne rapport viser at setoids ikke er nødvendige i universel algebra i HoTT. Implementeringen i denne rapport indeholder definitioner af under-algebra, produkt-algebra og kvotient-algebra. Disse definitioner er verificeret for korrekthed ved brug af kategori-teoretiske teknikker. Senere er definitionerne brugt til at bevise de tre isomorfi sætninger, hvilket kan anses som en milepæl. En nøglesætning i implementeringen viser at isomorfe algebraer er lig med hinanden i HoTT. Dette anvendes til at vise at præ-kategorien af algebraer for en signatur er en univalent kategori, og vi opnår ligheder fra isomorfi sætningerne.

Contents

Introduction

This text presents the beginning of a universal algebra development in Coq for the Homotopy Type Theory (HoTT) library [2]. The Coq formalization of this is located at `https://github.com/andreaslyn/hott-classes`. The work is based on the Math Classes library due to Spitters and van der Weegen [10], which was originally developed to serve as a basis for constructive analysis in Coq.

Universal algebra is important to mathematics because it provides general results about algebraic strucures. The isomorphism theorems in universal algebra are generalisations of the isomorphism theorems known from group theory and ring theory. In universal algebra these theorems apply to a broad collection of algebraic structures, including groups and rings and modules, hence proving these theorems once and for all. In computer science, universal algebra is used to characterize algebraic data types (known from functional languages) as initial algebras in certain categories of algebras. Birkhoff used universal algebra to study regular languages as algebras [3].

Part I of the text provides some background on category theory, universal algebra, and HoTT. The reader is assumed familiar with type theory. Part II presents the results of the universal algebra development for the Coq HoTT library.

Problem

Universal algebra has been formalized in Coq by Spitters and van der Weegen [10], and in Agda by Gunther, Gadea and Pagano [8]. In order to model quotient types and function extensionality, these developments are relying on setoids, a type together with an equivalence relation. Setoids complicate the theory because maps between setoids are required to respect the equivalence relations, and existing theorems relying on strict equality do not apply to setoids. Also, users of the library obtain results about setoids, which forces them to rely on setoids to some extent. This may escalate and add complexity to other developments as well.

The univalence axiom in HoTT implies function extensionality and higher inductive types can be used to define quotient types without the need for setoids.

This text develops universal algebra in HoTT using higher inductive types, so without relying on setoids. Section 7.3 contains a homotopy type theoretic definition of quotient algebra. A convenient practice in set theoretic foundations is to view isomorphic objects as being equal. A key result, Theorem 5.5, states that isomorphic algebras are formally equal in HoTT. This is used in Section 6 to show that the category of algebras for a signature forms a univalent category, and in Section 8 to obtain equalities from the isomorphism theorems.

Part I

Background

1 Category Theory

This section introduces elementary notions from category theory. Readers familiar with category theory can safely skip this section. The section is based on Steve Awodeys category theory book [1]. Throughout the section we will be working in set theory (with proper classes).

1.1 Definitions

Definition 1.1. A *category* $\mathbf{C}$ consists of

- a collection of *objects* $\mathbf{C}_0$,
- a collection of *morphisms* $\mathbf{C}_1$.

It is required that:

- For each morphism $f \in \mathbf{C}_1$ there are objects $\mathrm{dom}(f) \in \mathbf{C}_0$ and $\mathrm{cod}(f) \in \mathbf{C}_0$ called the *domain* and *codomain* of f.
- There is a binary *composition* operator $\circ$ defined for morphisms $f \in \mathbf{C}_1$ and $g \in \mathbf{C}_1$ where $\mathrm{cod}(f) = \mathrm{dom}(g)$, such that $g \circ f \in \mathbf{C}_1$ and $\mathrm{dom}(g \circ f) = \mathrm{dom}(f)$ and $\mathrm{cod}(g \circ f) = \mathrm{cod}(g)$.
- For any $A \in \mathbf{C}_0$ there is an *identity morphism* $1_A \in \mathbf{C}_1$ with $\mathrm{dom}(1_A) = A$ and $\mathrm{cod}(1_A) = A$.

Furthermore, the following laws hold:

- For all f, g, h in $\mathbf{C}_1$ where $\mathrm{cod}(f) = \mathrm{dom}(g)$ and $\mathrm{cod}(g) = \mathrm{dom}(h)$,
$$h \circ (g \circ f) = (h \circ g) \circ f \qquad \text{(associativity law).}$$
- For any $f \in \mathbf{C}_1$,
$$f \circ 1_{\mathrm{dom}(f)} = f = 1_{\mathrm{cod}(f)} \circ f \qquad \text{(unit laws).}$$

△

Notation 1.2. Given a category $\mathbf{C}$, it is convenient to write $f : A \to B$ to mean a morphism $f \in \mathbf{C}_1$ with $\mathrm{dom}(f) = A$ and $\mathrm{cod}(f) = B$. When there is no danger of ambiguity we will write $A \in \mathbf{C}$ instead of $A \in \mathbf{C}_0$, and similarly for morphisms. △

Example 1.3.

(i) A basic category is the category $\mathbf{1}$ consisting of a single object $\star \in \mathbf{1}$ and a single morphism $1_\star \in \mathbf{1}$.

(ii) There is a category $\mathbf{0}$ with no objects and no morphisms.

(iii) An example of a bigger category is the category **Set** of all sets. In this category the objects $\mathbf{Set}_0$ are sets and the morphisms $\mathbf{Set}_1$ are functions. Morphism composition is defined to be function composition and the identity morphisms are the identity functions. This is a large category because $\mathbf{Set}_1$ is a proper class. ◊

Definition 1.4. An *isomorphism* in a category $\mathbf{C}$ is a morphism $f : A \to B$ in $\mathbf{C}$ for which there exists an *inverse* morphism $g : B \to A$ in $\mathbf{C}$, such that

$$g \circ f = 1_A \qquad \text{and} \qquad f \circ g = 1_B.$$

If there exists such an inverse morphism we say that A and B are *isomorphic.* △

Definition 1.5. A *functor* is a map $F : \mathbf{C} \to \mathbf{D}$ between categories $\mathbf{C}$ and $\mathbf{D}$, where every object $A \in \mathbf{C}$ is associated to an object $F(C) \in \mathbf{D}$ and every morphism $f : B \to C$ in $\mathbf{C}$ is associated to a morphism $F(f) : F(B) \to F(C)$ in $\mathbf{D}$. A functor $F : \mathbf{C} \to \mathbf{D}$ must preserve identity and composition in the sense that

$$F(1_A) = 1_{F(A)} \qquad \text{and} \qquad F(g \circ f) = F(g) \circ F(f).$$

△

Definition 1.6. A *natural transformation* $\alpha : F \to G$ between functors $F, G : \mathbf{C} \to \mathbf{D}$ consists of morphisms $\alpha_A : F(A) \to G(A)$ for each object $A \in \mathbf{C}$, such that for any morphism $f : A \to B$ in $\mathbf{C}$ the following square commutes:

$$\begin{tikzcd}
F(A) \arrow[r, "\alpha_A"] \arrow[d, "F(f)"'] & G(A) \arrow[d, "G(f)"] \\
F(B) \arrow[r, "\alpha_B"] & F(B)
\end{tikzcd}$$

This means that α is required to satisfy $\alpha_B \circ F(f) = G(f) \circ \alpha_A$.

A *natural isomorphism* is a natural transformation α where each morphism α_A is an isomorphism. △

Remark 1.7. A category $\mathbf{C}$ gives rise to a *dual category* $\mathbf{C}^{\text{op}}$. For each object $A \in \mathbf{C}$ there is a corresponding dual object $A^{\text{op}} \in \mathbf{C}^{\text{op}}$ and for each morphism $f : A \to B$ in $\mathbf{C}$ there is a corresponding dual morphism $f^{\text{op}} : B^{\text{op}} \to A^{\text{op}}$. Identity morphisms $1_{A^{\text{op}}}$ are given by $(1_A)^{\text{op}}$ and composition $f^{\text{op}} \circ g^{\text{op}}$ is $(g \circ f)^{\text{op}}$. ◊

1.2 Universal properties

Definition 1.8.

(i) An object 0 in a category $\mathbf{C}$ is *initial* iff for every object $A \in \mathbf{C}$ there is a unique morphism $0 \to A$.

(ii) An object 1 in a category $\mathbf{C}$ is *terminal* iff for every object $A \in \mathbf{C}$ there is a unique morphism $A \to 1$. △

Example 1.9. In **Set** the empty set is initial and any singleton set is terminal. ◊

Definition 1.10.

(i) A *diagram* of shape $\mathbf{J}$ is a functor $F : \mathbf{J} \to \mathbf{C}$.

(ii) A *cone* over a diagram $F : \mathbf{J} \to \mathbf{C}$ is a natural transformation $\alpha : A \to F$ with *summit* A, an object in $\mathbf{C}$, which can be regarded as a constant functor. For i object in $\mathbf{J}$, we refer to the morphisms $\alpha_i : A \to F(i)$ as the *legs* of the cone. △

Definition 1.11. Suppose $F : \mathbf{J} \to \mathbf{C}$ is a diagram. There is a category $\text{Cone}(F)$ where the objects are cones over F. A morphism in $\text{Cone}(F)$ from a cone $\alpha : A \to F$ to

$\beta : B \to F$ corresponds to a morphism $\vartheta : A \to B$ in $\mathbf{C}$ satisfying $\alpha_i = \beta_i \circ \vartheta : A \to F(i)$ for all objects $i \in \mathbf{J}$. The identity morphism in $\mathrm{Cone}(F)$ of a cone $\alpha : A \to F$ is the identity morphism $1_A \in \mathbf{C}$, and composition in $\mathrm{Cone}(F)$ is composition in $\mathbf{C}$. △

Definition 1.12. A *limit* of a diagram $F : \mathbf{J} \to \mathbf{C}$ is a terminal object in $\mathrm{Cone}(F)$. △

Example 1.13. Consider a category $\mathbf{K}$ consisting of two objects 1 and 2, and the two required identity morphisms. Let $F : \mathbf{K} \to \mathbf{C}$ be a diagram and write $A = F(1)$ and $B = F(2)$. A limit of the diagram F is referred to as a *binary product*, we write $A \times B$ for the summit of the limit cone and $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$ for the legs. If $f : X \to A$ and $g : X \to B$ are morphisms in $\mathbf{C}$ then there is a cone $\alpha : X \to F$ with $\alpha_1 = f$ and $\alpha_2 = g$. Hence, there is a unique map $\langle f, g\rangle : X \to A \times B$ which satisfies $\pi_1 \circ \langle f, g\rangle = f$ and $\pi_2 \circ \langle f, g\rangle = g$, as indicated in the following diagram.

$$\begin{tikzcd}
 & A \times B \arrow[dl, "\pi_1"'] \arrow[dr, "\pi_2"] & \\
A & & B \\
 & X \arrow[ul, "f"] \arrow[ur, "g"'] \arrow[uu, dashed, "\langle f, g\rangle"] &
\end{tikzcd}$$

In the category **Set**, a binary product corresponds to the usual cartesian product where the projections $\pi_1(x, y) = x$ and $\pi_2(x, y) = y$ are the legs of the limit cone. ♢

Example 1.14. A limit of a diagram $F : \mathbf{0} \to \mathbf{C}$ is a terminal object in $\mathbf{C}$, since $\mathrm{Cone}(F) = \mathbf{C}$. ♢

Definition 1.15. Given a functor $F : \mathbf{C} \to \mathbf{D}$ there is a *dual functor* $F^{\mathrm{op}} : \mathbf{C}^{\mathrm{op}} \to \mathbf{D}^{\mathrm{op}}$ defined on objects and morphisms by

$$F^{\mathrm{op}}(X^{\mathrm{op}}) = F(X)^{\mathrm{op}} \qquad \text{and} \qquad F^{\mathrm{op}}(f^{\mathrm{op}}) = F(f)^{\mathrm{op}}.$$

△

Definition 1.16. Let $F : \mathbf{J} \to \mathbf{C}$ be a diagram. The category of cones $\mathrm{Cone}(F^{\mathrm{op}})$ has a dual category of *cocones* $\mathrm{Cocone}(F) = \mathrm{Cone}(F^{\mathrm{op}})^{\mathrm{op}}$. △

Remark 1.17. Cocones $\alpha \in \mathrm{Cocone}(F)$ are natural transformations $\alpha : F \to A$ where $A \in \mathbf{C}$ is an object called the *nadir*. A morphism from cocone $\alpha : F \to A$ to $\beta : F \to B$ corresponds to a morphism $\vartheta : B \to A$ in $\mathbf{C}$ such that $\alpha_i = \vartheta \circ \beta_i : F(i) \to A$ for all objects $i \in \mathbf{J}$. ♢

Definition 1.18. A *colimit* is an initial object in the category of cocones $\mathrm{Cocone}(F)$. △

Remark 1.19. Since a limit in $\mathbf{C}^{\mathrm{op}}$ is a terminal object in the category $\mathrm{Cone}(F^{\mathrm{op}})$ it corresponds to an initial object in the dual category $\mathrm{Cocone}(F)$. Hence a limit in $\mathbf{C}^{\mathrm{op}}$ corresponds to a colimit in $\mathbf{C}$. ♢

Example 1.20. A colimit of $F : \mathbf{0} \to \mathbf{C}$ is an initial object in $\mathbf{C}$ because $\mathrm{Cocone}(F) = \mathrm{Cone}(F^{\mathrm{op}})^{\mathrm{op}} = (\mathbf{C}^{\mathrm{op}})^{\mathrm{op}} = \mathbf{C}$. ♢

2 Universal algebra

This section presents set theoretic multi sorted universal algebra. Readers familiar with multi sorted universal algebra may want to just skim this section. The section is based on the Math Classes library [10] and the universal algebra book by Stanley and Sankappanavar [4].

2.1 Definitions

Definition 2.1. A *signature* σ consists of:

- A set of *sorts* $\mathcal{S}_\sigma$.
- A set of *function symbols* $\mathcal{F}_\sigma$.
- For each function symbol $\alpha \in \mathcal{F}_\sigma$, a function symbol type, which is a finite sequence $\mathcal{T}_\alpha = (s_n)_{n \leq \mathrm{ari}(\alpha)}$ of sorts $s_n \in \mathcal{S}_\sigma$, where $n \in \mathbb{N}_0$ and $\mathrm{ari}(\alpha) \in \mathbb{N}_0$.

The number $\mathrm{ari}(\alpha)$ is called the *arity* of the function symbol α. △

Definition 2.2. An *algebra* $\mathbf{A}$ for a signature σ consists of:

- A family of *carriers* $(\mathbf{A}_s)_{s \in \mathcal{S}_\sigma}$ indexed by $s \in \mathcal{S}_\sigma$.
- A family of *operations* $(\alpha^{\mathbf{A}})_{\alpha \in \mathcal{F}_\sigma}$ indexed by $\alpha \in \mathcal{F}_\sigma$. An operation for $\alpha \in \mathcal{F}_\sigma$ is an n-ary function (a constant when $n = 0$)

  $$\alpha^{\mathbf{A}} : (A_{s_1} \times A_{s_2} \times \cdots \times A_{s_n}) \to A_t$$

  where $n = \mathrm{ari}(\alpha)$ is the arity of the function symbol $\alpha \in \mathcal{F}_\sigma$ and $(s_1, s_2, \ldots, s_n, t) = \mathcal{T}_\alpha$ is the function symbol type of α.

△

Example 2.3. Any group G is an algebra for a signature with just one sort. A group G has a binary operation $\cdot : G \times G \to G$, a unary operation $(-)^{-1} : G \to G$ and a constant $1 \in G$. ◊

Definition 2.4. Given algebras $\mathbf{A}$ and $\mathbf{B}$ for some signature σ. An algebra *homomorphism* $f : \mathbf{A} \to \mathbf{B}$ is a family of functions

$$(f_s : \mathbf{A}_s \to \mathbf{B}_s)_{s \in \mathcal{S}_\sigma}, \text{ indexed by } s \in \mathcal{S}_\sigma,$$

satisfying

$$f_t(\alpha^{\mathbf{A}}(a_1, \ldots, a_n)) = \alpha^{\mathbf{B}}(f_{s_1}(a_1), \ldots, f_{s_n}(a_n))$$

for all function symbols $\alpha \in \mathcal{S}_\sigma$, where $(s_1, \ldots, s_n, t) = \mathcal{T}_\alpha$ is the function symbol type. △

Definition 2.5. An algebra *isomorphism* is a homomorphism $(f_s)_s$ where f_s is bijective for all sorts $s \in \mathcal{S}_\sigma$. If there exists an isomorphism $\mathbf{A} \to \mathbf{B}$ then we say $\mathbf{A}$ and $\mathbf{B}$ are *isomorphic.* △

Example 2.6. Group homomorphisms/isomorphisms are algebra homomorphisms/isomorphisms. ◊

Lemma 2.7. Let $\mathbf{A}, \mathbf{B}, \mathbf{C}$ be algebras a signature σ and suppose there exist homomorphisms $f = (f_s : \mathbf{A}_s \to \mathbf{B}_s)_s$ and $g = (g_s : \mathbf{B}_s \to \mathbf{C}_s)_s$. The family of composed functions

$$g \circ f := (g_s \circ f_s : \mathbf{A}_s \to \mathbf{C}_s)_{s \in \mathcal{S}_\sigma}$$

is a homomorphism $\mathbf{A} \to \mathbf{C}$. □

Definition 2.8. Let $\mathbf{A}$ and $\mathbf{B}$ be algebras for a signature σ. Then $\mathbf{B}$ is a *subalgebra* of $\mathbf{A}$ iff

- $\mathbf{B}_s \subseteq \mathbf{A}_s$ for all sorts $s \in \mathcal{S}_\sigma$,
- $\alpha^{\mathbf{B}} : (\mathbf{B}_{s_1} \times \cdots \times \mathbf{B}_{s_n}) \to B_t$ is the restriction of $\alpha^{\mathbf{A}} : (\mathbf{A}_{s_1} \times \cdots \times \mathbf{A}_{s_n}) \to A_t$ for all function symbols $\alpha \in \mathcal{F}_\sigma$ and all $(s_1, \ldots, s_n, t) = \mathcal{T}_\alpha$. $\triangle$

2.2 Isomorphism theorems

Normal subgroups play a central role in defining quotient groups and in the isomorphism theorems, which are fundamental to the development of group theory. Ideals play an analogous role in defining quotient rings and in the corresponding isomorphism theorems in ring theory. Given this parallel situation, it seems that there should be a general formulation of normal subgroup and ideal. In this subsection we will see that congruence is such a formulation, giving rise to generic versions of the isomorphism theorems.

Definition 2.9. Let $\mathbf{A}$ be an algebra for a signature σ. A family of equivalence relations $\sim_s \subseteq \mathbf{A}_s \times \mathbf{A}_s$, indexed by $s \in \mathcal{S}_\sigma$, is a *congruence* on $\mathbf{A}$ iff

$$\alpha^{\mathbf{A}}(a_1, \ldots, a_n) \sim_t \alpha^{\mathbf{A}}(b_1, \ldots, b_n), \text{ whenever } a_1 \sim_{s_1} b_1, \ldots, a_n \sim_{s_n} b_n,$$

for $\alpha \in \mathcal{F}_\sigma$ and $(s_1, \ldots, s_n, t) = \mathcal{T}_\alpha$ the function symbol type. $\triangle$

Definition 2.10. Suppose $\sim\, = (\sim_s)_{s \in \mathcal{S}_\sigma}$ is a congruence on an algebra $\mathbf{A}$ for some signature σ. The *quotient algebra* $\mathbf{A}/\!\sim$ is the algebra for σ with

- carriers $\mathbf{A}_s/\!\sim_s$, for each $s \in \mathcal{S}_\sigma$, the quotient set of $\mathbf{A}_s$ by $\sim_s$;
- operations $\alpha^{(\mathbf{A}/\sim)}([a_1], \ldots, [a_n]) = [\alpha^{\mathbf{A}}(a_1, \ldots, a_n)]$, for $\alpha \in \mathcal{F}_\sigma$ and equivalence classes $[a_i] \in \mathbf{A}_{s_i}/\!\sim_{s_i}$.

This algebra is well defined. $\triangle$

Example 2.11. If $\sim$ is a congruence on a group G with unit 1 then the equivalence class $N := [1] \in G/\!\sim$ is a normal subgroup of G, where the quotient group G/N and the quotient algebra $G/\!\sim$ coincide.

Conversely, if N is a normal subgroup of G then the relation $\sim$ given by

$$x \sim y \qquad \text{iff} \qquad xy^{-1} \in N$$

is a congruence on G where $N = [1] \in G/\!\sim$ and $G/N = G/\!\sim$. $\diamondsuit$

Definition 2.12. Let $\mathbf{A}, \mathbf{B}$ be algebras for a signature σ and suppose $f = (f_s : \mathbf{A}_s \to \mathbf{B}_s)_s$ is a homomorphism $\mathbf{A} \to \mathbf{B}$. The *kernel* $\ker(f)$ of f is a family of sets $\ker_t(f) \subseteq \mathbf{A}_t \times \mathbf{A}_t$, indexed by $t \in \mathcal{S}_\sigma$, defined by

$$\ker_t(f) = \{(a, b) \in \mathbf{A}_t \times \mathbf{A}_t \mid f_t(a) = f_t(b)\}.$$

$\triangle$

Remark 2.13. A function $f_t : \mathbf{A}_t \to \mathbf{B}_t$ of an algebra homomorphism $f : \mathbf{A} \to \mathbf{B}$ is injective if and only if $\ker_t(f)$ is the identity relation. $\diamondsuit$

Theorem 2.14 (First isomorphism theorem)**.** Suppose $\mathbf{A}$ and $\mathbf{B}$ are algebras for a signature σ. Let $f : \mathbf{A} \to \mathbf{B}$ be a homomorphism.

(i) The homomorphic image $f(\mathbf{A}) := (f_s(\mathbf{A}_s))_{s \in \mathcal{S}_\sigma}$ is a subalgebra of $\mathbf{B}$.
(ii) The kernel $\ker(f)$ is a congruence on $\mathbf{A}$.
(iii) The quotient algebra $\mathbf{A}/\ker(f)$ and the image algebra $f(\mathbf{A})$ are isomorphic. $\square$

Theorem 2.15 (Second isomorphism theorem)**.** Let $\mathbf{A}$ and $\mathbf{B}$ be algebras with $\mathbf{B}$ a subalgebra of $\mathbf{A}$ and assume $\varphi = (\varphi_s)_{s\in\mathcal{S}_\sigma}$ is a congruence on $\mathbf{A}$. For $s \in \mathcal{S}_\sigma$, write

$$\begin{aligned}\varphi_s^{\mathbf{B}} &= \varphi_s \cap (\mathbf{B} \times \mathbf{B}),\\ [\mathbf{B}]_s^{\varphi} &= \{[a] \in \mathbf{A}_s/\varphi_s \mid [a] \cap \mathbf{B}_s \neq \emptyset\}.\end{aligned}$$

(i) The family of relations $\varphi^{\mathbf{B}} := (\varphi_s^{\mathbf{B}})_{s\in\mathcal{S}_\sigma}$ is a congruence on $\mathbf{B}$.
(ii) The family of sets $[\mathbf{B}]^{\varphi} := ([\mathbf{B}]_s^{\varphi})_{s\in\mathcal{S}_\sigma}$ is a subalgebra of $\mathbf{A}/\varphi$.
(iii) The algebras $\mathbf{B}/\varphi^{\mathbf{B}}$ and $[\mathbf{B}]^{\varphi}$ are isomorphic. □

Theorem 2.16 (Third isomorphism theorem)**.** Let φ, ϑ be congruences on some algebra $\mathbf{A}$ where $\vartheta_s \subseteq \varphi_s$ for all $s \in \mathcal{S}_\sigma$. Set

$$\varphi_s/\vartheta_s = \{([a],[b]) \in (\mathbf{A}_s/\vartheta_s) \times (\mathbf{A}_s/\vartheta_s) \mid \varphi_s(a,b)\}, \qquad \text{for } s \in \mathcal{S}_\sigma.$$

(i) The family of relations $\varphi/\vartheta := (\varphi_s/\vartheta_s)_{s\in\mathcal{S}_\sigma}$ is a congruence on $\mathbf{A}/\vartheta$.
(ii) The algebras $(\mathbf{A}/\vartheta)/(\varphi/\vartheta)$ and $\mathbf{A}/\varphi$ are isomorphic. □

3 Homotopy Type Theory

This section is based on the HoTT book [11]. Readers already familiar with HoTT may want to skip to section 3.3 and skim it.

HoTT is an alternative to ZFC set theory as a foundation of mathematics. It is in particular distinguished from ZFC by being a type theory rather than a first order theory. HoTT allows for a convenient synthetic approach to homotopy theory where types are spaces, type inhabitants are points and identity types are paths. The cubical type theory gives a constructive interpretation of HoTT [7, 5]. An advantage of HoTT is that it formalizes the natural mathematical practice of identifying isomorphic objects, see for instance Theorem 5.5.

Section 3.1 introduces the basic type theory that the HoTT book is based on. Section 3.2 presents some of the elementary notions from HoTT. Section 3.3 introduces a couple of higher inductive types.

3.1 Type Theory

A *universe* is a type of types. All universes $\mathcal{U}_n$ come with an associated level $n \in \mathbb{N}$. There is a cumulative hierarchy of universes

$$\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \cdots$$

So universe $\mathcal{U}_n$ has type $\mathcal{U}_{n+k}$ for any $k \geq 1$. To simplify notation we leave the universe level implicit and write $\mathcal{U}$.

We use $\equiv$ for judgmental equality and $=$ for the identity type. The induction principle for the identity type is

$$\begin{aligned}&\mathrm{ind}_{=_A} : \prod_{\left(C:\prod_{(x,y:A)}(x=y)\to\mathcal{U}_i\right)} \Big(\prod_{(x:A)} C(x,x,\mathrm{refl}_x)\Big) \to \prod_{(x,y:a)}\prod_{(p:x=y)} C(x,y,p)\\ &\mathrm{ind}_{=_A}(C,c,x,x,\mathrm{refl}_x) \equiv c(x),\end{aligned}$$

where we write $f(a,b)$ for $f(a)(b)$ when the intention is clear.

Definition 3.1. Suppose $A : \mathcal{U}$ is a type and $x, y : A$ inhabitants. The identity type $x = y$ is called a *path* from x to y and the induction principle for the identity type is referred to as *path induction.* A term $p : x = y$ is viewed on as a path with *endpoints* x and y in a space A. △

Remark 3.2. The interpretation of identity types as paths is made precise in the simplicial model of univalent foundations [9]. ◊

Lemma 3.3. The path type is an equivalence relation. For let $x, y, z : A$ be inhabitants of a type $A : \mathcal{U}$, then

- $\mathrm{refl}_x : x = x$,
- $p : x = y$ implies $p^{-1} : y = x$,
- $p : x = y$ and $q : y = z$ implies $p \cdot q : x = z$. □

Definition 3.4. Let $p : x = y$ and $q : y = z$ be paths in some type A. We refer to $p^{-1} : y = x$ as the *inverse* path of p and $p \cdot q : x = z$ as the *composite* of p and q. △

3.2 Univalent foundations

The first definition in this section is fundamental and due to Voevodsky [12].

Definition 3.5. A type A is *contractible* if there exists a point $a : A$ and a dependent function $f : \prod_{(x:A)}(a = x)$ mapping $x : A$ to a path $a = x$,

$$\begin{aligned} &\mathrm{isContr} : \mathcal{U} \to \mathcal{U} \\ &\mathrm{isContr}(A) :\equiv \sum_{(a:A)} \prod_{(x:A)} (a = x). \end{aligned}$$

△

Remark 3.6. It is tempting to use a propositions-as-types interpretation and read the type $\mathrm{isContr}(A)$ as: there exists a basepoint $a : A$ such that for all $x : A$ there is a path $a = x$ from a to x. This makes it sound like A is just path-connected. It actually says something stronger. For an intuition, let A be a set theoretic topological space and $I = [0, 1]$ the unit interval. Suppose there exists a point $a \in A$ and a homotopy $f : A \times I \to X$ such that for all $x \in A$, $f(x, -) : I \to A$ is a path from a to x. Then f is a homotopy $a \simeq \mathrm{id}_A$ showing that the identity function on A is nullhomotopic. This says exactly that A is a contractible space. ◊

Example 3.7. The unit type $\mathbf{1}$ is a contractible type. Indeed

$$\begin{aligned} &\mathrm{unitIsContr} : \mathrm{isContr}(\mathbf{1}) \\ &\mathrm{unitIsContr} :\equiv (\star,\ \mathrm{ind}_{\mathbf{1}}(\lambda x.\ \star = x,\ \mathrm{refl}_\star)), \end{aligned}$$

where $\mathrm{ind}_{\mathbf{1}} : \prod_{(C:\star\to\mathcal{U})} C(\star) \to \prod_{(x:\mathbf{1})} C(x)$ is the induction principle for $\mathbf{1}$. ◊

Definition 3.8. A *mere proposition* is a type A for which $x = y : \mathcal{U}$ is contractible for all $x, y : A$,

$$\begin{aligned}&\mathrm{isProp} : \mathcal{U} \to \mathcal{U}\\&\mathrm{isProp}(A) :\equiv \prod_{(x,y:A)} \mathrm{isContr}(x = y).\end{aligned}$$

△

Example 3.9.

(i) Any contractible type is a mere proposition,

$$\begin{aligned}&\mathrm{contrIsProp} : \prod_{(A:\mathcal{U})} \mathrm{isContr}(A) \to \mathrm{isProp}(A)\\&\mathrm{contrIsProp}(A, (a, P))(x, y) :\equiv (P(x))^{-1} \cdot P(y), \qquad \text{since } P : \prod_{(z:A)} (a = z).\end{aligned}$$

(ii) The empty type $\mathbf{0}$ is a mere proposition, but it is not contractible.

(iii) Suppose $A : \mathcal{U}$ is a type and $B : A \to \mathcal{U}$ a type family such that $B(x)$ is a mere proposition for all $x : A$, then the dependent function type $\prod_{(x:A)} B(x)$ is a mere proposition as well. This is not the case for the Σ-type or coproduct. Section 3.3 below demonstrates how higher inductive types can be used to define a propositionally truncated Σ-type $\|\sum_{(x:A)} B(x)\|$, which is a mere proposition for any $A : \mathcal{U}$ and $B : A \to \mathcal{U}$.

◊

Definition 3.10. A *set* is a type that satisfies the uniqueness of identity proofs property, if $p : x = y$ and $q : x = y$ then $p = q$,

$$\begin{aligned}&\mathrm{isSet} : \mathcal{U} \to \mathcal{U}\\&\mathrm{isSet}(A) :\equiv \prod_{(x,y:A)} \mathrm{isProp}(x = y).\end{aligned}$$

△

Example 3.11.

(i) If $A : \mathcal{U}$ is a type and $B : A \to \mathcal{U}$ a type family where $B(x)$ is a set for all $x : A$, then the dependent function type $\prod_{(x:A)} B(x)$ is a set.

(ii) Let $A : \mathcal{U}$ be a type and $B : A \to \mathcal{U}$ a type family. If A is a set and $B(x)$ is a set for all $x : A$, then the Σ-type $\sum_{(x:A)} B(x)$ is a set. A similar statement holds for coproducts.

◊

Definition 3.12. Let $f : A \to B$ be a function and $x, y : A$ inhabitants. Define

$$\begin{aligned} &\mathrm{ap}_f : x = y \to f(x) = f(y) \\ &\mathrm{ap}_f(p) :\equiv \mathrm{ind}_{=_A}(C, c, x, y, p) \end{aligned}$$

where

$$\begin{aligned} C &: \prod_{(x,y:A)} \big(x = y \to \mathcal{U}\big), & C(x, y, q) &:\equiv f(x) = f(y) \\ c &: \prod_{(x:A)} \big(f(x) = f(x)\big), & c(x) &:\equiv \mathrm{refl}_{f(x)} \end{aligned}$$

△

Definition 3.13. Given a type family $P : A \to \mathcal{U}$ and a path $p : x = y$, where $x, y : A$. Then there is a function

$$\begin{aligned} &\mathrm{transport}(P, p, -) : P(x) \to P(y) \\ &\mathrm{transport}(P, p, -) :\equiv \mathrm{ind}_{=_A}(C, c, x, y, p). \end{aligned}$$

where

$$\begin{aligned} C &: \prod_{(x,y:A)} \big(x = y \to \mathcal{U}\big), & C(x, y, q) &:\equiv P(x) \to P(y) \\ c &: \prod_{(x:A)} \big(P(x) \to P(x)\big), & c(x)(h) &:\equiv h \end{aligned}$$

△

Definition 3.14. A function $f : A \to B$ is an *equivalence* iff there exist functions $g, h : B \to A$ such that $f(g(x)) = x$ for all $x : B$ and $h(f(x)) = x$ for all $x : A$,

$$\mathrm{isequiv}(f) :\equiv \Big(\sum_{(g:B\to A)} \prod_{(x:B)} \big(f(g(x)) = x\big)\Big) \times \Big(\sum_{(h:B\to A)} \prod_{(x:A)} \big(h(f(x)) = x\big)\Big).$$

For $A, B : \mathcal{U}$ types, we define

$$(A \simeq B) :\equiv \sum_{(f:A\to B)} \mathrm{isequiv}(f).$$

When $A \simeq B$ then we say A and B are *equivalent.* △

Remark 3.15. Given types $A, B : \mathcal{U}$ there is a function idtoequiv : $A = B \to A \simeq B$. The *univalence axiom* states that this function is an equivalence. ♢

Axiom 3.16 (Univalence axiom)**.** The function idtoequiv : $A = B \to A \simeq B$ is an equivalence, isequiv(idtoequiv). □

Remark 3.17. So equality is equivalent to equivalence, $(A = B) \simeq (A \simeq B)$. ♢

The univalence axiom implies function extensionality:

Theorem 3.18. There is an equivalence

$$(f = g) \simeq \Big(\prod_{(x:A)} \big(f(x) = g(x)\big)\Big), \qquad \text{for all } f, g : A \to B.$$

□

3.3 Higher inductive types

Higher inductive types are inductive types generated by constructors of inhabitants of the type, paths in the type and higher paths. This section introduces two higher inductive types that we will use in part II. Chapter 6 in the HoTT book [11] contains more information on higher inductive types.

Definition 3.19 (Propositional truncation)**.** Let A be any type. The propositional truncation $\|A\|$ of A is the higher inductive type with generating constructors:

(i) a function $|-| : A \to \|A\|$,

(ii) for all $x, y : \|A\|$, there is a path $\rho_{x,y} : x = y$.

There is an associated recursion principle. Given a type B and

- a function $g : A \to B$,
- for all $x, y : B$ there is a path $p_{x,y} : x = y$.

Then there is a function $f : \|A\| \to B$ such that $f(|a|) \equiv g(a)$ for all $a : A$. △

The propositional truncation type has an induction principle as well, but the recursion principle for propositional truncation implies the induction principle.

Example 3.20. The constructor (ii) of the propositional truncation type says that $\|A\|$ is a proposition. Using propositional truncation we have a mere proposition $\|\sum_{(x:A)} P(x)\|$ for any $P : A \to \mathcal{U}$. If there is a term $t : \|\sum_{(x:A)} P(x)\|$ and $B : \mathcal{U}$ is a mere proposition, by the recursion principle, we may assume an inhabitant $a : \sum_{(x:A)} P(x)$ to prove B. ◊

Definition 3.21. Let $f : A \to B$ be a function. We say that f is *surjective* iff

$$\prod_{(b:B)} \|\sum_{(a:A)} (f(a) = b)\|.$$

△

Remark 3.22. The above definition of surjective is a mere proposition. This would not generally be the case if we omitted the propositional truncation in the definition. ◊

Definition 3.23 (Set-quotient)**.** Let A be a type and $R : A \to A \to \mathcal{U}$ a family of mere propositions, such that $R(x, y)$ is a mere proposition for all $x, y : A$. The *set-quotient* A/R is the higher inductive type generated by the constructors:

(i) a function $q : A \to A/R$;

(ii) for $a, b : A$ such that $R(a, b)$, there is a path $q(a) = q(b)$;

(iii) if $x, y : A/R$ and $r, s : x = y$ then $r = s$. △

The set-quotient has a recursion principle and an induction principle, but we will not need the details. The constructor (i) of the set-quotient gives a quotient map $q : A \to A/R$. The constructor (ii) says that elements $a, b : A$ for which $R(a, b)$ holds are identified in A/R. Constructor (iii) implies that A/R is a set.

Part II
Universal algebra in HoTT

This part presents the universal algebra development for the Coq HoTT library. The formalization can be found at `https://github.com/andreaslyn/hott-classes`. The formalization contains proofs of all the lemmas and theorems presented below. The start of the formalization is part of a Coq project which was supervised by B. Spitters. That project is attached as Appendix A. It contains a proof of Theorem 5.5 below.

In section 7, category theory is used as a tool to verify our definitions. For example, we want a binary product of two algebras to be a binary product in the precategory of algebras and homomorphisms.

From hereon we switch to a pseudo code notation close to the Coq UTF-8 syntax. This makes it easier to relate the text to the formalization. The notation `x ≡ y` will denote `x` is judgmentally equal to `y` and `x = y` is the path type.

4 Algebra

This section gives the main definitions in the universal algebra development. They are explained in more detail in appendix A. The definitions are similar to those in Section 2, but they are homotopy type theoretic in this section. Before defining signature and algebra we will introduce a non-empty list datatype.

Definition 4.1 (`ne_list`)**.** *Non-empty list* is defined by

```
Inductive ne_list (T : Type) : Type :≡
  | one : T → ne_list T
  | cons : T → ne_list T → ne_list T.
Arguments one {T}.
Arguments cons {T}.
```
△

Notation 4.2. The `Arguments one {T}` statement above means that the `T:Type` argument to `one` should be left implicit. Hence `one : (∏ {T:Type}, T → ne_list T)` where curly braces in the type indicate implicit arguments. △

Definition 4.3. We will use the notation:

```
Global Notation "[: x :]" :≡ (one x) : ne_list_scope.
Global Notation "[: x ; .. ; y ; z :]"
    :≡ (cons x .. (cons y (one z)) ..) : ne_list_scope.
Global Infix ":::"
    :≡ cons (at level 60, right associativity) : ne_list_scope.
```
△

The non-empty list is used to define the function symbol type of function symbols.

Definition 4.4 (`Signature`)**.** A *signature* is defined by

```
Record Signature : Type :≡ BuildSignature
  { Sort : Type
  ; Symbol : Type
  ; symbol_types : Symbol → ne_list Sort }.
Definition SymbolType (σ : Signature) :≡ ne_list (Sort σ).
```

```
Global Coercion symbol_types : Signature >-> Funclass.
```
△

Notation 4.5. The above `Global Coercion` allows for using a signature `σ : Signature` as a function `σ u ≡ symbol_types σ u`, for all function symbols `u : Symbol σ`. △

The next definition is used to convert `σ u ≡ symbol_types σ u` into the type of the algebra operation corresponding to `u`.

Definition 4.6. The `Operation` function has type

```
Operation : ∏ {σ : Signature}, (Sort σ → Type) → SymbolType σ → Type.
```

For `A : Sort σ → Type` and `w : SymbolType σ` a symbol type, it is defined by

$$\texttt{Operation A w} :\equiv \texttt{A } \texttt{s}_1 \to \texttt{A } \texttt{s}_2 \to \cdots \to \texttt{A } \texttt{s}_n \to \texttt{A t}$$

where $\texttt{w} \equiv \texttt{[:}\texttt{s}_1\texttt{; }\texttt{s}_2\texttt{; ...; }\texttt{s}_n\texttt{; t:]}$ and $\texttt{s}_1\ \texttt{s}_2 \cdots \texttt{s}_n\ \texttt{t : Sort}\ \sigma$ are all sorts. △

Definition 4.7 (`Algebra`)**.** The type of (wild) *algebras* is defined by

```
Record Algebra {σ : Signature} : Type :≡ BuildAlgebra
  { carriers : Sort σ → Type
  ; operations : ∏ (u : Symbol σ), Operation carriers (σ u) }.
Arguments Algebra : clear implicits.
```

We also introduce an implicit coercion and notation:

```
Global Coercion carriers : Algebra >-> Funclass.
Global Notation "u ^^ A" :≡ (operations A u) (at level 60, no associativity).
```
△

Notation 4.8. The `Arguments Algebra : clear implicits` notation means that the `σ : Signature` argument to `Algebra` should not be implicit. Otherwise it would be implicit because it was given inside curly braces. The `σ : Signature` argument is still implicit for `carriers`, etc. △

An algebra `A : Algebra σ` for a signature `σ` consists of a type `A s ≡ carriers A s` for each sort `s : Sort σ`, and an *operation* `u^^A ≡ operations A u : Operation A (σ u)` for each function symbol `u : Symbol σ`.

The above (wild) algebra definition allows for carriers being arbitrary types. We will mainly be concerned with set-level algebras where the carriers are sets.

Definition 4.9 (`SetAlgebra`)**.** An algebra `A` for a signature `σ` satisfies `IsHSetAlgebra A` when `A s` is a set for all `s : Sort σ`. The type of *set-level algebras* is

```
Record SetAlgebra {σ : Signature} : Type :≡ BuildSetAlgebra
  { algebra_setalgebra : Algebra σ
  ; is_hset_algebra_setalgebra : IsHSetAlgebra algebra_setalgebra }.
Arguments SetAlgebra : clear implicits.
Global Coercion algebra_setalgebra : SetAlgebra >-> Algebra.
```
△

5 Homomorphism and isomorphism

In this section we let `A B: Algebra` σ denote two algebras for a signature σ`:Signature`.

Definition 5.1 (`Homomorphism`)**.** Let `f : (`$\prod$ `(s : Sort` σ`), A s` $\to$ `B s)` be a family of functions. Suppose α `: Operation A w` and β `: Operation B w` are operations of types given by `w`, see Definition 4.6. We define `OpPreserving f` α β `: Type` to be the type:

For all x_1 `: A` s_1, x_2 `: A` s_2, …, x_n `: A` s_n,

`f t (`α x_1 x_2 $\cdots$ x_n`) =` β `(f` s_1 x_1`) (f` s_2 x_2`)` $\cdots$ `(f` s_n x_n`)`

where `[:`s_1`;` s_2`; …;` s_n`; t:]` $\equiv$ σ `u` is the symbol type of `u`.

The type of algebra *homomorphisms* is

```
Record Homomorphism {σ} {A B : Algebra σ} : Type
  :≡ BuildHomomorphism
      { def_hom : ∏ (s : Sort σ), A s → B s
      ; is_hom : ∏ (u : Symbol σ) OpPreserving def_hom (u^^A) (u^^B) }.
Arguments Homomorphism {σ}.
Arguments BuildHomomorphism {σ} {A B : Algebra σ} def_hom {is_hom}.
Global Coercion def_hom : Homomorphism >-> Funclass.
```
△

Definition 5.2 (`IsIsomorphism`)**.** For `f : Homomorphism A B`, then `IsIsomorphism f` is defined to be the type:

For all `s : Sort` σ, `f s` is an equivalence.

We say that `f` is an *isomorphism* if `IsIsomorphism f` holds.

Write `A` $\cong$ `B` for the type of homomorphisms `f : Homomorphism A B` satisfying `Is-Isomorphism f`. We say `A` and `B` are *isomorphic* if `A` $\cong$ `B` is inhabited. △

Remark 5.3. When `A` is a set-level algebra, then this definition of isomorphism is equivalent with `f s` being both surjective and injective for all `s : Sort` σ. By surjective we mean Definition 3.21 and by injective we mean

`∏ (x y : A s), f s x = f s y → x = y.`

So this is similar to the set theoretic definition of isomorphism, Definition 2.5. ◊

Lemma 5.4.

(i) There is an *identity* homomorphism `hom_id :Homomorphism A A` satisfying

`hom_id (s : Sort σ) (x : A s) ≡ x`

The identity homomorphism is an isomorphism `IsIsomorphism hom_id`.

(ii) Suppose `f : Homomorphism A B` and `IsIsomorphism f`. Equivalences have inverse functions, so there is a family of inverse functions

λ `(s : Sort σ), (f s)`$^{-1}$.

There is an *inverse* homomorphism `hom_inv : Homomorphism B A` satisfying

`hom_inv (s : Sort σ) ≡ (f s)`$^{-1}$.

This homomorphism is an isomorphism `IsIsomorphism hom_inv`.

(iii) With `g : Homomorphism B C` and `f : Homomorphism A B` there is a *composition* homomorphism `hom_compose : Homomorphism A C` satisfying

`hom_compose (s : Sort σ) ≡ g s ∘ f s.`

If both `g` and `f` are isomorphisms then `hom_compose` is an isomorphism. □

Isomorphisms have an important property in HoTT:

Theorem 5.5 (`id_isomorphic`)**.** If `A B : Algebra` σ are isomorphic algebras for a signature σ, then `A = B`.

Proof. Follows from Theorem 5.6 in appendix A. □

6 Category of algebras

Definition 6.1. In HoTT (Section 9.1 in the HoTT book) a *precategory* `C : PreCategory` is defined by

- A type $\mathtt{C}_0$ of *objects*.
- Given two objects `a b :` $\mathtt{C}_0$, there is a set of morphisms `hom(a,b)`.
- For each `a :` $\mathtt{C}_0$, an identity morphism $\mathtt{1}_a$ `: hom(a,a)`.
- For `a b c :` $\mathtt{C}_0$, a composition function $\circ$ `: hom(a,b)` $\to$ `hom(b,c)` $\to$ `hom(a,c)`.
- For all `a b :` $\mathtt{C}_0$ and `f : hom(a,b)`, witnesses of `f` $\circ$ $\mathtt{1}_a$ `= f` and $\mathtt{1}_b$ $\circ$ `f = f`.
- A proof that composition is associative, `h` $\circ$ `(g` $\circ$ `f) = (h` $\circ$ `g)` $\circ$ `f`. △

Lemma 6.2 (`precategory_algebra`)**.** Given any signature σ `: Signature` there is a precategory `CatAlg` σ `: PreCategory` of algebras for σ.

- The type of objects is the type of set-level algebras `SetAlgebra` σ.
- Given `A B : SetAlgebra` σ, the set of morphisms `hom(A,B)` is `Homomorphism A B`.
- For `A : SetAlgebra` σ, the identity morphism is the identity homomorphism.
- For `A B C : SetAlgebra` σ, composition is homomorphism composition. □

Remark 6.3. A wild category is a precategory without the restriction that the type of morphisms `hom(a,b)` between objects `a,b` are sets. There is a wild category of algebras `Algebra` σ and homomorphisms, but we will not be concerned with this wild category here. ◊

Definition 6.4. Let `C : PreCategory` be a precategory and `f : hom(a,b)` some morphism in `C`. Then `IsCatIsomorphism f` holds iff there is a morphism `g : hom(b,a)` such that
`g` $\circ$ `f =` $\mathtt{1}_a$ and `f` $\circ$ `g =` $\mathtt{1}_b$.

We define `CatIsomorphic a b` :≡ $\sum$ `f : hom(a,b), IsCatIsomorphism f`, and say `a` and `b` are isomorphic when `CatIsomorphic a b` is inhabited. △

Lemma 6.5 (`isequiv_catiso_to_uaiso`)**.** For `A B : SetAlgebra` σ set-level algebras for a signature σ, there is a equivalence

$$\mathtt{CatIsomorphic\ A\ B} \simeq \mathtt{(A} \cong \mathtt{B)}.$$ □

Definition 6.6. Given any precategory `C : PreCategory` and objects `a b :` $\mathtt{C}_0$, there is a function `idtoiso : (a = b)` $\to$ `CatIsomorphic a b`, defined by path induction and the identity morphism, which is an isomorphism. △

Definition 6.7. A precategory `C : PreCategory` is said to be a univalent category if `idtoiso` is an equivalence for all `a b :` $\mathtt{C}_0$. △

Theorem 6.8 (`category_algebra`)**.** The precategory of algebras `CatAlg` σ is a univalent category.

Proof. See `category_algebra` in file `theory/ua_category.v` of the formalization. □

7 Algebra limits

7.1 Subalgebra

Definition 7.1 (`IsSubalgebraPredicate`)**.** Let `A : Algebra` σ be an algebra for a signature σ `: Signature` and suppose `P : (∏ (s : Sort` σ`), A s → Type)` such that `P s x` is a mere proposition for all `s` and `x`. Assume moreover that there is a term

$$\begin{aligned}&\Theta : \textstyle\prod\ (\mathtt{x}_1 : \mathtt{A}\ \mathtt{s}_1)\ (\mathtt{x}_2 : \mathtt{A}\ \mathtt{s}_2) \cdots (\mathtt{x}_n : \mathtt{A}\ \mathtt{s}_n),\\ &\quad \mathtt{P}\ \mathtt{s}_1\ \mathtt{x}_1 \to \mathtt{P}\ \mathtt{s}_2\ \mathtt{x}_2 \to \cdots \to \mathtt{P}\ \mathtt{s}_n\ \mathtt{x}_n \to \mathtt{P}\ \mathtt{t}\ ((\mathtt{u\hat{}\hat{}A})\ \mathtt{x}_1\ \mathtt{x}_2 \cdots \mathtt{x}_n)\end{aligned}$$

for all function symbols `u : Symbol` σ, where `[:`$\mathtt{s}_1$`;` $\mathtt{s}_2$`; ...;` $\mathtt{s}_n$`; t:]` $\equiv \sigma$ `u` is the symbol type of `u`. Then we refer to `P` as a *subalgebra predicate* for `A`. △

Definition 7.2 (`Subalgebra`)**.** Let σ `: Signature` and `A : Algebra` σ and suppose `P : (∏ (s : Sort` σ`), A s → Type)` is a subalgebra predicate for `A`. Then there is a *subalgebra* `A&P : Algebra` σ of `A`. The `carriers` of the subalgebra `A&P` are

$$\mathtt{(A\&P)}\ (\mathtt{s} : \mathtt{Sort}\ \sigma) \equiv \textstyle\sum\ \mathtt{x},\ \mathtt{P}\ \mathtt{s}\ \mathtt{x}$$

For each `u : Symbol` σ, the operation `u^^(A&P) : Operation (A&P) (`σ `u)` satisfies

$$\begin{aligned}&(\mathtt{u\hat{}\hat{}(A\&P)})\ (\mathtt{x}_1;\ \mathtt{p}_1)\ (\mathtt{x}_2;\ \mathtt{p}_2) \cdots (\mathtt{x}_n;\ \mathtt{p}_n)\\ &\quad \equiv ((\mathtt{u\hat{}\hat{}A})\ \mathtt{x}_1\ \mathtt{x}_2 \cdots \mathtt{x}_n\ ;\ \Theta\ \mathtt{x}_1\ \mathtt{x}_2 \cdots \mathtt{x}_n\ \mathtt{p}_1\ \mathtt{p}_2 \cdots \mathtt{p}_n)\end{aligned}$$

where `[:`$\mathtt{s}_1$`;` $\mathtt{s}_2$`; ...;` $\mathtt{s}_n$`; t:]` $\equiv \sigma$ `u` is the symbol type of `u` and `(_ ; _)` is notation for the Σ-type constructor, so that `(`$\mathtt{x}_i$`;` $\mathtt{p}_i$`) : (A&P)` $\mathtt{s}_i$. △

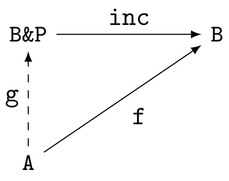
Remark 7.3. We think of the subalgebra carriers `(A&P) s : (∑ x, P s x)` as a subtype of `A s`, for each `s : Sort` σ. ◊

Lemma 7.4 (`hom_inc_subalgebra`)**.** Let σ `: Signature` and let `P : (∏ (s:Sort` σ`), A s → Type)` be a subalgebra predicate for an algebra `A : Algebra` σ. There is an inclusion homomorphism `inc : Homomorphism (A&P) A`,

$$\mathtt{inc}\ (\mathtt{s} : \mathtt{Sort}\ \sigma)\ ((\mathtt{x};\ \mathtt{p}) : (\mathtt{A\&P})\ \mathtt{s}) \equiv \mathtt{x}.$$

The function `inc s : (A&P) s → A s` is an embedding for all `s : Sort` σ. Note that embedding is defined in Section 4.6 of the HoTT book, it is equivalent with injection when `A` is a set-level algebra. □

The following lemma shows that the subalgebra together with the above inclusion homomorphism behaves in the expected way for set-level algebras `A B : SetAlgebra` σ. It says that for any subalgebra predicate `P : (∏ s, B s → Type)` and homomorphism `f : Homomorphism A B`, such that `P s (f s x)` holds for all `s : Sort` σ and `x : A s`, there exists a unique homomorphism `g : Homomorphism A (B&P)` making the following diagram commute:



$$\begin{tikzcd}\mathtt{B\&P} \arrow[r, "\mathtt{inc}"] & \mathtt{B} \\ \mathtt{A} \arrow[u, dashed, "\mathtt{g}"] \arrow[ur, "\mathtt{f}"'] & \end{tikzcd}$$

Lemma 7.5 (`ump_subalgebra`)**.** Suppose `A : Algebra` σ and `B : SetAlgebra` σ for a signature σ and `P : (`$\prod$` s, B s` $\to$ `Type)` is a subalgebra predicate. There is an equivalence

`Homomorphism A (B&P)` $\simeq$ `(`$\sum$` (f : Homomorphism A B), `$\prod$` s x, P s (f s x))`

induced by postcomposition with the inclusion homomorphism

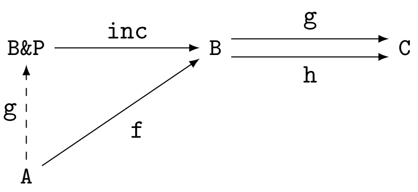
`inc : Homomorphism (B&P) B`

from lemma 7.4. □

Remark 7.6. In category theoretic terms, let `B C : SetAlgebra` σ be objects in the category of algebras `CatAlg` σ, and suppose `g h : Homomorphism B C` are morphisms. There is a subalgebra predicate `P : (`$\prod$` s, B s` $\to$ `Type)` satisfying

`P s y` $\equiv$ `(g s y = h s y).`

Given a morphism `f : Homomorphism A B` where `hom_compose g f = hom_compose h f`, then `P s (f s x)` holds for all `s : Sort` σ and `x : A s`. So by the preceding lemma we have a commutative diagram:



$$\begin{tikzcd}
\mathtt{B\&P} \arrow[r, "\mathtt{inc}"] & \mathtt{B} \arrow[r, shift left, "\mathtt{g}"] \arrow[r, shift right, "\mathtt{h}"'] & \mathtt{C} \\
\mathtt{A} \arrow[u, dashed, "\mathtt{g}"] \arrow[ru, "\mathtt{f}"'] & &
\end{tikzcd}$$

The above subalgebra `B&P` is summit of a limit cone over a diagram of type

$$\begin{tikzcd}
\bullet \arrow[r, shift left] \arrow[r, shift right] & \bullet
\end{tikzcd}$$

Such a limit cone is called an equaliser. This shows that the category of algebras for any signature σ `: Signature` has all equalisers. ◊

7.2 Product algebra

Definition 7.7 (`ProdAlgebra`)**.** Let `A : I` $\to$ `Algebra` σ be a family of algebras for some σ `: Signature` and `I : Type` an index type. The *product algebra* `Prod A : Algebra` σ has carriers

`Prod A (s : Sort` σ`)` $\equiv$ $\prod$ `(i:I), A i s.`

For all `u : Symbol` σ, the operation `u^^(Prod A) : Operation (Prod A)(`σ` u)` satisfies

`(u^^(Prod A)) (`p_1` : Prod A `s_1`) (`p_2` : Prod A `s_2`)` $\cdots$ `(`p_n` : Prod A `s_n`)`
$\equiv$ λ `(i:I), (u ^^ A i) (`p_1` i) (`p_2` i)` $\cdots$ `(`p_n` i)`

with `[:`s_1`; `s_2`; ...; `s_n`; t:]` $\equiv$ σ `u` the symbol type of `u`. △

Lemma 7.8 (`hom_projection_prod_algebra`)**.** Let `Prod A :Algebra` σ be the product algebra of a family of algebras `A : I` $\to$ `Algebra` σ. For each `i:I` there is a projection homomorphism π `i : Homomorphism (ProdAlgebra I A) (A i)`,

π `(i:I) (s : Sort` σ`) (p : (Prod A) s)` $\equiv$ `p i.` □

Remark 7.9. Suppose that `Prod A : Algebra` σ is the product algebra of the set-level algebras `A : Bool` $\to$ `SetAlgebra` σ. Given homomorphisms `f : Homomorphism B (A true)` and `g : Homomorphism B (A false)`, with `B : Algebra` σ, there is a homomorphism `h : Homomorphism B (Prod A)` satisfying

$$\texttt{h (s : Sort } \sigma\texttt{) (i:Bool)} \equiv \texttt{if i then f x else g x}$$

This homomorphism `h` is the unique homomorphism making the following diagram commute.

$$\begin{tikzcd}
& \texttt{Prod A} \arrow[ld, "\pi\ \texttt{true}"'] \arrow[rd, "\pi\ \texttt{false}"] & \\
\texttt{A true} & & \texttt{A false} \\
& B \arrow[lu, "\texttt{f}"] \arrow[ru, "\texttt{g}"'] \arrow[uu, dashed, "\texttt{h}"] &
\end{tikzcd}$$

From the above diagram we see that the category of algebras for any signature σ has all binary products `Prod A`, for any `A : Bool` $\to$ `SetAlgebra` σ. More generally, let `I:Type` be a type assigned discrete category structure. Then a family of set-level algebras `A : I` $\to$ `SetAlgebra` σ is a diagram, and `Prod A` is the limit of the diagram. This limit is called a product, so the category of algebras for σ has all products. This is stated in the following lemma. ◊

Lemma 7.10 (`ump_prod_algebra`)**.** Let `A : I` $\to$ `SetAlgebra` σ be a family of set-level algebras and `I:Type` an indexing type. There is an equivalence

$$\texttt{Homomorphism B (Prod A)} \simeq (\textstyle\prod \texttt{ i, Homomorphism B (A i))}$$

induced by mapping `f : Homomorphism B (Prod A)` to the family of homomorphisms

$$\lambda\ \texttt{(i:I), hom\_compose (}\pi\texttt{ i) f}$$

where π `i : Homomorphism (Prod A) (A i)` is the `i`th projection homomorphism. □

7.3 Quotient algebra

Notation 7.11. For `R : relation X` a relation on some type `X`, let

$$\texttt{is\_mere\_relation X R} :\equiv \textstyle\prod \texttt{ (x y : X), IsHProp (R x y)}$$

where `IsHProp` is isProp from Definition 3.8. △

Definition 7.12 (`IsCongruence`)**.** Let `A : Algebra` σ be an algebra for a signature σ. A family of relations Φ `: (`$\prod$ `(s : Sort` σ`), relation (A s))` satisfies `OpsCompatible A` Φ iff for all function symbols `u : Symbol` σ,

$$\Phi\ s_1\ x_1\ y_1 * \Phi\ s_2\ x_2\ y_2 * \cdots * \Phi\ s_n\ x_n\ y_n$$

implies

$$\Phi\ t\ \texttt{((u\^{}\^{}A)}\ x_1\ x_2\ \cdots\ x_n\texttt{)}\ \texttt{((u\^{}\^{}A)}\ y_1\ y_2\ \cdots\ y_n\texttt{)}$$

where `[:`s_1`;` s_2`; ...;` s_n`; t:]` $\equiv \sigma$ `u` is the symbol type, and x_i `: A` s_i and y_i `: A` s_i.

A *congruence* is a family of mere equivalence relations satisfying `OpsCompatible`,

```
Class IsCongruence {σ} (A : Algebra σ) (Φ : ∏ s, relation (A s)) :≡
 { is_mere_relation_cong : ∏ (s : Sort σ), is_mere_relation (A s) (Φ s)
 ; equiv_rel_cong : ∏ (s : Sort σ), EquivRel (Φ s)
 ; ops_compatible_cong : OpsCompatible A Φ }.
```
△

Definition 7.13. Suppose `R : relation X` is a relation on some type `X:Type`. In the Coq HoTT library `quotient R` is the name for the set-quotient from Definition 3.23. △

Definition 7.14 (`QuotientAlgebra`)**.** Let σ `: Signature` be a signature. Given an algebra `A : Algebra` σ and a congruence Φ `: (`$\prod s$`, relation (A s))` the *quotient algebra* `A/Φ` is a set-level algebra with `carriers`

$$\texttt{(A/}\Phi\texttt{) (s : Sort } \sigma\texttt{)} \equiv \texttt{quotient (}\Phi\texttt{ s)}$$

The `operations` of the quotient algebra `A/Φ` satisfy

$$\texttt{(u\^{}\^{}(A/}\Phi\texttt{)) [x}_1\texttt{] [x}_2\texttt{]} \cdots \texttt{[x}_n\texttt{] = [(u\^{}\^{}A) x}_1\texttt{ x}_2 \cdots \texttt{x}_n\texttt{]}$$

for all `u : Symbol` σ and x_i `: A` s_i, where `[:`s_1`;` s_2`; ...;` s_n`; t:]` $\equiv \sigma$ `u` is the symbol type of `u` and `[`x_i`] : quotient (Φ` s_i`)` is the equivalence class of x_i, Definition 3.23(i). △

Lemma 7.15 (`hom_quotient`)**.** For any algebra `A : Algebra` σ and congruence Φ `: (`$\prod s$`, relation (A s))` there is a homomorphism `hom_quotient :Homomorphism A (A/Φ)` satisfying

$$\texttt{hom\_quotient (s : Sort } \sigma\texttt{) (x : A s)} \equiv \texttt{[x]}$$

where `[x] :(A/Φ) s` is the equivalence class of `x`. This homomorphism is surjective. □

Remark 7.16. The quotient algebra `A/Φ` has the following universal property. Let `B : SetAlgebra` σ and `f : Homomorphism A B` a homomorphism respecting the congruence Φ in the sense that `Φ s x y` implies `f s x = f s y`, for all `s : Sort` σ and `x y : A s`. There is a unique homomorphism `k : Homomorphism (A/Φ) B` such that

```
hom_compose k hom_quotient = f
```

as indicated in the following diagram:

$$\begin{tikzcd}
\texttt{A} \arrow[r, "\texttt{hom\_quotient}"] \arrow[dr, "\texttt{f}"'] & \texttt{A/}\Phi \arrow[d, dashed, "\texttt{k}"] \\
 & \texttt{B}
\end{tikzcd}$$

◊

Lemma 7.17. Let Φ `: (`$\prod s$`, relation (A s))` be a congruence on an algebra `A : Algebra` σ. Let `B : SetAlgebra` σ be a set-level algebra. There is an equivalence

$$\begin{aligned}&\texttt{Homomorphism (A/}\Phi\texttt{) B}\\ &\quad\simeq \texttt{(}\textstyle\sum\texttt{ (f : Homomorphism A B), }\prod\texttt{ s x y, }\Phi\texttt{ s x y} \to \texttt{f s x = f s y)}\end{aligned}$$

induced by precomposition with `hom_quotient : Homomorphism A (A/Φ)`. □

Remark 7.18. For the categorical point of view, suppose `g h : Homomorphism A B` are homomorphisms between `A B : SetAlgebra` σ. There is a congruence `Φ : (∏s, relation (A s))` satisfying

```
Φ (s : Sort σ) (x : B s) (y : B s)
  ≡ ∏ (Ψ : ∏s, relation (A s)),
      IsCongruence A Ψ → (∏ t (a : A t), Ψ t (g t a) (h t a)) → Ψ s x y
```

This is the least congruence where `Φ s (g a) (h a)` for all `s : Sort` σ and `a : A s`. Let `f : Homomorphism B C` such that `hom_compose f g = hom_compose f h`. There is another congruence `Ψ : (∏s, relation (A s))` where

```
Ψ (s : Sort σ) (x : B s) (y : B s) ≡ (f s x = f s y).
```

It follows from `hom_compose f g = hom_compose f h` that `Ψ s (g a) (h a)` holds for all `a : A s`. Thus `Φ s x y` implies `f s x = f s y`. By Lemma 7.17 there exists a unique homomorphism `k : Homomorphism (B/Φ) C` making the following diagram commute.

$$\begin{tikzcd}
\mathtt{A} \arrow[r, shift left, "\mathtt{g}"] \arrow[r, shift right, "\mathtt{h}"'] & \mathtt{B} \arrow[rr, "\mathtt{hom\_quotient}"] \arrow[rrd, "\mathtt{f}"'] & & \mathtt{B}/\Phi \arrow[d, dashed, "\mathtt{k}"] \\
& & & \mathtt{C}
\end{tikzcd}$$

The above quotient algebra `B/Φ` is nadir of a colimit over a diagram of type

$$\begin{tikzcd}
\bullet \arrow[r, shift left] \arrow[r, shift right] & \bullet
\end{tikzcd}$$

This colimit is called a coequaliser, so the category of algebras for any signature σ has all coequalisers. ♢

8 Isomorphism theorems

This section presents homotopy type theoretic versions of the isomorphism theorems. Section 2.2 introduced the set theoretic isomorphism theorems. The isomorphism theorems in universal algebra are generalisations of the fundamental isomorphism theorems known from group theory and ring theory. Proofs of the theorems in this section can be found in the formalization, https://github.com/andreaslyn/hott-classes, in the `theory` directory. Before stating the theorems we will need a couple of definitions.

Definition 8.1. The term `hexists : (∏ {X:Type}, (X → Type) → Type)` is the Coq HoTT library name for the propositional truncation (Definition 3.19 above) of the Σ-type,

```
hexists P :≡ ‖∑ (x:X), P x‖.
```
△

Definition 8.2. Let `X:Type` be a type and `R : relation X` an equivalence relation where `R x y` is a mere proposition for all `x y : X`. Then there is a mere proposition `in_class : quotient R → X → Type` such that `in_class C x` holds if and only if `x:X` is in the equivalence class `C : quotient R`. △

Theorem 8.3 (`hom_first_isomorphism`). Let `A B : SetAlgebra` σ be set-level algebras for a signature σ `: Signature` and let `f : Homomorphism A B` be a homomorphism.

(i) There is a *kernel* congruence Φ `: (`$\prod s$`, relation (A s))` satisfying

```
Φ (s : Sort σ) (x : A s) (y : A s) ≡ (f s x = f s y).
```

(ii) There exists a subalgebra predicate `P : (∏ s, B s → Type)`, such that

```
P s y ≡ hexists (λ x, (f s x) = y).
```

(iii) There is an isomorphism `Homomorphism (A/Φ) (B&P)`.

(iv) This isomorphism induces a path `A/Φ = B&P`. $\square$

The first isomorphism theorem in this section is similar to that of section 2.2, where Φ corresponds to $\ker(f)$ from the first isomorphism theorem in Section 2.2, `B&P` corresponds to the homomorphic image $f(\mathbf{A})$. In HoTT we have the additional part (iv), which follows from Theorem 5.5.

Theorem 8.4 (`hom_second_isomorphism`). Let σ `: Signature` be a signature and `A : Algebra` σ an algebra for σ. Suppose `P : (∏ s, A s → Type)` is a subalgebra predicate for `A` and Φ `: (`$\prod s$`, relation (A s))` is a congruence on `A`. Let `inc : Homomorphism (A&P) A` denote the inclusion homomorphism from Lemma 7.4.

(i) There exists a *trace* congruence Ψ `: (`$\prod s$`, relation ((A&P) s))` where

```
Ψ (s : Sort σ) (x y : (A&P) s) ≡ Φ s (inc s x) (inc s y).
```

(ii) There is a subalgebra predicate `Q : (∏ s, (A/Φ) s → Type)` where

```
Q (s : Sort σ) (x : (A/Φ) s)
  ≡ hexists (λ (y : (A&P) s), in_class x (inc s y)).
```

(iii) There exists an isomorphism `Homomorphism ((A&P)/ Ψ) ((A/Φ) & Q)`.

(iv) Thus there is a path `(A&P) / Ψ = (A/Φ) & Q`. $\square$

Here Ψ corresponds to $\varphi^{\mathbf{B}}$ from the second isomorphism theorem in Section 2.2, and `((A/Φ) & Q)` corresponds to $[\mathbf{B}]^{\varphi}$. In HoTT we have the equality (iv), which we do not have in set theory.

Theorem 8.5 (`hom_third_isomorphism`). Let σ `: Signature` be a signature and `A : Algebra` σ an algebra. Suppose Φ Ψ `: (`$\prod s$`, relation (A s))` are two congruences such that `Ψ s x y` implies `Φ s x y`, for all `s : Sort` σ and `x y : A s`.

(i) There is a congruence Φ/Ψ `: (`$\prod s$`, relation (A/Ψ))` where

```
(Φ/Ψ) t (s : Sort σ) (a b : (A/Ψ) s)
  ≡ ∏ (x y : A s), in_class a x → in_class b y → Φ s x y.
```

(ii) There is an isomorphism `Homomorphism ((A/Ψ) / (Φ/Ψ)) (A/Φ)`.

(iii) So there is a path `(A/Ψ) / (Φ/Ψ) = A/Φ`. $\square$

Here Φ/Ψ corresponds to φ/ϑ from the third isomorphism theorem in Section 2.2. In HoTT we additionally get the path (iii).

9 Conclusions

This work demonstrates that one can develop universal algebra in HoTT without using setoids. We have seen subalgebra, product algebra, quotient algebra, and verified that they have the expected universal properties for set-level algebras.

Higher inductive types were used to define quotient algebra using the set-quotient type. An alternative to using higher inductive types is to define equivalence classes, as in set theory,

```
[a] :≡ ∑ (x:A), R a x
```

where `A:Type` and `a:A` and `R : A → A → Type` is a mere equivalence relation. Then for all `x y : A`,

$$\texttt{R x y} \leftrightarrow \texttt{R y x} \quad \text{iff} \quad \texttt{R x y} \simeq \texttt{R y x} \quad \text{iff} \quad \texttt{R x y = R y x}$$

where the first "iff" follows from `R x y` being a mere proposition, for all `x y : A`, and the last "iff" comes from the univalence axiom. This implies that `[x] = [y]` iff `R x y` holds, and we have an alternative quotient type. Using this quotient type we will need to assume the propositional resizing axiom. Sections 3.5 and 6.10 in the HoTT book [11] elaborates on this.

Towards the end of the report we saw the three isomorphism theorems. An appealing aspect of HoTT is that we obtain equalities from the isomorphism theorems, since isomorphic algebras are equal, Theorem 5.5.

10 Future work

A way to proceed is to define varieties. A variety is a subtype of `Algebra` σ given by equational laws. For example, the type of all groups forms a variety satisfying the group axioms/identities.

In the future we want to use higher inductive types to define free algebras. One can for instance define the infinite cyclic group Z (the free group on one generator) as a higher inductive type with the following constructors:

- A generating element $1 : Z$.
- A function $+ : Z \times Z \to Z$.
- An identity element $0 : Z$.
- An inversion function $- : Z \to Z$.
- For each $x, y, z : Z$, an equality $(x + y) + z = x + (y + z)$.
- For each $x : Z$, equalities $x + 0 = x$ and $0 + x = x$.
- For each $x : Z$, equalities $x + (-x) = 0$ and $(-x) + x = 0$.
- A 0-truncation constructor: for all $x, y : Z$ and $p, q : x = y$, an equality $p = q$.

The 0-truncation constructor states exactly that the higher inductive type Z is a set. The other constructors of equalities are stating the group axioms.

In this development we have mostly considered 0-truncated universal algebra, where the carrier types are sets. In HoTT there is the notion of an n-type, see the HoTT book [11] Section 3.1 for 1-types and Section 7.1 for the more general n-type. A way to continue is to consider what happens in 1-truncated universal algebra, where the carrier types are 1-types. One can also consider what happens in the case where the carrier types are arbitrary types. There is a wild category of such algebras, as stated in Remark 6.3. It would be interesting to study the properties of this wild category.

Appendices

Appendix A Universal algebra homomorphisms and isomorphisms in HoTT

This appendix is a self-contained report for a Coq project on universal algebra homomorphisms and isomorphisms in HoTT.

A.1 Introduction

In this report I present the beginnings of a port of the Math Classes library [10] to the Homotopy Type Theory (HoTT) library [2] for the Coq proof assistant. The Math Classes library is developed by Spitters and van der Weegen as a basis for constructive analysis in Coq. The focus of the development in this report has been on porting the Universal Algebra parts of Math Classes to HoTT. The Coq formalisation of this can be found at `https://github.com/andreaslyn/hott-classes`.

The reader is assumed familiar with HoTT [11] and the Coq HoTT library [2]. Knowledge of universal algebra is not required, but to appreciate the results, some universal algebra background is useful.

Since this is a Coq project I will use a pseudo code notation close to the Coq UTF-8 syntax. The notation `x ≡ y` will denote `x` is judgmentally equal to `y` and `x = y` is the path type.

Section A.2 presents a non-empty list data type used in later sections. Section A.3 defines what is meant by an algebra and other basic notions in universal algebra. This corresponds to the file `interfaces/ua_algebra.v` in the formalisation. Section A.4 introduces homomorphisms and isomorphisms and section A.5 contains a proof of the main theorem in this report:

If there is an isomorphism between two algebras `A` and `B` then `A = B`.

The sections A.4 and A.5 correspond to the file `theory/ua_homomorphism.v` in the formalisation. Apart from a few results, the report is devoted to the proof of the above statement. All preliminary results used in the proof are given in the report or can be found in the HoTT book [11]. Section A.6 concludes and compares the main theorem of this report to a similar theorem by Coquand and Danielsson [6].

A.2 Non-empty List

This section introduces a non-empty list implementation with accompanying notation used in the following sections.

Definition A.1. A non-empty list is defined by

```
Inductive ne_list (T : Type) : Type :≡
  | one : T → ne_list T
  | cons : T → ne_list T → ne_list T.
Arguments one {T}.
Arguments cons {T}.
```

For `ne_lists` we introduce the notation

```
Global Notation "[: x :]" :≡ (one x) : ne_list_scope.
Global Notation "[: x ; .. ; y ; z :]"
    :≡ (cons x .. (cons y (one z)) ..) : ne_list_scope.
Global Infix ":::"
    :≡ cons (at level 60, right associativity) : ne_list_scope.
```
△

The induction principle for the non-empty list is similar to that of the regular list. As an example, suppose `w : ne_list T` is a non-empty list and `P : ne_list T → Type` some predicate. To prove `P w` by induction we consider the base case `w ≡ [:x:]` and show that `P [:x:]` holds. Then, for the inductive step `w ≡ x ::: w'`, we assume `P w'` and show it implies `P (x ::: w')`.

A.3 Universal Algebra

In this section we develop the central definitions in universal algebra and provide a couple of useful results.

Definition A.2. A *signature* is defined by

```
Record Signature : Type :≡ BuildSignature
  { Sort : Type
  ; Symbol : Type
  ; symbol_types : Symbol → ne_list Sort }.
Definition SymbolType (σ : Signature) :≡ ne_list (Sort σ).
```
△

The intuition for this definition is that a signature specifies which operations (functions) an algebra for the signature is expected to provide.

- An algebra for `σ : Signature` provides a type for each *sort* `s : Sort σ`.
- The type `Symbol σ` consists of *function symbols*. For each function symbol `u : Symbol σ`, an algebra for the signature provides a corresponding operation.
- The field `symbol_types σ u` indicates which type the operation corresponding to `u` should to have.

Definition A.3. We introduce the implicit coercion

```
Global Coercion symbol_types : Signature >-> Funclass.
```
△

So with `σ : Signature` and `u : Symbol σ`, then `σ u ≡ symbol_types σ u` definitionally.
The `Operation` function

```
Operation : ∏ {σ : Signature}, (Sort σ → Type) → SymbolType σ → Type
```

is used to convert `σ u ≡ symbol_types σ u` into the type that the corresponding algebra operation to `u` should have.

Definition A.4. For `A : Sort` σ `→ Type` and `w : SymbolType` σ a symbol type,

$$\texttt{Operation A w} :\equiv \texttt{A } s_1 \to \texttt{A } s_2 \to \cdots \to \texttt{A } s_n \to \texttt{A t}$$

when `w` $\equiv$ `[:`s_1`;` s_2`; ...;` s_n`; t:]` for s_1 s_2 $\cdots$ s_n `t : Sort` σ. △

Lemma A.5. If `A s` is an `n`-type for all `s : Sort` σ, then `Operation A w` is an `n`-type for any `w : SymbolType` σ. In particular, if `A s` is a set for all `s`, then `Operation A w` is a set.

Proof. Induction on `w` and Theorem 7.1.9 in the HoTT book [11]. □

Definition A.6. An *algebra* is defined by

```
Record Algebra {σ : Signature} : Type :≡ BuildAlgebra
  { carriers : Sort σ → Type
  ; operations : ∏ (u : Symbol σ), Operation carriers (σ u) }.
Arguments Algebra : clear implicits.
Arguments BuildAlgebra {σ} carriers operations {hset_carriers_algebra}.
```
△

So an algebra `A : Algebra` σ for a signature σ consists of a type `carriers A s` for each sort `s : Sort` σ, and an *operation* `operations A u : Operation (carriers A) (`σ `u)` for each function symbol `u : Symbol` σ.

The following lemma has the same role as the `equality-pair-lemma` by Coquand and Danielsson [6].

Lemma A.7. Given two algebras `A B : Algebra` σ for a signature σ. To find a path `A = B`, it suffices to find paths between the carriers `p : carriers A = carriers B` and the operations `q : p#(operations A) = operations B`, where `p#` is transport along `p`,

```
p#(operations A) ≡ transport (λ C, ∏ u, Operation C (σ u)) p (operations A)
```

Proof. Records are Σ-types, see the `issig` tactic in `Types/Records.v` of the HoTT library. Hence the result follows from Theorem 2.7.2 in the HoTT book. □

Definition A.8.

```
Global Coercion carriers : Algebra >-> Funclass.
Global Notation "u ^^ A" :≡ (operations A u) (at level 60, no associativity)
                          : Algebra_scope.
```
△

Using the above implicit coercion with `A : Algebra` σ and `s : Sort` σ, we have `A s` $\equiv$ `carriers A s` by definition.

A.4 Homomorphisms and isomorphisms

This section defines homomorphism and isomorphism. Then we provide some results about homomorphisms and isomorphisms. In the end some elementary homomorphisms are defined. Throughout the section we let `A B : Algebra` σ denote two algebras for a signature σ `: Signature`.

Definition A.9. Let `f : (∏ (s : Sort σ), A s → B s)` be a family of functions. Suppose `α : Operation A w` and `β : Operation B w` are operations of types given by `w`, see Definition A.4. We define `OpPreserving f α β : Type` to be the type:

For all x_1 `: A` s_1, x_2 `: A` s_2, …, x_n `: A` s_n,

`f t (α` x_1 x_2 ⋯ x_n`) = β (f` s_1 x_1`) (f` s_2 x_2`)` ⋯ `(f` s_n x_n`)`,

where `[:`s_1`;` s_2`; …;` s_n`; t:] ≡ σ u` is the symbol type of `u`.

We define *homomorphism* by

```
Record Homomorphism {σ} {A B : Algebra σ} : Type
  :≡ BuildHomomorphism
     { def_hom : ∏ (s : Sort σ), A s → B s
     ; is_hom : ∏ (u : Symbol σ) OpPreserving def_hom (u^^A) (u^^B) }.
Arguments Homomorphism {σ}.
```

We add an implicit coercion

```
Global Coercion def_hom : Homomorphism >-> Funclass.
```
△

With the above implicit coercion we can apply a homomorphism without using `def_hom` explicitly.

Definition A.10. For `f : Homomorphism A B` a homomorphism, `IsIsomorphism f : Type` is defined as the type:

`∏ (s : Sort σ), IsEquiv f s`

We say that `f` is an *isomorphism* if `IsIsomorphism f` holds. △

Lemma A.11. `IsIsomorphism f` is a mere proposition.

Proof. Since `IsEquiv (f s)` is a mere proposition, Theorem 7.1.9 in HoTT [11] provides the result. □

For the rest of this section we introduce some elementary homomorphisms and isomorphisms. We omit the proofs of `OpPreserving` and `IsIsomorphism`, which can be found in the formalisation.

Lemma A.12. There is an *identity* homomorphism `hom_id : Homomorphism A A` induced from the family of identity functions,

`λ (s : Sort σ) (x : A s), x.`

The identity homomorphism is an isomorphism `IsIsomorphism hom_id`. □

Lemma A.13. Suppose `f : Homomorphism A B` and `IsIsomorphism f`. Equivalences have inverse functions, so there is a family of inverse functions

`λ (s : Sort σ), (f s)`$^{-1}$.

This family of functions gives rise to a homomorphism `hom_inv f :Homomorphism B A`, which is also an isomorphism `IsIsomorphism (hom_inv f)`. This homomorphism is also referred to as the *inverse* homomorphism of `f`. □

Lemma A.14. With `g : Homomorphism B C` and `f : Homomorphism A B` the *composition* homomorphism `hom_compose g f : Homomorphism A C` has family of functions

`λ (s : Sort σ), g s ∘ f s.`

If both `g` and `f` are isomorphisms then `hom_compose g f` is an isomorphism as well. □

A.5 Isomorphism is equality

This section proves the main theorem in this report. If `A B : Algebra` σ are two algebras for a signature σ and there is an isomorphism `Homomorphism A B`, then there exists a path `A = B`.

A.5.1 Preliminary results

We begin with `path_forall_recr_beta` from `Tactics.v` in the HoTT library [2].

Lemma A.15. Let `X : Type` be a type, `F : X → Type` a type family and `P : (∏ x, F x) → F x → Type`. Suppose `a : X` is a point and `f g : (∏ x, F x)` dependent functions. Assume moreover that there exists a homotopy `H : f ∼ g` and a witness `W : P f (f a)`. Then there is a path

```
transport (λ f, P f (f a)) (path_forall f g H) W
  = transport (λ h, P h (g a)) (path_forall f g H)
      (transport (λ y, P f y) (H a) W)
```

where `path_forall f g : f ∼ g → f = g` is function extensionality.

Proof. We will replace occurrences of `H` with `apD10 (path_forall f g H)`, where `apD10` is the HoTT library name for `happly` from the HoTT book. We achieve this by transporting along the path `H = apD10 (path_forall f g H)` which comes from the propositional computation rule in section 2.9 in the HoTT book [11]. By path induction we may assume judgmental equalities `path_forall f g H ≡ 1`$_{\texttt{f}}$ and `f ≡ g`, where `1`$_{\texttt{f}}$ `: f = f` is the identity path. It therefore suffices to show that

```
transport (λ f, P f (f a)) (path_forall f f (apD10 1_f)) W
  = transport (λ h, P h (f a)) (path_forall f f (apD10 1_f))
      (transport (λ y, P f y) (apD10 1_f a) W)
```

By definition `apD10 1`$_{\texttt{f}}$ `≡ (λ (x:X), 1`$_{\texttt{(f x)}}$`)`, so section 2.9 in HoTT [11] provides a path

```
(path_forall f g (apD10 1_f)) = 1_f
```

and `apD10 1`$_{\texttt{f}}$ `a ≡ 1`$_{\texttt{(f a)}}$ definitionally. Using this we get

```
transport (λ f, P f (f a)) (path_forall f f (apD10 1_f)) W
  = transport (λ f, P f (f a)) 1_f W
  ≡ W
```

and

```
transport (λ h, P h (f a)) (path_forall f f (apD10 1_f))
      (transport (λ y, P f y) (apD10 1_f a) W)
  = transport (λ h, P h (f a)) 1_f
      (transport (λ y, P f y) (apD10 1_f a) W)
  ≡ transport (λ y, P f y) (apD10 1_f a) W
  ≡ transport (λ y, P f y) 1_(f a) W
  ≡ W
```

□

Part (i) of the next lemma is `transport_arrow_toconst` from `Types/Arrow.v` in the HoTT library [2]. Path (ii) is `transport_forall_constant` from `Types/Forall.v` in the HoTT library.

Lemma A.16. Let `X Y : Type` be types. Assume there are inhabitants $\texttt{x}_1$ $\texttt{x}_2$ `: X` and a path `p :` $\texttt{x}_1$ `=` $\texttt{x}_2$.

(i) Suppose that `P : X -> Type` and `f : P` $\texttt{x}_1$ `→ Y` and `y : P` $\texttt{x}_2$. Then

```
transport (λ x, P x → Y) p f y = f (p^ # y).
```

where `p^ :` $\texttt{x}_2$ `=` $\texttt{x}_1$ denotes the inverse path and `p^ # y ≡ transport P p^ y` is transport along `p^`.

(ii) Let `y : Y` and `P : X → Y → Type` and `f : (∏ y, C` $\texttt{x}_1$ `y)`. There is a path

```
transport (λ x, ∏ z, P x z) p f y = transport (λ x, P x y) p (f y).
```

Proof. Both part (i) and (ii) follow from path induction. □

Part (i) of the above lemma is a version of equation (2.9.4) in the HoTT book where the codomain of `f` is non-dependent.

The proof of the next Lemma is inspired by the proof of `transport_path_universe_V_-uncurried` from `Types/Universe.v` in the HoTT library [2].

Lemma A.17. Let `X Y Z : Type` be types. If there is an equivalence `f : X ≃ Y` and function `g : X → Z` and a point `y : Y`, then

```
transport (λ (T:Type), T → Z) (path_universe f) g y = g (f^-1 y)
```

where `f`$^{-1}$ `: Y → X` denotes the inverse function and `path_universe f : X = Y` is univalence applied to `f`.

Proof. By concatenating with the path from Lemma A.16(i), it is sufficient to find a path of type

```
g ((path_universe f)^ # y) = g (f^-1 y)
```

where `(path_universe f)^ # y ≡ transport idmap (path_universe f)^ y`. It follows from section 2.10 in HoTT [11] that there is a path `f = transport idmap (path_universe f)`. Using this and path induction on `(path_universe f)`, we may assume `X ≡ Y` definitionally, and we just need to show that

```
g ((path_universe (transport idmap 1_X))^ # y) = g ((transport idmap 1_X)^-1 y)
```

Since `transport idmap 1`$_\texttt{X}$ is the identity function, the right hand side above is equal to `(g y)` judgmentally. The left hand side is equal to `(g y)` propositionally because section 2.10 in the HoTT book gives a path `path_universe (transport idmap 1`$_\texttt{X}$`) = 1`$_\texttt{X}$, so that

```
g ((path_universe (transport idmap 1_X))^ # y) = g (1_X^ # y) ≡ g y
```
□

Given a family of equivalences `f : (∏ i, F i ≃ G i)`, function extensionality composed with univalence gives a path `F = G`. We will need the definitional equality of this:

Lemma A.18.

```
Definition path_equiv_family {I} {F G : I → Type} (f : ∏ i, F i ≃ G i)
  : F = G
  :≡ path_forall F G (λ i, path_universe (f i)).
```
□

A.5.2 Isomorphisms induce paths

In this subsection we prove the main theorem. We let `A B : Algebra σ` be two algebras (Definition A.6) for some signature `σ : Signature` (Definition A.2).

Lemma A.19. Let `w : SymbolType σ` be a symbol type (Definition A.2) Suppose `α : Operation A w` and `β : Operation B w` are operations of types given by `w`, see Definition A.4 and the implicit coercion in definition A.8. Let `f : (∏ (s : Sort σ), A s ≃ B s)` be a family of equivalences between the carrier sets of `A` and `B`. Assume `OpPreserving f α β` (Definition A.9) holds. There is a path between the operations

```
transport (λ C, Operation C w) (path_equiv_family f) α = β.
```

where `path_equiv_family f : carriers A = carriers B` is given in Lemma A.18.

Proof. We proceed by induction on `w`. In case `w ≡ [:t:]`, then `Operation C w ≡ C t`, for any `C : Sort σ → Type`, and we have `OpPreserving f α β ≡ (f t α = β)`. Set

```
P :≡ (λ (C : Sort σ → Type) (X : Type), X)}.
```

Then we get the following chain (see the comments below).

```
transport (λ C, Operation C w) (path_equiv_family f) α
  ≡ transport (λ C, C t) (path_equiv_family f) α
  ≡ transport (λ C, P C (C t)) (path_equiv_family f) α
  = transport (λ C, P C (B t)) (path_equiv_family f)
      (transport (λ X, P A X) (path_universe (f t)) α)
  ≡ transport (λ C, B t) (path_equiv_family f)
      (transport idmap (path_universe (f t)) α)
  = transport idmap (path_universe (f t)) α
  = (f t) α
  = β.
```

The first `=` path comes from Lemma A.15. The second `=` path is from Lemma 2.3.5 in HoTT [11]. The third `=` path is from Section 2.10 in HoTT [11]. The last path is the assumption `OpPreserving f α β ≡ (f t α = β)`.

In case `w ≡ t ::: w'`, then `Operation C w ≡ (C t → Operation C w')`, for any `C : Sort σ → Type`, and we have

```
OpPreserving f α β ≡ ∏ (x : A t), OpPreserving f (α x) (β (f t x)).      (∗)
```

It suffices to find a path in `Operation B w'` of type

```
transport (λ C, Operation C w) (path_equiv_family f) α y = β y
```

where `y : B t` is some inhabitant. By (∗) above with `(f⁻¹ y)` we have

```
OpPreserving f (α (f⁻¹ y)) (β y)                                        (∗∗)
```

because

```
OpPreserving f (α (f⁻¹ y)) (β (f t (f⁻¹ y))) = OpPreserving f (α (f⁻¹ y)) (β y).
```

Write

```
P₁ :≡ λ (C : Sort σ → Type) (X : Type), X → Operation C w'
P₂ :≡ λ (C : Sort σ → Type) (z : B t), Operation C w'.
```

Then we have the following chain

```
transport (λ C, Operation C w) (path_equiv_family f) α y
  ≡ transport (λ C, C t → Operation C w’) (path_equiv_family f) α y
  ≡ transport (λ C, P_1 C (C t)) (path_equiv_family f) α y
  = transport (λ C, P_1 C (B t)) (path_equiv_family f)
      (transport (λ X, P_1 A X) (path_universe (f t)) α) y
  ≡ transport (λ C, B t → Operation C w’) (path_equiv_family f)
      (transport (λ X, X → Operation A w’) (path_universe (f t)) α) y
  ≡ transport (λ C, ∏ (z : B t), P_2 C z) (path_equiv_family f)
      (transport (λ X, X → Operation A w’) (path_universe (f t)) α) y
  = transport (λ C, P_2 C y) (path_equiv_family f)
      (transport (λ X, X → Operation A w’) (path_universe (f t)) α y)
  ≡ transport (λ C, Operation C w’) (path_equiv_family f)
      (transport (λ X, X → Operation A w’) (path_universe (f t)) α y)
  = transport (λ C, Operation C w’) (path_equiv_family f) (α (f^-1 y))
  = β y.
```

The first `=` path is Lemma A.15. The second `=` path is Lemma A.16(ii). The third `=` path is Lemma A.17. Using (∗∗) above, the last path follows by induction. □

Now we have tools to prove the main theorem.

Theorem A.20. If there is an isomorphism `f : Homomorphism A B` then `A = B`.

Proof. By Lemma A.7 we just need to find two paths

- `p : carriers A = carriers B` in `Sort σ → Type`,
- `q : p#(operations A) = operations B` in `∏ (u : Symbol σ), Operation B (σ u)`,

where

```
p#(operations A) ≡ transport (λ C, ∏ u, Operation C (σ u)) p (operations A)
```

For the path `p`, we can choose `path_equiv_family f : carriers A = carriers B` from Lemma A.18. For path `q : p#(operations A) = operations B`, set

```
R :≡ λ (C : Sort σ → Type) (u : Symbol σ), Operation C (σ u).
```

For any function symbol `v : Symbol σ`,

```
(p#(operations A)) v
  ≡ transport (λ C, ∏ u, R C u) p (operations A) v
  = transport (λ C, R C v) p (operations A v)
  ≡ transport (λ C, Operation C (σ v)) (path_equiv_family f) (v^^A)
  = v^^B
  ≡ operations B v.
```

The first `=` path follows from Lemma A.16(ii). The other `=` path follows from Lemma A.19 because `OpPreserving f (v^^A) (v^^B)` holds by Definition A.9. □

A.6 Conclusions and related work

This report presented a beginning of a Universal Algebra development for the HoTT library based on Math Classes [10]. The fact that isomorphic objects are equal is one of the things that distinguish HoTT from set theoretic and category theoretic foundations. Using this main theorem we can obtain paths from the isomorphism theorems, see the formalization `https://github.com/andreaslyn/hott-classes/tree/master/theory`.

There is a similar theorem by Coquand and Danielsson [6]. They work with a type `U : Type` called a universe. The universe has the role of characterising algebraic structures, similar to `Signature` in this report. This allows for more flexibility as to which algebraic structure is supported, but it requires a slightly different definition of isomorphism. They define it by

```
Definition IsIsomorphism
  (U : Type) (El : U → Type → Type)
  (resp : ∏ {a} {A B : Type}, (A ≃ B) → El a A → El a B)
  (resp_id : ∏ {a} {A : Type} (x : El a A), resp equiv_idmap x = x)
  {a} {A B : Type} (f : A ≃ B) (x : El a A) (y : El a B)
  :≡ resp f x = y.
```

The `U` argument is the universe. The value `El a A` is the type of the algebraic structure characterised by `a:U`. This corresponds to the type `∀ u, Operation C (σ u)` from Definition A.4 above. The `resp f` function is for transport of structure `El a A → El a B` by an equivalence `f : A ≃ B`. The `resp_id` argument is there to make sure `resp` is well behaved. They have a transport theorem which shows that such a `resp` function together with `resp_id` satisfies

```
resp f x = transport (El a) (path_universe f) x
```

An equivalence `f : A ≃ B` is an isomorphism if the algebra structure of `A` is transported by `resp f` to that of `B`. This corresponds to Lemma A.19 in this report. They avoid using `path_forall`, as in Lemma A.18 above, because they are working with single-sorted algebraic structures, just one carrier type. We have been working with multi-sorted algebraic structures `Carriers σ ≡ (Sort σ → Type)`, a family of carrier types.

References

[1] S. Awodey. *Category Theory*. Oxford University Press, Inc., New York, NY, USA, 2nd edition, 2010.

[2] A. Bauer, J. Gross, P. L. Lumsdaine, M. Shulman, M. Sozeau, and B. Spitters. The hott library: A formalization of homotopy type theory in coq. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 164–172. ACM, 2017.

[3] G. Birkhoff and J. D. Lipson. Heterogeneous algebras. *Journal of Combinatorial Theory*, 8(1):115 – 133, 1970.

[4] S. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer-Verlag, 2012.

[5] E. Cavallo and R. Harper. Higher Inductive Types in Cubical Computational Type Theory. *Proc. ACM Program. Lang.*, 3(POPL):1:1–1:27, Jan. 2019.

[6] T. Coquand and N. A. Danielsson. Isomorphism is equality. *Indagationes Mathematicae*, 24(4):1105 – 1120, 2013. In memory of N.G. (Dick) de Bruijn (1918–2012), http://www.sciencedirect.com/science/article/pii/S0019357713000694.

[7] T. Coquand, S. Huber, and A. Mörtberg. On Higher Inductive Types in Cubical Type Theory. *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '18*, 2018.

[8] E. Gunther, A. Gadea, and M. Pagano. Formalization of Universal Algebra in Agda. *Electronic Notes in Theoretical Computer Science*, 338:147–166, 10 2018.

[9] C. Kapulkin and P. LeFanu Lumsdaine. The Simplicial Model of Univalent Foundations (after Voevodsky). *arXiv e-prints*, page arXiv:1211.2851, Nov. 2012.

[10] B. Spitters and E. van der Weegen. Type Classes for Mathematics in Type Theory. *MSCS, special issue on 'Interactive theorem proving and the formalization of mathematics'*, 21:1–31, 2011.

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

[12] V. Voevodsky. *Univalent Foundations Project*. `http://www.math.ias.edu/vladimir/files/univalent_foundations_project.pdf`, Oct. 2010. 12 pages.