Internal Parametricity for Cubical Type Theory

Evan Cavallo
Carnegie Mellon University, USA
ecavallo@cs.cmu.edu

Robert Harper
Carnegie Mellon University, USA
rwh@cs.cmu.edu

Abstract
We define a computational type theory combining the contentful equality structure of cartesian cubical type theory with internal parametricity primitives. The combined theory supports both univalence and its relational equivalent, which we call relativity. We demonstrate the use of the theory by analyzing polymorphic functions between higher inductive types, and we give an account of the identity extension lemma for internal parametricity.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases parametricity, cubical type theory, higher inductive types

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements We thank Carlo Angiuli, Steve Awodey, Daniel Gratzer, Kuen-Bang Hou (Favonia), Dan Licata, Anders Mørberg, and Jonathan Sterling for their comments and insights. Conversations at the 2017 Mathematics Research Communities workshop and with Emily Riehl and Christian Sattler afterwards were instrumental to the first author’s understanding of relativity. We gratefully acknowledge the support of the Air Force Office of Scientific Research through MURI grant FA9550-15-1-0053. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR.

1 Introduction
Cubical type theory [15, 3, 2] is a recent extension of type theory with contentful equality (or paths). The central concept of cubical type theory, inherited from homotopy type theory [26], is that terms can be equal in multiple ways, each of which is a method of translating results between them. The motivating example of contentful equality is structured isomorphism. In informal mathematical practice, it is common to treat isomorphic objects as if they were “the same,” because any interesting property of one will also hold of the other. Cubical type theory makes this informal practice formal: equality of types is isomorphism, which is to say that we have Voevodsky’s univalence axiom [28]. For this to be possible, it is essential that equality be contentful, because two objects can be isomorphic in many ways. The key feature of cubical type theory, in comparison with homotopy type theory, is that it is a programming language, not just a logical formalism; in particular, uses of contentful equality compute.

A user of cubical type theory can also define types with custom equality structure using higher inductive types (HITs) [16, 12]. A simultaneous generalization of inductive types and quotient types, a HIT is freely generated by a collection of constructors, each of which may introduce not only elements but also paths between elements. For example, the following specification defines a type \( \mathbb{Z}/2\mathbb{Z} \) of integers modulo 2 from a type \( \mathbb{Z} \) of integers.

```plaintext
data \( \mathbb{Z}/2\mathbb{Z} \) : \text{Type} where
| in(n : \mathbb{Z}) : \mathbb{Z}/2\mathbb{Z} |
| mod(n : \mathbb{Z}, x : \mathbb{I}) : \mathbb{Z}/2\mathbb{Z} [x = 0 \leftrightarrow \text{in}(n) \mid x = 1 \leftrightarrow \text{in}(n + 2)]
```

© Evan Cavallo and Robert Harper; licensed under Creative Commons License CC-BY 42nd Conference on Very Important Topics (CVIT 2016). 
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
This type has two constructors, \text{in} and \text{mod}. The first introduces an element of \( \mathbb{Z}/2\mathbb{Z} \) for each element of \( \mathbb{Z} \), while the second identifies \( \text{in}(n) \) and \( \text{in}(n+2) \) for every \( n : \mathbb{Z} \). To construct a function out of \( \mathbb{Z}/2\mathbb{Z} \), we simply explain where to send \text{in} and \text{mod}, in direct analogy with the induction principle of an ordinary inductive type. In addition to quotients, HITs permit the definition of higher-dimensional objects, enabling the use of type theory as a domain-specific language for formalizing homotopy-theoretic mathematics [26].

While contentful equality creates new possibilities, it also introduces new obligations. The type of equalities between any pair of objects has its own equality structure, which means that every type actually contains an infinite tower of structure: paths, paths between paths, and so on. A user of HITs is often forced to wrestle with this higher-dimensional structure. A particularly vicious example is provided by the smash product [26, §6.8], a key construction in homotopy theory. The smash product is a binary operator \(-\wedge-\) on the universe \( \mathcal{U}_* := (X : \mathcal{U}) \times X \) of pointed types. Defined as a HIT (see Section 4.3), it is the natural notion of product for \( \mathcal{U}_* \). As such, it satisfies properties such as commutativity and associativity up to isomorphism: for any \( X,Y,Z : \mathcal{U}_* \), we have \( X \wedge Y \simeq Y \wedge X \) and \((X \wedge Y) \wedge Z \simeq X \wedge (Y \wedge Z)\). However, these laws are not so simple to prove. The associator, in particular, involves two stacked applications of \( \wedge \); this means the programmer must wrangle with two-dimensional structure to define functions back and forth, then three-dimensional structure to prove they are inverses. Worse yet, these are only the first level of an infinite hierarchy of laws satisfied by \( \wedge \). For example, Mac Lane’s pentagon relates the two different ways of re-associating the product of four types; to prove it requires building four-dimensional terms. Despite concerted effort [27, 10], a complete formal proof that the smash product is a symmetric monoidal product has yet to be produced.

For all the suffering, these properties seem “obvious” in a way familiar to computer scientists: they look like consequences of parametricity [21]. Parametricity is Reynolds’ crystallization of a property enjoyed by many type theories: programs behave uniformly in their type variables. Reynolds captures this uniformity in the existence of a relational interpretation of type theory that expresses the invariance of type constructions under a broad class of relations. With parametricity, it is often possible to derive what Wadler dubs “free theorems” [29], naturality properties enjoyed by any term of a given type. In our case, we can hope that there are only so many functions \( \alpha : (X,Y,Z: \mathcal{U}_*) \rightarrow (X \wedge Y) \wedge Z \rightarrow X \wedge (Y \wedge Z) \). Perhaps any definable function of this type satisfies Mac Lane’s pentagon?

Contributions

We present a cubical type theory with internal parametricity, a further extension to type theory introduced by Bernardy and Moulin [6, 7, 5, 18, 20, 19]. Just as cubical type theory makes the fact that all constructions act on isomorphisms available to the user, internally parametric type theory exposes that all constructions act on relations, providing an operational account of Reynolds’s denotational presentation of parametricity. In fact, the two are based on the same design principle: the use of dimension variables.

Our contribution can be viewed in two ways. On the one hand, we bring parametricity to bear on problems in cubical type theory, giving in particular a characterization of maps between smash products. On the other, we provide an internally parametric theory that enjoys the extensionality principles of cubical type theory. A lack of function extensionality, for one, is acutely visible when working with Church encodings. By relying on univalence, we are able to eliminate a technical device used by Bernardy, Coquand, and Moulin in their presheaf model [5]. We also explore the use of internal parametricity beyond the initial forays of Bernardy et al., developing the sub-universe of bridge-discrete types as a substitute for the
traditional identity extension lemma. We show by example that the relational interpretations of inductive types can be characterized. Finally, we use this paper as an opportunity to compare and contrast the mechanisms underlying cubical and parametric type theory.

We begin in Section 2 by introducing cubical type theory. In Section 3, we mix in the parametricity primitives. With the theory complete, we make use of it in Section 4, proving results about the smash product and probing the status of the identity extension lemma. In Section 5, we go into more detail on the meaning of the judgments and canonicity. In Section 6, we briefly sketch a presheaf model. We close in Section 7 with a discussion of related work. Complete proofs of our results, and in particular a detailed development of the computational interpretation, can be found in our companion technical report [13].

2 Cubical type theory

Cubical type theory, in its various incarnations, is a means of organizing the data of a type equipped with path structure. Homotopy theory suggests various ways of representing this kind of structure. Empirically, cubical structure is most convenient for the design of type theories, because n-dimensional elements of a type can be represented by terms in a context of n dimension variables. In this section, we recall cartesian cubical type theory [3, 2]: one can substitute De Morgan cubical type theory [15] or a theory based on the Bezem-Coquand-Huber (BCH) substructural cubical sets model [8, 9] without difficulty.

2.1 Path dimensions

Like ordinary type theory, cubical type theory is based on four judgment forms, expressing typehood, type equality, elementhood, and element equality.

\[
\Gamma \vDash A \text{ type} \quad \Gamma \vDash A = B \text{ type} \quad \Gamma \vDash M \in A \quad \Gamma \vDash M = N \in A
\]

For us, these judgments are behavioral specifications on terms of an untyped programming language. Roughly, a program A is a type when, for any instantiation of its hypotheses, it computes to a name in some prescribed set of value types, while M is in A when its instantiations compute to values in the type named by A. Types and elements are equal when they compute to the same values, where values equality is prescribed in advance. To more quickly give the reader a feel for the system, however, we will defer precise definitions of the judgments to Section 5, and instead first present a collection of rules they satisfy.

Cubical type theory is distinguished by the addition of path dimensions, for which we write r,s. These are specified by judgments \(\Gamma \vDash r \text{ pdim}\) and \(\Gamma \vDash r = s \text{ pdim}\) and populated by two constants.

\[
\Gamma \vDash 0 \text{ pdim} \quad \Gamma \vDash 1 \text{ pdim}
\]

We likewise add dimension variable assumptions to the grammar of contexts. We write them in the form \(x : I\), but this is merely suggestive notation: “I” is not the name of a type. We also allow dimension equality assumptions.

\[
\text{ctx} \quad \Gamma \vDash A \text{ type} \quad \Gamma \vDash M \in A \quad \Gamma \vDash M = N \in A \quad \Gamma \vDash r \text{ pdim} \quad \Gamma \vDash r = s \text{ pdim}
\]

A path dimension variable can be pictured as varying in the real unit interval [0, 1]: as \(x\) varies in a term \(x : I \vDash M \in A\), the term M draws out a “line” in A. The line’s “endpoints”
For us, an internal parametricity for cubical type theory. The path apparatus equips each type with some kind of infinite-dimensional relation. It is obtained by substituting 0 and 1 for $x$: writing $-\langle r/x \rangle$ for path dimension substitution, we have $M(0/x) \in A(0/x)$ and $M(1/x) \in A(1/x)$. Path dimension variables behave just like ordinary variables, satisfying the usual structural rules.

Path dimensions provide a judgmental notion of contentful equality: a path between $M_0 \in A$ and $M_1 \in A$ is a term $x : I \gg P \in A$ such that $P(0/x) = M_0$ and $P(1/x) = M_1$. Paths are internalized via path types, shown in Figure 1. Aside from the indices $M_0$ and $M_1$, they behave as “functions out of $I$”: they are introduced by abstraction, eliminated by application, and satisfy β- and η-rules. In general, path types are heterogeneous, meaning that they are dependent functions: they take the form $\text{Path}_{x,A}(M_0, M_1)$ where $x : I \gg A$ type, and applying $Q \in \text{Path}_{x,A}(M_0, M_1)$ at $r$ yields $Q @ r \in A(r/x)$. When $A$ does not depend on $x$, we simply write $\text{Path}_A(M_0, M_1)$.

Using just these few principles of cubical type theory, we can already observe that path types validate function extensionality. Given $F_0, F_1 \in (a:A) \to B$, a pointwise path between them is a term $H \in (a:A) \to \text{Path}_B(F_0a, F_1a)$. Given such an $H$, we obtain a path between $F_0$ and $F_1$ by simply flipping the order of abstraction: $\lambda^2 x. \lambda a. H a @ x \in \text{Path}_{(a:A) \to B}(F_0, F_1)$.

### 2.2 Coercion and composition

The path apparatus equips each type with some kind of infinite-dimensional relation. It is reflexive: given any $M \in A$, we have $\lambda^1 x. M \in \text{Path}_A(M, M)$. However, there is as yet no reason for it to be symmetric or transitive, nor for all constructions to respect it. These properties are ensured by adding two operations: coercion and composition.

The coercion operation turns paths between types into isomorphisms,\(^1\) implementing one direction of the correspondence required by the univalence axiom. Given a line $x.A$ and an

\(^1\) For us, an isomorphism is a function with a left and right inverse.

\(A \simeq B := (f : A \to B) \times (l : B \to A) \times (r : B \to A) \times ((a:A) \to \text{Path}_A(l(fa), a)) \times ((b:B) \to \text{Path}_B(f(rb), b))\)

This is often called an equivalence in the literature, and can be defined in a number of equivalent ways [26, §4].

---

\(\Gamma, x : I \gg A\) type \(\Gamma \gg M_0 \in A(0/x)\) \(\Gamma \gg M_1 \in A(1/x)\) \(\Gamma \gg \text{Path}_{x,A}(M_0, M_1)\) type

\[\Gamma, x : I \gg P \in A\quad \Gamma \gg P(0/x) = M_0 \in A(0/x)\quad \Gamma \gg P(1/x) = M_1 \in A(1/x)\quad \Gamma \gg \lambda^1 x. P \in \text{Path}_{x,A}(M_0, M_1)\]

\[\Gamma \gg Q \in \text{Path}_{x,A}(M_0, M_1)\quad \varepsilon \in \{0, 1\}\quad \Gamma \gg Q @ \varepsilon = M_\varepsilon \in A(\varepsilon/x)\quad \Gamma \gg Q \in \text{Path}_{x,A}(M_0, M_1)\]

\[\Gamma, x : I \gg P \in A\quad \Gamma \gg (\lambda^2 x.P) @ r = P(r/x) \in A(r/x)\quad \Gamma \gg Q \in \text{Path}_{x,A}(M_0, M_1)\]
element \( M \in A(r/x) \) at some index \( r \), coercion produces an element at any other index \( s \).

\[
\Gamma, \ x : \mathbb{I} \gg A \text{ type} \quad \Gamma \gg r, \ s \ pdim \quad \Gamma \gg M \in A(r/x)
\]

We moreover impose the equation \( \text{coe}_{A}(r/s)(M) = M \in A(r/x) \). From this, one can show that \( \lambda a.\text{coe}_{r,s}(a) \in A(r/x) \to A(s/x) \) is in fact an isomorphism. Using coercion, we can see that all constructions respect paths: if we have some property \( B \in A \to U \), a proof \( N \in BM_0 \) that \( B \) holds of some \( M_0 \in A_0 \), and a path \( Q \in \text{Path}_A(M_0, M_1) \), we get a proof \( \text{coe}_{r,N地说}^{x}(Q) \in BM_1 \) that \( B \) holds of \( M_1 \). This fact can be used to invert and compose paths, establishing that path equality is symmetric and transitive.

While coercion gives us all we need of equality, it is not a strong enough “induction hypothesis.” Operationally, the evaluation of a coercion term is guided by the outermost constructor of type line. To explain the reduction of coercion for \( \text{Path} \) types, a second operation, \textit{(homogeneous) composition}, is required to obtain a term with the correct endpoints.

As the purpose of this operation is essentially technical, however, we defer to [3, 2] for details.

### 2.3 Paths in the universe: \( V \) types and univalence

Coercion converts paths of types into isomorphisms. What is left is to go in the other direction: convert isomorphisms into paths. This is accomplished by a new type former: \textit{Glue types} in [15, 2], \textit{V types} in [3], and \textit{G types} in [9]. We use \( V \) types, which are a sufficient special case of Glue types.

The \( V \) constructor does not, strictly speaking, convert an isomorphism into a path. Rather, it takes an isomorphism and a path as input, and produces a second path which is obtained by composing the two inputs. Precisely, it takes a dimension \( r \ pdim \), some \( r = 0 \gg A \text{ type} \) defined at its 0 endpoint, some \( B \text{ type} \), and an isomorphism \( r = 0 \gg E \in A \simeq B \). These inputs form a V-shape.

\[
\begin{array}{c}
A \\
V_r(A, B, E) \\
B_0 \xrightarrow{r} B_1
\end{array}
\]

The output is a type \( V_r(A, B, E) \) satisfying \( V_0(A, B, E) = A \) and \( V_1(A, B, E) = B \).

To transform an isomorphism \( E \in A \simeq B \) of types \( A, B \in U \) into a path, we simply take \( \lambda x.V_x(A, B, E) \in \text{Path}_U(A, B) \). This is the special case of the \( V \) type where \( B \) is constant in the direction of the output (here \( x \)).

### 2.4 Higher inductive types

As described in the introduction, cubical type theory also supports \textit{higher inductive types}, types generated by constructors that may take dimension arguments and be attached at their boundaries to other elements. We refer to [16, 12] for formal treatments of HITs in cubical type theory; for this paper, we will only need an intuitive understanding.

### 3 Internalizing parametricity

We now add \textit{internal parametricity} primitives, closely following Bernardy, Coquand, and Moulin [5]. Reynolds’ parametricity captures the vague concept of “uniformity in type
variables" by the precise concept of *acting on relations*. To say that all constructions act on relations is essentially to say that type theory has a relational semantics. With internal parametricity, we make that interpretation visible inside the theory.

With cubical type theory, the goal is to ensure that all constructions act on isomorphisms. The solution is to equip each type with equality structure via dimension variables, then to identify lines between types with isomorphisms. For internal parametricity, we ensure that constructions act on relations with the same technique, but now identifying lines between types with *type-valued relations*. Where we use the word *path* in cubical type theory, we will use *bridge* in parametric type theory, following Nuyts et al. [20].

### 3.1 Bridge dimensions

Second verse, same as the first: we introduce *bridge dimensions* $r, s, \ldots$ by judgments $\Gamma \triangleright r \triangleright bdim$ and $\Gamma \triangleright r = s \triangleright bdim$ with two constants $\Gamma \triangleright 0, 1 \triangleright bdim$. We use **bold type** to distinguish bridge from path dimensions. We likewise add bridge dimension and equality assumptions—but this time, only equations where one side is a constant.

\[
\begin{align*}
\Gamma \text{ ctx} & \quad \Gamma \triangleright r \triangleright bdim & \in \{0, 1\} \\
\Gamma, x : 2 \text{ ctx} & \quad \Gamma \triangleright r \triangleright bdim & \quad \Gamma, r = \varepsilon \text{ ctx}
\end{align*}
\]

The distinguishing feature of bridge dimensions is that they are *substructural*: they do not support contraction. For any $\Gamma \text{ ctx}$ and $\Gamma \triangleright r \triangleright bdim$, we write $\Gamma^{\langle \varepsilon \rangle}$ for the prefix of $\Gamma$ occurring before $r$ if it is a variable: that is, we define $\Gamma^{\langle \varepsilon \rangle} := \Gamma$ and $(\Gamma, x : 2, \Gamma')^{\langle \varepsilon \rangle} := \Gamma$.\footnote{Here we follow the approach developed by Cheney for nominal dependent type theory [14], except that we (inessentially) use an operation $\Gamma^{\langle \varepsilon \rangle} = \Gamma'$ rather than a judgment $\Gamma \vdash r : 2 \setminus \Gamma'$.}

Then we have the following structural rules for bridge dimension variables.

\[
\begin{align*}
\Gamma, x : 2 & \triangleright x \triangleright bdim & \quad \text{BHyp} \\
\Gamma^{\langle r \rangle}, x : 2 & \triangleright \Gamma' \text{ ctx} & \quad \text{BWeak} \\
\Gamma \triangleright r \triangleright bdim & \quad \Gamma^{\langle r \rangle}, x : 2 \triangleright \Gamma' \triangleright J & \quad \text{BCut}
\end{align*}
\]

The first two rules are unsurprising, but the third contains an essential restriction. To substitute $r$ for $x$ in a judgment $\Gamma, x : 2, \Gamma' \triangleright J$, we must know that neither $\Gamma'$ or $J$ refers to $r$ (if it is a variable). In other words, $r$ must be fresh for $\Gamma'$ and $J$.

The BCH cubical set model uses substructural dimension variables, but recent work on cubical type theories has focused on structural variables. The primary motivation for the shift is to support HITs, whose development remains an open problem in the substructural setting; ease of implementation is another factor. For internal parametricity, however, substructurality is essential to ensure the correct characterization of bridges in function types (Section 3.2) and in the universe (Section 3.3).

As with paths, we introduce types $\text{Bridge}_{x:A}(M_0, M_1)$ of bridges over $x:A$ from $M_0$ to $M_1$. We write $\lambda^2 x.P$ for the values of these types. In accordance with the judgmental structure, a bridge may only be applied to a fresh variable: if $\Gamma^{\langle r \rangle} \triangleright Q \in \text{Bridge}_{x:A}(M_0, M_1)$, then $\Gamma \triangleright Q \circ r \in A(r/x)$. The *Bridge* type constructor implements one direction of our desired correspondence between bridges of types and binary relations: for any line $x:A$, we have the relation $\text{Bridge}_{x:A}(\langle -,- \rangle)$ on $A(0/x)$ and $A(1/x)$.
We call this as a line in direction \( r \). There are three cases: either \( r = 0 \) or \( 1 \), in which case \( M \) is simply a point, or \( r \) is a variable \( x \), in which case \( M \) is a bridge in direction \( x \). The operator takes an argument for each \( x \). Recall that \( a \) being abstracted when \( x \) is in scope, ranges over terms that may mention \( x \). We would like, then, to think of \( x \) as a variable, so \( a(0/x) = a \). We therefore instead introduce an operator, \( \text{extent} \), that performs the substitutions and capture when \( a \) is instantiated. This operator takes the following form; it satisfies rules listed in Figure 2.

\[
\begin{align*}
\Gamma \Rightarrow r \ bdim & \quad \Gamma \vdash r, x: 2 \Rightarrow A \text{ type} & \quad \Gamma \vdash r, x: 2, a: A \Rightarrow B \text{ type} & \quad \Gamma \Rightarrow M \in A(r/x) \\
\Gamma \vdash r, a_0 : A(0/x) & \Rightarrow N_0 \in B(0/x)[a_0/a] & \quad \Gamma \vdash r, a_1 : A(1/x) & \Rightarrow N_1 \in B(1/x)[a_1/a] \\
\Gamma \vdash r, a_0 : A(0/x), a_1 : A(1/x), \pi : \text{Bridge}_{A} & \Rightarrow N \in \text{Bridge}_{A} & \Rightarrow M \in A(r/x) \quad \text{Bridge}_{A} \models (N_0, N_1)
\end{align*}
\]

\[
\begin{align*}
\Gamma \Rightarrow \text{extent}_{r}(M; a_0, N_0, a_1, N_1, a_0, a_1, \pi, N) & \in B(r/x)[M/a] \\
\text{extent}_{r}(M; \cdots) & = N_0[M/a_0] \in B(0/x)[M/a] \\
\text{extent}_{r}(M; \cdots) & = N_1[M/a_1] \in B(1/x)[M/a] \\
\Gamma \vdash r, x: 2 & \Rightarrow M' \in A \\
\Gamma \Rightarrow \text{extent}_{r}(M'(r/x); \cdots) & = N[M'(0/x)/a_0][M'(1/x)/a_1][\lambda^2 x.M'/\pi]@r \in B(r/x)[M'/a]
\end{align*}
\]

Figure 2 The extent operator. We omit straightforwardly inferrable premises for readability.

### 3.2 Bridges at function type: extent

In the standard relational interpretation of type theory, two functions are related when they take related arguments to related results. In other words, \( \text{Bridge}_{A,B}(F_0, F_1) \) should be isomorphic to the following.

\[
(a_0, A(0/x))(a_1, A(1/x))(q: \text{Bridge}_{A}, (a_0, a_1)) \rightarrow \text{Bridge}_{B}[q@x/a](F_0a_0, F_1a_1)
\]  

The first role of substructurality is to validate this principle. As we have seen, structural dimensions give rise to a different principle: \( \text{Bridge}_{A,B}(F_0, F_1) \cong (a: A) \rightarrow \text{Bridge}_{B}(F_0a, F_1a) \) when \( A \) does not depend on \( x \). The proof, sketched in Section 2.1, uses the interchangeability of terms \( \lambda a. \lambda^2 x. P \) and \( \lambda^2 x. \lambda a. P \). Substructurally, however, these are not interchangeable: in the former, \( x \) ranges over dimensions that are fresh for \( a \), whereas no such restriction is in play in the latter. Conversely, structural dimensions do not permit the derivation of a term of type \( \text{Bridge}_{A,B}(F_0, F_1) \) from a term of type (1). The two principles are both derivable in cubical type theory, but only thanks to the presence of coercion, an operation with no equivalent in internal parametricity.

To see how we can get from (1) to \( \text{Bridge}_{A,B}(F_0, F_1) \) with structural dimensions, let \( H \) in the former be given. Abstracting \( x \) and \( a \), our goal is to exhibit a term of type \( B \) that is equal to \( F_0a \) when \( x = 0 \) and \( F_1a \) when \( x = 1 \). Recall that \( a \), being abstracted when \( x \) is in scope, ranges over terms that may mention \( x \). We would like, then, to think of \( a \) as a bridge in direction \( x \), and to supply that bridge to \( H \). In syntax, we would like to write “\( H(a(0/x))(a(1/x))(\lambda^2 x. a) \)”.

This does not quite make sense: \( a \) is a variable, so \( a(0/x) = a \). We therefore instead introduce an operator, \( \text{extent} \), that performs the substitutions and capture when \( a \) is instantiated. This operator takes the following form; it satisfies rules listed in Figure 2.

\[
\text{extent}_{r}(M; a_0, N_0, a_1, N_1, a_0, a_1, \pi, N)
\]

We call this \( \text{extent} \) because it reveals the extent of the term \( M \) as a line in direction \( r \).
The final ingredient is the equivalent of univalence: a characterization of bridges in the universe:

\[ \text{Bridge} : \text{relativistic types} \rightarrow \text{binary type-valued relations} \]

We call this property \textit{structure}, motivated by its use of variable capture: the mapping \( \lambda^2 : (x, M) \mapsto \lambda^2 x. M \) is not stable under all dimension substitutions. If \( M = M'(x, y) \), for example, then applying \( \lambda^2 \) after substituting \( y/x \) results in \( \lambda^2 y . M'(y, y) \), while applying \( \lambda^2 \) before substituting \( y/x \) results in the inequivalent \( \lambda^2 x . M'(x, y) \). However, variable capture does commute with substitution of fresh variables. Returning to the original motivation, \textit{extention} is exactly what is needed to get from (1) to \text{Bridge}_{\lambda_0 : (a : A) \rightarrow (F_0, F_1)}.

\[\begin{align*}
H & \mapsto \lambda^2 x. \lambda a. \text{extent}_x(a; a_0, F_0 a_0, a_1, F_1 a_1, a_0, a_1) \cdot H a_0 a_1) : \text{Bridge}_{\lambda_0 : (a : A) \rightarrow (F_0, F_1)}
\end{align*}\]

It is straightforward to show that this map is in fact an isomorphism.

### 3.3 Bridges in the universe: Gel types and relativity

The final ingredient is the equivalent of univalence: a characterization of bridges in the universe as binary type-valued relations. We call this property \textit{relativity}.

\begin{definition}
A \textit{universe} \( \mathcal{U} \) is relativistic when for every pair of types \( A, B : \mathcal{U} \), the map \( \lambda C. \text{Bridge}_{\lambda_0 : C \rightarrow a.(-, -) : \text{Bridge}_{\lambda_0 : (a : A) \rightarrow (a \times a) \rightarrow \mathcal{U}}} \) is an isomorphism.
\end{definition}

As in cubical type theory, we implement the inverse map with a new type constructor: Gel, so named because it resembles the G types of the BCH model but applies to relations rather than isomorphisms. Rules for Gel types—omitting those for coercion and composition—are displayed in Figure 3. In contrast to V types, coercion and composition in Gel types is simple; this is because the direction of a coercion or composition is always a path dimension and therefore orthogonal to the direction of the Gel type.

Given a dimension \( r \) and a relation \( \Gamma^r : a : B \Rightarrow R \) for which \( r \) is fresh, we obtain a type \( \text{Gel}_r(A, B, a.b.R) \) satisfy \( \text{Gel}_0(A, B, a.b.R) = A \) and \( \text{Gel}_1(A, B, a.b.R) = B \). Its values take the form \( \text{gel}_r(M, N, P) \) where \( M \in A, N \in B, \) and \( P \) is a proof the two are related by \( R \); we have \( \text{gel}_0(M, N, P) = M \in A \) and \( \text{gel}_1(M, N, P) = N \in B \). Given a bridge \( \Gamma, x : 2 \Rightarrow Q \in \text{Gel}_x(A, B, a.b.R) \) over a Gel type, we can project the proof...
We have completed a formulation of internally parametric cubical type theory. Now, we will present a classic application of parametricity: the characterization of bridge-discrete types, and finally, a characterization of maps between smash products.

4 The practice of internal parametricity

We have completed a formulation of internally parametric cubical type theory. Now, we will use it. As a warm-up, we prove that \(\text{bool}\) is isomorphic to its Church encoding. Then we move on to novel results: a characterization of bridges in \(\text{bool}\), the definition and applications of bridge-discrete types, and finally, a characterization of maps between smash products.

4.1 The basics: booleans

A classic application of parametricity is the characterization of Church encodings, definitions of inductive types by their universal properties. Our universes are predicative, so Church encodings likely cannot be used to obtain data types \(\text{ex nihilo}\). If we know that a data type exists, however, we can show that it is isomorphic to its Church encoding.

\[\text{Theorem 3.} \quad \text{The type } B = (X \times U) \rightarrow X \rightarrow X \rightarrow X \text{ is isomorphic to } \text{bool}.\]
Proof. It is easy to exhibit functions in either direction.

\[ \lambda b. (\lambda x. t \cdot \lambda f. \text{if } x(b, t, f)) \in \text{bool} \to B \quad \lambda g. g(\text{true})(\text{false}) \in B \to \text{bool} \]

One inverse condition is immediate. For the other, we must construct a path from \( \lambda X. \lambda t. \lambda f. \text{if } X(b, t, f) \) to \( g \) for all \( g : B \). Applying function extensionality, we assume \( X : \mathcal{U} \) and \( t, f : X \) and aim to connect \( \text{if } X(g(\text{true})(\text{false}); t, f) \) to \( gXtf \).

Intuitively, we want to say that \( g \) is natural in its type argument; specifically, with respect to the map \( \text{if } X(-; t, f) : \text{bool} \to X \). Accordingly, we define the relation given by the graph of this map: \( R : \text{bool} \times X \to \mathcal{U} \) given by \( R(b, a) := \text{Path}_X(\text{if } X(b, t, f), a) \). To obtain the relational interpretation of \( g \) at \( R \), we introduce a dimension \( x \) and apply \( g \) at its Gel type.

\[ g(\text{Gel}(\text{bool}, X, R)) \in \text{Gel}(\text{bool}, X, R) \to \text{Gel}(\text{bool}, X, R) \to \text{Gel}(\text{bool}, X, R) \]

Next, we apply this to the related pairs \((\text{true}, t)\) and \((\text{false}, f)\), in the form of \( \text{gel} \) terms.

\[ g(\text{Gel}(\text{bool}, X, R))(\text{gel}(\text{true}, t, x^I_t))(\text{gel}(\text{false}, f, x^I_f)) \in \text{Gel}(\text{bool}, X, R) \]

Call this term \( W \). If we substitute \( 0 \) for \( x \) in \( W \), each Gel or gel term steps to its first argument; thus \( W(0/x) = g(\text{false}) \). Likewise, \( W(1/x) = gXtf \). The term \( \text{ungel}(x.W) \) is then a proof that these are related by \( R \), which is precisely what we need. \( \blacksquare \)

We can also characterize the bridges in \text{bool}. Intuitively, these are all trivial: \text{bool}'s only elements are the zero-dimensional \true{} and \false{}. To show this, we use parametricity, an interesting parallel with the use of univalence for characterizing paths in HITs.

\textbf{Theorem 4.} For any \( b_0, b_1 : \text{bool} \), we have \( \text{Bridge}_{\text{bool}}(b_0, b_1) \sim \text{Path}_{\text{bool}}(b_0, b_1) \).

\textbf{Sketch.} We will only construct the forward map; for a full proof, see [13, Theorem 11.5]. Let \( q : \text{Bridge}_{\text{bool}}(b_0, b_1) \). Given \( x \), we have \( P_x := \text{Gel}_x(\text{bool}, \text{bool}, \text{Path}_{\text{bool}}(-, -)) \) corresponding to the identity relation on \text{bool}, as well as a map \( F_x : \text{bool} \to P_x \) defined as follows.

\[ F_x := \lambda b. \text{if}_{P_x}(b; \text{gel}_x(\text{true}, \text{true}, x^I_{\text{true}}), \text{gel}_x(\text{false}, \text{false}, x^I_{\text{false}})) \]

Note that \( F_0 = F_1 = \lambda b. \text{if}_{\text{bool}}(b; \text{true}, \text{false}) \). By applying \( F \) pointwise to \( q \), we thus obtain \( \lambda x. F_x(\text{gel}_x(\text{true}, \text{false})) \). It is easy to show that for any \( b : \text{bool} \), there is a path from \( \text{if}_{\text{bool}}(b; \text{true}, \text{false}) \) to \( b \); using coercion, we obtain some \( T \in \text{Bridge}_x P_x(b_0, b_1) \). Applying ungel gives an element of \( \text{Path}_{\text{bool}}(b_0, b_1) \). \( \blacksquare \)

The definition of the forward map uses parametricity; showing that it is an isomorphism uses \textit{iterated parametricity}, which is to say two-bridge-dimensional types. Just as a one-dimensional bridge corresponds to a relation indexed by its boundary (the two endpoint types), relativity and the characterization of bridges at function type suffice to show that two-dimensional bridges satisfy the same characterization, the boundary now being given by four types and four relations in a square shape.

### 4.2 Bridge-discrete types

The booleans are one example of a \textit{bridge-discrete} type, a type with trivial bridge structure.

\textbf{Definition 5.} A type \( A \) is \textit{bridge-discrete} when for all pairs \( a_0, a_1 : A \), the canonical map \( \lambda p. \text{coerce}_{\text{Bridge}_x P_x(a_0, a_1)}(\lambda x. \text{if}_{P_x}(x; a_0, a_1)) \in \text{Path}_x(a_0, a_1) \to \text{Bridge}_x(a_0, a_1) \) is an isomorphism. We write \( \text{isBDisc}(A) \) for the type of proofs that \( A \) is bridge-discrete.
To show that \( A \) is bridge-discrete, it suffices to exhibit any family of isomorphisms between \( \text{Path}_A \) and \( \text{Bridge}_A \) [13, Corollary 10.7]. The type \( \text{isBDisc}(A) \) is a homotopy proposition [26, §3.3]: all proofs of \( \text{isBDisc}(A) \) are equal up to a path.

Bridge-discrete types are worth identifying because they provide an analogue to the identity extension lemma, a standard lemma in parametricity stating that the relational interpretation of a closed type is its identity relation. This does not hold in our theory, in the sense that a homogeneous bridge type \( \text{Bridge}_A(M_0, M_1) \) is not necessarily isomorphic to \( \text{Path}_A(M_0, M_1) \) (take \( A = \mathcal{U} \)). However, we can instead work by inserting bridge-discreteness hypotheses where the lemma would be required. For example, in imitation of Abel et al. [1], we can prove that path equality in a bridge-discrete type is isomorphic to Leibniz equality.

\[ \text{Proposition 6 ([13, §11.2])} \] Let \( A \) be bridge-discrete. For any \( a_0, a_1 : A \), \( \text{Path}_A(a_0, a_1) \) is isomorphic to \( (P : A \to \mathcal{U}) \to P_{a_0} \to P_{a_1} \).

Fortunately, most type constructors preserve bridge-discreteness. We can check that \( \mathcal{U}_{\text{BDisc}} := (X : \mathcal{U}) \times \text{isBDisc}(X) \) is closed under almost all the type formers, with the obvious exception of universes.

\[ \text{Proposition 7.} \] The sub-universe \( \mathcal{U}_{\text{BDisc}} \) is closed under product, function, \( \text{Path} \), and \( \text{Bridge} \) types. It is univalent and relativistic (i.e., closed under \( \forall \) and \( \text{Gel} \) types).

\[ \text{Proof.} \] For the first statement, see [13, Lemma 10.9]. Any sub-universe of a univalent universe is univalent. For the proof of relativicity, see [13, Theorem 10.12].

Per Section 4.1, we also expect that \( \mathcal{U}_{\text{BDisc}} \) is closed under inductive types. Proposition 7 implies that we can also use parametricity in \( \mathcal{U}_{\text{BDisc}} \): for example, one can repeat the proof of Theorem 3 to show that \( \text{bool} \) is isomorphic to \( (X \mathcal{U}) \to \text{isBDisc}(X) \to X \to X \to X \).

Finally, we observe as a consequence of Theorem 4 that the excluded middle for homotopy propositions [26, §3.4] is refuted by internal parametricity. (The excluded middle for all types is already refuted by univalence [26, Corollary 3.27].)

\[ \text{Definition 8.} \] Write \( \text{isProp}(A) := (a_0, a_1 : A) \to \text{Path}_A(a_0, a_1) \). The excluded middle for homotopy propositions is \( \text{LEM} := (X \mathcal{U}) \to \text{isProp}(X) \to (b : \text{bool}) \times \delta_{\text{is}}(b, X, \neg X) \).

\[ \text{Lemma 9.} \] If \( A \) is bridge-discrete, then any function \( f : \mathcal{U} \to A \) is constant.

\[ \text{Proof.} \] For any pair of types \( X, Y : \mathcal{U} \), we have a bridge \( B := \lambda^2 x. \text{Gel}_x(X, Y, \varnothing, X) \) between them, thus a bridge \( \lambda^2 x. f(B \circ x) : \text{Bridge}_A(fX, fY) \) between their images by \( f \), thus a path from \( fX \) to \( fY \) by bridge-discreteness of \( A \).

\[ \text{Theorem 10.} \] There is a term of type \( \text{LEM} \to \bot \).

\[ \text{Proof.} \] We refute the weak excluded middle \( \text{WLEM} := (X \mathcal{U}) \to (b : \text{bool}) \times \delta_{\text{is}}(b; \neg X, \neg \neg X) \), the special case of \( \text{LEM} \) that decides negated types. (Any negated type is a homotopy proposition.) Let \( f : \text{WLEM} \). Then \( \lambda X. \text{fst}(fX) \in \mathcal{U} \to \text{bool} \). By Theorem 4 and Lemma 9, this map is constant. But \( \text{fst}(\bot) \) and \( \text{fst}(\top) \) cannot be equal, so we have a contradiction.

### 4.3 The smash product

We adopt the convention of writing \( A_* : \mathcal{U}_* := (X : \mathcal{U}) \times X \) for a pointed type, \( A := \text{fst}(A_*) \) for its underlying type, and \( a_0 := \text{snd}(A_*) \) for its basepoint. Given \( A_*, B_* \in \mathcal{U}_* \), we write \( A_* \to_* B_* := (f : A \to B) \times \text{Path}_B(fa_0, b) \) for the type of basepoint-preserving functions from \( A_* \) to \( B_* \). Given \( A_*, B_* : \mathcal{U}_* \), their smash product is the following HIT.
data $A_\ast \land B_\ast : \mathcal{U}$ where
| pair($a : A, b : B) : A_\ast \land B_\ast$
| baser : $A_\ast \land B_\ast$
| gluel(b : B, x : 1) : $A_\ast \land B_\ast$ [x = 0 \iff baser | x = 1 \iff pair(a, b)]
| gluer(a : A, x : 1) : $A_\ast \land B_\ast$ [x = 0 \iff baser | x = 1 \iff pair(a, b)]

In words, the smash product of $A_\ast$ and $B_\ast$ is the cartesian product of their underlying types modulo the relation equating all pairs of the form $(a_0, b)$ or $(a, b_0)$. The smash product can itself be made a pointed type $A_\ast \land B_\ast$ with basepoint pair$(a_0, b_0)$.

Our goal is to characterize the polymorphic pointed endofunctions on $n$-ary smash products. Here, we will only consider the binary case, and we will only sketch the proof. We give detailed proofs of the arguments that use parametricity directly in [13, Appendix C], and we have formalized the purely cubical arguments in the redtt cubical proof assistant [25, cool.smash].

**Theorem 11.** Any function $f_\ast : (X_\ast, Y_\ast : \mathcal{U}_\ast) \rightarrow X_\ast \land Y_\ast \rightarrow_\ast X_\ast \land Y_\ast$ is connected by a path to either the polymorphic identity or the polymorphic constant function.

That we cannot squeeze a complete proof into this space may seem to undermine our case for parametricity’s usefulness. However, our argument is not that it is *easy*, but that it *scales*. After establishing the above, we will argue that no combinatorial explosion results from generalizing to $n$-ary smash products, in contrast to the more direct approaches.

**Definition 12.** Given $f : A \rightarrow B$, write $\text{Gr}_r(A, B, f) := \text{Gel}_r(A, B, a.b.\text{Path}_B(fa, b)).$

Given $f_\ast : A_\ast \rightarrow B_\ast$, define $\text{Gr}_r^\ast(A_\ast, B_\ast, f_\ast) := \langle \text{Gr}_r(A, B, f), \text{Gel}_r(a_0, b_0, f_0) \rangle \in \mathcal{U}_\ast$.

The smash product has an action: from $f_\ast : A_\ast \rightarrow C_\ast$ and $g_\ast : B_\ast \rightarrow D_\ast$, we obtain $f_\ast \land g_\ast \in A_\land B_\land C_\land D_\land$. The following extracts results from products of Gr types.

**Lemma 13** (Graph Lemma for $\land$). Let pointed types $A_\ast, B_\ast, C_\ast, D_\ast : \mathcal{U}_\ast$ and pointed functions $f_\ast : A_\ast \rightarrow C_\ast$, $g_\ast : B_\ast \rightarrow D_\ast$ be given. For any $x$, there is a map of type

\[
\text{Gr}_{x_\ast}^\ast(A, C, f) \land \text{Gr}_{x_\ast}^\ast(B, D, g) \rightarrow \text{Gr}_x(A_\land B_\land C_\land D_\land, f_\ast \land g_\ast)
\]

equal to the identity function on $A_\land B_\land$ when $x = 0$ and on $C_\land D_\land$ when $x = 1$.

**Sketch.** This is analogous to Theorem 4; we define the map by smash product induction. We use extent and unglue to extract relation witnesses in the pair, gluel, and gluer cases.

**Proposition 14.** Any element of $\text{bool}_\ast \land \text{bool}_\ast$ is either $\text{pair}(\text{true}, \text{true})$ or $\text{pair}(\text{false}, \text{false})$.

**Lemma 15.** Any $f : (X_\ast, Y_\ast : \mathcal{U}_\ast) \rightarrow X \rightarrow Y \rightarrow X_\ast \land Y_\ast$ is connected by a path to either

1. the pairing function $\lambda(X, x_0).\lambda(Y, y_0).\lambda.a.\lambda.b.\text{pair}(a, b)$, or
2. the constant basepoint function $\lambda(X, x_0).\lambda(Y, y_0).\lambda.a.\lambda.x_0.\text{pair}(x_0, y_0)$.

**Proof.** Let $X_\ast, Y_\ast : \mathcal{U}_\ast$, $a : X$, and $b : Y$ be given. We have a function $g_\ast^X \in \text{bool} \rightarrow_\ast X_\ast$ taking $\text{true}$ to $x_0$ and $\text{false}$ to $a$, likewise $g_\ast^Y \in \text{bool} \rightarrow_\ast Y_\ast$ taking $\text{true}$ to $y_0$ and $\text{false}$ to $b$.

Fix a fresh bridge dimension $x$. By taking the Gel types for the graphs of the above functions, we obtain pointed types $G^X_\ast := \text{Gr}_{x_\ast}^\ast(\text{bool}, X, g^X)$ and $G^Y_\ast := \text{Gr}_{x_\ast}^\ast(\text{bool}, Y, g^Y)$. We apply $f$ with the elements of $G^X$ and $G^Y$ corresponding to $a$ and $b$.

\[
f(G^X_\ast)(G^Y_\ast)(\text{gel}_x(\text{false}, a, \lambda^1_\ast.a))(\text{gel}_x(\text{false}, b, \lambda^1_\ast.b)) \in G^X_\ast \land G^Y_\ast
\]
At \( x = 0 \), this is equal to \( f(\text{bool},)\)(\text{false}))(\text{false}) \in \text{bool} \land \text{bool} \); at \( x = 1 \), it is equal to \( fX,Y,ab \in X \land Y \). By Lemma 13, we obtain a term in \( \text{Gr}_a(\text{bool} \land \text{bool}, X \land Y, gX^2 \land gY^2) \) with the same endpoints. Applying ungel, we get a proof that these endpoints are in the graph of \( gX^2 \land gY^2 \), i.e., that \( (gX^2 \land gY^2)(f(\text{bool},)\)(\text{false}))(\text{false}) \) is path-equivalent to \( fX,Y,ab \).

By Proposition 14, \( f(\text{bool},)\)(\text{false}))(\text{false}) \) has two possible values. If it is \( \text{pair(true, true)} \), then \( fX,Y,ab \) must be \( \text{pair}(x_0, y_0) \); if it is \( \text{pair(false, false)} \), then \( fX,Y,ab \) is \( \text{pair}(a, b) \).

**Sketch of Theorem 11.** To characterize \( f_\ast : (X_\ast, Y_\ast) \rightarrow X_\ast \land Y_\ast \rightarrow X_\ast \land Y_\ast \), we must characterize its behavior on each constructor, as well as the proof that it preserves the basepoint of \( X_\ast \land Y_\ast \). Given \( X,Y \), write \( fX,Y \) for the function underlying \( f_\ast X,Y \).

Write \( P \coloneqq (X_\ast, Y_\ast, \mathcal{U}_\ast) \rightarrow X \rightarrow Y \rightarrow X \land Y \). First, we isolate the behavior of \( f_\ast \) on the \( \text{pair} \) constructor: \( \lambda X_\ast.Y_\ast.\lambda a.\lambda b.fX,Y_\ast(\text{pair}(a, b)) \in P \). By Lemma 15, this is one of two functions. We aim to show that this is the only degree of freedom available to \( f_\ast \).

The values of \( f \) on the \text{basel} and \text{baser} constructors are uniquely determined up to a path by the fact that \( f_\ast X,Y_\ast \) is basepoint-preserving, as \text{basel} and \text{baser} are connected to the basepoint of \( X_\ast \land Y_\ast \) by \( \text{gluel}(y_0, -) \) and \( \text{gluer}(x_0, -) \) respectively.

For \text{gluel}, we consider the term \( H \coloneqq \lambda X_\ast.\lambda Y_\ast.\lambda a.\lambda b.fX,Y_\ast(\text{gluel}(b, x)) \), which is a path in \( P \) from \( \lambda X_\ast.\lambda Y_\ast.\lambda a.\lambda b.fX,Y_\ast(\text{basel}) \) to \( \lambda X_\ast.\lambda Y_\ast.\lambda a.\lambda b.fX,Y_\ast(\text{pair}(x_0, b)) \). By Lemma 15, we know that \( P \) is isomorphic to \text{bool}, which means in particular that it is a homotopy set: its path types are all homotopy propositions. So \( H \), and therefore the behavior of \( f_\ast \) on \text{gluel} terms, is uniquely determined (up to a path). The same applies to \text{gluer}.

Finally, write \( f_0 : (X_\ast, Y_\ast, \mathcal{U}_\ast) \rightarrow \text{Path}_{X_\ast \land Y_\ast}(fX,Y_\ast(\text{pair}(x_0, y_0))), \text{pair}(x_0, y_0)) \) for the proof that \( f \) preserves the basepoint of \( X_\ast \land Y_\ast \). As with \text{gluel}, we prove that \( f_0 \) is uniquely determined by recasting it as a path in \( P \), namely the path \( \lambda x.\lambda X_\ast.\lambda Y_\ast.\lambda a.\lambda b.f_0 X,Y_\ast(\text{pair}(a, b)) \) that connects \( \lambda X_\ast.\lambda Y_\ast.\lambda a.\lambda b.fX,Y_\ast(\text{pair}(x_0, y_0)) \) to \( \lambda X_\ast.\lambda Y_\ast.\lambda a.\lambda b.\text{pair}(x_0, y_0) \).

To prove the \( n \)-ary generalization of this theorem, we can proceed by an inductive argument, showing for each \( i \leq n \) that there are exactly two maps of the following type.

\[
(X_1, \ldots, X_n, \mathcal{U}_\ast) \rightarrow X_1 \rightarrow \cdots X_{n-i-1} \land X_i \rightarrow X_{n-i} \rightarrow \Lambda X_1 \land X_i
\]

For the base case \( i = 0 \), we use an easy generalization of Lemma 15; for the inductive step, the argument in the proof of Theorem 11. What is key is that we never use an iterated induction argument on the elements of stacked smash products, so we never find ourselves dealing with a two-dimensional case like that of \( \text{gluel}(\text{gluel}(c, x), y) \in X_\ast \land (Y_\ast \land Z_\ast) \).

To see how we can apply the theorem, consider the case of commutativity. If we have \( K : (X_\ast, Y_\ast, \mathcal{U}_\ast) \rightarrow X_\ast \land Y_\ast \rightarrow Y_\ast \land X_\ast \), then \( \lambda X_\ast.\lambda Y_\ast.K_{X_\ast,Y_\ast} \circ K_{Y_\ast,X_\ast} \) is a polymorphic endofunction on smash products. By Theorem 11, we can show it is the identity simply by showing it is not constant, which we can do by instantiating \( X_\ast \land Y_\ast \) with \text{bool}, \text{bool}, and testing it on \text{pair}(false,false). If it is indeed non-constant, we see that \( K \) is an isomorphism. Similar techniques show that any non-constant associator is an isomorphism and satisfies Mac Lane’s pentagon.

## 5 Computational meaning of the judgments

We now explain more precisely the meanings of the typing judgments. We follow the work of Angiuli et al. [3]. The central idea is that well-typed terms need not only evaluate to values, but evaluate in a way that is coherent with respect to substitution of dimensions.

First, we have an operational semantics, specified by judgments \( M \text{ val} \) and \( M \rightarrow M' \).

We write \( M \downarrow V \) when \( M \rightarrow^* V \) and \( V \text{ val} \). The “closed” types and terms—those to which...
the operational semantics applies—are those whose free variables are all dimensions, i.e.,
those well-formed in some \( \Phi \Psi \) context where \( \Phi = x_1 : 2, \ldots, x_n : 2 \) and \( \Psi = y_1 : \|, \ldots, y_m : I \). A type system on this language is a five-place relation \( \tau(\Phi, \Psi, A_0, A_0', \varphi) \). It specifies, for each context \( \Phi \Psi \), those values \( A_0 \) and \( A_0' \) that are equal value types in that context, and associates to each such pair a partial equivalence relation (PER) \( \varphi \) on values in context \( \Phi \Psi \). To be a type system, \( \tau \) is required to satisfy laws ensuring symmetry, transitivity, and so forth. To interpret the type formers described in Sections 2 and 3, we can define an appropriate \( \tau \) by a fixed-point construction populating it with the various constructors.

Two terms \( A \) and \( A' \) in context \( \Phi \Psi \) are then equal types when they evaluate to equal value types in a way that commutes with dimension substitution.

Definition 16. Given \( \tau \), define a five-place relation \( PTy(\tau)(\Phi, \Psi, A, A', \alpha) \) on context \( \Phi \Psi \), terms \( A, A' \), and families \( (\alpha)_{\psi: \Psi \cdot \Phi \rightarrow \Phi} \) indexed by substitutions into \( \Phi \Psi \), as follows. \( PTy(\tau)(\Phi, \Psi, A, A', \alpha) \) holds when for all \( \psi_1 : \Phi_1 \Psi_1 \rightarrow \Phi \Psi \) and \( \psi_2 : \Phi_2 \Psi_2 \rightarrow \Phi_1 \Psi_1 \), we have

\[
A_{\psi_1} \downarrow_{A_1} A_{\psi_2} \downarrow_{A_2} A_{\psi_1'} \downarrow_{A_1'} A_{\psi_2'} \downarrow_{A_2'} A_{\psi_1'} A_{\psi_2'} \downarrow_{A_{12}}
\]

with \( \tau(\Phi_2, \Psi_2, V, V', \alpha_{\psi_1 \psi_2}) \) for all \( V \in \{A_2, A_{12}\} \) and \( V' \in \{A_2, A_{12}'\} \).

We write \([\alpha]\) for the \( \alpha \) such that \( PTy(\tau)(\Phi_2, \Psi_2, A, A, \alpha) \) when it exists; the laws required of type systems ensure its uniqueness. We now say that \( \Phi \Psi \gg A = A' \) type when \( PTy(\tau)(\Phi, \Psi, A, A', \alpha) \) for some \( \alpha \) (satisfying a certain coherence condition). As a special case, we say that \( \Phi \Psi \gg A \) type when \( \Phi \Psi \gg A = A \) type. Element equality is defined analogously: \( M \gg M' \) are equal in \( A \) when they coherently evaluate to equal values in \([A]\). Finally, the closed judgments are extended to open judgments by functionality: \( \Gamma \gg A = A' \) holds when \( \Phi \Psi \gg A \gamma = A' \gamma' \) type for all \( \Phi \Psi \gg \gamma = \gamma' \in \Gamma \).

By definition of \( \Phi \Psi \gg M \in \text{bool} \), we obtain a canonicity theorem.

Theorem 17 (Canonicity). If \( \Phi \Psi \gg M \in \text{bool} \), then either \( M \mapsto^* \text{true} \) or \( M \mapsto^* \text{false} \). Additionally, either \( \Phi \Psi \gg M = \text{true} \in \text{bool} \) or \( \Phi \Psi \gg M = \text{false} \in \text{bool} \).

6 Presheaf model

As we have said, the rules presented in Sections 2 and 3 can be used as a formalism for reasoning in parametric cubical type theory. Following prior work on cubical type theory [15, 2] and internal parametricity [5], this formalism also supports a presheaf semantics.

The two base categories at play, defined using the notation of Buchholtz and Morehouse [11], are the BCH cube category \( \mathcal{BCH} = C_{(\text{wec,})} \) and the cartesian cube category \( \mathcal{C}_{\text{wec}} \). We model parametric cubical type theory in the category \( C := \text{Set}(\mathcal{BCH}, \mathcal{BCH})^{\text{op}} \) of presheaves on the product of the two. We can immediately obtain an interpretation of the standard and cubical type formers (including the universe) by applying a result of Angiuli et al. [2, Theorem 1]. For the interval, we take \((x : \|) \in \mathcal{BCH} \times \mathcal{BCH} \); for the generating cofibrations \( \text{Cof} \subseteq \Omega_{\text{dec}} \), we take finite unions of equations of the form \( r = s \) and \( r = e \). It is straightforward to check that these choices satisfy the axioms required to apply their theorem.

On the parametric side, we follow the BCH model. Given a context \( \Gamma \) interpreted as some \([\Gamma] : C \), its extension \( \Gamma, x : 2 \) is interpreted as the separated product \( [\Gamma] \otimes [x : 2, .]. \) whose elements at \( (\Phi, \Psi) \in \mathcal{BCH} \times \mathcal{BCH} \) are pairs \((\gamma, r)\) with \( r \in \Phi \cup \{0, 1\} \) and \( \gamma \in [\Gamma'][\Phi \setminus \{r\}, \Psi] \). Bridge types are interpreted à la BCH path types; the inclusion of equations \( r = e \) in \( \text{Cof} \) ensures that they support coercion and composition. Gel types can be interpreted similarly to BCH G types (though coercion and composition are much simpler in our case).
One advantage of univalence is that we can obtain a relativistic universe without replacing sets with so-called $I$-sets, as is done by Bernardy et al. [5]. In their theory, the isomorphism implemented by gel and ungel is replaced by an equality $\text{Bridge}_U(\text{Gel}_{(A,B,R)}(M,N), R(M,N))$. To ensure that the interpretations of these types have exactly the same elements, sets are everywhere replaced with $I$-sets. In our notation, these would be $\Phi$-sets: for $\Phi \in \mathcal{P}_B$, a $\Phi$-element is a family indexed by subcontexts $\Phi' \subseteq \Phi$, and a $\Phi$-set is a set of $\Phi$-elements. Interpreting types as families of $I$-sets makes it possible to give an interpretation of Gel and Bridge types that validates the above equation. In our work, on the contrary, the equality is replaced by an isomorphism, obviating the need for $I$-sets. By univalence, we can still obtain a path and so establish the target isomorphism between $\text{Bridge}_U(A,B)$ and $A \times B \rightarrow \mathcal{U}$.

7 Related and future work

Our parametric cubical type theory is, for the most part, simply the combination of Angiuli et al.'s cartesian cubical type theory [3] and Bernardy et al.'s internally parametric type theory [5]. (As our notation differs substantially from the latter's, we include a translation dictionary in [13, Figure 7].) We work with binary rather than unary parametricity, but as Bernardy et al. themselves remark, this requires only cosmetic changes. There is little interaction between the two halves of the theory; we need only to check that coercion and composition can be defined for the types introduced by internal parametricity. For bridge types, this requires a minor change to the definition of composition (the addition of $r = \varepsilon$ tube equations). As discussed in Section 6, our Gel types satisfy fewer equations than the corresponding types in [5]; accordingly, our proof of relativity is novel.

A second approach to internal parametricity has been developed by Nuyts, Vezzosi, and Devriese [20]. Like ours, their system is based on paths and bridges. Where ours are almost entirely separate, however, theirs are connected by a modality, which mediates between variable uses in continuous and parametric positions. Intuitively, their goal is to internalize the independence of element-level calculation from terms at the type level, whereas ours is merely to internalize the relational interpretation. The divergence of aims leads to very different considerations; in particular, their bridge dimensions are structural. In later work [19], Nuyts and Devriese generalize from paths and bridges to an infinite tower of relations.

Parametric cubical type theory also strongly resembles Riehl and Shulman’s directed type theory [23]. Where ours has semantics in presheaves on $\mathcal{P}_B \times \mathcal{P}_P$, theirs is aimed at presheaves on $\Delta \times \Delta$, two copies of the simplex category. In both cases, one half of the base category is used for equality, while the other endows types with a relational structure. For directed type theory, the goal is to identify those types for which the relational structure is actually a category structure, meaning that concatenable bridges have path-unique composites. Interestingly, their model does not appear to admit a relativistic universe [22].

Stepping outside the realm of internalization, there has been general interest in proof-relevant and higher-dimensional generalizations of parametricity, for example in the work of Benton et al. [4], Ghani et al. [17], and Sojakova and Johann [24].

One unsolved problem looms large: to what extent do results in parametric type theory translate into ordinary type theory? Naturally, not all theorems translate—$(f : \mathcal{U} \rightarrow \text{bool}) \rightarrow \text{Path}_{\text{bool}}(f \top, f \bot)$ is provable in parametric type theory but refuted in classical cubical sets—but one reasonable conjecture is that proofs $\Gamma \Rightarrow M \in A$ translate when $\Gamma$ and $A$ use no function types. It appears fairly straightforward to obtain results of this kind in semantics, but syntactic results are much murkier; similar conservativity questions for homotopy and cubical type theory over ordinary type theory remain open.
References


