

A Machine-checked Formal Proof of Birkhoff's Variety Theorem in Martin-Löf Type Theory

William DeMeo ✉ 

<https://williamdemeo.org>

Jacques Carette ✉ 

McMaster University

1 Introduction

The Agda Universal Algebra Library (*agda-algebras*) is a collection of types and programs (theorems and proofs) formalizing the foundations of universal algebra in dependent type theory using the *Agda* programming language and proof assistant. The *agda-algebras* library now includes a substantial collection of definitions, theorems, and proofs from universal algebra and equational logic and as such provides many examples that exhibit the power of inductive and dependent types for representing and reasoning about general algebraic and relational structures.

The first major milestone of the *agda-algebras* project is a new formal proof of *Birkhoff's variety theorem* (also known as the *HSP theorem*), the first version of which was completed in January of 2021. To the best of our knowledge, this was the first time Birkhoff's theorem had been formulated and proved in dependent type theory and verified with a proof assistant.

In this paper, we present a subset of the *agda-algebras* library that culminates in a complete, self-contained, formal proof of the HSP theorem. In the course of our exposition of the proof, we discuss some of the more challenging aspects of formalizing universal algebra in type theory and the issues that arise when attempting to constructively prove some of the basic results in that area. We demonstrate that dependent type theory and Agda, despite the demands they place on the user, are accessible to working mathematicians who have sufficient patience and a strong enough desire to constructively codify their work and formally verify the correctness of their results.

Our presentation may be viewed as a sobering glimpse at the painstaking process of doing mathematics in the languages of dependent types and Agda. Nonetheless we hope to make a compelling case for investing in these languages. Indeed, we are excited to share the gratifying rewards that come with some mastery of type theory and interactive theorem proving technologies.

2 Preliminaries

2.1 Logical foundations

An Agda program typically begins by setting some language options and by importing types from existing Agda libraries. The language options are specified using the *OPTIONS pragma* which affect control the way Agda behaves by controlling the deduction rules that are available to us and the logical axioms that are assumed when the program is type-checked by Agda to verify its correctness. Every Agda program in the *agda-algebras* library begins with the following line.

```
{-# OPTIONS -without-K -exact-split -safe #-}
```

Here we provide a brief overview (along with links to more details) of how these options affect our foundational assumptions.



This work and the *agda-algebras* library by William DeMeo and the *agda-algebras* team is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

- `without-K` disables Streicher’s `K` axiom ; see also the section on axiom `K` in the Agda Language Reference Manual.
- `exact-split` makes Agda accept only those definitions that behave like so-called *judgmental* equalities. Martín Escardó explains this by saying it “makes sure that pattern matching corresponds to Martin-Löf eliminators;” see also the Pattern matching and equality section of the Agda Tools documentation.
- `safe` ensures that nothing is postulated outright—every non-MLTT axiom has to be an explicit assumption (e.g., an argument to a function or module); see also this section of the Agda Tools documentation and the Safe Agda section of the Agda Language Reference.

2.2 Agda Modules

The `OPTIONS` pragma is usually followed by the start of a module. Indeed, the `HSP.lagda` program that is subject of the present paper begins with a list of `import` directives. We won’t produce all of these here. Rather we show only those imports that rename objects from the standard library to our own notation which might be less standard.

```
open import Agda.Primitive using ( _⊔_ ; !suc ) renaming ( Set to Type )
open import Data.Product using ( _×_ ; !_ ; Σ ) renaming ( proj₁ to fst ; proj₂ to snd )
open import Function using ( id ; _∘_ ; flip ; Injection ; Surjection ) renaming ( Func to _→_ )
open import Relation.Binary.PropositionalEquality as ≡ using ( _≡_ )
import Function.Definitions as FD
import Relation.Binary.Reasoning.Setoid as SetoidReasoning
open _→_ using ( cong ) renaming ( f to _⟦$⟧_ )
```

Note, in particular, we rename the `Set` and `Func` types of the standard library to `Type` and the infix long arrow symbol `_→_`, respectively, and we use `fst` and `snd` in place of `proj₁` and `proj₂` for the first and second projections out of the product type `_×_`. In addition, when it improves readability of the code, we use the alternative notation `|_|` and `||_|` (defined elsewhere) for the first and second projections.

2.3 Setoids

A *setoid* is a type packaged with an equivalence relation on the collection of inhabitants of that type. Setoids are useful for representing classical (set-theory-based) mathematics in a constructive, type-theoretic way because most mathematical structures are assumed to come equipped with some (often implicit) equivalence relation manifesting a notion of equality of elements, and therefore a type-theoretic representation of such a structure should also model its equality relation.

The `agda-algebras` library was first developed without the use of setoids, opting instead for specially constructed experimental quotient types. However, this approach resulted in code that was hard to comprehend and it became difficult to determine whether the resulting proofs were fully constructive. In particular, our initial proof of the Birkhoff variety theorem required postulating function extensionality, an axiom that is not provable in pure Martin-Löf type theory.[reference needed]

In contrast, our current approach using setoids makes the equality relation of a given type explicit and this transparency can make it easier to determine the correctness and constructivity of the proofs. Using setoids we need no additional axioms beyond Martin-Löf type theory; in particular, no function extensionality axioms are postulated in our current formalization of Birkhoff’s variety theorem.

2.4 Inverses of setoid functions

We define a data type that represent the semantic concept of the *image* of a function (cf. the `Overture.Func.Inverses` module of the `agda-algebras` library).

```
module _ {A : Setoid α ρa} {B : Setoid β ρb} where
  open Setoid B using ( _≈_ ; sym ) renaming ( Carrier to B )

  data Image_⊃_ (f : A → B) : B → Type (α ⊔ β ⊔ ρb) where
    eq : {b : B} → ∀ a → b ≈ (f $) a → Image f ⊃ b
  open Image_⊃_
```

An inhabitant of `Image f ⊃ b` is a dependent pair (a, p) , where $a : A$ and $p : b \approx f a$ is a proof that f maps a to b . Since the proof that b belongs to the image of f is always accompanied by a witness $a : A$, we can actually *compute* a (pseudo)inverse of f . For convenience, we define this inverse function, which we call `Inv`, and which takes an arbitrary $b : B$ and a (witness, proof)-pair, $(a, p) : \text{Image } f \ni b$, and returns the witness a .

```
Inv : (f : A → B) {b : B} → Image f ⊃ b → Carrier A
Inv _ (eq a _) = a

InvIsInverser : {f : A → B} {b : B} (q : Image f ⊃ b) → (f $) (Inv f q) ≈ b
InvIsInverser (eq _ p) = sym p
```

`InvIsInverser` proves that `Inv f` is the range-restricted right-inverse of the setoid function f .

2.5 Injective and surjective setoid functions

If $f : A \rightarrow B$ is a setoid function from $A = (A, \approx_0)$ to $B = (B, \approx_1)$, then we call f *injective* provided $\forall (a_0 a_1 : A), f \$ a_0 \approx_1 f \$ a_1$ implies $a_0 \approx_0 a_1$; we call f *surjective* provided $\forall (b : B), \exists (a : A)$ such that $f \$ a \approx_1 b$. We codify these definitions in Agda and prove some of their properties inside the next submodule where we first set the stage by declaring two setoids A and B , naming their equality relations, and making some definitions from the standard library available.

```
module _ {A : Setoid α ρa} {B : Setoid β ρb} where
  open Setoid A using () renaming ( _≈_ to _≈1_ ; Carrier to A )
  open Setoid B using () renaming ( _≈_ to _≈2_ ; Carrier to B )
  open Surjection renaming ( f to _$ )
  open FD _≈1_ _≈2_
```

The `Agda Standard Library` represents injective functions on bare types by the type `Injective`, which we now use to define `IsInjective` representing the property of being an injective setoid function. We then define the type `IsSurjective` to represent the property of being a surjective setoid function. Finally, we define `SurjInv` to represent the *right-inverse* of a surjective function. The definitions are as follows (cf. `Overture.Func.Injective` and `Overture.Func.Surjective` in the `agda-algebras` library).

```
IsInjective : (A → B) → Type (α ⊔ ρa ⊔ ρb)
IsInjective f = Injective (_$) f

IsSurjective : (A → B) → Type (α ⊔ β ⊔ ρb)
```

```

IsSurjective F =  $\forall \{y\} \rightarrow \text{Image } F \ni y$  where open Image_ $\ni$ _
SurjInv : (f : A  $\rightarrow$  B)  $\rightarrow$  IsSurjective f  $\rightarrow$  B  $\rightarrow$  A
SurjInv f fE b = Inv f (fE {b})

```

2.5.1 Composition of injective and surjective setoid functions

Proving that the composition of injective setoid functions is again injective is simply a matter of composing the two assumed witnesses to injectivity. Proving that surjectivity is preserved under composition is only slightly more involved.

```

module _ {A : Setoid  $\alpha$   $\rho^a$ } {B : Setoid  $\beta$   $\rho^b$ } {C : Setoid  $\gamma$   $\rho^c$ } (f : A  $\rightarrow$  B) (g : B  $\rightarrow$  C) where
  o-IsInjective : IsInjective f  $\rightarrow$  IsInjective g  $\rightarrow$  IsInjective (g  $\langle \circ \rangle$  f)
  o-IsInjective finj ginj = finj  $\circ$  ginj

  o-IsSurjective : IsSurjective f  $\rightarrow$  IsSurjective g  $\rightarrow$  IsSurjective (g  $\langle \circ \rangle$  f)
  o-IsSurjective fE gE {y} = Goal
  where
    mp : Image g  $\ni y \rightarrow$  Image g  $\langle \circ \rangle$  f  $\ni y$ 
    mp (eq c p) =  $\eta$  fE
    where
      open Setoid C using ( trans )
       $\eta$  : Image f  $\ni c \rightarrow$  Image g  $\langle \circ \rangle$  f  $\ni y$ 
       $\eta$  (eq a q) = eq a (trans p (cong g q))

    Goal : Image g  $\langle \circ \rangle$  f  $\ni y$ 
    Goal = mp gE

```

2.6 Kernels

The *kernel* of a function $f : A \rightarrow B$ (where A and B are bare types) is defined informally by $\{(x, y) \in A \times A : f x = f y\}$. This can be represented in Agda in a number of ways, but for our purposes we find it most convenient to define the kernel as an inhabitant of a (unary) predicate over the square of the function’s domain.

```

module _ {A : Type  $\alpha$ } {B : Type  $\beta$ } where
  kernel : Rel B  $\rho \rightarrow$  (A  $\rightarrow$  B)  $\rightarrow$  Pred (A  $\times$  A)  $\rho$ 
  kernel  $\approx$  f (x , y) = f x  $\approx$  f y

```

The kernel of a setoid function $f : A \rightarrow B$ is defined informally by $\{(x, y) \in A \times A : f \langle \$ \rangle x \approx f \langle \$ \rangle y\}$, where \approx denotes the equality of B .

```

module _ {A : Setoid  $\alpha$   $\rho^a$ } {B : Setoid  $\beta$   $\rho^b$ } where
  open Setoid A using () renaming ( Carrier to A )

  ker : (A  $\rightarrow$  B)  $\rightarrow$  Pred (A  $\times$  A)  $\rho^b$ 
  ker g (x , y) = g  $\langle \$ \rangle$  x  $\approx$  g  $\langle \$ \rangle$  y where open Setoid B using (  $\approx$  )

```

3 Algebras

In this section we define the notion of an algebraic structure whose *domain* (or “carrier” or “universe”) is a setoid. Our first goal is to develop a working vocabulary and formal types for

classical (single-sorted, set-based) universal algebra.

3.1 Signatures of an algebra

In [model theory](#), the *signature* $S = (C, F, R, \rho)$ of a structure consists of three (possibly empty) sets C , F , and R —called *constant symbols*, *function symbols*, and *relation symbols*, respectively—along with a function $\rho : C + F + R \rightarrow \mathbb{N}$ that assigns an *arity* to each symbol. Often, but not always, \mathbb{N} is taken to be the set of natural numbers.

As our focus here is universal algebra, we are more concerned with the restricted notion of an *algebraic signature* (or *signature* for algebraic structures), by which we mean a pair $S = (F, \rho)$ consisting of a collection F of *operation symbols* and an *arity function* $\rho : F \rightarrow \mathbb{N}$ that maps each operation symbol to its arity; here, \mathbb{N} denotes the *arity type*. Heuristically, the arity ρf of an operation symbol $f \in F$ may be thought of as the “number of arguments” that f takes as “input”.

If the arity of f is n , then we call f an n -ary operation symbol. In case n is 0 (or 1 or 2 or 3, respectively) we call the function *nullary* (or *unary* or *binary* or *ternary*, respectively).

If A is a set and f is a (ρf) -ary operation on A , then the arguments of f form a (ρf) -tuple, say, $(a_0, a_1, \dots, a_{(\rho f)-1})$, which may be viewed as the graph of a function, say, $a : \rho f \rightarrow A$. When the codomain of ρ is \mathbb{N} , we may view ρf as the finite set $\{0, 1, \dots, \rho f - 1\}$. Thus, by identifying the ρf -th power of A with the type $\rho f \rightarrow A$ of functions from $\{0, 1, \dots, \rho f - 1\}$ to A , we identify the collection of all tuples of arguments of f with the function type $(\rho f \rightarrow A) \rightarrow A$.

3.1.1 Signature type

The [agda-algebras](#) library represents a *signature* as an inhabitant of the following dependent pair type.

```
Signature : (ℳ ℳ' : Level) → Type (lsuc (ℳ ⊔ ℳ'))
Signature ℳ ℳ' = Σ[ F ∈ Type ℳ ] (F → Type ℳ')
```

Using our special syntax, $| _ |$ and $\| _ \|$, for the first and second projections, respectively, if $S : \text{Signature } \mathcal{M} \ \mathcal{M}'$ is a signature, then

- $| S |$ denotes the set of operation symbols;
- $\| S \|$ denotes the arity function.

Thus, if $f : | S |$ is an operation symbol in the signature S , then $\| S \| f$ is the arity of f .

First we define an operator that translates an ordinary signature into a signature over a setoid domain.

```
EqArgs : {S : Signature ℳ ℳ'} {ξ : Setoid α ρa}
  → ∀ {f g} → f ≡ g → (‖ S ‖ f → Carrier ξ) → (‖ S ‖ g → Carrier ξ) → Type (ℳ ⊔ ρa)

EqArgs {ξ = ξ} ≡ refl u v = ∀ i → u i ≈ v i
  where open Setoid ξ using ( _≈_ )

module _ where
  open Setoid using ( _≈_ )
  open IsEquivalence using ( refl ; sym ; trans )

  (⟦_⟧) : Signature ℳ ℳ' → Setoid α ρa → Setoid _ _
```

```

Carrier (( S ) ξ) = Σ[ f ∈ | S | ] ((|| S || f) → ξ .Carrier)
_≈_ (( S ) ξ) (f , u) (g , v) = Σ[ eqv ∈ f ≡ g ] EqArgs{ξ = ξ} eqv u v
refl (isEquivalence (( S ) ξ)) = ≡.refl , λ _ → Setoid.refl ξ
sym (isEquivalence (( S ) ξ)) (≡.refl , g) = ≡.refl , λ i → Setoid.sym ξ (g i)
trans (isEquivalence (( S ) ξ)) (≡.refl , g)(≡.refl , h) = ≡.refl , λ i → Setoid.trans ξ (g i) (h i)

```

3.2 Algebra type

Informally, an *algebraic structure in the signature* $S = (F, \rho)$ (or *S-algebra*) is typically denoted by $\mathbf{A} = (A, F^A)$ and consists of

- A := a *nonempty* set (or type), called the *domain* (or *carrier* or *universe*) of the algebra;
- $F^A := \{ f^A \mid f \in F, f^A : (\rho f \rightarrow A) \rightarrow A \}$, a collection of *operations* on A ;
- a (potentially empty) collection of *identities* satisfied by elements and operations of A .

We represent an algebra in Agda using a record type with two fields:

- **Domain** is a setoid denoting the underlying *universe* of the algebra (informally, the set of elements of the algebra);
- **Interp** represents the *interpretation* in the algebra of each operation symbol of the given signature. The record type **Func** from the Agda Standard Library provides what we need for an operation on the domain setoid.

Let us present the definition of the **Algebra** type and then discuss the definition of the **Func** type that provides the interpretation of each operation symbol.

```

record Algebra α ρ : Type (0 ⊔ ℵ ⊔ Isuc (α ⊔ ρ)) where
  field
    Domain : Setoid α ρ
    Interp : (( S ) Domain) → Domain

```

The **Interp** field actually has type **Func** $((S) \text{Domain}) \text{Domain}$ (recall we renamed **Func** as the infix long-arrow symbol). The **Func** type is from the standard library and is defined as a record type with two fields. In the present instance, the fields are

1. a function $f : \text{Carrier } ((S) \text{Domain}) \rightarrow \text{Carrier Domain}$
2. a proof $\text{cong} : f \text{ Preserves } _ \approx_1 _ \rightarrow _ \approx_2 _$ that f preserves the underlying setoid equalities.

Thus **Interp** gives, for each operation symbol in the signature S , a setoid function f —namely, a function where the domain is a power of **Domain** and the codomain is **Domain**—along with a proof that all operations so interpreted respect the underlying setoid equality on **Domain**.

Next we define some syntactic sugar that will make our Agda code easier to read and comprehend. If \mathbf{A} is an algebra, then

- $\mathbb{D}[\mathbf{A}]$ will denote the setoid **Domain** \mathbf{A} ,
- $\mathbb{U}[\mathbf{A}]$ will be the underlying carrier or “universe” of the algebra \mathbf{A} , and
- $f^\mathbf{A}$ will denote the interpretation in the algebra \mathbf{A} of the operation symbol f .

```

open Algebra

ℰ[ ] : Algebra α ρa → Type α

```

```

U[ A ] = Carrier (Domain A)

D[ ] : Algebra α ρa → Setoid α ρa
D[ A ] = Domain A

_ ^ _ : (f : | S |)(A : Algebra α ρa) → (|| S || f → U[ A ]) → U[ A ]

f ^ A = λ a → (Interp A) (⌞$⌟ (f , a))

```

3.3 Universe lifting of algebra types

A technical aspect of dealing with the noncumulativity of the hierarchy of type levels in Agda...

3.4 Product Algebras

(cf. the `Algebras.Func.Products` module of the `Agda Universal Algebra Library`.)

```

module _ {ι : Level}{I : Type ι} where

  Π : (A : I → Algebra α ρa) → Algebra (α ⊔ ι) (ρa ⊔ ι)

  Domain (Π A) =
    record { Carrier = ∀ i → U[ A i ]
          ; _≈_ = λ a b → ∀ i → (Setoid._≈_ D[ A i ]) (a i)(b i)
          ; isEquivalence =
              record { refl = λ i → IsEquivalence.refl (isEquivalence D[ A i ])
                    ; sym = λ x i → IsEquivalence.sym (isEquivalence D[ A i ])(x i)
                    ; trans = λ x y i → IsEquivalence.trans (isEquivalence D[ A i ])(x i)(y i) }}

  Interp (Π A) ⌞$⌟ (f , a) = λ i → (f ^ (A i)) (flip a i)
  cong (Interp (Π A)) (≡.refl , f=g) = λ i → cong (Interp (A i)) (≡.refl , flip f=g i)

```

4 Homomorphisms

4.1 Basic definitions

Here are some useful definitions and theorems extracted from the `Homomorphisms.Func.Basic` module of the `Agda Universal Algebra Library`.

```

module _ (A : Algebra α ρa)(B : Algebra β ρb) where
  private ov = 0 ⊔ ℳ ; a = α ⊔ ρa ; b = β ⊔ ρb ; c = 0 ⊔ ℳ ⊔ α ⊔ ρa ⊔ β ⊔ ρb

  compatible-map-op : (D[ A ] → D[ B ]) → | S | → Type (ℳ ⊔ α ⊔ ρb)
  compatible-map-op h f = ∀ {a} → (h ⌞$⌟ ((f ^ A) a)) ≈2 ((f ^ B) (λ x → (h ⌞$⌟ (a x))))
    where open Setoid D[ B ] using () renaming ( _≈_ to _≈2_ )

  compatible-map : (D[ A ] → D[ B ]) → Type (ov ⊔ α ⊔ ρb)
  compatible-map h = ∀ {f} → compatible-map-op h f

  - The property of being a homomorphism.
  record IsHom (h : D[ A ] → D[ B ]) : Type (ov ⊔ α ⊔ ρb) where
    field compatible : compatible-map h

  - The type of homomorphisms.

```

```

hom : Type (ov  $\sqcup$  a  $\sqcup$  b)
hom =  $\Sigma$  ( $\mathbb{D}[A] \rightarrow \mathbb{D}[B]$ ) IsHom

```

4.2 Monomorphisms and epimorphisms

```

record IsMon (h :  $\mathbb{D}[A] \rightarrow \mathbb{D}[B]$ ) : Type (ov  $\sqcup$  a  $\sqcup$   $\rho^b$ ) where
  field isHom : IsHom h ; isInjective : IsInjective h

  HomReduct : hom
  HomReduct = h , isHom

mon : Type (ov  $\sqcup$  a  $\sqcup$  b)
mon =  $\Sigma$  ( $\mathbb{D}[A] \rightarrow \mathbb{D}[B]$ ) IsMon

record IsEpi (h :  $\mathbb{D}[A] \rightarrow \mathbb{D}[B]$ ) : Type (ov  $\sqcup$   $\alpha$   $\sqcup$  b) where
  field isHom : IsHom h ; isSurjective : IsSurjective h

  HomReduct : hom
  HomReduct = h , isHom

epi : Type (ov  $\sqcup$  a  $\sqcup$  b)
epi =  $\Sigma$  ( $\mathbb{D}[A] \rightarrow \mathbb{D}[B]$ ) IsEpi

open IsHom ; open IsMon ; open IsEpi

module _ (A : Algebra  $\alpha$   $\rho^a$ )(B : Algebra  $\beta$   $\rho^b$ ) where

  mon $\rightarrow$ intohom : mon A B  $\rightarrow \Sigma$ [ h  $\in$  hom A B ] IsInjective | h |
  mon $\rightarrow$ intohom (hh , hhM) = (hh , isHom hhM) , isInjective hhM

  epi $\rightarrow$ ontohom : epi A B  $\rightarrow \Sigma$ [ h  $\in$  hom A B ] IsSurjective | h |
  epi $\rightarrow$ ontohom (hh , hhE) = (hh , isHom hhE) , isSurjective hhE

```

4.3 Basic properties of homomorphisms

Here are some definitions and theorems extracted from the `Homomorphisms.Func.Properties` module of the `Agda Universal Algebra Library`.

4.3.1 Composition of homomorphisms

The composition of homomorphisms is again a homomorphism. Similarly, the composition of epimorphisms is again an epimorphism.

```

module _ {A : Algebra  $\alpha$   $\rho^a$ } {B : Algebra  $\beta$   $\rho^b$ } {C : Algebra  $\gamma$   $\rho^c$ }
  {g :  $\mathbb{D}[A] \rightarrow \mathbb{D}[B]$ } {h :  $\mathbb{D}[B] \rightarrow \mathbb{D}[C]$ } where
  open Setoid  $\mathbb{D}[C]$  using ( trans )

  o-is-hom : IsHom A B g  $\rightarrow$  IsHom B C h  $\rightarrow$  IsHom A C (h  $\langle \circ \rangle$  g)
  o-is-hom ghom hhom = record { compatible = c }
    where
      c : compatible-map A C (h  $\langle \circ \rangle$  g)
      c = trans (cong h (compatible ghom)) (compatible hhom)

  o-is-epi : IsEpi A B g  $\rightarrow$  IsEpi B C h  $\rightarrow$  IsEpi A C (h  $\langle \circ \rangle$  g)
  o-is-epi gE hE =

```



```

record { isHom = o-is-hom (isHom gE) (isHom hE)
      ; isSurjective = o-IsSurjective g h (isSurjective gE) (isSurjective hE) }

module _ {A : Algebra α ρa} {B : Algebra β ρb} {C : Algebra γ ρc} where

o-hom : hom A B → hom B C → hom A C
o-hom (h , hhom) (g , ghom) = (g ⟨o⟩ h) , o-is-hom hhom ghom

o-epi : epi A B → epi B C → epi A C
o-epi (h , hepi) (g , gepi) = (g ⟨o⟩ h) , o-is-epi hepi gepi

```

4.3.2 Universe lifting of homomorphisms

First we define the identity homomorphism for setoid algebras and then we prove that the operations of lifting and lowering of a setoid algebra are homomorphisms.

4.4 Factorization of homomorphisms

If $g : \text{hom } A \ B$, $h : \text{hom } A \ C$, h is surjective, and $\ker h \subseteq \ker g$, then there exists $\varphi : \text{hom } C \ B$ such that $g = \varphi \circ h$ (cf. the `Homomorphisms.Func.Factor` module of the `Agda Universal Algebra Library`).

```

module _ {A : Algebra α ρa} {B : Algebra β ρb} {C : Algebra γ ρc}
  (gh : hom A B) (hh : hom A C) where
  open Setoid ℙ [ B ] using ( ) renaming ( _≈_ to _≈2_ ; sym to sym2 )
  open Setoid ℙ [ C ] using ( trans ) renaming ( _≈_ to _≈3_ ; sym to sym3 )
  open SetoidReasoning ℙ [ B ]
  private gfunc = | gh | ; g = _⟨$⟩_ gfunc ; hfunc = | hh | ; h = _⟨$⟩_ hfunc

  HomFactor : kernel _≈3_ h ⊆ kernel _≈2_ g → IsSurjective hfunc
  → ∑ [ φ ∈ hom C B ] ∀ a → (g a) ≈2 | φ | ⟨$⟩ (h a)

  HomFactor Khg hE = (φmap , φhom) , gφh
  where
  kerpres : ∀ a0 a1 → h a0 ≈3 h a1 → g a0 ≈2 g a1
  kerpres a0 a1 hyp = Khg hyp

  h-1 : ℙ [ C ] → ℙ [ A ]
  h-1 = SurjInv hfunc hE

  η : ∀ {c} → h (h-1 c) ≈3 c
  η = InvIsInverser hE

  ζ : ∀ {x y} → x ≈3 y → h (h-1 x) ≈3 h (h-1 y)
  ζ xy = trans η (trans xy (sym3 η))

  φmap : ℙ [ C ] → ℙ [ B ]
  _⟨$⟩_ φmap = g ∘ h-1
  cong φmap = Khg ∘ ζ
  open _→_ φmap using ( ) renaming ( cong to φcong )

  gφh : (a : ℙ [ A ]) → g a ≈2 φmap ⟨$⟩ (h a)
  gφh a = Khg (sym3 η)

  φcomp : compatible-map C B φmap

```

```

 $\varphi\text{comp } \{f\}\{c\} =$ 
begin
   $\varphi\text{map } \langle \$ \rangle ((f \hat{=} C) c) \approx^\sim \langle \varphi\text{cong } (\text{cong } (\text{Interp } C) (\equiv.\text{refl} , (\lambda \_ \rightarrow \eta))) \rangle$ 
   $g (h^{-1} ((f \hat{=} C)(h \circ (h^{-1} \circ c)))) \approx^\sim \langle \varphi\text{cong } (\text{compatible } \parallel \text{hh } \parallel) \rangle$ 
   $g (h^{-1} (h ((f \hat{=} A)(h^{-1} \circ c)))) \approx^\sim \langle g\varphi h ((f \hat{=} A)(h^{-1} \circ c)) \rangle$ 
   $g ((f \hat{=} A)(h^{-1} \circ c)) \approx \langle \text{compatible } \parallel gh \parallel \rangle$ 
   $(f \hat{=} B)(g \circ (h^{-1} \circ c)) \blacksquare$ 

 $\varphi\text{hom} : \text{IsHom } C B \varphi\text{map}$ 
 $\text{compatible } \varphi\text{hom} = \varphi\text{comp}$ 

```

4.5 Isomorphisms

(cf. the `Homomorphisms.Func.Isomorphisms` of the Agda Universal Algebra Library.)

Two structures are *isomorphic* provided there are homomorphisms going back and forth between them which compose to the identity map.

```

module _ (A : Algebra  $\alpha$   $\rho^a$ ) (B : Algebra  $\beta$   $\rho^b$ ) where
  open Setoid D[ A ] using ( sym ; trans ) renaming (  $\approx$  to  $\approx_1$  )
  open Setoid D[ B ] using () renaming (  $\approx$  to  $\approx_2$  ; sym to sym2 ; trans to trans2 )

  record  $\cong$  : Type (0  $\sqcup$   $\mathcal{V}$   $\sqcup$   $\alpha$   $\sqcup$   $\beta$   $\sqcup$   $\rho^a$   $\sqcup$   $\rho^b$  ) where
    constructor mkiso
    field
      to : hom A B
      from : hom B A
      to $\sim$ from :  $\forall b \rightarrow (| \text{to} | \langle \$ \rangle (| \text{from} | \langle \$ \rangle b)) \approx_2 b$ 
      from $\sim$ to :  $\forall a \rightarrow (| \text{from} | \langle \$ \rangle (| \text{to} | \langle \$ \rangle a)) \approx_1 a$ 

  toIsSurjective : IsSurjective | to |
  toIsSurjective {y} = eq (| from |  $\langle \$ \rangle$  y) (sym2 (to $\sim$ from y))

  toIsInjective : IsInjective | to |
  toIsInjective {x} {y} xy = Goal
  where
     $\xi : | \text{from} | \langle \$ \rangle (| \text{to} | \langle \$ \rangle x) \approx_1 | \text{from} | \langle \$ \rangle (| \text{to} | \langle \$ \rangle y)$ 
     $\xi = \text{cong } | \text{from} | xy$ 
    Goal :  $x \approx_1 y$ 
    Goal = trans (sym (from $\sim$ to x)) (trans  $\xi$  (from $\sim$ to y))

  fromIsSurjective : IsSurjective | from |
  fromIsSurjective {y} = eq (| to |  $\langle \$ \rangle$  y) (sym (from $\sim$ to y))

  fromIsInjective : IsInjective | from |
  fromIsInjective {x} {y} xy = Goal
  where
     $\xi : | \text{to} | \langle \$ \rangle (| \text{from} | \langle \$ \rangle x) \approx_2 | \text{to} | \langle \$ \rangle (| \text{from} | \langle \$ \rangle y)$ 
     $\xi = \text{cong } | \text{to} | xy$ 
    Goal :  $x \approx_2 y$ 
    Goal = trans2 (sym2 (to $\sim$ from x)) (trans2  $\xi$  (to $\sim$ from y))

  open  $\cong$ 

```

4.5.1 Properties of isomorphisms

```

≅-refl : Reflexive (≅_ {α}{ρa})
≅-refl {α}{ρa}{A} = mkiso id id (λ b → refl) λ a → refl
  where open Setoid D[ A ] using ( refl )

≅-sym : Sym (≅_ {β}{ρb}) (≅_ {α}{ρa})
≅-sym φ = mkiso (from φ) (to φ) (from~to φ) (to~from φ)

≅-trans : Trans (≅_ {α}{ρa})(≅_ {β}{ρb})(≅_ {α}{ρa}{γ}{ρc})
≅-trans {ρc = ρc}{A}{B}{C} ab bc = mkiso f g τ ν
  where
    open Setoid D[ A ] using () renaming ( ≅_ to ≅1 ; trans to trans1 )
    open Setoid D[ C ] using () renaming ( ≅_ to ≅3 ; trans to trans3 )
    f : hom A C
    f = o-hom (to ab) (to bc)
    g : hom C A
    g = o-hom (from bc) (from ab)
    τ : ∀ b → (| f | $) (| g | $) b) ≈3 b
    τ b = trans3 (cong | to bc | (to~from ab (| from bc | $) b))) (to~from bc b)
    ν : ∀ a → (| g | $) (| f | $) a) ≈1 a
    ν a = trans1 (cong | from ab | (from~to bc (| to ab | $) a))) (from~to ab a)

```

Fortunately, the lift operation preserves isomorphism (i.e., it's an *algebraic invariant*). As our focus is universal algebra, this is important and is what makes the lift operation a workable solution to the technical problems that arise from the noncumulativity of Agda's universe hierarchy.

4.6 Homomorphic Images

We begin with what for our purposes is the most useful way to represent the class of *homomorphic images* of an algebra in dependent type theory (cf. the `Homomorphisms.Func.HomomorphicImages` module of the `Agda Universal Algebra Library`). (The first definition is merely a short-hand.)

```

ov : Level → Level
ov α = 0 ⊔ ℳ ⊔ Isuc α

_!sHomImageOf_ : (B : Algebra β ρb)(A : Algebra α ρa) → Type (0 ⊔ ℳ ⊔ α ⊔ β ⊔ ρa ⊔ ρb)
B !sHomImageOf A = Σ[ φ ∈ hom A B ] !sSurjective | φ |

HomImages : Algebra α ρa → Type (α ⊔ ρa ⊔ ov (β ⊔ ρb))
HomImages {β = β}{ρb = ρb} A = Σ[ B ∈ Algebra β ρb ] B !sHomImageOf A

```

These types should be self-explanatory, but just to be sure, let's describe the Sigma type appearing in the second definition. Given an S -algebra $A : \text{Algebra } \alpha \ \rho$, the type `HomImages A` denotes the class of algebras $B : \text{Algebra } \beta \ \rho$ with a map $\varphi : |A| \rightarrow |B|$ such that φ is a surjective homomorphism.

5 Subalgebras

5.1 Basic definitions

$\underline{\leq} : \text{Algebra } \alpha \rho^a \rightarrow \text{Algebra } \beta \rho^b \rightarrow \text{Type } (\mathbb{G} \sqcup \mathcal{V} \sqcup \alpha \sqcup \rho^a \sqcup \beta \sqcup \rho^b)$
 $\mathbf{A} \leq \mathbf{B} = \Sigma[\mathbf{h} \in \text{hom } \mathbf{A} \mathbf{B}] \text{IsInjective } | \mathbf{h} |$

5.2 Basic properties

$\leq\text{-reflexive} : \{ \mathbf{A} : \text{Algebra } \alpha \rho^a \} \rightarrow \mathbf{A} \leq \mathbf{A}$

$\leq\text{-reflexive } \{ \mathbf{A} = \mathbf{A} \} = \text{id} , \text{id}$

$\text{mon} \rightarrow \leq : \{ \mathbf{A} : \text{Algebra } \alpha \rho^a \} \{ \mathbf{B} : \text{Algebra } \beta \rho^b \} \rightarrow \text{mon } \mathbf{A} \mathbf{B} \rightarrow \mathbf{A} \leq \mathbf{B}$

$\text{mon} \rightarrow \leq \{ \mathbf{A} = \mathbf{A} \} \{ \mathbf{B} \} \mathbf{x} = \text{mon} \rightarrow \text{intohom } \mathbf{A} \mathbf{B} \mathbf{x}$

$\text{module } _ \{ \mathbf{A} : \text{Algebra } \alpha \rho^a \} \{ \mathbf{B} : \text{Algebra } \beta \rho^b \} \{ \mathbf{C} : \text{Algebra } \gamma \rho^c \} \text{ where}$

$\leq\text{-trans} : \mathbf{A} \leq \mathbf{B} \rightarrow \mathbf{B} \leq \mathbf{C} \rightarrow \mathbf{A} \leq \mathbf{C}$

$\leq\text{-trans } (\mathbf{f} , \text{finj}) (\mathbf{g} , \text{ginj}) = (\text{o-hom } \mathbf{f} \mathbf{g}) , \text{o-IsInjective } | \mathbf{f} | | \mathbf{g} | \text{finj ginj}$

$\cong\text{-trans-}\leq : \mathbf{A} \cong \mathbf{B} \rightarrow \mathbf{B} \leq \mathbf{C} \rightarrow \mathbf{A} \leq \mathbf{C}$

$\cong\text{-trans-}\leq \mathbf{A} \cong \mathbf{B} (\mathbf{h} , \text{hinj}) = (\text{o-hom } (\text{to } \mathbf{A} \cong \mathbf{B}) \mathbf{h}) , (\text{o-IsInjective } | \text{to } \mathbf{A} \cong \mathbf{B} | | \mathbf{h} | (\text{tolIsInjective } \mathbf{A} \cong \mathbf{B}) \text{hinj})$

5.3 Products of subalgebras

$\text{module } _ \{ \iota : \text{Level} \} \{ \mathbf{I} : \text{Type } \iota \} \{ \mathcal{A} : \mathbf{I} \rightarrow \text{Algebra } \alpha \rho^a \} \{ \mathcal{B} : \mathbf{I} \rightarrow \text{Algebra } \beta \rho^b \} \text{ where}$

$\sqcap \leq : (\forall \mathbf{i} \rightarrow \mathcal{B} \mathbf{i} \leq \mathcal{A} \mathbf{i}) \rightarrow \sqcap \mathcal{B} \leq \sqcap \mathcal{A}$

$\sqcap \leq \mathbf{B} \leq \mathbf{A} = \mathbf{h} , \mathbf{hM}$

where

$\mathbf{h} : \text{hom } (\sqcap \mathcal{B}) (\sqcap \mathcal{A})$

$\mathbf{h} = \text{hfunc} , \text{hhom}$

where

$\mathbf{hi} : \forall \mathbf{i} \rightarrow \text{hom } (\mathcal{B} \mathbf{i}) (\mathcal{A} \mathbf{i})$

$\mathbf{hi} \mathbf{i} = | \mathbf{B} \leq \mathbf{A} \mathbf{i} |$

$\text{hfunc} : \mathbb{D}[\sqcap \mathcal{B}] \rightarrow \mathbb{D}[\sqcap \mathcal{A}]$

$(\text{hfunc } \langle \$ \rangle \mathbf{x}) \mathbf{i} = | \mathbf{hi} \mathbf{i} | \langle \$ \rangle (\mathbf{x} \mathbf{i})$

$\text{cong hfunc} = \lambda \mathbf{xy} \mathbf{i} \rightarrow \text{cong } | \mathbf{hi} \mathbf{i} | (\mathbf{xy} \mathbf{i})$

$\text{hhom} : \text{IsHom } (\sqcap \mathcal{B}) (\sqcap \mathcal{A}) \text{hfunc}$

$\text{compatible hhom} = \lambda \mathbf{i} \rightarrow \text{compatible } || \mathbf{hi} \mathbf{i} ||$

$\mathbf{hM} : \text{IsInjective } | \mathbf{h} |$

$\mathbf{hM} = \lambda \mathbf{xy} \mathbf{i} \rightarrow || \mathbf{B} \leq \mathbf{A} \mathbf{i} || (\mathbf{xy} \mathbf{i})$

6 Terms

6.1 Basic definitions

Fix a signature S and let \mathbf{X} denote an arbitrary nonempty collection of variable symbols. Assume the symbols in \mathbf{X} are distinct from the operation symbols of S , that is $\mathbf{X} \cap | S | = \emptyset$.

By a *word* in the language of S , we mean a nonempty, finite sequence of members of $\mathbf{X} \cup | S |$. We denote the concatenation of such sequences by simple juxtaposition.

Let S_0 denote the set of nullary operation symbols of S . We define by induction on n the sets T_n of *words* over $\mathbf{X} \cup | S |$ as follows (cf. Bergman (2012) Def. 4.19):

$T_0 := \mathbf{X} \cup S_0$ and $T_{n+1} := T_n \cup \mathcal{T}_n$

where \mathcal{T}_n is the collection of all $f \ t$ such that $f : | S |$ and $t : || S || f \rightarrow T_n$. (Recall, $|| S || f$ is the arity of the operation symbol f .)

We define the collection of *terms* in the signature S over X by $\text{Term } X := \bigcup_n T_n$. By an *S-term* we mean a term in the language of S .

The definition of $\text{Term } X$ is recursive, indicating that an inductive type could be used to represent the semantic notion of terms in type theory. Indeed, such a representation is given by the following inductive type.

```
data Term (X : Type χ) : Type (ov χ) where
  g : X → Term X – (g for "generator")
  node : (f : | S |)(t : || S || f → Term X) → Term X
open Term
```

This is a very basic inductive type that represents each term as a tree with an operation symbol at each **node** and a variable symbol at each leaf (**generator**).

Notation. As usual, the type X represents an arbitrary collection of variable symbols. Recall, $\text{ov } \chi$ is our shorthand notation for the universe level $\mathcal{O} \sqcup \mathcal{V} \sqcup \text{lsuc } \chi$.

6.2 Equality of terms

We take a different approach here, using Setoids instead of quotient types. That is, we will define the collection of terms in a signature as a setoid with a particular equality-of-terms relation, which we must define. Ultimately we will use this to define the (absolutely free) term algebra as a Algebra whose carrier is the setoid of terms.

```
module _ {X : Type χ} where
  data _≐_ : Term X → Term X → Type (ov χ) where
    rfl : {x y : X} → x ≡ y → (g x) ≐ (g y)
    gnl : ∀ {f}{s t : || S || f → Term X} → (∀ i → (s i) ≐ (t i)) → (node f s) ≐ (node f t)
```

It is easy to show that the equality-of-terms relation $_ \doteq _$ is an equivalence relation, so we omit the formal proof. (See the `Terms.Func.Basic` module of the `agda-algebras` library for details.)

6.3 The term algebra

For a given signature S , if the type $\text{Term } X$ is nonempty (equivalently, if X or $| S |$ is nonempty), then we can define an algebraic structure, denoted by $\mathbf{T } X$ and called the *term algebra in the signature S over X* . Terms are viewed as acting on other terms, so both the domain and basic operations of the algebra are the terms themselves.

- For each operation symbol $f : | S |$, denote by $f^\wedge(\mathbf{T } X)$ the operation on $\text{Term } X$ that maps a tuple $t : || S || f \rightarrow | \mathbf{T } X |$ to the formal term $f \ t$.
- Define $\mathbf{T } X$ to be the algebra with universe $| \mathbf{T } X | := \text{Term } X$ and operations $f^\wedge(\mathbf{T } X)$, one for each symbol f in $| S |$.

In Agda the term algebra can be defined as simply as one might hope.

```
TermSetoid : (X : Type χ) → Setoid (ov χ) (ov χ)
TermSetoid X = record { Carrier = Term X ; _≈_ = _≐_ ; isEquivalence = ≐-isEquiv }
```

```

T : (X : Type  $\chi$ ) → Algebra (ov  $\chi$ ) (ov  $\chi$ )
Algebra.Domain (T X) = TermSetoid X
Algebra.Interp (T X) <$> (f , ts) = node f ts
cong (Algebra.Interp (T X)) (≡.refl , ss≐ts) = gnl ss≐ts

```

6.4 Interpretation of terms

The approach to terms and their interpretation in this module was inspired by Andreas Abel's formal proof of Birkhoff's completeness theorem.

A substitution from X to Y associates a term in X with each variable in Y .

– *Parallel substitutions.*

```

Sub : Type  $\chi$  → Type  $\chi$  → Type (ov  $\chi$ )
Sub X Y = (y : Y) → Term X

```

– *Application of a substitution.*

```

_[]_ : {X Y : Type  $\chi$ } {t : Term Y} (σ : Sub X Y) → Term X
(g x) [ σ ] = σ x
(node f ts) [ σ ] = node f (λ i → ts i [ σ ])

```

An environment for Γ maps each variable $x : \Gamma$ to an element of A , and equality of environments is defined pointwise.

```

module Environment (A : Algebra  $\alpha$   $\ell$ ) where
  open Setoid  $\mathbb{D}[A]$  using ( refl ; sym ; trans ) renaming ( _≈_ to _≈a_ )
  Env : Type  $\chi$  → Setoid _ _
  Env X = record { Carrier = X →  $\mathbb{U}[A]$ 
                  ; _≈_ = λ ρ ρ' → (x : X) → ρ x ≈a ρ' x
                  ; isEquivalence =
                      record { refl = λ _ → refl
                              ; sym = λ h x → sym (h x)
                              ; trans = λ g h x → trans (g x) (h x) }}

  []_ : {X : Type  $\chi$ } {t : Term X} → (Env X) →  $\mathbb{D}[A]$ 
  [ g x ] <$> ρ = ρ x
  [ node f args ] <$> ρ = (Interp A) <$> (f , λ i → [ args i ] <$> ρ)
  cong [ g x ] u≈v = u≈v x
  cong [ node f args ] x≈y = cong (Interp A) (≡.refl , λ i → cong [ args i ] x≈y )

```

An equality between two terms holds in a model if the two terms are equal under all valuations of their free variables (cf. Andreas Abel's formal proof of Birkhoff's completeness theorem).

```

Equal : ∀ {X : Type  $\chi$ } (s t : Term X) → Type _
Equal {X = X} s t = ∀ (ρ : Carrier (Env X)) → [ s ] <$> ρ ≈a [ t ] <$> ρ

⇒→Equal : {X : Type  $\chi$ } (s t : Term X) → s ≐ t → Equal s t
⇒→Equal .(g _) .(g _) (rfl ≡.refl) = λ _ → refl
⇒→Equal (node _ s) (node _ t) (gnl x) =
  λ ρ → cong (Interp A) (≡.refl , λ i → ⇒→Equal (s i) (t i) (x i) ρ)

```

The proof that `Equal` is an equivalence relation is trivial, so we omit it. (See the `Varieties.Func.SoundAndComplete` module of the `agda-algebras` library for details.)

Evaluation of a substitution gives an environment (cf. Andreas Abel’s formal proof of Birkhoff’s completeness theorem)

```

[⟦_⟧]s : {X Y : Type χ} → Sub X Y → Carrier (Env X) → Carrier (Env Y)
[⟦ σ ⟧]s ρ x = [⟦ σ x ⟧] ⟨$⟩ ρ

```

Next we prove that $\llbracket t[\sigma] \rrbracket \rho \simeq \llbracket t \rrbracket \llbracket \sigma \rrbracket \rho$ (cf. Andreas Abel’s formal proof of Birkhoff’s completeness theorem).

```

substitution : {X Y : Type χ} → (t : Term Y) (σ : Sub X Y) (ρ : Carrier (Env X))
  → [⟦ t [ σ ] ⟧] ⟨$⟩ ρ ≈a [⟦ t ⟧] ⟨$⟩ ([⟦ σ ⟧]s ρ)

substitution (g x) σ ρ = refl
substitution (node f ts) σ ρ = cong (Interp A)(≡.refl , λ i → substitution (ts i) σ ρ)

```

6.5 Compatibility of terms

We now prove two important facts about term operations. The first of these, which is used very often in the sequel, asserts that every term commutes with every homomorphism.

```

module _ {X : Type χ} {A : Algebra α ρa} {B : Algebra β ρb} (hh : hom A B) where
  open Setoid D[ B ] using ( _≈_ ; refl )
  open SetoidReasoning D[ B ]
  private hfunc = | hh | ; h = _⟦$⟧_ hfunc
  open Environment A using () renaming ( [⟦_⟧] to [⟦_⟧]1 )
  open Environment B using () renaming ( [⟦_⟧] to [⟦_⟧]2 )

  comm-hom-term : (t : Term X) (a : X → U[ A ]) → h ([⟦ t ⟧]1 ⟨$⟩ a) ≈ [⟦ t ⟧]2 ⟨$⟩ (h ∘ a)
  comm-hom-term (g x) a = refl
  comm-hom-term (node f t) a = goal
  where
    goal : h ([⟦ node f t ⟧]1 ⟨$⟩ a) ≈ ([⟦ node f t ⟧]2 ⟨$⟩ (h ∘ a))
    goal = begin h ([⟦ node f t ⟧]1 ⟨$⟩ a) ≈⟨ compatible || hh || ⟩
      (f ^ B)(λ i → h ([⟦ t i ⟧]1 ⟨$⟩ a)) ≈⟨ cong (Interp B)(≡.refl , λ i → comm-hom-term (t i) a) ⟩
      (f ^ B)(λ i → [⟦ t i ⟧]2 ⟨$⟩ (h ∘ a)) ≈⟨ refl ⟩
      ([⟦ node f t ⟧]2 ⟨$⟩ (h ∘ a)) ■

```

7 Model Theory and Equational Logic

(cf. the `Varieties.Func.SoundAndComplete` module of the `Agda Universal Algebra Library`)

7.1 Basic definitions

Let S be a signature. By an *identity* or *equation* in S we mean an ordered pair of terms in a given context. For instance, if the context happens to be the type $X : \text{Type } \chi$, then an equation will be a pair of inhabitants of the domain of term algebra $\mathbf{T} X$.

We define an equation in Agda using the following record type with fields denoting the left-hand and right-hand sides of the equation, along with an equation “context” representing

the underlying collection of variable symbols (cf. Andreas Abel's formal proof of Birkhoff's completeness theorem).

```
record Eq : Type (ov χ) where
  constructor _≈·_
  field
    {cxt} : Type χ
    lhs : Term cxt
    rhs : Term cxt

open Eq public
```

We now define a type representing the notion of an equation $p \approx \cdot q$ holding (when p and q are interpreted) in algebra \mathbf{A} .

If \mathbf{A} is an S -algebra we say that \mathbf{A} *satisfies* $p \approx q$ provided for all environments $\rho : X \rightarrow |\mathbf{A}|$ (assigning values in the domain of \mathbf{A} to variable symbols in X) we have $\llbracket p \rrbracket \langle \$ \rangle \rho \approx \llbracket q \rrbracket \langle \$ \rangle \rho$. In this situation, we write $\mathbf{A} \models (p \approx \cdot q)$ and say that \mathbf{A} *models* the identity $p \approx q$.

If \mathcal{K} is a class of algebras, all of the same signature, we write $\mathcal{K} \models (p \approx \cdot q)$ if, for every $\mathbf{A} \in \mathcal{K}$, we have $\mathbf{A} \models (p \approx \cdot q)$.

Because a class of structures has a different type than a single structure, we must use a slightly different syntax to avoid overloading the relations \models and \approx . As a reasonable alternative to what we would normally express informally as $\mathcal{K} \models p \approx q$, we have settled on $\mathcal{K} \models (p \approx \cdot q)$ to denote this relation. To reiterate, if \mathcal{K} is a class of S -algebras, we write $\mathcal{K} \models (p \approx \cdot q)$ provided every $\mathbf{A} \in \mathcal{K}$ satisfies $\mathbf{A} \models (p \approx \cdot q)$.

```
_|=_ : (A : Algebra α ρa) (term-identity : Eq{χ}) → Type _
A ⊨ (p ≈· q) = Equal p q where open Environment A

_||=_ : Pred (Algebra α ρa) ℓ → Eq{χ} → Type (ℓ ⊔ χ ⊔ ov(α ⊔ ρa))
ℳ ⊨ equ = ∀ A → ℳ A → A ⊨ equ
```

We denote by $\mathbf{A} \models \mathcal{E}$ the assertion that the algebra \mathbf{A} models every equation in a collection \mathcal{E} of equations.

```
_|=_ : (A : Algebra α ρa) → {ℓ : Level} {I : Type ℓ} → (I → Eq{χ}) → Type _
A ⊨ ℰ = ∀ i → Equal (lhs (ℰ i)) (rhs (ℰ i)) where open Environment A
```

7.2 Equational theories and models

If \mathcal{K} denotes a class of structures, then $\text{Th } \mathcal{K}$ represents the set of identities modeled by the members of \mathcal{K} .

```
Th : {X : Type χ} → Pred (Algebra α ρa) ℓ → Pred (Term X × Term X) _
Th ℳ = λ (p , q) → ℳ ⊨ (p ≈· q)

Mod : {X : Type χ} → Pred (Term X × Term X) ℓ → Pred (Algebra α ρa) _
Mod ℰ A = ∀ {p q} → (p , q) ∈ ℰ → Equal p q where open Environment A
```

7.3 The entailment relation

Based on Andreas Abel's Agda formalization of Birkhoff's completeness theorem.)


```

module _ {χ ι : Level} where

data _⊢_▷_≈_ {l : Type ι} {ℳ : l → Eq} (X : Type χ) (p q : Term X) → Type (ι ⊔ ov χ) where
  hyp : ∀ i → let p ≈ · q = ℳ i in ℳ ⊢ _▷ p ≈ q
  app : ∀ {ps qs} → (∀ i → ℳ ⊢ Γ▷ ps i ≈ qs i) → ℳ ⊢ Γ▷ (node f ps) ≈ (node f qs)
  sub : ∀ {p q} → ℳ ⊢ Δ▷ p ≈ q → ∀ (σ : Sub Γ Δ) → ℳ ⊢ Γ▷ (p [ σ ]) ≈ (q [ σ ])

  ⊢refl : ∀ {p} → ℳ ⊢ Γ▷ p ≈ p
  ⊢sym : ∀ {p q : Term Γ} → ℳ ⊢ Γ▷ p ≈ q → ℳ ⊢ Γ▷ q ≈ p
  ⊢trans : ∀ {p q r : Term Γ} → ℳ ⊢ Γ▷ p ≈ q → ℳ ⊢ Γ▷ q ≈ r → ℳ ⊢ Γ▷ p ≈ r

  ⊢≈IsEquiv : {X : Type χ} {l : Type ι} {ℳ : l → Eq} → IsEquivalence (ℳ ⊢ X▷_≈_)
  ⊢≈IsEquiv = record { refl = ⊢refl ; sym = ⊢sym ; trans = ⊢trans }

```

7.4 Soundness

In any model \mathbf{A} that satisfies the equations \mathcal{E} , derived equality is actual equality (cf. Andreas Abel's Agda formalization of Birkhoff's completeness theorem.)

```

module Soundness {χ α ι : Level} {l : Type ι} {ℳ : l → Eq{χ}}
  (A : Algebra α ρα) – We assume an algebra A
  (V : A ⊢ ℳ) – that models all equations in ℳ.
  where

  open SetoidReasoning D[ A ]
  open Environment A
  open IsEquivalence using ( refl ; sym ; trans )

  sound : ∀ {p q} → ℳ ⊢ Γ▷ p ≈ q → A ⊢ (p ≈ · q)
  sound (hyp i) = V i
  sound (app {f = f} es) ρ = cong (Interp A) (≡.refl , λ i → sound (es i) ρ)
  sound (sub {p = p} {q} Epq σ) ρ =
    begin
      [ p [ σ ] ] <$> ρ ≈< substitution p σ ρ >
      [ p ] <$> [ σ ]s ρ ≈< sound Epq ([ σ ]s ρ) >
      [ q ] <$> [ σ ]s ρ ≈< substitution q σ ρ >
      [ q [ σ ] ] <$> ρ ■

  sound (⊢refl {p = p}) = refl EqualsEquiv {x = p}
  sound (⊢sym {p = p} {q} Epq) = sym EqualsEquiv {x = p} {q} (sound Epq)
  sound (⊢trans {p = p} {q} {r} Epq Eqr) = trans EqualsEquiv {i = p} {q} {r} (sound Epq) (sound Eqr)

```

8 The Closure Operators H, S, P and V

Fix a signature S , let \mathcal{K} be a class of S -algebras, and define

- $H \mathcal{K}$ = algebras isomorphic to a homomorphic image of a member of \mathcal{K} ;
- $S \mathcal{K}$ = algebras isomorphic to a subalgebra of a member of \mathcal{K} ;
- $P \mathcal{K}$ = algebras isomorphic to a product of members of \mathcal{K} .

A straight-forward verification confirms that H , S , and P are *closure operators* (expansive, monotone, and idempotent). A class \mathcal{K} of S -algebras is said to be *closed under the taking of homomorphic images* provided $H \mathcal{K} \subseteq \mathcal{K}$. Similarly, \mathcal{K} is *closed under the taking of*

subalgebras (resp., *arbitrary products*) provided $S \mathcal{K} \subseteq \mathcal{K}$ (resp., $P \mathcal{K} \subseteq \mathcal{K}$). The operators H, S, and P can be composed with one another repeatedly, forming yet more closure operators.

An algebra is a homomorphic image (resp., subalgebra; resp., product) of every algebra to which it is isomorphic. Thus, the class $H \mathcal{K}$ (resp., $S \mathcal{K}$; resp., $P \mathcal{K}$) is closed under isomorphism.

A *variety* is a class of S -algebras that is closed under the taking of homomorphic images, subalgebras, and arbitrary products. To represent varieties we define types for the closure operators H, S, and P that are composable. Separately, we define a type V which represents closure under all three operators, H, S, and P.

8.1 Basic definitions

We now define the type H to represent classes of algebras that include all homomorphic images of algebras in the class—i.e., classes that are closed under the taking of homomorphic images—the type S to represent classes of algebras that closed under the taking of subalgebras, and the type P to represent classes of algebras closed under the taking of arbitrary products.

```

H :  $\forall \ell \rightarrow \text{Pred}(\text{Algebra } \alpha \rho^a) (\alpha \sqcup \rho^a \sqcup \text{ov } \ell) \rightarrow \text{Pred}(\text{Algebra } \beta \rho^b) (\beta \sqcup \rho^b \sqcup \text{ov}(\alpha \sqcup \rho^a \sqcup \ell))$ 
H { $\alpha$ }{ $\rho^a$ } _  $\mathcal{K}$  B =  $\Sigma [A \in \text{Algebra } \alpha \rho^a] A \in \mathcal{K} \times B \text{ IsHomImageOf } A$ 

S :  $\forall \ell \rightarrow \text{Pred}(\text{Algebra } \alpha \rho^a) (\alpha \sqcup \rho^a \sqcup \text{ov } \ell) \rightarrow \text{Pred}(\text{Algebra } \beta \rho^b) (\beta \sqcup \rho^b \sqcup \text{ov}(\alpha \sqcup \rho^a \sqcup \ell))$ 
S { $\alpha$ }{ $\rho^a$ } _  $\mathcal{K}$  B =  $\Sigma [A \in \text{Algebra } \alpha \rho^a] A \in \mathcal{K} \times B \leq A$ 

P :  $\forall \ell \iota \rightarrow \text{Pred}(\text{Algebra } \alpha \rho^a) (\alpha \sqcup \rho^a \sqcup \text{ov } \ell) \rightarrow \text{Pred}(\text{Algebra } \beta \rho^b) (\beta \sqcup \rho^b \sqcup \text{ov}(\alpha \sqcup \rho^a \sqcup \ell \sqcup \iota))$ 
P { $\alpha$ }{ $\rho^a$ } _  $\iota$   $\mathcal{K}$  B =  $\Sigma [I \in \text{Type } \iota] (\Sigma [A \in (I \rightarrow \text{Algebra } \alpha \rho^a)] (\forall i \rightarrow A i \in \mathcal{K}) \times (B \cong \prod A))$ 

```

A class \mathcal{K} of S -algebras is called a *variety* if it is closed under each of the closure operators H, S, and P defined above. The corresponding closure operator is often denoted \mathbb{V} or \mathcal{V} , but we will denote it by V .

```

module _ { $\alpha \rho^a \beta \rho^b \gamma \rho^c \delta \rho^d : \text{Level}$ } where
  private a =  $\alpha \sqcup \rho^a$  ; b =  $\beta \sqcup \rho^b$  ; c =  $\gamma \sqcup \rho^c$  ; d =  $\delta \sqcup \rho^d$ 

  V :  $\forall \ell \iota \rightarrow \text{Pred}(\text{Algebra } \alpha \rho^a) (a \sqcup \text{ov } \ell) \rightarrow \text{Pred}(\text{Algebra } \delta \rho^d) (d \sqcup \text{ov}(a \sqcup b \sqcup c \sqcup \ell \sqcup \iota))$ 
  V  $\ell \iota$   $\mathcal{K}$  =  $H\{\gamma\}\{\rho^c\}\{\delta\}\{\rho^d\} (a \sqcup b \sqcup \ell \sqcup \iota) (S\{\beta\}\{\rho^b\} (a \sqcup \ell \sqcup \iota) (P \ell \iota \mathcal{K}))$ 

  module _ { $\alpha \rho^a \ell : \text{Level}$ } ( $\mathcal{K} : \text{Pred}(\text{Algebra } \alpha \rho^a) (\alpha \sqcup \rho^a \sqcup \text{ov } \ell)$ )
    (A : Algebra ( $\alpha \sqcup \rho^a \sqcup \ell$ ) ( $\alpha \sqcup \rho^a \sqcup \ell$ )) where
      private  $\iota = \text{ov}(\alpha \sqcup \rho^a \sqcup \ell)$ 

      V- $\cong$ -lc : Lift-Alg A  $\iota \iota \in V\{\beta = \iota\}\{\iota\} \ell \iota \mathcal{K} \rightarrow A \in V\{\gamma = \iota\}\{\iota\} \ell \iota \mathcal{K}$ 
      V- $\cong$ -lc (A' , spA' , lAimgA') = A' , (spA' , AimgA')
      where
        AimgA' : A IsHomImageOf A'
        AimgA' = Lift-HomImage-lemma lAimgA'

```

8.2 Properties

8.2.1 Idempotence of S

S is a closure operator. The facts that S is monotone and expansive won't be needed, so we omit the proof of these facts. However, we will make use of idempotence of S, so we prove that property as follows.

S-idem : $\{\mathcal{K} : \text{Pred } (\text{Algebra } \alpha \rho^a)(\alpha \sqcup \rho^a \sqcup \text{ov } \ell)\}$
 $\rightarrow \mathbf{S}\{\beta = \gamma\}\{\rho^c\} (\alpha \sqcup \rho^a \sqcup \ell) (\mathbf{S}\{\beta = \beta\}\{\rho^b\} \ell \mathcal{K}) \subseteq \mathbf{S}\{\beta = \gamma\}\{\rho^c\} \ell \mathcal{K}$

S-idem $(\mathbf{A}, (\mathbf{B}, \text{sB}, \mathbf{A} \leq \mathbf{B}), \mathbf{x} \leq \mathbf{A}) = \mathbf{B}, (\text{sB}, \leq\text{-trans } \mathbf{x} \leq \mathbf{A} \mathbf{A} \leq \mathbf{B})$

8.2.2 Algebraic invariance of \models

The binary relation \models would be practically useless if it were not an *algebraic invariant* (i.e., invariant under isomorphism). Let us now verify that the models relation we defined above has this essential property.

```
module _ {X : Type χ} {A : Algebra α ρa} (B : Algebra β ρb) (p q : Term X) where

  |<-I-invar : A ⊨ (p ≈ · q) → A ≅ B → B ⊨ (p ≈ · q)
  |<-I-invar Apq (mkiso fh gh f~g g~f) ρ =
    begin
      [p]₂ <$> ρ ≈~< cong [p]₂ (λ x → f~g (ρ x)) >
      [p]₂ <$> (ff ∘ (g ∘ ρ)) ≈~< comm-hom-term fh p (g ∘ ρ) >
      ff ([p]₁ <$> (g ∘ ρ)) ≈< cong | fh | (Apq (g ∘ ρ)) >
      ff ([q]₁ <$> (g ∘ ρ)) ≈< comm-hom-term fh q (g ∘ ρ) >
      [q]₂ <$> (ff ∘ (g ∘ ρ)) ≈< cong [q]₂ (λ x → f~g (ρ x)) >
      [q]₂ <$> ρ ■
    where
      open Environment A using () renaming ([_] to [ ]₁)
      open Environment B using () renaming ([_] to [ ]₂)
      open SetoidReasoning D[ B ]
      private ff = _<$>_ | fh | ; g = _<$>_ | gh |
```

8.2.3 Subalgebraic invariance of \models

Identities modeled by an algebra \mathbf{A} are also modeled by every subalgebra of \mathbf{A} , which fact can be formalized as follows.

```
module _ {X : Type χ} {A : Algebra α ρa} {B : Algebra β ρb} {p q : Term X} where

  |<-S-invar : A ⊨ (p ≈ · q) → B ≤ A → B ⊨ (p ≈ · q)
  |<-S-invar Apq B ≤ A b = goal
    where
      open Environment A using () renaming ([_] to [ ]₁)
      open Setoid D[ A ] using ( _≈_ )
      open SetoidReasoning D[ A ]
      open Environment B using () renaming ([_] to [ ]₂)
      open Setoid D[ B ] using ( _≈_ to _≈₂_ )
      hh : hom B A
      hh = | B ≤ A |
      h = _<$>_ | hh |
      ξ : ∀ b → h ([p]₂ <$> b) ≈ h ([q]₂ <$> b)
      ξ b = begin
        h ([p]₂ <$> b) ≈< comm-hom-term hh p b >
        [p]₁ <$> (h ∘ b) ≈< Apq (h ∘ b) >
        [q]₁ <$> (h ∘ b) ≈~< comm-hom-term hh q b >
```

```

      h ([ q ]2 <$> b) ■
goal : [ p ]2 <$> b ≈2 [ q ]2 <$> b
goal = [ B≤A ] (ξ b)

```

8.2.4 Product invariance of \models

An identity satisfied by all algebras in an indexed collection is also satisfied by the product of algebras in that collection.

```

module _ {X : Type} {I : Type} {ℓ} {A : I → Algebra α ρa} {p q : Term X} where

  ⊢-P-invar : (∀ i → A i ⊢ (p ≈* q)) → [ A ] ⊢ (p ≈* q)
  ⊢-P-invar A p q a = goal
  where
    open Environment ([ A ]) using () renaming ([ ] to [ ]1)
    open Environment using ([ ] )
    open Setoid D[ [ A ] ] using ([ ≈- ])
    open SetoidReasoning D[ [ A ] ]
    ξ : (λ i → ([ A i ] p) <$> (λ x → (a x) i)) ≈ (λ i → ([ A i ] q) <$> (λ x → (a x) i))
    ξ = λ i → A p q i (λ x → (a x) i)
    goal : [ p ]1 <$> a ≈ [ q ]1 <$> a
    goal = begin
      [ p ]1 <$> a ≈ (interp-prod A p a)
      (λ i → ([ A i ] p) <$> (λ x → (a x) i)) ≈ (ξ)
      (λ i → ([ A i ] q) <$> (λ x → (a x) i)) ≈ (interp-prod A q a)
      [ q ]1 <$> a ■

```

8.2.5 $\mathbf{PS} \subseteq \mathbf{SP}$

Another important fact we will need about the operators S and P is that a product of subalgebras of algebras in a class \mathcal{K} is a subalgebra of a product of algebras in \mathcal{K} . We denote this inclusion by $\mathbf{PS} \subseteq \mathbf{SP}$, which we state and prove as follows.

```

module _ {K : Pred (Algebra α ρa) (α ⊔ ρa ⊔ ov ℓ)} where
  private
    a = α ⊔ ρa
    oal = ov (a ⊔ ℓ)

  PS⊆SP : P (a ⊔ ℓ) oal (S {β = α} {ρa} ℓ K) ⊆ S oal (P ℓ oal K)
  PS⊆SP {B} (I, (A, sA, B ≅ [A])) = Goal
  where
    B : I → Algebra α ρa
    B i = [ sA i ]
    kB : (i : I) → B i ∈ K
    kB i = fst [ sA i ]
    [A] ≤ [B] : [ A ] ≤ [ B ]
    [A] ≤ [B] = [ ≤ ] λ i → snd [ sA i ]
    Goal : B ∈ S {β = oal} {oal} oal (P {β = oal} {oal} ℓ oal K)
    Goal = [ B, (I, (B, (kB, ≅-refl))), (≅-trans-≤ B ≅ [A] [A] ≤ [B]) ]

```

8.3 Identity preservation

The classes \mathcal{H} , \mathcal{S} , \mathcal{P} , and \mathcal{V} all satisfy the same set of equations. We will only use a subset of the inclusions used to prove this fact. (For a complete proof, see the `Varieties.Func.Preservation` module of the `Agda Universal Algebra Library`.)

8.3.1 \mathcal{H} preserves identities

First we prove that the closure operator \mathcal{H} is compatible with identities that hold in the given class.

```

module _ {X : Type} {K : Pred (Algebra α ρa) (α ⊔ ρa ⊔ ov ℓ)} {p q : Term X} where

H-id1 : K ⊨ (p ≈ · q) → (H {β = α} {ρa} ℓ K) ⊨ (p ≈ · q)
H-id1 σ B (A , kA , BimgOfA) ρ = B⊨pq
  where
    IH : A ⊨ (p ≈ · q)
    IH = σ A kA
    open Environment A using () renaming ( [ ] to [ ]1 )
    open Environment B using ( [ ] )
    open Setoid B using ( ≈ )
    open SetoidReasoning B using ( ≈ )

    φ : hom A B
    φ = | BimgOfA |
    φE : IsSurjective | φ |
    φE = || BimgOfA ||
    φ-1 : U[ B ] → U[ A ]
    φ-1 = SurjInv | φ | φE

    ζ : ∀ x → (| φ | ( $ ) (φ-1 ∘ ρ) x) ≈ ρ x
    ζ = λ _ → InvlInverser φE

    B⊨pq : ([ p ] ( $ ) ρ) ≈ ([ q ] ( $ ) ρ)
    B⊨pq = begin
      [ p ] ( $ ) ρ
      ≈~ ( cong [ p ] ζ )
      [ p ] ( $ ) (λ x → (| φ | ( $ ) (φ-1 ∘ ρ) x)) ≈~ ( comm-hom-term φ p (φ-1 ∘ ρ) )
      | φ | ( $ ) ([ p ]1 ( $ ) (φ-1 ∘ ρ)) ≈~ ( cong | φ | (IH (φ-1 ∘ ρ)) )
      | φ | ( $ ) ([ q ]1 ( $ ) (φ-1 ∘ ρ)) ≈~ ( comm-hom-term φ q (φ-1 ∘ ρ) )
      [ q ] ( $ ) (λ x → (| φ | ( $ ) (φ-1 ∘ ρ) x)) ≈~ ( cong [ q ] ζ )
      [ q ] ( $ ) ρ

```

8.3.2 \mathcal{S} preserves identities

```

S-id1 : K ⊨ (p ≈ · q) → (S {β = α} {ρa} ℓ K) ⊨ (p ≈ · q)
S-id1 σ B (A , kA , B≤A) = ⊨-S-invar {p = p} {q} (σ A kA) B≤A

```

The obvious converse is barely worth the bits needed to formalize it, but we will use it below, so let's prove it now.

```

S-id2 : S ℓ K ⊨ (p ≈ · q) → K ⊨ (p ≈ · q)
S-id2 Spq A kA = Spq A (A , (kA , ≤-reflexive))

```

8.3.3 P preserves identities

```

P-id1 : ∀ {ι} → ℳ ⊨ (p ≈ · q) → P {β = α} {ρa} ℓ ι ℳ ⊨ (p ≈ · q)
P-id1 σ A (I , A , kA , A ≅ ⊔ A) = ⊨-I-invar A p q IH (≅-sym A ≅ ⊔ A)
  where
  ih : ∀ i → A i ⊨ (p ≈ · q)
  ih i = σ (A i) (kA i)
  IH : ⊔ A ⊨ (p ≈ · q)
  IH = ⊨-P-invar A {p} {q} ih

```

8.3.4 V preserves identities

Finally, we prove the analogous preservation lemmas for the closure operator V .

```

module _ {X : Type} {χ} {ι : Level} {ℳ : Pred (Algebra α ρa) (α ⊔ ρa ⊔ ov ℓ)} {p q : Term X} where
  private
    aℓι = α ⊔ ρa ⊔ ℓ ⊔ ι

  V-id1 : ℳ ⊨ (p ≈ · q) → V ℓ ι ℳ ⊨ (p ≈ · q)
  V-id1 σ B (A , (⊔ A , p ⊔ A , A ≤ ⊔ A) , BimgA) =
    H-id1 {ℓ = aℓι} {ℳ = S aℓι (P {β = α} {ρa} ℓ ι ℳ)} {p = p} {q} spK ⊨ pq B (A , (spA , BimgA))
    where
      spA : A ∈ S aℓι (P {β = α} {ρa} ℓ ι ℳ)
      spA = ⊔ A , (p ⊔ A , A ≤ ⊔ A)
      spK ⊨ pq : S aℓι (P ℓ ι ℳ) ⊨ (p ≈ · q)
      spK ⊨ pq = S-id1 {ℓ = aℓι} {p = p} {q} (P-id1 {ℓ = ℓ} {ℳ = ℳ} {p = p} {q} σ)

```

8.3.5 Th ℳ ⊆ Th (V ℳ)

From V-id1 it follows that if \mathcal{K} is a class of algebras, then the set of identities modeled by the algebras in \mathcal{K} is contained in the set of identities modeled by the algebras in $V \mathcal{K}$. In other terms, $\text{Th } \mathcal{K} \subseteq \text{Th } (V \mathcal{K})$. We formalize this observation as follows.

```

classIds-⊆-VIds : ℳ ⊨ (p ≈ · q) → (p , q) ∈ Th (V ℓ ι ℳ)
classIds-⊆-VIds pKq A = V-id1 pKq A

```

9 Free Algebras

9.1 The absolutely free algebra $T X$

The term algebra $T X$ is *absolutely free* (or *universal*, or *initial*) for algebras in the signature S . That is, for every S -algebra A , the following hold.

1. Every function from X to $|A|$ lifts to a homomorphism from $T X$ to A .
2. The homomorphism that exists by item 1 is unique.

We now prove this in Agda, starting with the fact that every map from X to $|A|$ lifts to a map from $|T X|$ to $|A|$ in a natural way, by induction on the structure of the given term.

```

module _ {X : Type} {A : Algebra α ρa} (h : X → U [ A ]) where
  open Setoid D [ A ] using ( _≈_ ; reflexive ; refl ; trans )

```

```

free-lift :  $\mathbb{U}[\mathbf{T} \times] \rightarrow \mathbb{U}[\mathbf{A}]$ 
free-lift ( $\underline{g} \ x$ ) =  $\underline{h} \ x$ 
free-lift ( $\text{node } f \ t$ ) = ( $f \hat{\ } \mathbf{A}$ ) ( $\lambda \ i \rightarrow \text{free-lift } (t \ i)$ )

free-lift-func :  $\mathbb{D}[\mathbf{T} \times] \rightarrow \mathbb{D}[\mathbf{A}]$ 
free-lift-func  $\langle \$ \rangle x = \text{free-lift } x$ 
cong free-lift-func = flcong
where
  flcong :  $\forall \{s \ t\} \rightarrow s \doteq t \rightarrow \text{free-lift } s \approx \text{free-lift } t$ 
  flcong ( $\_ \doteq \_ \text{.rfl } x$ ) = reflexive ( $\equiv \text{.cong } h \ x$ )
  flcong ( $\_ \doteq \_ \text{.gnl } x$ ) = cong (Interp  $\mathbf{A}$ ) ( $\equiv \text{.refl } , (\lambda \ i \rightarrow \text{flcong } (x \ i))$ )

```

Naturally, at the base step of the induction, when the term has the form generator x , the free lift of h agrees with h . For the inductive step, when the given term has the form $\text{node } f \ t$, the free lift is defined as follows: Assuming (the induction hypothesis) that we know the image of each subterm $t \ i$ under the free lift of h , define the free lift at the full term by applying $f \hat{\ } \mathbf{A}$ to the images of the subterms.

The free lift so defined is a homomorphism by construction. Indeed, here is the trivial proof.

```

lift-hom : hom ( $\mathbf{T} \times$ )  $\mathbf{A}$ 
lift-hom = free-lift-func , hhom
where
  hfunc :  $\mathbb{D}[\mathbf{T} \times] \rightarrow \mathbb{D}[\mathbf{A}]$ 
  hfunc = free-lift-func

hcomp : compatible-map ( $\mathbf{T} \times$ )  $\mathbf{A}$  free-lift-func
hcomp {f}{a} = cong (Interp  $\mathbf{A}$ ) ( $\equiv \text{.refl } , (\lambda \ i \rightarrow (\text{cong free-lift-func})\{a \ i\} \doteq \text{isRefl})$ )

hhom : IsHom ( $\mathbf{T} \times$ )  $\mathbf{A}$  hfunc
hhom = record { compatible =  $\lambda \{f\}\{a\} \rightarrow \text{hcomp}\{f\}\{a\}$  }

module  $\_ \{X : \text{Type } \chi\} \{A : \text{Algebra } \alpha \ \rho^a\}$  where
  open Setoid  $\mathbb{D}[\mathbf{A}]$  using ( $\_ \approx \_ ; \text{refl}$ )
  open Environment  $\mathbf{A}$  using ( $\llbracket \_ \rrbracket$ )

  free-lift-interp : ( $\eta : X \rightarrow \mathbb{U}[\mathbf{A}]$ )( $p : \text{Term } X$ )  $\rightarrow \llbracket p \rrbracket \langle \$ \rangle \eta \approx (\text{free-lift } \{A = \mathbf{A}\} \eta) \ p$ 
  free-lift-interp  $\eta$  ( $\underline{g} \ x$ ) = refl
  free-lift-interp  $\eta$  ( $\text{node } f \ t$ ) = cong (Interp  $\mathbf{A}$ ) ( $\equiv \text{.refl } , (\text{free-lift-interp } \eta) \circ t$ )

```

9.2 The relatively free algebra \mathbb{F}

We now define the algebra $\mathbb{F}[X]$, which plays the role of the relatively free algebra, along with the natural epimorphism $\text{epiF} : \text{epi } (\mathbf{T} \times) \mathbb{F}[X]$ from $\mathbf{T} \times$ to $\mathbb{F}[X]$.

```

module FreeAlgebra { $\chi : \text{Level}$ }{ $\iota : \text{Level}$ }{ $\mathbf{I} : \text{Type } \iota$ }{ $\mathcal{G} : \mathbf{I} \rightarrow \text{Eq}$ } where
  open Algebra

  FreeDomain :  $\text{Type } \chi \rightarrow \text{Setoid } \_ \_$ 
  FreeDomain  $X = \text{record } \{ \text{Carrier} = \text{Term } X$ 
    ;  $\_ \approx \_ = \mathcal{G} \vdash X \triangleright \_ \approx \_$ 
    ;  $\text{isEquivalence} = \vdash \triangleright \approx \text{isEquiv} \}$ 

```

The interpretation of an operation is simply the operation itself. This works since $\mathcal{E} \vdash X$ $\triangleright \approx$ is a congruence.

```
FreeInterp : ∀ {X} → ⟨ S ⟩ (FreeDomain X) → FreeDomain X
FreeInterp ($) (f , ts) = node f ts
cong FreeInterp (≡.refl , h) = app h

ℱ[ ] : Type χ → Algebra (ov χ) (ι ⊔ ov χ)
Domain ℱ[ X ] = FreeDomain X
Interp ℱ[ X ] = FreeInterp
```

9.3 Basic properties of free algebras

In the code below, X will play the role of an arbitrary collection of variables; it would suffice to take X to be the cardinality of the largest algebra in \mathcal{K} , but since we don't know that cardinality, we leave X arbitrary for now.

```
module FreeHom (χ : Level) {K : Pred(Algebra α ρa) (α ⊔ ρa ⊔ ov ℓ)} where
  private ι = ov(χ ⊔ α ⊔ ρa ⊔ ℓ)
  open Eq

  J : Type ι - indexes the collection of equations modeled by K
  J = Σ[ eq ∈ Eq{χ} ] K || (lhs eq ≈ · (rhs eq))

  E : J → Eq
  E (eqv , p) = eqv

  E ⊢ ThK : (X : Type χ) → ∀{p q} → E ⊢ X ⊔ p ≈ q → K || (p ≈ · q)
  E ⊢ [ X ] ⊔ ThK x A kA = sound (λ i ρ → || i || A kA ρ) x
    where open Soundness E A
  open FreeAlgebra {ι = ι}{I = J} E using ( ℱ[ ] )
```

9.3.1 The natural epimorphism from $\mathbf{T} X$ to $\mathbb{F}[X]$

Next we define an epimorphism from $\mathbf{T} X$ onto the relatively free algebra $\mathbb{F}[X]$. Of course, the kernel of this epimorphism will be the congruence of $\mathbf{T} X$ defined by identities modeled by $(S \mathcal{K}, \text{ hence } \mathcal{K})$.

```
epiℱ[ ] : (X : Type χ) → epi (T X) ℱ[ X ]
epiℱ[ X ] = h , hepi
  where
    open Algebra (T X) using () renaming (Domain to TX)
    open Setoid TX using () renaming ( _≈_ to _≈0_ ; refl to refl0 )
    open Algebra ℱ[ X ] using () renaming ( Domain to F )
    open Setoid F using () renaming ( _≈_ to _≈1_ ; refl to refl1 )
    open _≐_

    c : ∀ {x y} → x ≈0 y → x ≈1 y
    c (rfl {x}{y} ≡.refl) = refl1
    c (gnl {f}{s}{t} x) = cong (Interp ℱ[ X ]) (≡.refl , c ∘ x)

    h : TX → F
    h = record { f = id ; cong = c }
```



```

hepi : IsEpi (T X) F[ X ] h
compatible (isHom hepi) = cong h refl0
isSurjective hepi {y} = eq y refl1

homF[ ] : (X : Type  $\chi$ ) → hom (T X) F[ X ]
homF[ X ] = IsEpi.HomReduct || epiF[ X ] ||

homF[ ]-is-epic : (X : Type  $\chi$ ) → IsSurjective | homF[ X ] |
homF[ X ]-is-epic = IsEpi.isSurjective (snd (epiF[ X ]))

```

9.3.2 The kernel of the natural epimorphism

```

class-models-kernel : ∀{X p q} → (p , q) ∈ ker | homF[ X ] | →  $\mathcal{K} \models (p \approx \cdot q)$ 
class-models-kernel {X = X}{p}{q} pKq =  $\mathcal{K} \vdash [X] \triangleright \text{Th } \mathcal{K} \text{ pKq}$ 

kernel-in-theory : {X : Type  $\chi$ } → ker | homF[ X ] | ⊆ Th (V  $\ell \iota \mathcal{K}$ )
kernel-in-theory {X = X} {p , q} pKq vkA x = classIds ⊆ VIds { $\ell = \ell$ } {p = p}{q}
  (class-models-kernel pKq) vkA x

module _ {X : Type  $\chi$ } {A : Algebra  $\alpha \rho^a$ } {sA : A ∈ S { $\beta = \alpha$ } { $\rho^a$ }  $\ell \mathcal{K}$ } where
  open Environment A using (Equal)
  kerF ⊆ Equal : ∀{p q} → (p , q) ∈ ker | homF[ X ] | → Equal p q
  kerF ⊆ Equal {p = p}{q} x = S-id1 { $\ell = \ell$ } {p = p}{q} ( $\mathcal{K} \vdash [X] \triangleright \text{Th } \mathcal{K} x$ ) A sA

 $\mathcal{K} \models \rightarrow \mathcal{K} \vdash$  : {X : Type  $\chi$ } → ∀{p q} →  $\mathcal{K} \models (p \approx \cdot q)$  →  $\mathcal{K} \vdash X \triangleright p \approx q$ 
 $\mathcal{K} \models \rightarrow \mathcal{K} \vdash$  {p = p} {q} pKq = hyp ((p ≈ · q) , pKq) where open  $\_ \vdash \_ \approx \_$  using (hyp)

```

9.3.3 The universal property

```

module _ {A : Algebra ( $\alpha \sqcup \rho^a \sqcup \ell$ ) ( $\alpha \sqcup \rho^a \sqcup \ell$ )}
  { $\mathcal{K}$  : Pred(Algebra  $\alpha \rho^a$ ) ( $\alpha \sqcup \rho^a \sqcup \text{ov } \ell$ )} where
  private  $\iota$  = ov( $\alpha \sqcup \rho^a \sqcup \ell$ )
  open FreeHom { $\ell = \ell$ } ( $\alpha \sqcup \rho^a \sqcup \ell$ ) { $\mathcal{K}$ }
  open FreeAlgebra { $\iota = \iota$ } { $\ell = \mathcal{J}$ }  $\mathcal{K}$  using (F[ ])
  open Setoid  $\mathbb{D}[A]$  using (trans ; sym ; refl) renaming (Carrier to A)

  F-ModTh-epi : A ∈ Mod (Th (V  $\ell \iota \mathcal{K}$ ))
    → epi F[ A ] A
  F-ModTh-epi A ∈ ModThK =  $\varphi$  , isEpi
  where
     $\varphi$  :  $\mathbb{D}[F[A]] \rightarrow \mathbb{D}[A]$ 
     $\_ \langle \$ \rangle \_ \varphi$  = free-lift{A = A} id
    cong  $\varphi$  {p} {q} pq = trans (sym (free-lift-interp{A = A} id p))
      (trans (A ∈ ModThK {p = p}{q} (kernel-in-theory pq) id)
        (free-lift-interp{A = A} id q))

    isEpi : IsEpi F[ A ] A  $\varphi$ 
    compatible (isHom isEpi) = cong (Interp A) ( $\equiv$ .refl , ( $\lambda \_ \rightarrow$  refl))
    isSurjective isEpi {y} = eq ( $\varphi$  y) refl

  F-ModTh-epi-lift : A ∈ Mod (Th (V  $\ell \iota \mathcal{K}$ )) → epi F[ A ] (Lift-Alg A  $\iota$ )
  F-ModTh-epi-lift A ∈ ModThK = o-epi (F-ModTh-epi ( $\lambda$  {p q} → A ∈ ModThK {p = p}{q})) ToLift-epi

```

10 Products of classes of algebras

We want to pair each (\mathbf{A}, ρ) (where $\rho : \mathbf{X} \rightarrow |\mathbf{A}|$) so that we can quantify over all algebras *and* all assignments of values in the domain $|\mathbf{A}|$ to variables in \mathbf{X} .

```

module _ (K : Pred (Algebra  $\alpha$   $\rho^a$ ) ( $\alpha \sqcup \rho^a \sqcup \ell$ )) {X : Type ( $\alpha \sqcup \rho^a \sqcup \ell$ )} where
  private  $\iota = \text{ov}(\alpha \sqcup \rho^a \sqcup \ell)$ 
  open FreeHom { $\ell = \ell$ } ( $\alpha \sqcup \rho^a \sqcup \ell$ ) {K}
  open FreeAlgebra { $\iota = \iota$ } { $\ell = \ell$ }  $\mathcal{F}$  using (  $\mathbb{F}[\_]$  )
  open Environment using ( Env )

   $\mathcal{J}^+ : \text{Type } \iota$ 
   $\mathcal{J}^+ = \Sigma [ \mathbf{A} \in (\text{Algebra } \alpha \rho^a) ] (\mathbf{A} \in \mathbf{S} \ell \mathcal{K}) \times (\text{Carrier } (\text{Env } \mathbf{A} \mathbf{X}))$ 

   $\mathcal{A}^+ : \mathcal{J}^+ \rightarrow \text{Algebra } \alpha \rho^a$ 
   $\mathcal{A}^+ i = | i |$ 

   $\mathcal{C} : \text{Algebra } \iota \iota$ 
   $\mathcal{C} = \prod \mathcal{A}^+$ 

```

Next we define a useful type, `skEqual`, which we use to represent a term identity $p \approx q$ for any given $i = (\mathbf{A}, s\mathbf{A}, \rho)$ (where \mathbf{A} is an algebra, $s\mathbf{A} : \mathbf{A} \in \mathbf{S} \mathcal{K}$ is a proof that \mathbf{A} belongs to $\mathbf{S} \mathcal{K}$, and ρ is a mapping from \mathbf{X} to the domain of \mathbf{A}). Then we prove `AllEqual \subseteq ker \mathbb{F}` which asserts that if the identity $p \approx q$ holds in all $\mathbf{A} \in \mathbf{S} \mathcal{K}$ (for all environments), then $p \approx q$ holds in the relatively free algebra $\mathbb{F}[\mathbf{X}]$; equivalently, the pair (p, q) belongs to the kernel of the natural homomorphism from $\mathbf{T} \mathbf{X}$ onto $\mathbb{F}[\mathbf{X}]$. We will use this fact below to prove that there is a monomorphism from $\mathbb{F}[\mathbf{X}]$ into \mathcal{C} , and thus $\mathbb{F}[\mathbf{X}]$ is a subalgebra of \mathcal{C} , so belongs to $\mathbf{S} (\mathbf{P} \mathcal{K})$.

```

skEqual : (i :  $\mathcal{J}^+$ )  $\rightarrow \forall \{p q\} \rightarrow \text{Type } \rho^a$ 
skEqual i {p}{q} = [ [ p ] <$> snd || i ||  $\approx$  [ [ q ] <$> snd || i || ]
  where
    open Setoid  $\mathbb{D}[\mathcal{A}^+ i]$  using (  $\_ \approx \_$  )
    open Environment ( $\mathcal{A}^+ i$ ) using (  $[\_]$  )

AllEqual $\subseteq$ ker $\mathbb{F}$  :  $\forall \{p q\} \rightarrow (\forall i \rightarrow \text{skEqual } i \{p\}\{q\}) \rightarrow (p, q) \in \text{ker } | \text{hom } \mathbb{F}[\mathbf{X}] |$ 
AllEqual $\subseteq$ ker $\mathbb{F}$  {p} {q} x = Goal
  where
    open Algebra  $\mathbb{F}[\mathbf{X}]$  using () renaming ( Domain to F )
    open Setoid F using (  $\_ \approx \_$  )
     $\mathcal{SK} \models pq : \mathbf{S} \{ \beta = \alpha \} \{ \rho^a \} \ell \mathcal{K} \models (p \approx \cdot q)$ 
     $\mathcal{SK} \models pq \mathbf{A} s\mathbf{A} \rho = x (\mathbf{A}, s\mathbf{A}, \rho)$ 
    Goal :  $p \approx q$ 
    Goal =  $\mathcal{K} \models \rightarrow \mathcal{E} \vdash (\text{S-id2} \{ \ell = \ell \} \{ p = p \} \{ q \} \mathcal{SK} \models pq)$ 

hom $\mathcal{C} : \text{hom } (\mathbf{T} \mathbf{X}) \mathcal{C}$ 
hom $\mathcal{C} = \prod \text{-hom-co } \mathcal{A}^+ h$ 
  where
    h :  $\forall i \rightarrow \text{hom } (\mathbf{T} \mathbf{X}) (\mathcal{A}^+ i)$ 
    h i = lift-hom (snd || i ||)

ker $\mathbb{F} \subseteq \text{ker } \mathcal{C} : \text{ker } | \text{hom } \mathbb{F}[\mathbf{X}] | \subseteq \text{ker } | \text{hom } \mathcal{C} |$ 

```

```

kerF ⊆ kerC {p , q} pKq (A , sA , ρ) = Goal
  where
    open Setoid D[ A ] using ( _≈_ ; sym ; trans )
    open Environment A using ( [ ] )
    fl : ∀ t → [ t ] ($) ρ ≈ free-lift ρ t
    fl t = free-lift-interp {A = A} ρ t
    subgoal : [ p ] ($) ρ ≈ [ q ] ($) ρ
    subgoal = kerF ⊆ Equal {A = A} {sA} pKq ρ
    Goal : (free-lift {A = A} ρ p) ≈ (free-lift {A = A} ρ q)
    Goal = trans (sym (fl p)) (trans subgoal (fl q))

homF C : hom F[ X ] C
homF C = | HomFactor C homC homF[ X ] kerF ⊆ kerC homF[ X ]-is-epic |

kerC ⊆ kerF : ∀ {p q} → (p , q) ∈ ker | homC | → (p , q) ∈ ker | homF[ X ] |
kerC ⊆ kerF {p}{q} pKq = E⊢pq
  where
    pqEqual : ∀ i → skEqual i {p}{q}
    pqEqual i = goal
      where
        open Environment (A+ i) using () renaming ( [ ] to [ ]i )
        open Setoid D[ A+ i ] using ( _≈_ ; sym ; trans )
        goal : [ p ]i ($) snd || i || ≈ [ q ]i ($) snd || i ||
        goal = trans (free-lift-interp {A = | i |} (snd || i ||) p)
          (trans (pKq i) (sym (free-lift-interp {A = | i |} (snd || i ||) q)))
    E⊢pq : C ⊢ X ▷ p ≈ q
    E⊢pq = AllEqual ⊆ kerF pqEqual

monF C : mon F[ X ] C
monF C = | homF C , isMon
  where
    isMon : IsMon F[ X ] C | homF C |
    isHom isMon = || homF C ||
    isInjective isMon {p} {q} φpq = kerC ⊆ kerF φpq

```

Now that we have proved the existence of a monomorphism from $F[X]$ to C we are in a position to prove that $F[X]$ is a subalgebra of C , so belongs to $S(P\mathcal{K})$. In fact, we will show that $F[X]$ is a subalgebra of the *lift* of C , denoted ℓC .

```

F ≤ C : F[ X ] ≤ C
F ≤ C = mon → ≤ monF C

SPF : F[ X ] ∈ S ι (P ℓ ι K)
SPF = S-idem SSPF
  where
    PS C : C ∈ P (α ⊔ ρa ⊔ ℓ) ι (S ℓ K)
    PS C = J+ , (A+ , ((λ i → fst || i ||) , ≅-refl))
    SP C : C ∈ S ι (P ℓ ι K)
    SP C = PS ⊆ SP {ℓ = ℓ} PS C
    SSPF : F[ X ] ∈ S ι (S ι (P ℓ ι K))
    SSPF = C , (SP C , F ≤ C)

```

11 The HSP Theorem

Finally, we are in a position to prove Birkhoff's celebrated variety theorem.

```

module _ {  $\mathcal{K} : \text{Pred}(\text{Algebra } \alpha \rho^a) (\alpha \sqcup \rho^a \sqcup \text{ov } \ell)$  } where
  private  $\iota = \text{ov}(\alpha \sqcup \rho^a \sqcup \ell)$ 
  open FreeHom {  $\ell = \ell$  } {  $\alpha \sqcup \rho^a \sqcup \ell$  } {  $\mathcal{K}$  }
  open FreeAlgebra {  $\iota = \iota$  } {  $\mathcal{I} = \mathcal{J}$  }  $\mathcal{G}$  using (  $\mathbb{F}[\_]$  )

  Birkhoff :  $\forall \mathbf{A} \rightarrow \mathbf{A} \in \text{Mod } (\text{Th } (\mathbf{V } \ell \iota \mathcal{K})) \rightarrow \mathbf{A} \in \mathbf{V } \ell \iota \mathcal{K}$ 
  Birkhoff  $\mathbf{A} \text{ ModThA} = \mathbf{V}\text{-}\cong\text{-lc}\{\alpha\}\{\rho^a\}\{\ell\} \mathcal{K} \mathbf{A} \text{ VIA}$ 
  where
    open Setoid  $\mathbb{D}[\mathbf{A}]$  using () renaming ( Carrier to  $\mathbf{A}$  )
    spFA :  $\mathbb{F}[\mathbf{A}] \in \mathbf{S}\{\iota\} \iota (\mathbf{P } \ell \iota \mathcal{K})$ 
    spFA =  $\text{SP}\mathbb{F}\{\ell = \ell\} \mathcal{K}$ 
    epiFA :  $\text{epi } \mathbb{F}[\mathbf{A}] (\text{Lift-Alg } \mathbf{A} \iota \iota)$ 
    epiFA =  $\mathbb{F}\text{-ModTh-epi-lift}\{\ell = \ell\} (\lambda \{p \ q\} \rightarrow \text{ModThA}\{p = p\}\{q\})$ 
    IAimgFA :  $\text{Lift-Alg } \mathbf{A} \iota \iota \text{ IsHomImageOf } \mathbb{F}[\mathbf{A}]$ 
    IAimgFA =  $\text{epi} \rightarrow \text{ontohom } \mathbb{F}[\mathbf{A}] (\text{Lift-Alg } \mathbf{A} \iota \iota) \text{ epiFA}$ 
    VIA :  $\text{Lift-Alg } \mathbf{A} \iota \iota \in \mathbf{V } \ell \iota \mathcal{K}$ 
    VIA =  $\mathbb{F}[\mathbf{A}] , \text{spFA} , \text{IAimgFA}$ 

```

The converse inclusion, $\mathbf{V } \mathcal{K} \subseteq \text{Mod } (\text{Th } (\mathbf{V } \mathcal{K}))$, is a simple consequence of the fact that Mod Th is a closure operator. Nonetheless, completeness demands that we formalize this inclusion as well, however trivial the proof.

```

module _ {  $\mathbf{A} : \text{Algebra } \alpha \rho^a$  } where
  Birkhoff-converse :  $\mathbf{A} \in \mathbf{V}\{\alpha\}\{\rho^a\}\{\alpha\}\{\rho^a\}\{\ell \iota \mathcal{K}\} \rightarrow \mathbf{A} \in \text{Mod}\{X = \bigcup[\mathbf{A}]\} (\text{Th } (\mathbf{V } \ell \iota \mathcal{K}))$ 
  Birkhoff-converse  $\text{vA } p\text{Thq} = p\text{Thq } \mathbf{A} \text{ vA}$ 

```

We have thus proved that every variety is an equational class.

Readers familiar with the classical formulation of the Birkhoff HSP theorem as an “if and only if” assertion might worry that the proof is still incomplete. However, recall that in the `Varieties.Func.Preservation` module we proved the following identity preservation lemma:

$\text{V-id1} : \mathcal{K} \models p \approx \cdot q \rightarrow \mathbf{V } \mathcal{K} \models p \approx \cdot q$

Thus, if \mathcal{K} is an equational class—that is, if \mathcal{K} is the class of algebras satisfying all identities in some set—then $\mathbf{V } \mathcal{K} \subseteq \mathcal{K}$. On the other hand, we proved that \mathbf{V}' is expansive in the `Varieties.Func.Closure` module:

$\text{V-expa} : \mathcal{K} \subseteq \mathbf{V } \mathcal{K}$

so $\mathcal{K} (= \mathbf{V } \mathcal{K} = \text{HSP } \mathcal{K})$ is a variety.

Taken together, `V-id1` and `V-expa` constitute formal proof that every equational class is a variety.

This completes the formal proof of Birkhoff's variety theorem.