A Synopsis of Morphoid Type Theory

David McAllester
TTI-Chicago
mcallester@ttic.edu

Abstract

Morphoid type theory (MTT) is a type-theoretic foundation for mathematics supporting the concept of isomorphism and the substitution of isomorphics. Unlike homotopy type theory (HoTT), which also supports isomorphism, morphoid type theory is a direct extension of classical predicate calculus and avoids the intuitionistic constructs of propositions-as-types, path induction and squashing. Although HoTT is capable of supporting classical inference, MTT's thoroughly classical treatment is expected to be more comfortable for those who take a Platonic or realist approach to the practice of mathematics.

1 Introduction

The central issue in both homotopy type theory (HoTT-Authors, 2013) and morphoid type theory (McAllester, 2014) is isomorphism. The notion of isomorphism in mathematics seems related to the notion of an application programming interface (API) in computer software. An API specifies what information and behavior an object provides. Two different implementations can produce identical behavior when interaction is restricted to that allowed by the API. For example, textbooks on real analysis typically start from axioms involving multiplication, addition, and ordering. Addition, multiplication and ordering define an abstract interface — the well formed statements about real numbers are limited to those that can be defined in terms of the operations of the interface. We can implement real numbers in different ways — as Dedekind cuts or Cauchy sequences. However, these different implementations provide identical behavior as viewed through the interface — the different implementations are isomorphic as ordered fields. The axioms of real analysis specify the reals up to isomorphism for ordered fields. Peano's axioms (the second order version) similarly specify the structure of the natural numbers up to isomorphism.

The general notion of isomorphism is best illustrated by considering dependent pair types. Here we will write a dependent pair type as $\mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x])$ where the instances of this type are the pairs $\mathbf{Pair}(x,y)$ where x is an instance of the type σ and y is an instance of $\tau[x]$. The type of directed graphs can be written as $\mathbf{PairOf}\,(\mathcal{N}{:}\mathbf{type},\ P{:}(\mathcal{N}\times\mathcal{N})\to\mathbf{Bool})$ where $\mathcal{N}$ is a type representing the set of nodes of the graph and P is a binary predicate on the nodes giving the edge relation. Two directed graphs $\mathbf{Pair}(\mathcal{N},P)$ and $\mathbf{Pair}(\mathcal{M},Q)$ are isomorphic if there exists a bijection from $\mathcal{N}$ to $\mathcal{M}$ that carries P to Q. Some bijections will carry P to Q while others will not. Two pairs $\mathbf{Pair}(x,y)$ and $\mathbf{Pair}(u,w)$ of a general dependent pair type $\mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x])$ are isomorphic if there is a σ-isomorphism from x to u that carries y to w. Some σ-isomorphisms from x to u will carry y to w while others will not. This implies that to define isomorphism at general dependent pairs types we need that for any type σ, and for any two isomorphic values x and u of type σ, we can define the full set of σ-isomorphisms from x to u. An interesting special case is the full set of σ-isomorphisms from x to x. This is the symmetry group of x.

Proceedings of the 14th Meeting on the Mathematics of Language (MoL 14), pages 75–86,
Chicago, USA, July 25–26, 2015.

Both Homotopy type theory (HoTT) and morphoid type theory (MTT) are intended as type-theoretic foundations for mathematics supporting a concept of isomorphism. HoTT is an extension of constructive logic while MTT is an extension of classical predicate calculus. More specifically, HoTT is a version of Martin Löf type theory (Martin-Löf, 1971; Coquand and Huet, 1988; Sambin and Smith, 1998) extended to includes Voevodsky's univalence axiom (HoTT-Authors, 2013). Martin-Löf type theory involves propositions-as-types and path induction, neither of which are used in MTT. To accommodate classical (nonconstructive) inference, HoTT can be extended with a version of the law of the excluded middle. However, even in the classical version propositions continue to be represented as types rather than Boolean-valued expressions. To accommodate classical inference HoTT also includes squashing — a technicality required to allow propositions-types to be treated more like Boolean-valued expressions. In MTT all propositions are Boolean-valued and there is no need for squashing.

Perhaps the most significant difference between HoTT and MTT involves the abstraction barrier imposed on types. In MTT two types are type-isomorphic if there exists a bijection between them. In MTT two types with the same cardinality (number of equivalence classes) cannot be distinguished by well-typed predicates on types. In HoTT, however, types with the same cardinality can still be distinguished by well-typed predicates. In HoTT two types are equivalent only when they have the same higher-order groupoid structure. For example, two graphs fail to be isomorphic unless the node types have the same higher order groupoid structure. This can be interpreted as implementation details of a type leaking from the abstraction barrier on types. This leakage interpretation is discussed more explicitly in section 3.

HoTT allows one to block the leakage of type implementations by squashing types to "sets". A set is a type whose internal groupoid structure is effectively suppressed. One can construct the type of topological space whose point types are sets. In this case we get the familiar notion isomorphism (homeomorphism) where two topological spaces are homeomorphic if there is *any* bijection between their points that identifies their open sets. We can then define the groupoid of topological spaces to be the category consisting of the topological spaces and the homeomorphisms between them. This is the "first order" groupoid of topological spaces. If we take the point types of topological spaces to be first order groupoids rather than sets, and restrict the point bijections to functors, we get the second order groupoid of topological spaces. We can then define a third order groupoid and so on. In HoTT we can even have ω-order groupoids.

In MTT the internal structure of types is approached in a different way. In MTT natural mappings are distinguished from general functions. For example, there is an isomorphism (a linear bijection) from a finite dimensional vector space to its dual. However, there is no natural isomorphism. Although not covered in this synopsis, MTT takes a function to be natural if it can be written as a lambda expression. Lambda expressions (natural functions) have commutation properties not shared by general functions. Two types σ and τ are cryptomorphic (in the sense of Birkoff or Rota) if there exists a pair of natural functions (lambda expressions) $f : \sigma \to \tau$ and $g : \tau \to \sigma$ such that $f \circ g$ and $g \circ f$ are both identity functions (viewed as functions on the isomorphism classes of σ and τ respectively). MTT does not attempt to handle higher order groupoid structure.

This synopsis of MTT is preliminary and many of the features described here go beyond the features covered by soundness proofs in version 4 of (McAllester, 2014). This synopsis should be viewed as a plan, or program, for the next version of (McAllester, 2014).

2 The Core Rules of Morphoid Type Theory

Morphoid type theory starts from the syntax and semantics of classical predicate calculus. In sorted first order logic every term has a sort and each function symbol f specifies the sorts of its arguments and the sort of its value. We write $f : \sigma_1 \times \cdots \times \sigma_n \to \tau$ to indicate that f is a function that takes n arguments of sort $\sigma_1, \ldots, \sigma_n$ respectively and which produces a value of sort τ. The syntax of sorted first order logic can be defined by the following grammar where function and predicate applications must sat-

$$\epsilon \vdash \mathbf{type}_j : \mathbf{type}_i \text{ for } j < i \qquad \frac{\Sigma \vdash \tau : \mathbf{type}_i \quad x \text{ not declared in } \Sigma}{\Sigma;\ x:\tau \vdash x:\tau} \qquad \frac{\Sigma \vdash \tau : \mathbf{type}_i \quad \Sigma \vdash \sigma : \mathbf{type}_i}{\Sigma \vdash (\tau \to \sigma) : \mathbf{type}_i} \qquad \frac{\Sigma \vdash f : \sigma \to \tau \quad \Sigma \vdash e : \sigma}{\Sigma \vdash f(e) : \tau}$$

$$\epsilon \vdash \mathbf{Bool} : \mathbf{type}_i \qquad \frac{\Sigma \vdash \Phi : \mathbf{Bool}}{\Sigma; \Phi \vdash \Phi} \qquad \frac{\Sigma; \Theta \vdash \Theta \quad \Sigma \vdash \Psi}{\Sigma; \Theta \vdash \Psi}$$

$$\frac{\Sigma \vdash \Phi : \mathbf{Bool} \quad \Sigma \vdash \Psi : \mathbf{Bool}}{\Sigma \vdash (\Phi \vee \Psi) : \mathbf{Bool}} \qquad \frac{\Sigma \vdash \Phi : \mathbf{Bool}}{\Sigma \vdash \neg\Phi : \mathbf{Bool}} \qquad \frac{\Sigma;\ x:\tau \vdash \Phi[x] : \mathbf{Bool}}{\Sigma \vdash (\forall x:\tau\ \Phi[x]) : \mathbf{Bool}} \qquad \frac{\Sigma \vdash w : \tau \quad \Sigma \vdash u : \tau}{\Sigma \vdash (w =_\tau u) : \mathbf{Bool}}$$

Figure 1: **Predicate Calculus Expressions**. Here $\mathbf{type}_0$, $\mathbf{type}_1$, $\mathbf{type}_2$, ... are distinct constants and ϵ is a constant denoting the empty context. The first two rules of the first row allow us to derive $\epsilon; \alpha : \mathbf{type}_j \vdash \alpha : \mathbf{type}_j$ thereby declaring primitive types. We can then declare additional symbols such as $c : \alpha$ or $P : \alpha \to \mathbf{Bool}$. The requirement of $j < i$ in the first rule is needed to avoid Russel's paradox. The second rule of the second row allows Boolean assumptions to be introduced into contexts.

isfy the sort constraints associated with the function and predicate symbols.

$$\begin{array}{rcl} t & ::= & x \mid c \mid f(t_1, \ldots, t_n) \\ \Phi & ::= & P(t_1, \ldots, t_n) \mid t_1 =_\sigma t_2 \\ & & \mid \Phi_1 \vee \Phi_2 \mid \neg\Phi \mid \forall x:\sigma\ \Phi[x] \end{array}$$

Note that in the above grammar the equality symbol $=_\sigma$ is subscripted with a sort σ to which it applies. The labeling of equality with sorts is important for the treatment of isomorphism.

Given this basic grammar of terms and formulas it is standard to introduce the following abbreviations.

$$\begin{array}{rcl} \Phi \wedge \Psi & \equiv & \neg(\neg\Phi \vee \neg\Psi) \\ \Phi \Rightarrow \Psi & \equiv & \neg\Phi \vee \Psi \\ \exists x:\sigma\ \Phi[x] & \equiv & \neg\forall x:\sigma\ \neg\Phi[x] \\ \exists! x:\sigma\ \Phi[x] & \equiv & \left\{ \begin{array}{l} \exists x:\sigma\ \Phi[x] \\ \wedge \forall x, y:\sigma \\ \quad \Phi[x] \wedge \Phi[y] \Rightarrow x =_\sigma y \end{array} \right. \\ (\exists x:\sigma) & \equiv & \exists x:\sigma\ x =_\sigma x \end{array}$$

We now replace the word "sort" with the word "type". To define the set of well formed terms and formulas we need to specify primitive types and a set of constant and function symbols each with specified argument and value types. In formal type systems this is done with symbol declarations. We write $\Sigma \vdash t : \sigma$ to indicate that the symbol declarations in Σ imply that t is a well-formed expression of type σ. For example we have the following.

$$\left.\begin{array}{l} \alpha : \mathbf{type}; \\ \beta : \mathbf{type}; \\ c : \alpha; \\ f : \alpha \to \beta \end{array}\right\} \vdash f(c) : \beta$$

$$\left.\begin{array}{l} \alpha : \mathbf{type}; \\ c : \alpha; \\ f : \alpha \to \alpha; \\ P : \alpha \to \mathbf{Bool} \end{array}\right\} \vdash P(f(f(c))) : \mathbf{Bool}$$

An expression of the form $\Sigma \vdash \Theta$ is called a *sequent* where Σ is called the context and Θ is called the judgement. The sequent $\Sigma \vdash \Theta$ says that judgement Θ holds in context Σ. We allow a context to contain both symbol declarations and Boolean assumptions. For example we have

$$\left.\begin{array}{l} \alpha : \mathbf{type};\ a : \alpha;\ b : \alpha; \\ f : \alpha \times \alpha \to \alpha; \\ \forall x:\alpha\ \forall y:\alpha \\ \quad f(x, y) =_\alpha f(y, x) \end{array}\right\} \vdash f(a, b) =_\alpha f(b, a)$$

$$\frac{\Sigma \vdash \Phi : \mathbf{Bool} \quad \Sigma \vdash \Psi : \mathbf{Bool} \quad \Sigma; \Phi \vdash \Psi \quad \Sigma; \neg\Phi \vdash \Psi}{\Sigma \vdash \Psi}$$

$$\frac{\Sigma \vdash \Phi : \mathbf{Bool} \quad \Sigma \vdash \Psi : \mathbf{Bool} \quad \Sigma \vdash \Phi}{\Sigma \vdash \Phi \vee \Psi \quad \Sigma \vdash \neg\neg\Phi}$$

$$\frac{\Sigma \vdash \Phi : \mathbf{Bool} \quad \Sigma \vdash \Psi : \mathbf{Bool} \quad \Sigma \vdash \Psi}{\Sigma \vdash \Phi \vee \Psi}$$

$$\frac{\Sigma \vdash \Phi : \mathbf{Bool} \quad \Sigma \vdash \Psi : \mathbf{Bool} \quad \Sigma \vdash \neg\Psi \quad \Sigma \vdash \neg\Phi}{\Sigma \vdash \neg(\Phi \vee \Psi)}$$

$$\frac{\Sigma \vdash \forall x : \tau\ \Phi[x] \quad \Sigma \vdash e : \tau}{\Sigma \vdash \Phi[e]}$$

$$\frac{\Sigma; x : \tau \vdash \Phi[x] : \mathbf{Bool} \quad \Sigma; x : \tau \vdash \Phi[x]}{\Sigma \vdash \forall x : \tau\ \Phi[x]}$$

$$\frac{\Sigma \vdash \exists! x : \sigma\ \Phi[x]}{\Sigma \vdash \mathbf{The}(x : \sigma,\ \Phi[x]) : \sigma \quad \Sigma \vdash \Phi[\mathbf{The}(x : \sigma,\ \Phi[x])]}$$

$$\frac{\Sigma \vdash e : \tau}{\Sigma \vdash e =_\tau e} \qquad \frac{\Sigma \vdash u =_\tau w}{\Sigma \vdash w =_\tau u}$$

$$\frac{\Sigma \vdash u =_\tau w \quad \Sigma \vdash w =_\tau s}{\Sigma \vdash u =_\tau s}$$

$$\frac{\Sigma \vdash \tau : \mathbf{type}_i \quad \Sigma;\ x : \sigma \vdash e[x] : \tau \quad \Sigma \vdash w =_\sigma u \quad \Sigma, \sigma, \tau \text{ and } e[x] \text{ are pure}}{\Sigma \vdash e[w] =_\tau e[u]}$$

$$\frac{\Sigma \vdash f, g : \sigma \to \tau \quad \Sigma \vdash \forall x : \sigma\ f(x) =_\tau g(x)}{\Sigma \vdash f =_{\sigma\to\tau}\ g}$$

$$\frac{\Sigma \vdash \tau : \mathbf{type}_i \quad \Sigma \vdash \forall x : \sigma\ \exists y : \tau\ \Phi[x,\ y] \quad \Sigma, \sigma, \tau \text{ and } \Phi[x, y] \text{ are pure}}{\Sigma \vdash \exists f : \sigma \to \tau\ \forall x : \sigma\ \Phi[x, f(x)]}$$

Figure 2: **Predicate Calculus Inference Rules.** The first row is a complete set of rules for Boolean logic. A rule with two conclusions abbreviates two rules each with the same antecedents. The third rule in the second row handles definite descriptions (Hilbert's ι-operator). An expression is "pure" if does not involve any of the constructs introduced in figure 6. The last row gives the axioms of extensionality and choice.

In higher order predicate calculus the type system is extended to include not only primitive types but also function types and we can write, for example, $P(f)$ where we have $f : \sigma \to \tau$ and $P : (\sigma \to \tau) \to \mathbf{Bool}$. In the higher order case we can use the following standard abbreviations due to Curry.

$$\begin{aligned} \sigma_1 \times \sigma_2 \to \tau &\equiv \sigma_1 \to (\sigma_2 \to \tau) \\ f(a, b) &\equiv f(a)(b) \end{aligned}$$

This extends in the obvious way to abbreviations of the form $\sigma_1 \times \cdots \times \sigma_n \to \tau$. Without loss of generality we then need consider only single argument functions.

Figure 1 gives a set of inference rules for forming the expressions of higher order predicate calculus. Each rule allows for the derivation of the sequent below the line provided that the sequents above the line are derivable. A rule with no antecedents is written as a single derivable sequent.

Figure 2 gives inference rules for predicate calculus including definite descriptions of the form $\mathbf{The}(x : \sigma,\ \Phi[x])$ (Hilbert's ι-operator) and rules representing the axiom of extensionality and the axiom of choice.

Figure 3 gives inference rules for dependent pair types, subtypes, and existential types. A dependent pair type has the form $\mathbf{PairOf}\ (x : \sigma,\ y : \tau[x])$ and is the type whose instances are the pairs $\mathbf{Pair}(x, y)$ where x is an instance of σ and y is an instance of $\tau[x]$. A subtype expression has the form $\mathbf{SubType}(x : \sigma,\ \Phi[x])$ where $\Phi[x]$ is a Boolean expression. This expression denotes the type whose elements are those elements x in σ such that $\Phi[x]$ holds. We let $\mathbf{PairOf}(x : \sigma,\ y : \tau[x]$ s. t. $\Phi[x, y])$

$$\frac{\Sigma \vdash \sigma : \mathbf{type}_i \quad \Sigma;\ x{:}\sigma \vdash \tau[x] : \mathbf{type}_i}{\Sigma \vdash \mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x]) : \mathbf{type}_i}$$

$$\frac{\Sigma \vdash \mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x]) : \mathbf{type}_i \quad \Sigma \vdash u : \sigma \quad \Sigma \vdash w : \tau[u]}{\Sigma \vdash \mathbf{Pair}(u,w) : \mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x]) \quad \Sigma \vdash \pi_1(\mathbf{Pair}(u,w)) \doteq u \quad \Sigma \vdash \pi_2(\mathbf{Pair}(u,w)) \doteq w}$$

$$\frac{\Sigma \vdash p : \mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x])}{\Sigma \vdash \pi_1(p) : \sigma \quad \Sigma \vdash \pi_2(p) : \tau[\pi_1(p)] \quad \Sigma \vdash p \doteq \mathbf{Pair}(\pi_1(p),\ \pi_2(p))}$$

$$\Sigma \vdash e \doteq e$$

$$\frac{\Sigma \vdash u \doteq w}{\Sigma \vdash w \doteq u}$$

$$\frac{\Sigma \vdash u \doteq w \quad \Sigma \vdash w \doteq s}{\Sigma \vdash u \doteq s}$$

$$\frac{\Sigma \vdash u \doteq w \quad \Sigma \vdash \Theta[u]}{\Sigma \vdash \Theta[w]}$$

$$\frac{\Sigma \vdash \tau : \mathbf{type}_i \quad \Sigma;\ x{:}\tau \vdash \Phi[x] : \mathbf{Bool}}{\Sigma \vdash \mathbf{SubType}\,(x{:}\tau,\ \Phi[x]) : \mathbf{type}_i}$$

$$\frac{\Sigma \vdash \mathbf{SubType}\,(x{:}\tau,\ \Phi[x]) : \mathbf{type}_i \quad \Sigma \vdash e : \tau \quad \Sigma \vdash \Phi[e]}{\Sigma \vdash e : \mathbf{SubType}\,(x{:}\tau,\ \Phi[x])}$$

$$\frac{\Sigma \vdash e : \mathbf{SubType}\,(x{:}\tau,\ \Phi[x])}{\Sigma \vdash e : \tau \quad \Sigma \vdash \Phi[e]}$$

$$\frac{\Sigma \vdash \sigma : \mathbf{type}_i \quad \Sigma;\ x{:}\sigma \vdash \tau[x] : \mathbf{type}_i}{\Sigma \vdash (\exists x{:}\sigma\ \tau[x]) : \mathbf{type}_i}$$

$$\frac{\Sigma \vdash (\exists x{:}\sigma\ \tau[x]) : \mathbf{type}_i \quad \Sigma \vdash w : \sigma \quad \Sigma \vdash e : \tau[w]}{\Sigma \vdash e : (\exists x{:}\sigma\ \tau[x])}$$

$$\frac{\Sigma \vdash e : (\exists x{:}\sigma\ \tau[x]) \quad \Sigma;\ y{:}\sigma; z{:}\tau[y] \vdash \Theta[z] \quad y \text{ does not occur free in } \Theta[z]}{\Sigma \vdash \Theta[e]}$$

Figure 3: **Pair Types, Subtypes and Existential Types.** Note the use of absolute equality (judgemental equality) $\doteq$ in the rules for pair types. We can have two distinct but isomorphic things — we can have $a =_\sigma b$ with $a \neq b$. It is important that absolute equalities are not Boolean expressions — otherwise the substitution of isomorphics would yield that $a =_\sigma b$ implies $a \doteq b$.

abbreviate

$$\mathbf{SubType}\left(\begin{array}{l} z{:}\mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x])\,, \\ \Phi[\pi_1(z), \pi_2(z)] \end{array}\right).$$

The type of groups, abbreviated $\mathbf{Group}$, can then be written as

$$\mathbf{PairOf}\,(\alpha{:}\mathbf{type},\ f{:}(\alpha \times \alpha) \to \alpha \text{ s. t. } \Phi[\alpha, f])$$

where $\Phi[\alpha, f]$ states the group axioms. For example, the group axiom that an identity element exists can be written as

$$\exists x{:}\alpha\ \forall y{:}\alpha \quad f(x,y) =_\alpha y \quad \wedge \quad f(y,x) =_\alpha y.$$

The type of topological spaces, denoted $\mathbf{TOP}$, can be written as

$$\mathbf{PairOf}\left(\begin{array}{l} \alpha{:}\mathbf{type}, \\ \mathbf{Open}{:}(\alpha \to \mathbf{Bool}) \to \mathbf{Bool}, \\ \text{ s. t. } \Psi[\alpha, \mathbf{Open}] \end{array}\right)$$

where $\Psi[\alpha, \mathbf{Open}]$ states the topology axioms. Here the open sets of the topological space are represented by predicates. Note that the types $\mathbf{Group}$ and $\mathbf{TOP}$ are closed type expressions — these type expressions do not contain free variables.

We should note that subtypes are literally subsets and, for example, we can derive the sequent

$$G{:}\mathbf{AbelianGroup} \vdash G{:}\mathbf{Group}.$$

Existential types have the form $\exists x{:}\sigma\ \tau[x]$ where $\tau[x]$ is a type expression. This is the type whose members are those values v such that there exists a value $u : \sigma$ such that v is in the type $\tau[u]$. Existential types allow one to express the type of permutation groups as distinct from the type of groups. A permutation group is a group whose group elements are permutations of an underlying set. The type of permutation groups, denoted $\mathbf{PermGroup}$, has the

form

$$\exists\alpha\!:\!\mathbf{type}\ \exists P\!:\!(\mathbf{Permutation}[\alpha]\to\mathbf{Bool})\ \tau[\alpha,P].$$

Here the predicate P represents a set of permutations on the type α and $\tau[\alpha,P]$ is a pair type specifying a group whose elements are the permutations of α satisfying P and where the group operation is functional composition. Again note that $\mathbf{PermGroup}$ is a closed type expression. The rules for existential types then allow the derivation of the sequent

$$G\!:\!\mathbf{PermGroup}\vdash G\!:\!\mathbf{Group}.$$

We also have the representation theorem

$$\vdash\forall G\!:\!\mathbf{Group}\ \exists H\!:\!\mathbf{PermGroup}\ G=_{\mathbf{Group}}H.$$

However, the isomorphism relations $=_{\mathbf{Group}}$ and $=_{\mathbf{PermGroup}}$ are different. Two permutation groups can be group-isomorphic while operating on underlying sets of different sizes. Such permutation groups are group-isomorphic but not permutation-group-isomorphic.

3 Observational Equivalence

Our intention now is to interpret an equation such as $G=_{\mathbf{Group}}H$ as stating that G and H are group-isomorphic. The significance of isomorphism arises from the substitution rule of figure 2. The rules of equality — reflexivity, symetry, transitivity and substitution — support the congruence closure algorithm for reasoning about equality. The ability to apply congruence closure to the isomorphism relation should be of great value in automated reasoning.[1]

The core rules define a notion of "observational equivalence" — two closed terms a and b of type σ are observationally σ-equivalent if for every predicate expression $P\!:\!\sigma\to\mathbf{Bool}$ (typable by the core rules) we have $P(a)$ if and only if $P(b)$. We want $=_\sigma$ to be as course as possible subject to the constraint that $a=_\sigma b$ implies that a and b are observationally σ-equivalent.

The desire to be as coarse as possible while staying within observational equivalence motivates the interpretation of type-isomorphism as same-cardinality. Predicates on types that are well formed under the core rules cannot distinguish between types of the same cardinality.

[1] It is tempting to suggest that congruence closure is of great value in subconscious human thought.

4 Morphoids

The semantics of morphoid type theory is developed within a meta-theory of Platonic mathematics — we adopt the position that it is meaningful to discuss actual (Platonic) mathematical objects.

The semantics of morphoid type theory is based on a class of values called morphoids. A rigorous definition of a class of morphoids for a subset of the language can be found in (McAllester, 2014) version 4. Here we give some intuition for morphoids and state some formal properties.

Morphoids are built from "points". Morphoid points are analogous to the ur-elements of some early versions of set theory. General morphoids are built from points in a manner analogous to the way that sets are constructed from ur-elements.

A morphoid point has the form $\mathbf{Point}(i,j)$ where i is called the left index and j is the right index of the point. We define the operations of $\mathbf{Left}$, $\mathbf{Right}$, inverse and composition on points as follows.

$$\begin{aligned}\mathbf{Left}(\mathbf{Point}(i,j)) &= \mathbf{Point}(i,i)\\ \mathbf{Right}(\mathbf{Point}(i,j)) &= \mathbf{Point}(j,j)\\ \mathbf{Point}(i,j)^{-1} &= \mathbf{Point}(j,i)\\ \mathbf{Point}(i,j)\circ\mathbf{Point}(j,k) &= \mathbf{Point}(i,k)\end{aligned}$$

Here we have that $x\circ y$ is defined only in the case where $\mathbf{Right}(x)=\mathbf{Left}(y)$.

Every morphoid value is either a Boolean value ($\mathbf{True}$ or $\mathbf{False}$), a point, a morphoid type, a pair of morphoid values, or a morphoid function. The operations of $\mathbf{Left}$, $\mathbf{Right}$, inverse and composition are defined recursively on all morphoid values where $x\circ y$ is defined when $\mathbf{Right}(x)=\mathbf{Left}(y)$. We consider each kind of value in turn.

A morphoid type is a set σ of morphoid values satisfying certain properties defined in (McAllester, 2014). A fundamental property is the following.

(M) For $x,y,x\in\sigma$ with $x\circ y^{-1}\circ z$ defined we have $x\circ y^{-1}\circ z\in\sigma$.

The following equations define the morphoid operations on types where $\sigma\circ\tau$ is defined only when

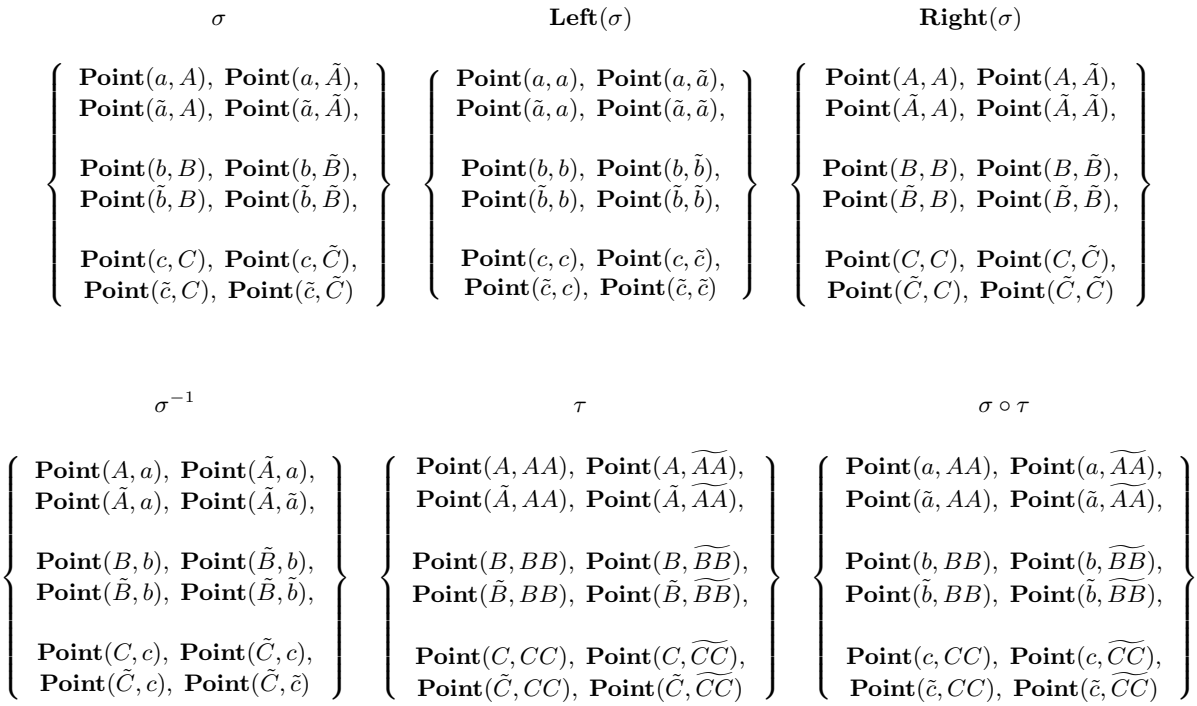

Figure 4: **The operations of Left, Right, inverse and composition on point types.**

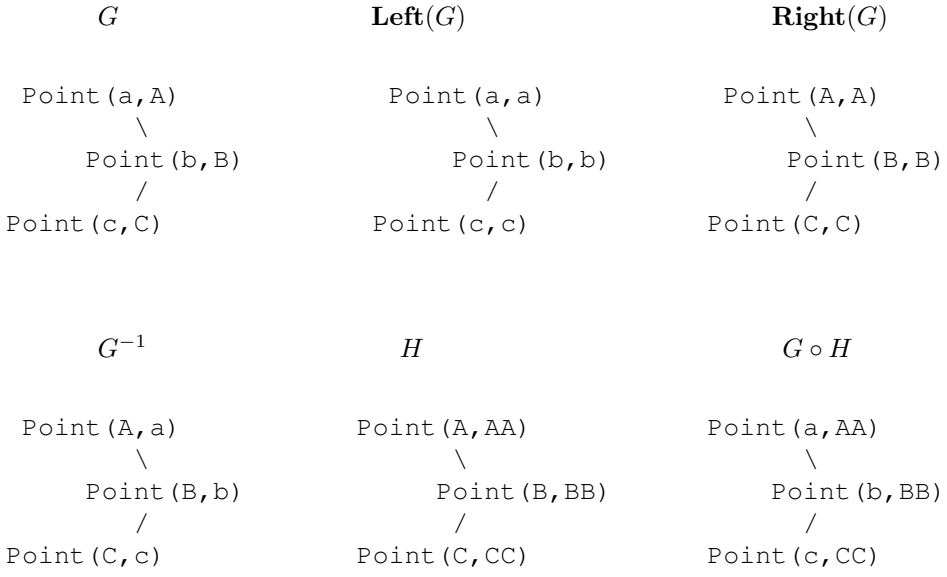

Figure 5: **The operations of Left, Right, inverse and composition on abstract morphoid graphs.**

$\mathbf{Right}(\sigma) = \mathbf{Left}(\tau)$.

$$
\begin{aligned}
\mathbf{Left}(\sigma) &= \left\{ \begin{array}{l} x_1 \circ x_2^{-1} :\ x_1, x_2 \in \sigma, \\ \mathbf{Right}(x_1) = \mathbf{Right}(x_2) \end{array} \right\} \\
\mathbf{Right}(\sigma) &= \left\{ \begin{array}{l} x_1^{-1} \circ x_2 :\ x_1, x_2 \in \sigma, \\ \mathbf{Left}(x_1) = \mathbf{Left}(x_2) \end{array} \right\} \\
\sigma \circ \tau &= \left\{ \begin{array}{l} x \circ y :\ x \in \sigma,\ y \in \tau, \\ \mathbf{Right}(x) = \mathbf{Left}(y) \end{array} \right\} \\
\sigma^{-1} &= \{x^{-1} :\ x \in \sigma\}
\end{aligned}
$$

A morphoid type whose elements are points is called a point type. Figure 4 gives examples of morphoid point types and examples of the morphoid operations applied to point types. The morphoid closure condition (M) implies that for any morphoid type σ we have that $\mathbf{Left}(\sigma)$ and $\mathbf{Right}(\sigma)$ are equivalence relations (see figure 4). Note that morphoid types are not required to be closed under inverse. This allows types to be directed from left to right. Again consider the types in figure 4. Furthermore, property (M) implies that any morphoid type σ defines a bijection between the equivalence classes of $\mathbf{Left}(\sigma)$ and the equivalence classes of $\mathbf{Right}(\sigma)$.

A morphoid pair is simply a pair of morphoids. The morphoid operations on pairs are defined as follows where again $x \circ y$ is defined only when $\mathbf{Right}(x) = \mathbf{Left}(y)$.

$$
\begin{aligned}
\mathbf{Left}(\mathbf{Pair}(x,y)) &= \mathbf{Pair}(\mathbf{Left}(x), \mathbf{Left}(y)) \\
\mathbf{Right}(\mathbf{Pair}(x,y)) &= \mathbf{Pair}(\mathbf{Right}(x), \mathbf{Right}(y)) \\
\mathbf{Pair}(x,y)^{-1} &= \mathbf{Pair}(x^{-1}, y^{-1}) \\
\mathbf{Pair}(x,y) \circ \mathbf{Pair}(z,w) &= \mathbf{Pair}(x \circ z,\ y \circ w)
\end{aligned}
$$

The treatment of morphoid functions involves subtleties. In this synopsis we note only that for any two morphoid types σ and τ we can define the type $\sigma \to \tau$ such that for $f \in \sigma \to \tau$ and $x \in \sigma$ we can define the application $f(x)$ so that we have $f(x) \in \tau$. Furthermore, these definitions are such that the elements of the type $\sigma \to \tau$ represents *all* functions from the equivalence classes of σ to the equivalence classes of τ. Details, including the definitions of the morphoid operations on functions, can be found in (McAllester, 2014).

It is possible to prove that morphoids satisfy the following properties where properties (G4), (G5), (G6), and (G9) apply when the compositions are defined.

(G1) For any morphoid x we have that $\mathbf{Left}(x)$, $\mathbf{Right}(x)$ and x^{-1} are also morphoids.

(G2) For any morphoids x and y we have that $x \circ y$ is defined if and only if $\mathbf{Right}(x) = \mathbf{Left}(y)$ and when $x \circ y$ is defined we have that $x \circ y$ is a morphoid.

(G3) $\mathbf{Left}(x^{-1}) = \mathbf{Right}(x)$ and $\mathbf{Right}(x^{-1}) = \mathbf{Left}(x)$

(G4) $\mathbf{Left}(x \circ y) = \mathbf{Left}(x)$ and $\mathbf{Right}(x \circ y) = \mathbf{Right}(y)$.

(G5) $(x \circ y) \circ z = x \circ (y \circ z)$.

(G6) $x^{-1} \circ x \circ y = y$ and $x \circ y \circ y^{-1} = x$.

(G7) $\mathbf{Right}(x) = x^{-1} \circ x$ and $\mathbf{Left}(x) = x \circ x^{-1}$.

(G8) $(x^{-1})^{-1} = x$.

(G9) $(x \circ y)^{-1} = y^{-1} \circ x^{-1}$.

Properties (G1) through (G9) state that the class of morphoids forms a groupoid under the morphoid operations. Figure 5 shows morphoid operations on graphs whose nodes are points.

5 Abstraction and Isomorphism

The semantic definition of isomorphism relies on an additional operation on morphoids — the operation of abstraction. As an example we consider vector spaces. In morphoid type theory an abstract vector space is one in which the vectors are points. The space $\mathbb{R}^n$ is a vector space whose vectors are n-tuples of real numbers. A tuple of real numbers is an implementation of a vector — a tuple of real numbers is not a point. However, we can define an abstraction operation such that for any morphoid value x we have that $x@\mathbf{Point}$ is a point. Details can be found in (McAllester, 2014).

There is an abstraction ordering on morphoids where $x \preceq y$ if x can be converted to y by abstracting parts of x to points. For every type there is a set of maximally abstract elements of that type. The maximally abstract graphs are the graphs whose nodes are points. The maximally abstract vector

spaces are those vector spaces in which the vectors are points. The maximally abstract types are the point types. For each morphoid type σ it is possible to define an abstraction operation mapping $x \in \sigma$ to $x@\sigma$ where $x@\sigma$ is a maximally abstract member of σ. For example, for any morphoid type $\sigma \in \mathbf{type}_i$ we have that $\sigma@\mathbf{type}_i$ is the point type whose members are the points of the form $x@\mathbf{Point}$ for $x \in \sigma$. Details can be found in (McAllester, 2014).

The isomorphism relation $x =_\sigma y$ is defined to mean that $x \in \sigma$, $y \in \sigma$, and there exists $z \in \sigma$ such that $(x@\sigma) \circ z^{-1} \circ (y@\sigma)$ is defined. Condition (M) on morphoid types guarantees that this is an equivalence relation on the elements of σ.

6 The Semantic Value Function

The semantics of morphoid type theory is an extension of the sementics of predicate calculus. The semantics involves three concepts — variable interpretations, semantic entailment, and a semantic value function. These three concepts are defined by mutual recursion where the recursion reduces the size of the expressions involved. A variable interpretation assigns a value to each variable declared in a given context. More formally, for any well-formed context Σ we write $\mathcal{V}[\![\Sigma]\!]$ for the set of variable interpretations consistent with declarations and Boolean assertions in Σ. We define $\mathcal{V}[\![\Sigma]\!]$ by the following rules where this is undefined if no rule applies.

- We define $\mathcal{V}[\![\epsilon]\!]$ to be the set containing the empty variable interpretation.
- $\mathcal{V}[\![\Sigma; x:\tau]\!]$ is defined if $\mathcal{V}[\![\Sigma]\!]$ is defined, x is not declared in Σ, and $\Sigma \models \tau : \mathbf{type}_i$ in which case $\mathcal{V}[\![\Sigma; x:\tau]\!]$ is defined to be the set of variable interpretations of the form $\rho[x \leftarrow v]$ for $\rho \in \mathcal{V}[\![\Sigma]\!]$ and $v \in \mathcal{V}_\Sigma[\![\tau]\!]\,\rho$.
- $\mathcal{V}[\![\Sigma; \Phi]\!]$ is defined if $\mathcal{V}[\![\Sigma]\!]$ is defined and $\Sigma \models \Phi : \mathbf{Bool}$ in which case $\mathcal{V}[\![\Sigma; \Phi]\!]$ is defined to be the set of all $\rho \in \mathcal{V}[\![\Sigma]\!]$ such that $\mathcal{V}_\Sigma[\![\Phi]\!]\,\rho = \mathbf{True}$.

A semantic entailment is written as $\Sigma \models \Theta$ and this holds if $\mathcal{V}[\![\Sigma]\!]$ is defined and Θ holds under all variable interpretations in $\mathcal{V}[\![\Sigma]\!]$. The entailment relation $\Sigma \models \Theta$ holds if one of the following clauses applies.

- The entailment $\Sigma \models e:\tau$ holds if $\mathcal{V}[\![\Sigma]\!]$, $\mathcal{V}_\Sigma[\![e]\!]$ and $\mathcal{V}_\Sigma[\![\tau]\!]$ are all defined and for all $\rho \in \mathcal{V}[\![\Sigma]\!]$ we have that $\mathcal{V}_\Sigma[\![\tau]\!]\,\rho$ is a morphoid type and $\mathcal{V}_\Sigma[\![e]\!]\,\rho \in \mathcal{V}_\Sigma[\![\tau]\!]\,\rho$.
- For a Boolean expression Φ, i.e., for $\Sigma \models \Phi : \mathbf{Bool}$, we have that $\Sigma \models \Phi$ holds if for all $\rho \in \mathcal{V}[\![\Sigma]\!]$ we have $\mathcal{V}_\Sigma[\![\Phi]\!]\,\rho = \mathbf{True}$.
- We write $\Sigma \models e_1 \doteq e_2$ if $\mathcal{V}[\![\Sigma]\!]$, $\mathcal{V}_\Sigma[\![e_1]\!]$ and $\mathcal{V}_\Sigma[\![e_2]\!]$ are all defined and for $\rho \in \mathcal{V}[\![\Sigma]\!]$ we have that $\mathcal{V}_\Sigma[\![e_1]\!]\,\rho$ and $\mathcal{V}_\Sigma[\![e_2]\!]\,\rho$ are the same value.

For $\mathcal{V}[\![\Sigma]\!]$ defined and for an expression e that is well-formed in the context Σ, we have a semantic value function $\mathcal{V}_\Sigma[\![e]\!]$. The semantic value function $\mathcal{V}_\Sigma[\![e]\!]$ maps a variable interpretation $\rho \in \mathcal{V}[\![\Sigma]\!]$ to a morphoid value $\mathcal{V}_\Sigma[\![e]\!]\,\rho$. For $\mathcal{V}[\![\Sigma]\!]$ defined, the following clauses state when $\mathcal{V}_\Sigma[\![e]\!]$ is defined and, when it is defined, define the value $\mathcal{V}_\Sigma[\![e]\!]\,\rho$ for $\rho \in \mathcal{V}[\![\Sigma]\!]$.

• x. For x declared in Σ and for $\rho \in \mathcal{V}[\![\Sigma]\!]$ we have that $\mathcal{V}_\Sigma[\![x]\!]$ is defined with $\mathcal{V}_\Sigma[\![x]\!]\,\rho = \rho(x)$.

• $\mathbf{Bool}$. We have that $\mathcal{V}_\Sigma[\![\mathbf{Bool}]\!]\,\rho$ is the type containing the two Boolean values $\mathbf{True}$ and $\mathbf{False}$.

• $\mathbf{type}_i$. We have $\mathcal{V}_\Sigma[\![\mathbf{type}_i]\!]\,\rho$ is the type whose members are all morphoid types in the set-theoretic universe V_{κ_i} where κ_i is the ith inaccessible cardinal.

• $\sigma \to \tau$. If $\Sigma \models \sigma : \mathbf{type}_i$, and $\Sigma \models \tau : \mathbf{type}_i$, then $\mathcal{V}_\Sigma[\![\sigma \to \tau]\!]$ is defined with $\mathcal{V}_\Sigma[\![\sigma \to \tau]\!]\,\rho = (\mathcal{V}_\Sigma[\![\sigma]\!]\,\rho) \to (\mathcal{V}_\Sigma[\![\tau]\!]\,\rho)$.

• $f(e)$. If $\mathcal{V}_\Sigma[\![f]\!]$ and $\mathcal{V}_\Sigma[\![e]\!]$ are defined and for all $\rho \in \mathcal{V}[\![\Sigma]\!]$ we have that $\mathcal{V}_\Sigma[\![f]\!]\,\rho$ can be applied to $\mathcal{V}_\Sigma[\![e]\!]\,\rho$ then $\mathcal{V}_\Sigma[\![f(e)]\!]$ is defined with $\mathcal{V}_\Sigma[\![f(e)]\!]\,\rho = (\mathcal{V}_\Sigma[\![f]\!]\,\rho)(\mathcal{V}_\Sigma[\![e]\!]\,\rho)$.

• $\forall\ x:\tau\ \Phi[x]$. If $\Sigma;\ y:\tau \models \Phi[y] : \mathbf{Bool}$ then $\mathcal{V}_\Sigma[\![\forall\, x:\tau\ \Phi[x]]\!]$ is defined with $\mathcal{V}_\Sigma[\![\forall\, x:\tau\ \Phi[x]]\!]\,\rho$ being $\mathbf{True}$ if for all $v \in \mathcal{V}_\Sigma[\![\tau]\!]\,\rho$ we have $\mathcal{V}_{\Sigma;y:\tau}[\![\Phi[y]]\!]\,\rho[y \leftarrow v] = \mathbf{True}$.

• $\Phi \vee \Psi$. If $\Sigma \models \Phi : \mathbf{Bool}$ and $\Sigma \models \Psi : \mathbf{Bool}$ then $\mathcal{V}_\Sigma[\![\Phi \vee \Psi]\!]$ is defined with $\mathcal{V}_\Sigma[\![\Phi \vee \Psi]\!]\,\rho = \mathcal{V}_\Sigma[\![\Phi]\!]\,\rho \vee \mathcal{V}_\Sigma[\![\rho]\!]\,\rho$.

• $\neg\Phi$. If $\Sigma \models \Phi : \mathbf{Bool}$ then $\mathcal{V}_\Sigma[\![\neg\Phi]\!]$ is defined with $\mathcal{V}_\Sigma[\![\neg\Phi]\!]\,\rho = \neg\mathcal{V}_\Sigma[\![\Phi]\!]\,\rho$.

$$\mathbf{Bijection}[\sigma,\tau] \equiv \mathbf{SubType}(f{:}\sigma\to\tau,\ \forall y{:}\tau\ \exists!x{:}\sigma\ f(x) =_\tau y) \qquad a \rightsquigarrow_\sigma b \ \equiv\ \exists z{:}\mathbf{iso}(\sigma,a,b)$$

$$\frac{\Sigma\vdash\sigma{:}\mathbf{type}_i \quad \Sigma\vdash a{:}\tau \quad \Sigma\vdash b{:}\eta}{\Sigma\vdash \mathbf{iso}(\sigma,a,b){:}\mathbf{type}_i} \qquad \frac{\Sigma\vdash\sigma,\tau{:}\mathbf{type}_i \quad \Sigma\vdash f{:}\mathbf{Bijection}[\sigma,\tau]}{\Sigma\vdash\updownarrow(\sigma,\tau,f){:}\mathbf{iso}(\mathbf{type}_i,\sigma,\tau) \quad \Sigma\vdash\forall x{:}\sigma\ \ x\ \rightsquigarrow_{\updownarrow(\sigma,\tau,f)} f(x)}$$

$$\frac{\Sigma\vdash\eta{:}\mathbf{iso}(\mathbf{type}_i,\sigma,\tau)}{\Sigma\vdash\uparrow_{\eta\to\sigma}{:}\mathbf{Bijection}[\eta,\sigma] \quad \Sigma\vdash\downarrow_{\eta\to\tau}{:}\mathbf{Bijection}[\eta,\tau] \quad \Sigma\vdash \begin{cases} \forall x{:}\sigma\ \forall y{:}\tau \\ \ (x\rightsquigarrow_\eta y)\ \Leftrightarrow \\ \ \ \exists z{:}\eta\ \uparrow_{\eta\to\sigma}(z) =_\sigma x \wedge \\ \qquad \downarrow_{\eta\to\tau}(z) =_\tau y \end{cases}}$$

$$\frac{\Sigma\vdash c{:}\mathbf{iso}(\sigma,a,b)}{\Sigma\vdash c{:}\sigma} \qquad \frac{\Sigma\vdash a{:}\sigma,\ b{:}\sigma \quad \Sigma\vdash c{:}\mathbf{iso}(\sigma,a,b)}{\Sigma\vdash a =_\sigma c \quad \Sigma\vdash b =_\sigma c} \qquad \frac{\Sigma\vdash a =_\sigma b}{\Sigma\vdash a\rightsquigarrow_\sigma b}$$

$$\frac{\Sigma;x{:}\sigma;y{:}\gamma[x]\vdash e[x,y]{:}\tau[x,y] \quad \Sigma\vdash a_1{:}\sigma,\ a_2{:}\sigma,\ a_3{:}\mathbf{iso}(\sigma,a_1,a_2) \quad \Sigma\vdash b_1{:}\gamma[a_1],\ b_2{:}\gamma[a_2],\ b_3{:}\mathbf{iso}(\gamma[a_3],b_1,b_2)}{\Sigma\vdash e[a_3,b_3]{:}\mathbf{iso}(\tau[a_3,b_3],e[a_1,b_1],e[a_2,b_2])}$$

$$\frac{\Sigma\vdash\sigma_3{:}\mathbf{iso}(\mathbf{type}_i,\sigma_1,\sigma_2) \quad \Sigma\vdash f_1{:}\sigma_1\to\tau_1,\ f_2{:}\sigma_2\to\tau_2,\ f_3{:}\sigma_3\to\tau_3 \quad \Sigma;\ z{:}\sigma_3\vdash f_3(z){:}\mathbf{iso}(\tau_3,\ f_1(\uparrow_{\sigma_3\to\sigma_1}(z)),\ f_2(\downarrow_{\sigma_3\to\sigma_2}(z)))}{\Sigma\vdash f_3{:}\mathbf{iso}(\sigma_3\to\tau_3,\ f_1,\ f_2)}$$

$$\frac{\Sigma\vdash c{:}\mathbf{iso}(\sigma,a,b) \quad \Sigma\vdash\Phi[c]}{\Sigma\vdash c{:}\mathbf{iso}(\mathbf{SubType}(x{:}\sigma,\ \Phi[x]),\ a,\ b)} \qquad \frac{\Sigma\vdash c{:}\mathbf{iso}(\tau[d],a,b) \quad \Sigma\vdash d{:}\sigma}{\Sigma\vdash c{:}\mathbf{iso}(\exists x{:}\sigma\ \tau[x],\ a,\ b)}$$

Figure 6: **Internalizing Isomorphism.** Here $\mathbf{iso}(\sigma,x,y)$ is the type whose elements are the σ-isomorphisms from x to y. The fourth row gives the rules of iso-substition and iso-extensionality.

• $s =_\sigma w$. If $\Sigma \models s:\sigma$ and $\Sigma \models w:\sigma$ then $\mathcal{V}_\Sigma[\![s =_\sigma w]\!]$ is defined with $\mathcal{V}_\Sigma[\![s =_\sigma w]\!]\,\rho$ being **True** if $\mathcal{V}_\Sigma[\![s]\!]\,\rho\ =_{\mathcal{V}_\Sigma[\![\sigma]\!]\rho}\ \mathcal{V}_\Sigma[\![w]\!]\,\rho$.

• $\mathbf{The}(x{:}\sigma,\ \Phi[x])$. If $\Sigma\models\exists!y{:}\sigma\ \Phi[y]$ then

$$\mathcal{V}_\Sigma[\![\mathbf{The}(x{:}\sigma,\ \Phi[x])]\!]\,\rho =$$
$$\mathbf{The}(\mathcal{V}_\Sigma[\![\mathbf{SubType}(x{:}\sigma,\ \Phi[x])]\!]\,\rho).$$

• $\mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x])$. If $\Sigma \models \sigma : \mathbf{type}_i$ and $\Sigma; z : \sigma \models \tau[z] : \mathbf{type}_i$ then $\mathcal{V}_\Sigma[\![\mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x])]\!]$ is defined with $\mathcal{V}_\Sigma[\![\mathbf{PairOf}\,(x{:}\sigma,\ y{:}\tau[x])]\!]\,\rho$ being the type containing the pairs $\mathbf{Pair}(v,w)$ for $v \in \mathcal{V}_\Sigma[\![\sigma]\!]\,\rho$ and $w \in \mathcal{V}_{\Sigma;\,z:\sigma}[\![\tau[z]]\!]\,\rho[z\leftarrow v]$.

• $\mathbf{Pair}(u,w)$. If $\mathcal{V}_\Sigma[\![u]\!]$ and $\mathcal{V}_\Sigma[\![w]\!]$ are defined then $\mathcal{V}_\Sigma[\![\mathbf{Pair}(u,w)]\!]$ is defined with $\mathcal{V}_\Sigma[\![\mathbf{Pair}(u,w)]\!]\,\rho = \mathbf{Pair}(\mathcal{V}_\Sigma[\![u]\!]\,\rho,\ \mathcal{V}_\Sigma[\![w]\!]\,\rho)$.

• $\pi_i(e)$. If $\mathcal{V}_\Sigma[\![e]\!]$ is defined and for all $\rho \in \mathcal{V}[\![\Sigma]\!]$ we have that $\mathcal{V}_\Sigma[\![e]\!]\,\rho$ is a pair then $\mathcal{V}_\Sigma[\![\pi_i(e)]\!]$ is defined with $\mathcal{V}_\Sigma[\![\pi_i(e)]\!]\,\rho = \pi_i(\mathcal{V}_\Sigma[\![e]\!]\,\rho)$.

• $\mathbf{SubType}\,(x{:}\sigma,\ \Phi[x])$. If $\Sigma \models \sigma : \mathbf{type}_i$ and $\Sigma;\ y : \sigma \models \Phi[y] : \mathbf{Bool}$ then $\mathcal{V}_\Sigma[\![\mathbf{SubType}(x{:}\sigma,\ \Phi[x])]\!]$ is defined with $\mathcal{V}_\Sigma[\![\mathbf{SubType}\,(x{:}\sigma,\ \Phi[x])]\!]\,\rho$ being the type whose members are those values $v \in \mathcal{V}_\Sigma[\![\sigma]\!]\,\rho$ with $\mathcal{V}_{\Sigma;\,y:\sigma}[\![\Phi[y]]\!]\,\rho[y\leftarrow v] = \mathbf{True}$.

• $\exists x : \sigma\ \tau[x]$. If $\Sigma;\ y : \sigma \models \tau[y] : \mathbf{type}_i$ then $\mathcal{V}_\Sigma[\![\exists x{:}\sigma\ \tau[x]]\!]$ is defined with $\mathcal{V}_\Sigma[\![\exists x{:}\sigma\ \tau[x]]\!]\,\rho$ being the type containing those values w such that there exists $u \in \mathcal{V}_\Sigma[\![\sigma]\!]\,\rho$ with $w \in \mathcal{V}_\Sigma[\![\tau[y]]\!]\,\rho[y\leftarrow u]$.

The morphoid operations can also be defined on variable interpretations. For $\rho \in \mathcal{V}[\![\Sigma]\!]$ we have

$$\frac{\Sigma;\alpha:\mathbf{type}_i \vdash \gamma[\alpha]:\mathbf{type}_i \quad \Sigma \vdash \sigma:\mathbf{type}_i,\ \tau:\mathbf{type}_i,\ f:\mathbf{Bijection}[\sigma,\tau] \quad \Sigma \vdash a:\gamma[\sigma],\ b:\gamma[\tau],\ a \rightsquigarrow_{\gamma[\mathfrak{I}(\sigma,\tau,f)]} b}{\Sigma \vdash \mathbf{Pair}(\sigma,a) =_{\mathbf{PairOf}(\alpha:\mathbf{type}_i,\ y:\gamma[\alpha])} \mathbf{Pair}(\tau,b)}$$

$$\frac{\Sigma;\ \alpha:\mathbf{type}_i \vdash \delta[\alpha],\eta[\alpha]:\mathbf{type}_i \quad \Sigma \vdash \sigma,\tau:\mathbf{type}_i \quad \Sigma \vdash f:\mathbf{Bijection}[\sigma,\tau] \quad \Sigma \vdash a:\mathbf{PairOf}(\delta[\sigma],\ \eta[\sigma]) \quad \Sigma \vdash b:\mathbf{PairOf}(\delta[\tau],\ \eta[\tau])}{\Sigma \vdash \left\{\begin{array}{l}(a \rightsquigarrow_{\mathbf{PairOf}(\delta[\mathfrak{I}(\sigma,\tau,f)],\ \eta[\mathfrak{I}(\sigma,\tau,f)])} b) \\ \quad \Leftrightarrow \\ \pi_1(a) \rightsquigarrow_{\delta[\mathfrak{I}(\sigma,\tau,f)]} \pi_1(b) \wedge \\ \pi_2(a) \rightsquigarrow_{\eta[\mathfrak{I}(\sigma,\tau,f)]} \pi_2(b)\end{array}\right.}$$

$$\frac{\Sigma;\ \alpha:\mathbf{type}_i \vdash \delta[\alpha],\eta[\alpha]:\mathbf{type}_i \quad \Sigma \vdash \sigma,\tau:\mathbf{type}_i \quad \Sigma \vdash f:\mathbf{Bijection}[\sigma,\tau] \quad \Sigma \vdash g:\delta[\sigma] \to \eta[\sigma] \quad \Sigma \vdash h:\delta[\tau] \to \eta[\tau]}{\Sigma \vdash \left\{\begin{array}{l}(g \rightsquigarrow_{\delta[\mathfrak{I}(\sigma,\tau,f)]\to\eta[\mathfrak{I}(\sigma,\tau,f)]} h) \\ \quad \Leftrightarrow \\ \forall x_1:\delta[\sigma] \\ \ \forall x_2:\delta[\tau] \\ \ (x_1 \rightsquigarrow_{\delta[\mathfrak{I}(\sigma,\tau,f)]} x_2) \\ \quad \Rightarrow g(x_1) \rightsquigarrow_{\eta[\mathfrak{I}(\sigma,\tau,f)]} h(x_2)\end{array}\right.}$$

Figure 7: **Some Derived Rules.** The rules in this figure can be derived from the rules in figure 6. The first rule constructs isomorphism relations at types of the form $\mathbf{PairOf}(\alpha:\mathbf{type}_i,\ y:\gamma[\alpha])$. The rule states that $\mathbf{Pair}(\sigma,a)$ is isomorphic to $\mathbf{Pair}(\tau,b)$ if there exists a bijection f from σ to τ that carries a to b. The rules in the second row allow one to determine whether f carries a to b in the case where $\gamma[\alpha]$ is a simple type over α (see the text). There are two base cases not listed in the figure. For $\gamma[\alpha] = \alpha$ we have that $a \rightsquigarrow_{\updownarrow(\sigma,\tau,f)} b$ if and only if $f(a) =_\tau b$. If $\gamma[\alpha]$ does not depend on α we have $a \rightsquigarrow_\gamma b$ if and only if $a =_\gamma b$.

that $\mathbf{Left}(\rho)$ is the variable interpretation that maps x to $\mathbf{Left}(\rho(x))$. $\mathbf{Right}(\rho)$ is defined similarly. For $\rho \in \mathcal{V}[\![\Sigma]\!]$ we have that ρ^{-1} maps x to $\rho(x)^{-1}$. For $\rho,\gamma \in \mathcal{V}[\![\Sigma]\!]$ we have that $\rho \circ \gamma$ is defined if and only if $\mathbf{Right}(\rho) = \mathbf{Left}(\gamma)$ in which case $\rho \circ \gamma$ is the variable interpretation mapping x to $\rho(x) \circ \gamma(x)$.

A fundamental property of Morphoid type theory is that if $\mathcal{V}[\![\Sigma]\!]$ is defined then it is closed under inverse and composition — for $\rho \in \mathcal{V}[\![\Sigma]\!]$ we have $\rho^{-1} \in \mathcal{V}[\![\Sigma]\!]$ and for $\rho,\gamma \in \mathcal{V}[\![\Sigma]\!]$ with $\rho \circ \gamma$ defined we have $(\rho \circ \gamma) \in \mathcal{V}[\![\Sigma]\!]$. Furthermore for $\mathcal{V}_\Sigma[\![e]\!]$ defined and for $\rho,\gamma \in \mathcal{V}[\![\Sigma]\!]$ we have

$$\mathcal{V}_\Sigma[\![e]\!](\rho \circ \gamma) = (\mathcal{V}_\Sigma[\![e]\!]\,\rho) \circ (\mathcal{V}_\Sigma[\![e]\!]\,\gamma)$$

and

$$\mathcal{V}_\Sigma[\![e]\!](\rho^{-1}) = (\mathcal{V}_\Sigma[\![e]\!]\,\rho)^{-1}.$$

Another fundamental property of the value function involves the abstraction ordering. The abstraction ordering can be extended to variable interpretations where we have $\rho \preceq \gamma$ if ρ and γ are defined on the same set of variables and for each variable x we have $\rho(x) \preceq \gamma(x)$. The value function is monotone with respect to the abstraction ordering — for $\rho \preceq \gamma$ we have $\mathcal{V}_\Sigma[\![e]\!]\,\rho \preceq \mathcal{V}_\Sigma[\![e]\!]\,\gamma$.

7 Internalizing Isomorphism

Figure 6 gives inference rules for isomorphism. Figure 7 gives rules which can be derived from the rules in figure 6. The first rule in figure 7 derives isomorphisms at types of the form $\mathrm{PairOf}(\alpha:\mathbf{type}_i,\ \tau[\alpha])$. We note that types **Group** and **TOP** as defined in section 2 can be written as subtypes of pair types of this form. The rule states that two objects $\mathbf{Pair}(\sigma,a)$ and $\mathbf{Pair}(\tau,b)$ of type $\mathrm{PairOf}(\alpha:\mathbf{type}_i,\ \tau[\alpha])$ are isomorphic if there exists a bijection f from σ to τ which carries a to b. The two rules in the second row of figure 7 allow one to determine whether or not f carries a to b in the case where $\tau[\alpha]$ is a "simple type" over

α. To define the simple types we first introduce the notation $\mathbf{PairOf}(\gamma, \eta)$ as an abbreviation for $\mathbf{PairOf}(x : \gamma,\ y : \eta)$ in the case where x does not occur in η. A simple type expression $\gamma[\alpha]$ over the type variable α is then defined to be either the variable α itself, a type γ not containing α, or a type of the form $\mathbf{PairOf}(\delta[\alpha],\ \eta[\alpha])$ or $\delta[\alpha] \rightarrow \eta[\alpha]$ where $\delta[\alpha]$ and $\eta[\alpha]$ are recursively simple type expressions over α. Subtypes of pair types of the form $\mathbf{PairOf}(\alpha : \mathbf{type}_i,\ \gamma[\alpha])$ where $\gamma[\alpha]$ is a simple type over α covers the types $\mathbf{Group}$ and $\mathbf{TOP}$ as well as many other mathematical concepts. We leave the derivation of the rules in the second row of figure 7 as a (tricky and tedious) exercise for the reader.

While the rules in figure 7 are adequate in many situations, they do not cover types such as

$$\mathbf{Pairof}(p{:}\mathbf{Pairof}(\mathbf{type}_i, \mathbf{type}_i),\ y{:}\gamma[p])$$

or

$$\mathbf{Pairof}(G{:}\mathbf{Group},\ H{:}\tau[G])$$

or

$$\mathbf{Pairof}(f{:}\mathbf{type}_i \rightarrow \mathbf{type}_i,\ A{:}\tau[f]).$$

For the general case we need the rules of figure 6.

8 Summary

Morphoid type theory is a type-theoretic foundation for mathematics supporting the concept of isomorphism and the substitution of isomorphics. Morphoid type theory is an extension of classical predicate calculus that avoids the use of propositions-as-types, path induction or squashing. Morphoid type theory may be more comfortable for mathemticians who take a realist or Platonic approach to the practice of mathematics.

References

Thierry Coquand and Gerard Huet. 1988. The calculus of constructions. *Information and computation*, 76(2):95–120.

Martin Hofmann and Thomas Streicher. 1994. The groupoid model refutes uniqueness of identity proofs. In *Logic in Computer Science, 1994. LICS'94. Proceedings., Symposium on*, pages 208–212. IEEE.

HoTT-Authors. 2013. Homotopy type theory, univalent foundations of mathematics. http://hottheory.files.wordpress.com/2013/03/hott-online-611-ga1a258c.pdf.

Per Martin-Löf. 1971. A theory of types.

David McAllester. 2014. Morphoid type theory. *CoRR*, abs/1407.7274.

John C. Reynolds. 1983. Types, abstraction and parametric polymorphism. In *IFIP Congress*, pages 513–523.

Giovanni Sambin and Jan M Smith. 1998. *Twenty Five Years of Constructive Type Theory*, volume 36. Oxford University Press.