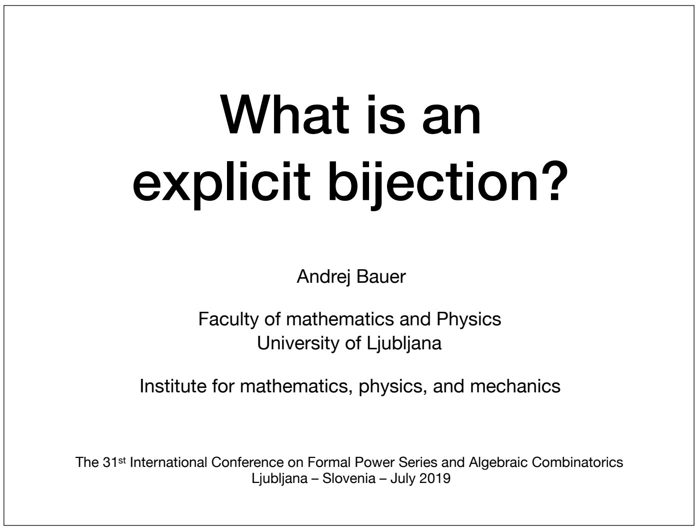


What is an explicit bijection?

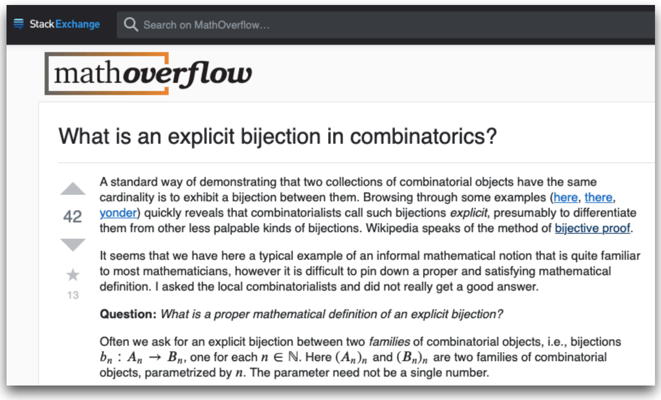
Andrej Bauer

Faculty of mathematics and Physics
University of Ljubljana

Institute for mathematics, physics, and mechanics

The 31st International Conference on Formal Power Series and Algebraic Combinatorics
Ljubljana – Slovenia – July 2019

Thank you very much for the invitation to give the "outsider" talk. I am honored.
My area of work spans mathematics and computer science, but in this talk I would like to speak about the foundations of mathematics, namely homotopy type theory, or "HoTT".
My goal here is to explain why HoTT is interesting for the ordinary mathematician in the street. But where to start?



What is an explicit bijection in combinatorics?

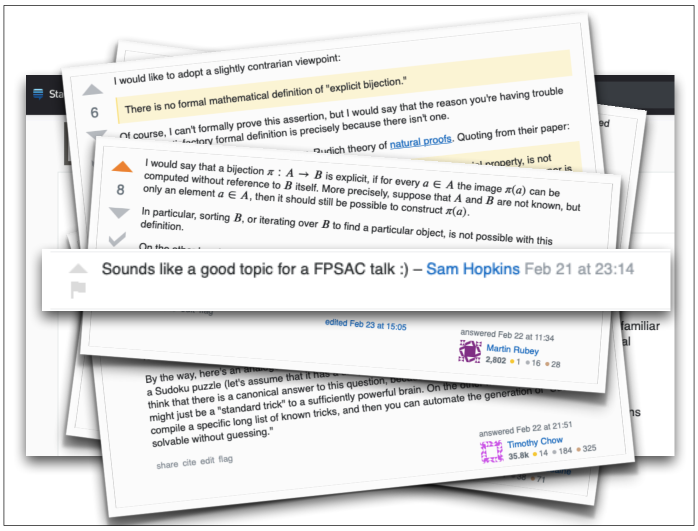
A standard way of demonstrating that two collections of combinatorial objects have the same cardinality is to exhibit a bijection between them. Browsing through some examples (here, there, yonder) quickly reveals that combinatorialists call such bijections *explicit*, presumably to differentiate them from other less palpable kinds of bijections. Wikipedia speaks of the method of bijective proof.

It seems that we have here a typical example of an informal mathematical notion that is quite familiar to most mathematicians, however it is difficult to pin down a proper and satisfying mathematical definition. I asked the local combinatorialists and did not really get a good answer.

Question: *What is a proper mathematical definition of an explicit bijection?*

Often we ask for an explicit bijection between two *families* of combinatorial objects, i.e., bijections $b_n : A_n \to B_n$, one for each $n \in \mathbb{N}$. Here $(A_n)_n$ and $(B_n)_n$ are two families of combinatorial objects, parametrized by n. The parameter need not be a single number.

One task of mathematical foundations is making precise notions that one sees in practice, but which lack a proper mathematical treatment. The idea of *explicit bijection* in combinatorics fits the bill, I think. It is well understood and used in practice, but the notion does not seem to have a proper definition (compare for instance with “continuous function”). Just to be sure, I asked about it on MathOverflow.



6

I would like to adopt a slightly contrarian viewpoint:

There is no formal mathematical definition of "explicit bijection."

Of course, I can't formally prove this assertion, but I would say that the reason you're having trouble
formal definition is precisely because there isn't one.

Rudich theory of natural proofs. Quoting from their paper:

property, is not

8

I would say that a bijection $\pi : A \to B$ is explicit, if for every $a \in A$ the image $\pi(a)$ can be computed without reference to B itself. More precisely, suppose that A and B are not known, but only an element $a \in A$, then it should still be possible to construct $\pi(a)$.

In particular, sorting B, or iterating over B to find a particular object, is not possible with this definition.

Sounds like a good topic for a FPSAC talk :) – Sam Hopkins Feb 21 at 23:14

edited Feb 23 at 15:05

answered Feb 22 at 11:34
Martin Rubey
2,802 ● 1 ● 16 ● 28

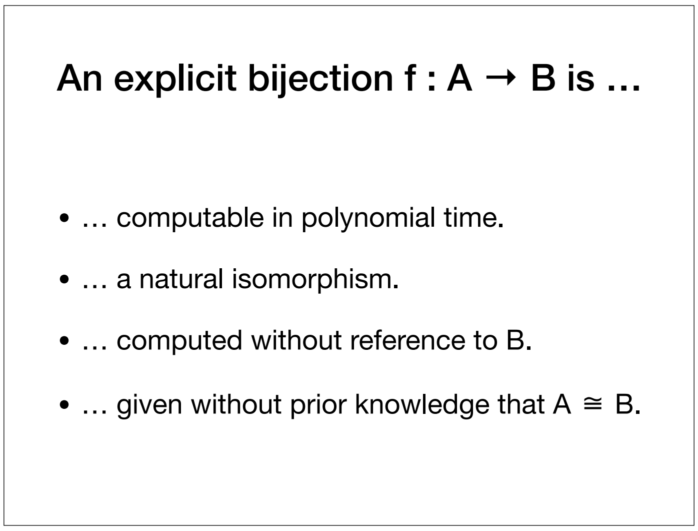
By the way, here's an
a Sudoku puzzle (let's assume that it has
think that there is a canonical answer to this question,
might just be a "standard trick" to a sufficiently powerful brain. On the other
compile a specific long list of known tricks, and then you can automate the generation of
solvable without guessing."

share cite edit flag

answered Feb 22 at 21:51
Timothy Chow
35.8k ● 14 ● 184 ● 325

The question sparked interest (it’s the second most popular question asked by me) and many insightful comments were given, but no definitive answer. So I thought “this would make a catchy title”.

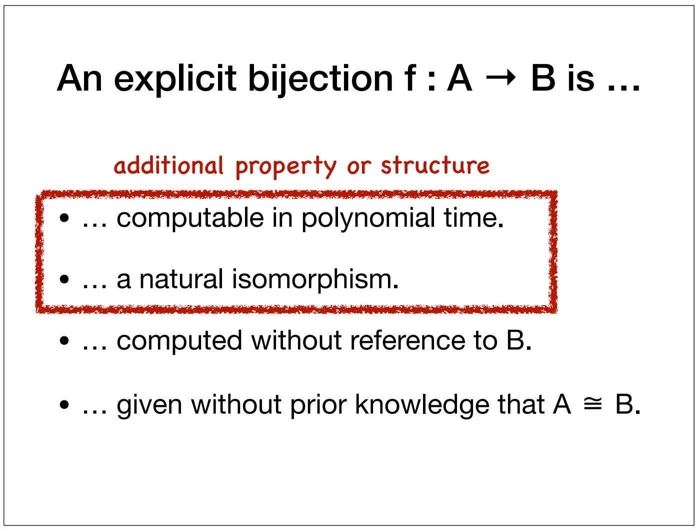
I shall not give you the definitive answer either, but will give *an* answer, and along the way we will see how foundations can shape our understanding of mathematics.



An explicit bijection f : A → B is …

- … computable in polynomial time.
- … a natural isomorphism.
- … computed without reference to B.
- … given without prior knowledge that A ≅ B.

So what is an explicit bijection? The suggestions I got in the MathOverflow answers were mainly of two flavors.

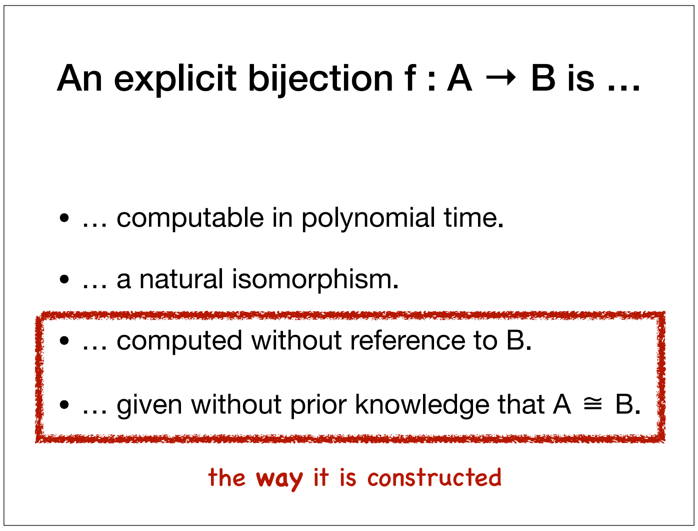


An explicit bijection f : A → B is …

additional property or structure

- … computable in polynomial time.
- … a natural isomorphism.
- … computed without reference to B.
- … given without prior knowledge that A ≅ B.

We may attempt to define "explicitness" as a property, or structure, of a bijection, for instance by requiring computational efficiency or structural properties. These read as proper mathematical definitions. But what if I prove by contradiction that a polynomial-time bijection exists, is it still explicit? I think not.



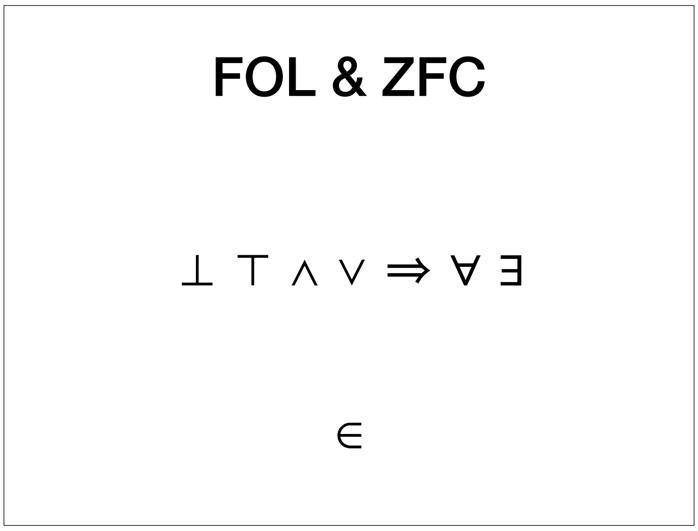
An explicit bijection f : A → B is …

- … computable in polynomial time.
- … a natural isomorphism.
- … computed without reference to B.
- … given without prior knowledge that A ≅ B.

the way it is constructed

The second flavor of answers got closer to the heart of the matter. They put requirements on *how* the bijection was constructed or defined, for instance without reference to a certain object, or without prior knowledge etc. These answers feel more correct, but also less mathematical. What does it mean to construct a bijection without reference to its codomain, or without prior knowledge that the domain and codomain are isomorphic?

Can foundations of mathematics make these ideas precise?



FOL & ZFC

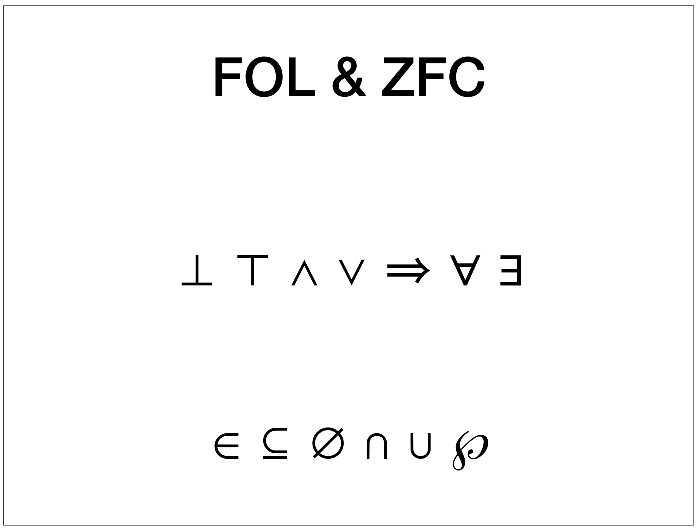
$\bot \quad \top \quad \wedge \quad \vee \quad \Rightarrow \quad \forall \quad \exists$

$\in$

We could look at what is normally considered the “official” foundation: first-order logic and the Zermelo-Fraenkel set theory ZFC.

This formalism has served us extremely well in the study the meta-theoretic properties of set theory, and it has been tremendously influential in unifying the 20th century mathematics, but in its raw form is a bit too removed from how mathematicians work in practice.

It is *possible* to encode ordinary mathematics in ZFC, but the encoding obfuscates and destroys relevant structure. In particular, there is no notion of *construction* in pure ZFC. There is only existence of sets, and *everything* has to be expressed in terms of the membership relation $\in$. For example, you cannot even mention the empty set directly, you have to say “there exists a set such that no x is an element of it” every single time.



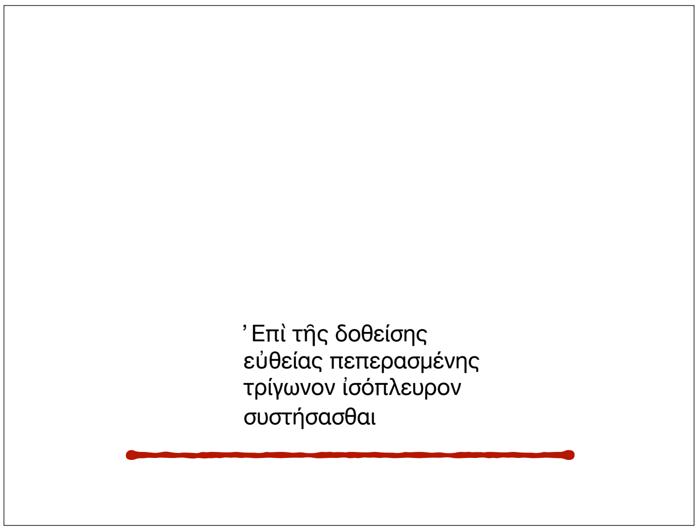
FOL & ZFC

$\bot \quad \top \quad \wedge \quad \vee \quad \Rightarrow \quad \forall \quad \exists$

$\in \quad \subseteq \quad \varnothing \quad \cap \quad \cup \quad \wp$

Of course, there are meta-theorems that allow us to introduce new symbols and functions in set theory. They guarantee that the new symbols can be eliminated and do not change the expressivity of the theory. Still, these meta-theorems are not flexible enough (in particular, it’s not so straightforward to introduce a notation for subset formation because it mixes terms and formulas).

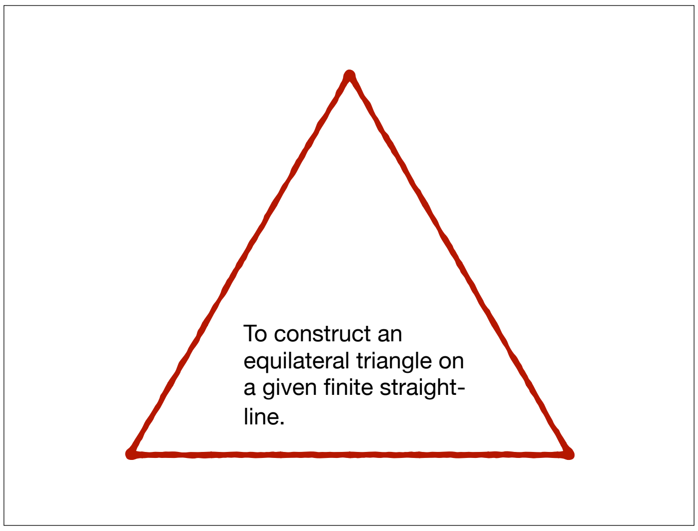
Since we are interested in how bijections, and mathematical objects in general, are constructed, we should have a proper *theory of constructions*.



’Επὶ τῆς δοθείσης
εὐθείας πεπερασμένης
τρίγωνον ἰσόπλευρον
συστήσασθαι

Here is a favorite fact of mine: the first proposition of the first book of Euclid’s elements is *not* a statement of fact. It is a *construction*. It literally says: “to construct an equilateral triangle on a given finite straight-line”.

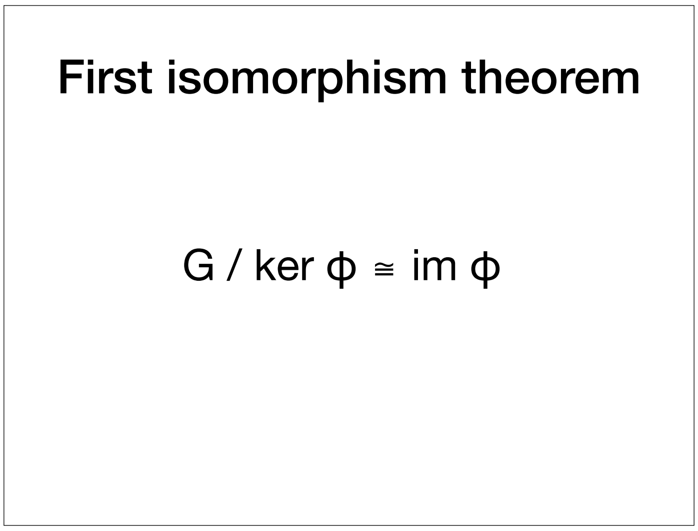
It does *not* claim that there exists such a triangle.


To construct an equilateral triangle on a given finite straight-line.

The solution is not a proof. It is a method by which we create the desired equilateral triangle.

Of course, what I am saying is quite obvious and natural because constructions have always been the bread and butter of mathematics. But the ghost of first-order logic shows up in how we write mathematics. We often make statements of truth where a construction is more appropriate.

Let me give you an example.

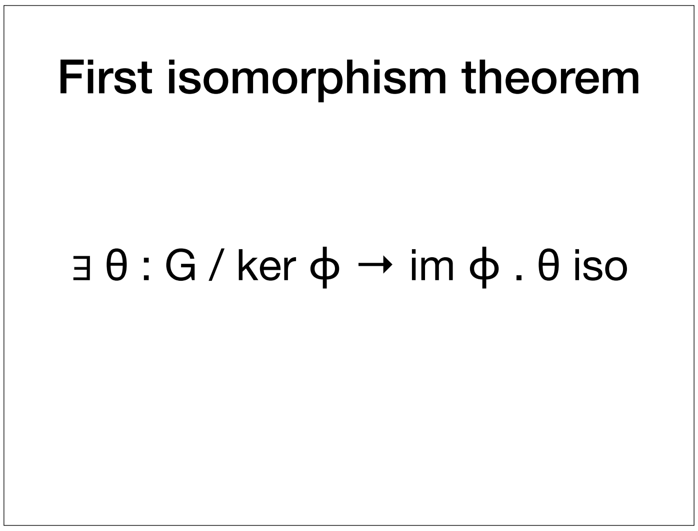


First isomorphism theorem

$$G / \ker \phi \cong \operatorname{im} \phi$$

Here is a well-known theorem from group theory, where G is a group and ϕ a homomorphism from G to some other group.

I am going to be a bit nit-picky, so please bare with me. If you read this as a logical statement, then its form is an existential, i.e., we can unwrap it as follows.

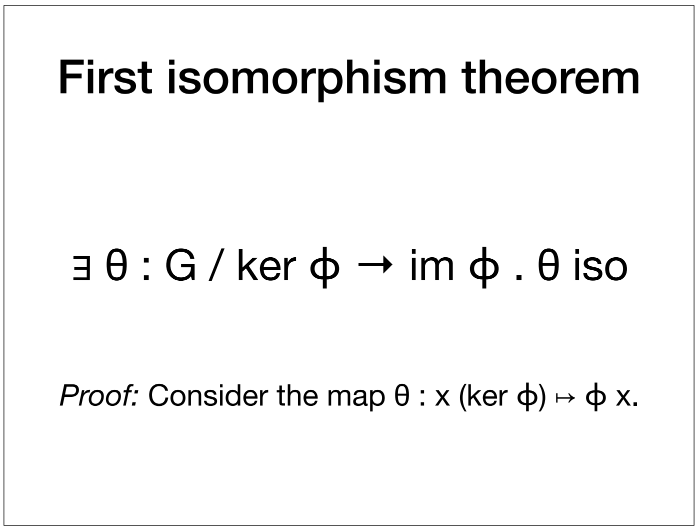


First isomorphism theorem

$\exists\ \theta : G / \ker \phi \to \mathrm{im}\ \phi\ .\ \theta$ iso

We are asked to prove the existence of a certain isomorphism.

As stated, the theorem does not specify *which* isomorphism we are supposed exhibit. In fact, we could prove the theorem by contradiction, and that would be a valid proof that would not exhibit any particular isomorphism at all.



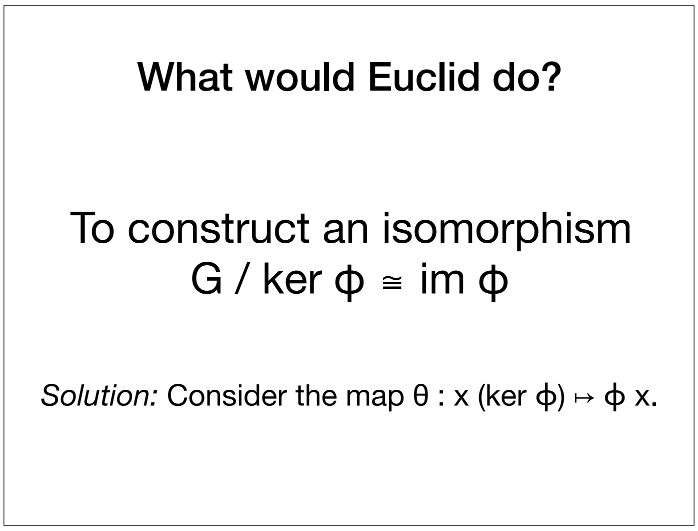
First isomorphism theorem

$$\exists\ \theta : G / \ker \phi \to \mathrm{im}\ \phi\ .\ \theta \text{ iso}$$

Proof: Consider the map $\theta : x\,(\ker \phi) \mapsto \phi\, x$.

But of course, nobody ever does that. *Every* account of the first isomorphism theorem uses a particular map, namely the one induced by ϕ. And everybody know this, and everybody reads both the theorem and the proof in this way.

However, speaking formally, when we prove an existential statement, the witness of existence is *hidden* in the proof. There is no way that it can be extracted – that is simply how the existential quantifier works. And the situation has nothing to do with classical vs. intuitionistic logic – they both have precisely the same inference rule for the existential quantifier.



What would Euclid do?

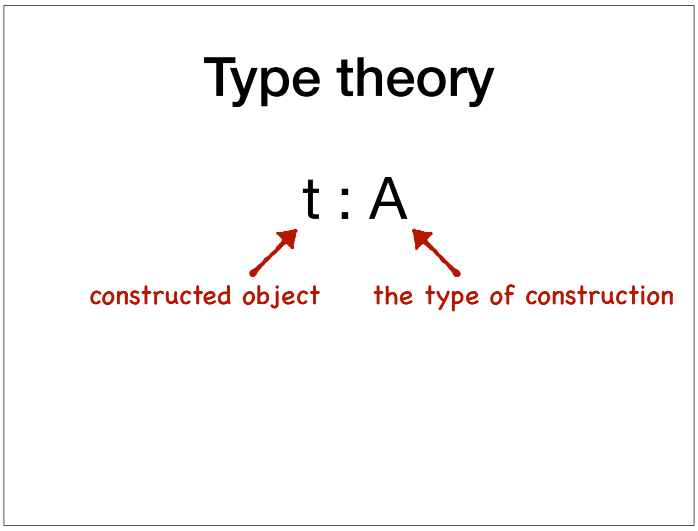
To construct an isomorphism
$G / \ker \phi \cong \operatorname{im} \phi$

Solution: Consider the map $\theta : x (\ker \phi) \mapsto \phi\, x$.

We have a case of discrepancy between formalism and practice. It is an example of how people write one thing but mean another. This is not something that one notices easily, but if you try to formalize group theory and have it all verified by a computer, the computer will make you clean it all up. It's a lot of extra work.

But what would Euclid do? Well, quite simply, he would have asked for a construction: to construct such-and-such isomorphism. The solution would be the isomorphism everyone always gives – but this time it would be an honest construction.

Let us then turn the tables and start with a theory of constructions. No logic, no sets, just bare constructions.



Type theory

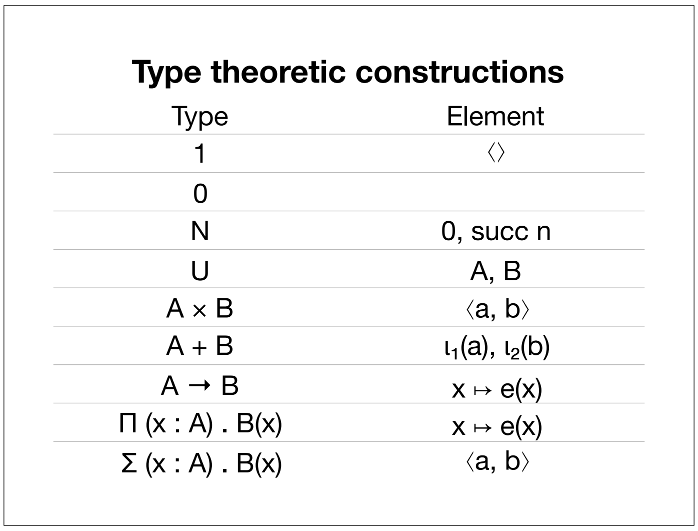
t : A

constructed object　　the type of construction

Type theory is a general theory of constructions, where we should understand the word “construction” in the most general sense, abstractly. We might be constructing points in a space, or combinatorial objects, or elements of a set, or values of a datatype.

It is not about what exists, like set theory, but about how to construct things. To say in type theory that we have an element t of type A means that t was built according to the rules of type A. Of course, to each type A we may associate its *extension*, which is the collection of its elements, but such “collections” are strictly speaking outside the realm of type theory.

We need a good collection of (types of) constructions that will let us do mathematics.



Type theoretic constructions

Type	Element
1	$\langle\rangle$
0	
N	0, succ n
U	A, B
$A \times B$	$\langle a, b\rangle$
$A + B$	$\iota_1(a)$, $\iota_2(b)$
$A \to B$	$x \mapsto e(x)$
$\Pi\,(x : A)\,.\,B(x)$	$x \mapsto e(x)$
$\Sigma\,(x : A)\,.\,B(x)$	$\langle a, b\rangle$

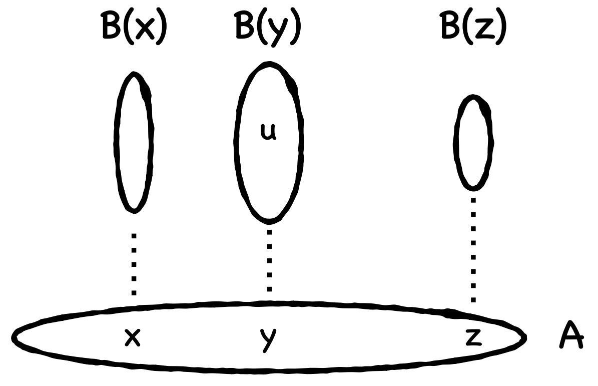
Here is a table of the basic type-theoretic constructions. Each of these comes with some inference rules explaining how to build and use the elements of each type. I emphasize again that there is no underlying logic or set theory. We really start from scratch and give a formal system of rules, through which I am not going to drag you.

First we postulate some basic types, such as the singleton type 1, the empty type 0, and the type of natural numbers N. There is also the type of types U, called a *universe*, whose elements are types. The universe embodies the idea that types themselves are constructions. The usual question of whether a universe is an element of itself comes up. There are various flavors of type theory, some of which allow U : U, but today we will follow the more traditional path and avoid a universe that contains itself. There can be many universes, and they may form a hierarchy.

There are also operations for forming new types: the cartesian product, the disjoint sum, and the type of functions. Hopefully these are familiar, for instance the elements of a cartesian product are pairs.

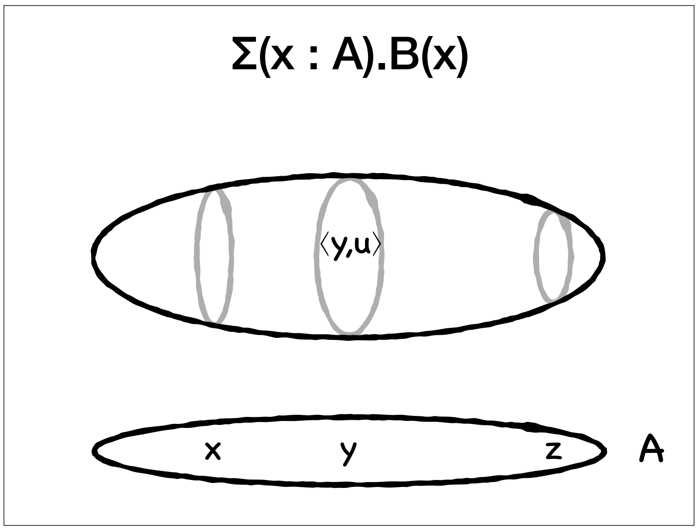
Let us look at the last two lines a little closer.

Dependent type B over A

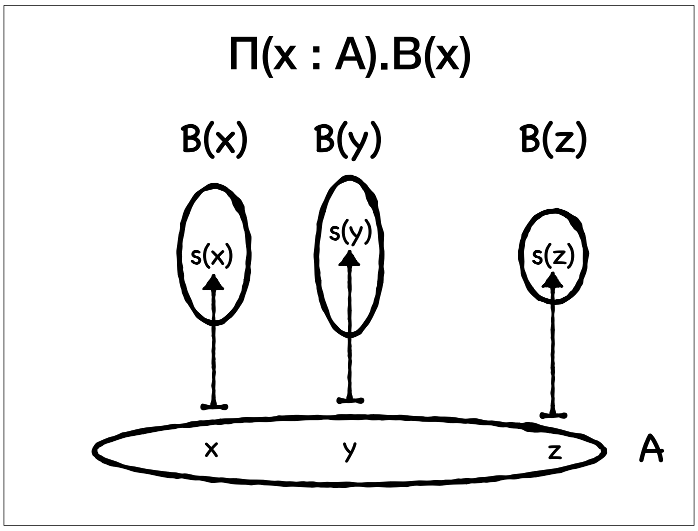


In mathematics sets, or spaces, are in general *parametrized*. If I say “consider an element of the permutation group S_n” the group is parametrized by n. If I say “the closed interval [a,b]”, again, the interval is parametrized by its endpoints a and b.

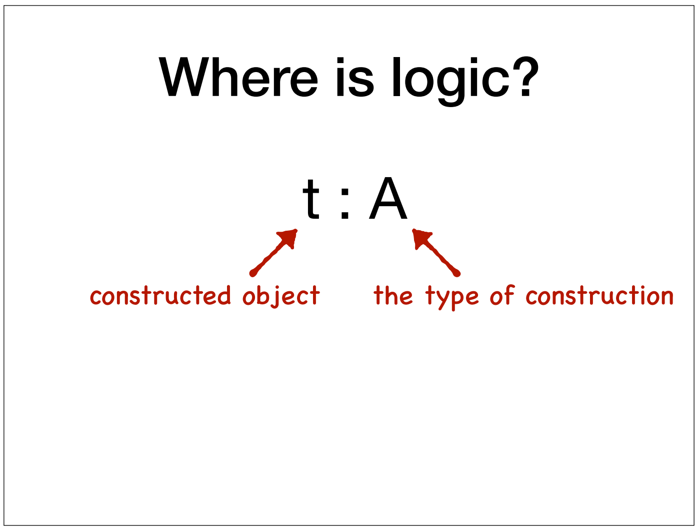
We can picture a type B parametrized by the elements of type A as a *family* of types: for each element x of A we get a type B(x).



The dependent sum collects a type family into a single type whose elements are pairs ⟨y, u⟩, with y from A and u from B(y). A topologist will recognize this as the *total space* of a bundle, while a set-theorist will just call it coproduct or disjoint sum of the family.



The dependent product of a family is the type of all sections, or choice functions. In the picture we see one such element s, which maps each element x in A to an element of B(x).

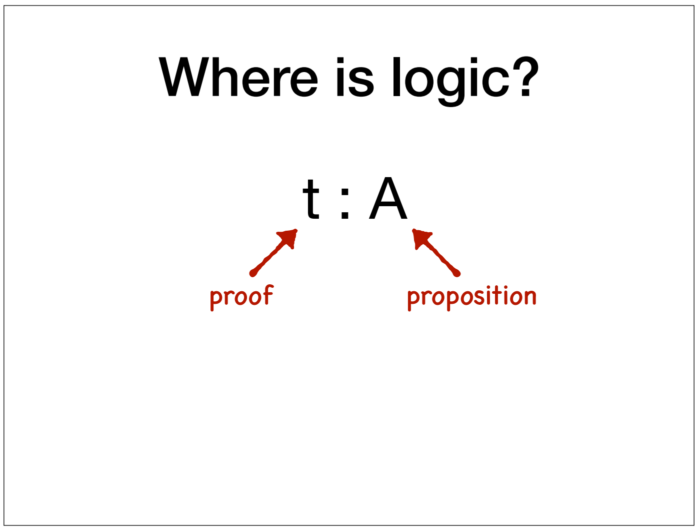


Where is logic?

$$\underset{\text{constructed object}}{t} : \underset{\text{the type of construction}}{A}$$

Do do mathematics, we also need some logic. We cannot just keep building mathematical objects without ever making any observations about them.

The matter is actually quite simple. A proof is just another kind of construction: start with basic axioms and using the rules of inference, construct a proof which shows that a given proposition holds.

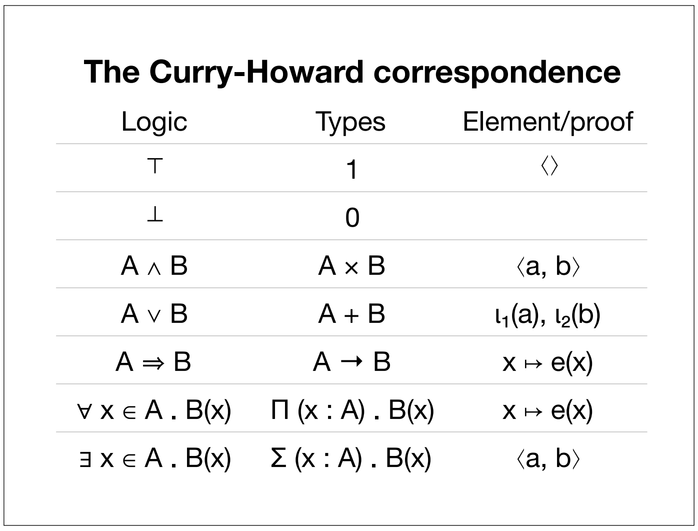


Where is logic?

$t : A$

proof
proposition

So in fact, logic is subsumed by type theory, if we think of proofs as elements of the statements that they prove. This observation was made a while ago already and goes under the name *Curry-Howard correspondence* or *propositions-as-types*.



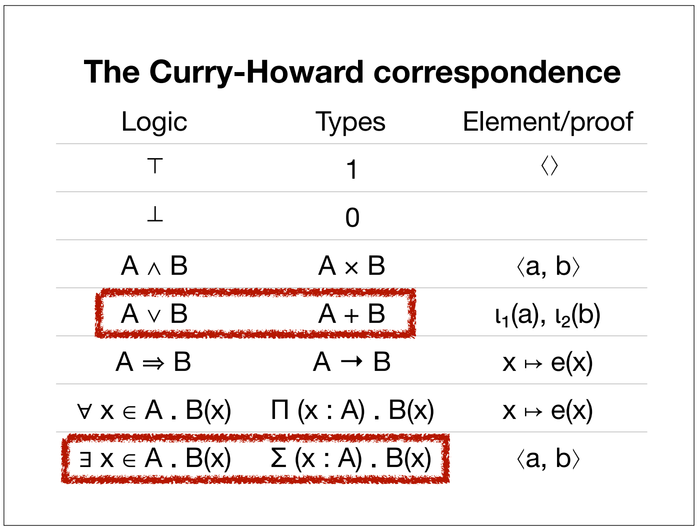
The Curry-Howard correspondence

Logic	Types	Element/proof
$\top$	1	$\langle\rangle$
$\bot$	0	
$A \wedge B$	$A \times B$	$\langle a, b\rangle$
$A \vee B$	$A + B$	$\iota_1(a)$, $\iota_2(b)$
$A \Rightarrow B$	$A \to B$	$x \mapsto e(x)$
$\forall\, x \in A\,.\, B(x)$	$\Pi\,(x : A)\,.\, B(x)$	$x \mapsto e(x)$
$\exists\, x \in A\,.\, B(x)$	$\Sigma\,(x : A)\,.\, B(x)$	$\langle a, b\rangle$

Even better, the basic logical connectives correspond to the basic type-theoretic constructions.

For example, the cartesian product corresponds to conjunction because in both proofs of $A \wedge B$ and elements of $A \times B$ are pairs (of proofs, or elements, as the case may be). A proof of an implication $A \Rightarrow B$ can be viewed as a map which takes proofs of A to proofs of B, and so on. This correspondence is also known as *propositions-as-types*.

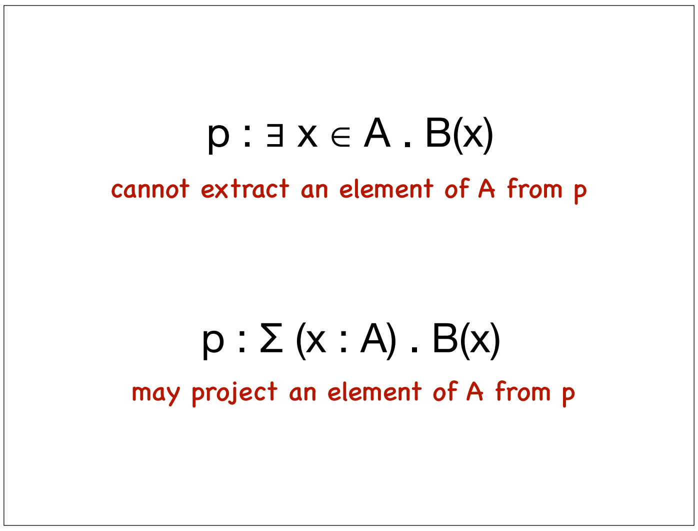
The displayed table is how the correspondence was understood in the last millennium. It is not quite right, because the disjunction and the existential quantifiers do not correspond precisely to the binary and the general disjoint sum.



The Curry-Howard correspondence

Logic	Types	Element/proof
$\top$	1	$\langle\rangle$
$\bot$	0	
$A \wedge B$	$A \times B$	$\langle a, b\rangle$
$\boxed{A \vee B}$	$\boxed{A + B}$	$\iota_1(a), \iota_2(b)$
$A \Rightarrow B$	$A \to B$	$x \mapsto e(x)$
$\forall x \in A . B(x)$	$\Pi (x : A) . B(x)$	$x \mapsto e(x)$
$\boxed{\exists x \in A . B(x)}$	$\boxed{\Sigma (x : A) . B(x)}$	$\langle a, b\rangle$

Let us look at the existential quantifier and the dependent sum.



$p : \exists\, x \in A \,.\, B(x)$

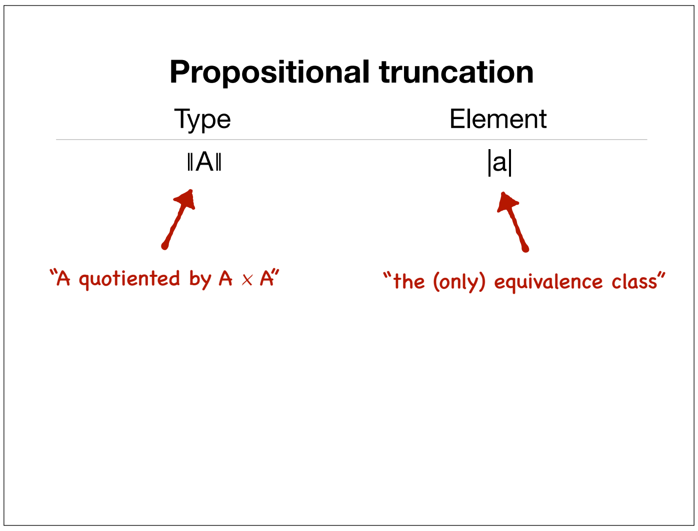
cannot extract an element of A from p

$p : \Sigma\, (x : A) \,.\, B(x)$

may project an element of A from p

When we prove the existence of x : A such that B(x), the x is *hidden* or *abstracted* by the proof, it is not visible outside the proof. But an element of the dependent sum is a pair ⟨a, b⟩ from which we may project the first component to obtain an element of A.

This is the crux of the matter. It is the difference between showing that something exists abstractly, without explicitly giving it, versus constructing an object. We should have the ability to speak about both of these.



Propositional truncation

Type	Element
‖A‖	|a|

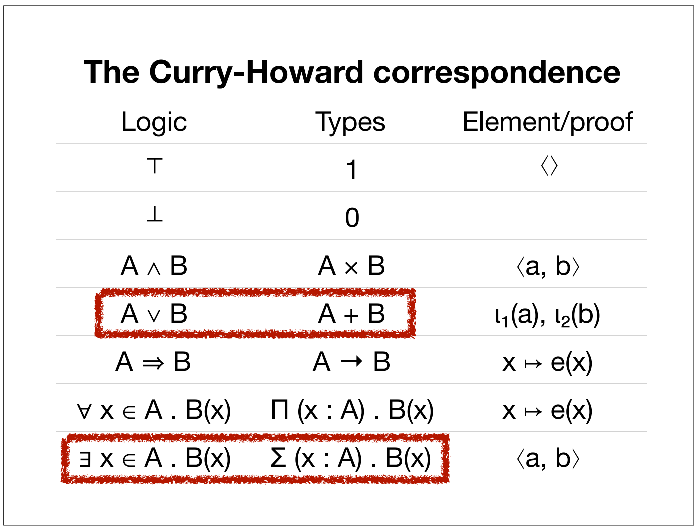
“A quotiented by A × A”
“the (only) equivalence class”

The solution is to add another type theoretic construction, called *propositional truncation*.

Speaking set-theoretically, it is the operation which takes a set A and quotients it by the largest equivalence relation on A. This may seem silly? If A is empty they we will get the empty set, and if a is non-empty, we will get a singleton set. Precisely, but so far such a construction has not been available in type theory.

It is the idea of “abstracting” or “hiding” an element in its pure form, as it takes an element a in A and yields (the only) equivalence class |a|, from which a cannot be uniquely recovered.

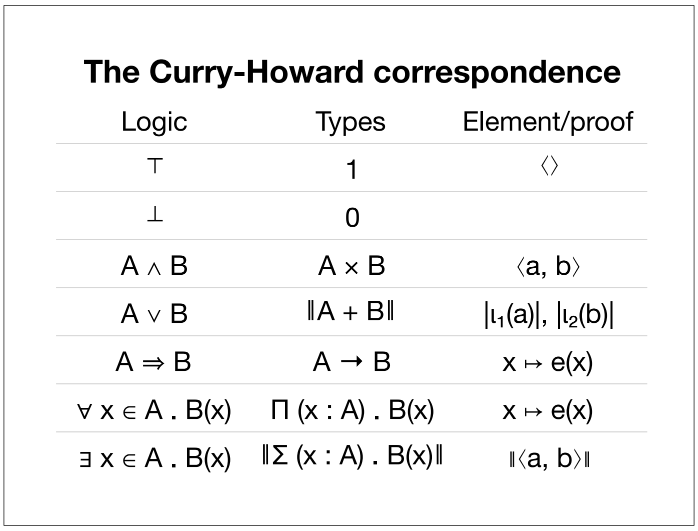
(If you’re still paying attention, you might be asking how to say that all the element of ‖A‖ are “equal”, we will come to that in a moment.)



The Curry-Howard correspondence

Logic	Types	Element/proof
$\top$	1	$\langle\rangle$
$\bot$	0	
$A \wedge B$	$A \times B$	$\langle a, b\rangle$
$A \vee B$	$A + B$	$\iota_1(a), \iota_2(b)$
$A \Rightarrow B$	$A \to B$	$x \mapsto e(x)$
$\forall\, x \in A\, .\, B(x)$	$\Pi\, (x : A)\, .\, B(x)$	$x \mapsto e(x)$
$\exists\, x \in A\, .\, B(x)$	$\Sigma\, (x : A)\, .\, B(x)$	$\langle a, b\rangle$

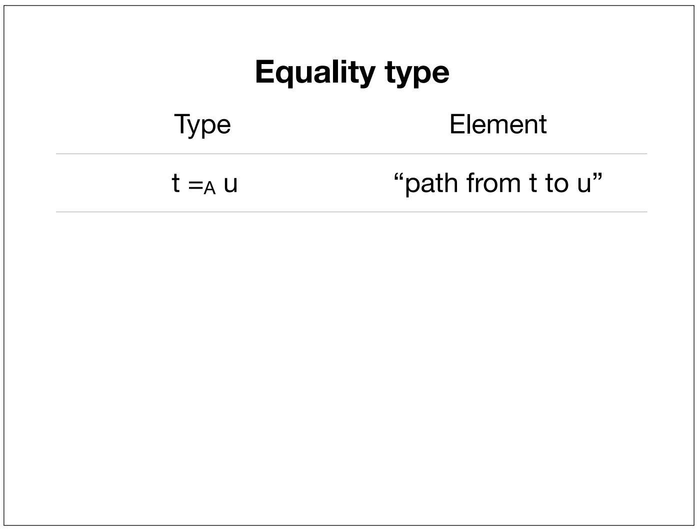
We can now fix the Curry-Howard correspondence by truncating the sums.



The Curry-Howard correspondence

Logic	Types	Element/proof
$\top$	1	$\langle\rangle$
$\bot$	0	
$A \wedge B$	$A \times B$	$\langle a, b\rangle$
$A \vee B$	$\Vert A + B \Vert$	$\vert\iota_1(a)\vert$, $\vert\iota_2(b)\vert$
$A \Rightarrow B$	$A \to B$	$x \mapsto e(x)$
$\forall\, x \in A\,.\, B(x)$	$\Pi\,(x : A)\,.\, B(x)$	$x \mapsto e(x)$
$\exists\, x \in A\,.\, B(x)$	$\Vert\Sigma\,(x : A)\,.\, B(x)\Vert$	$\Vert\langle a, b\rangle\Vert$

With this we have truly subsumed logic inside type theory.

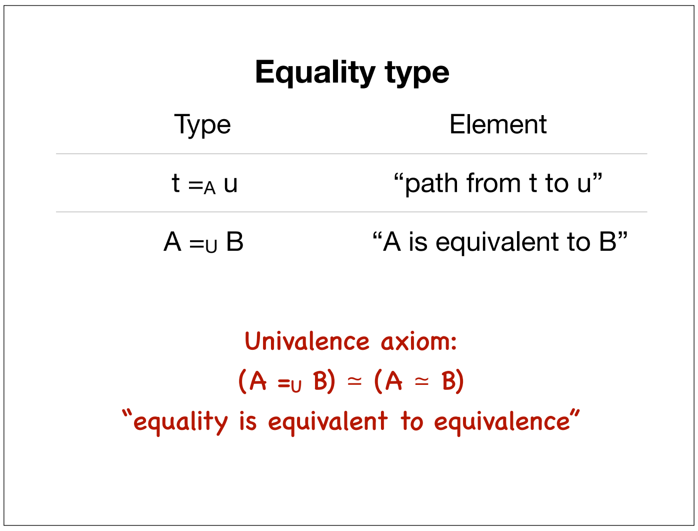


Equality type

Type	Element
$t =_A u$	“path from t to u”

Unfortunately, we are not quite done yet. To complete the picture we also have to discuss equality, which is the most interesting part of homotopy type theory, and it’s what gave ordinary type theory the qualifier “homotopy”. In order to avoid a cognitive overload, here is a one-slide summary.

If everything is supposed to be a construction, what do we do about equality? Is it not a relation? How does one “construct” an “equality” a = b? Topologists have an answer: think of two points in a space as being “equal” if there is a path between them. Technically speaking, it is better to think of homotopy classes of paths.



Equality type

Type	Element
$t =_A u$	"path from t to u"
$A =_U B$	"A is equivalent to B"

Univalence axiom:

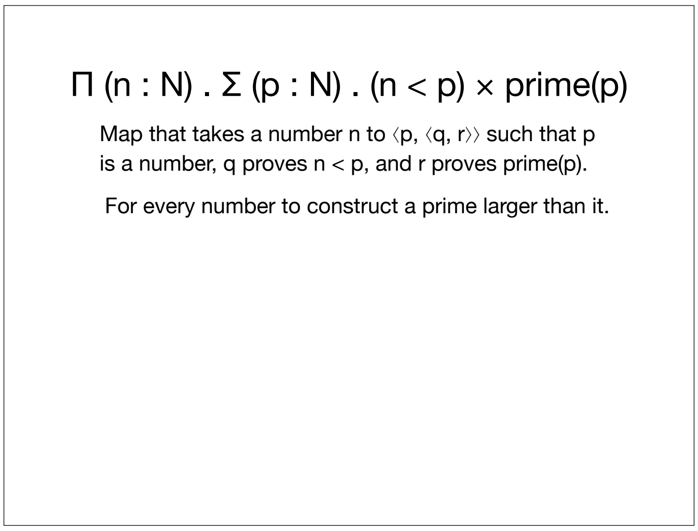
$(A =_U B) \simeq (A \simeq B)$

"equality is equivalent to equivalence"

To get equalty-as-paths idea really going, we need to know more about equality of types, i.e., what do paths in the universe look like? Here an axiom by Vladimir Voevodsky, the univalence axiom, saves the day. It states that "equality of types is equivalent to equivalence of types".

For the purposes of this lecture you can read "equivalence" as saying "isomorphism", with the definition of isomorphism being "a map whose inverse images of points are singletons".

The univalence axiom has many important consequences. It would take a whole lecture to explain its importance, so we shall instead move on and just mention some of its consequences.



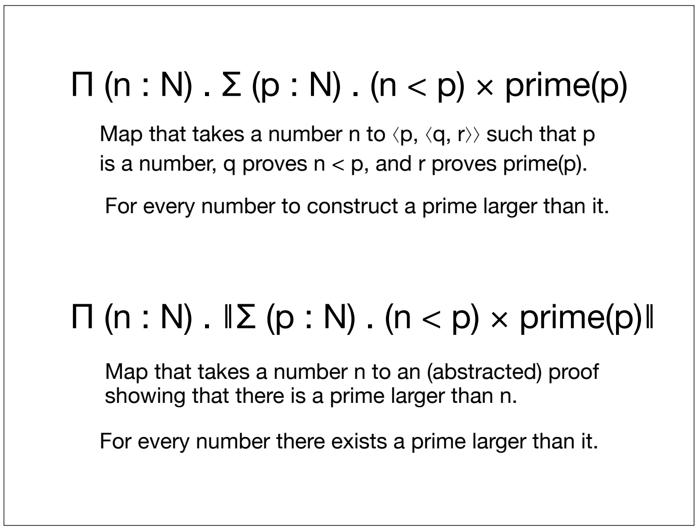
$$\Pi\ (n : \mathbb{N})\ .\ \Sigma\ (p : \mathbb{N})\ .\ (n < p) \times \mathrm{prime}(p)$$

Map that takes a number n to ⟨p, ⟨q, r⟩⟩ such that p is a number, q proves $n < p$, and r proves prime(p).

For every number to construct a prime larger than it.

Finally, let us look at some examples.

Let us start with something familiar, the infinitude of primes. Without truncation, here is how one would write down the infinitude of primes, as a construction. (We assume that we already figured out how to define < and primality.)



$\Pi\ (n : N)\ .\ \Sigma\ (p : N)\ .\ (n < p) \times \mathrm{prime}(p)$

Map that takes a number n to ⟨p, ⟨q, r⟩⟩ such that p is a number, q proves n < p, and r proves prime(p).

For every number to construct a prime larger than it.

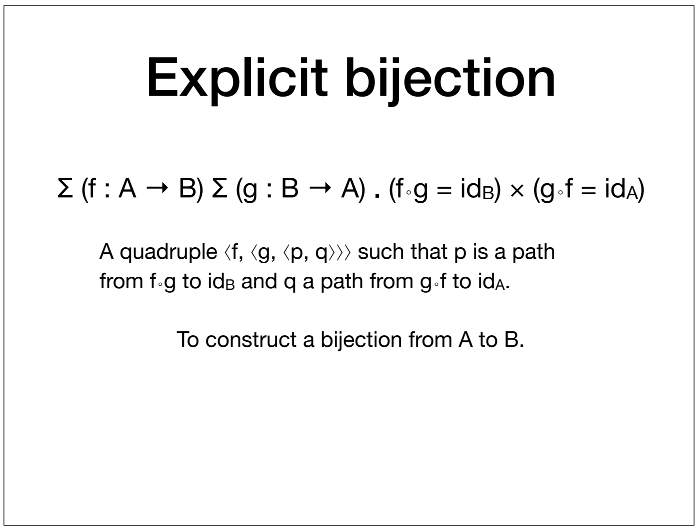
$\Pi\ (n : N)\ .\ \|\Sigma\ (p : N)\ .\ (n < p) \times \mathrm{prime}(p)\|$

Map that takes a number n to an (abstracted) proof showing that there is a prime larger than n.

For every number there exists a prime larger than it.

With truncation, we obtain the *statement* that there are infinitely many primes.

How about *explicit bijection?*



Explicit bijection

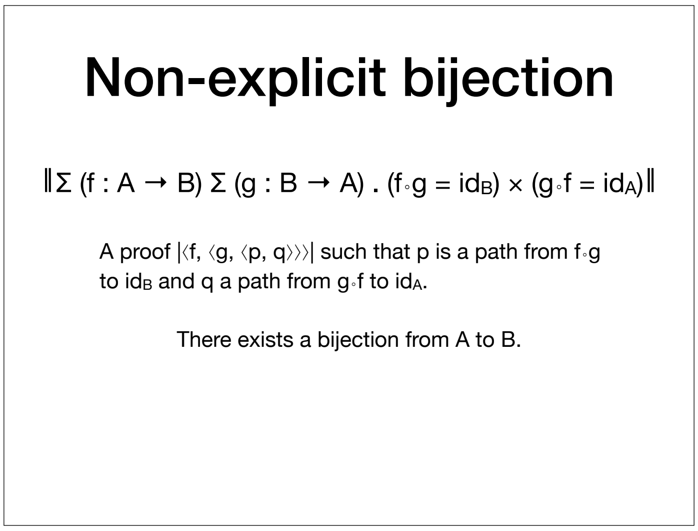
$\Sigma\,(f : A \to B)\ \Sigma\,(g : B \to A)\ .\ (f \circ g = id_B) \times (g \circ f = id_A)$

A quadruple $\langle f, \langle g, \langle p, q\rangle\rangle\rangle$ such that p is a path from $f \circ g$ to id_B and q a path from $g \circ f$ to id_A.

To construct a bijection from A to B.

With dependent sums we get the explicit version.

An element of the given type is a quadruple whose first two components are maps, and the other two show that the maps are inverses of each other (up to homotopy!)



Non-explicit bijection

$\|\Sigma\,(f : A \to B)\ \Sigma\,(g : B \to A)\ .\ (f \circ g = id_B) \times (g \circ f = id_A)\|$

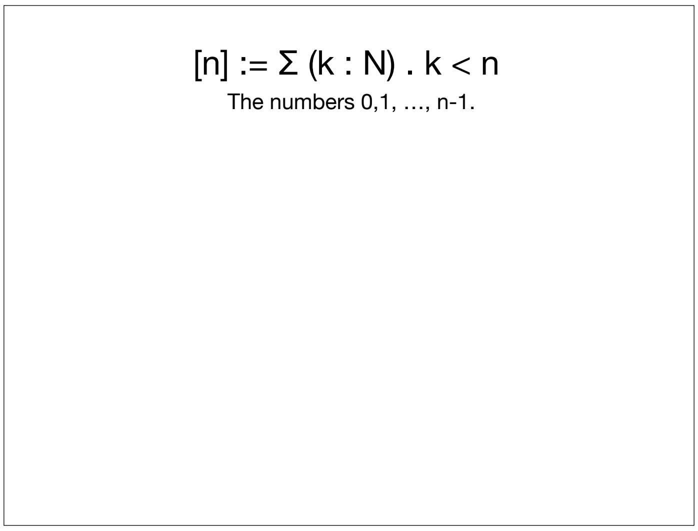
A proof $|\langle f, \langle g, \langle p, q\rangle\rangle\rangle|$ such that p is a path from $f \circ g$ to id_B and q a path from $g \circ f$ to id_A.

There exists a bijection from A to B.

To get the non-explicit version, just truncate.

This is probably not the final word on explicit bijections. What I find important in the whole matter is the *awareness* of the fact that some piece of mathematics does not have a precise formal definition, and the search for such a definition.

We may not have resolved the question in the title, but we have a new way of thinking and organizing mathematics. To conclude, let me show you how to do combinatorial spices in homotopy type theory. You can read more about it in Brent Yorgey's PhD thesis.

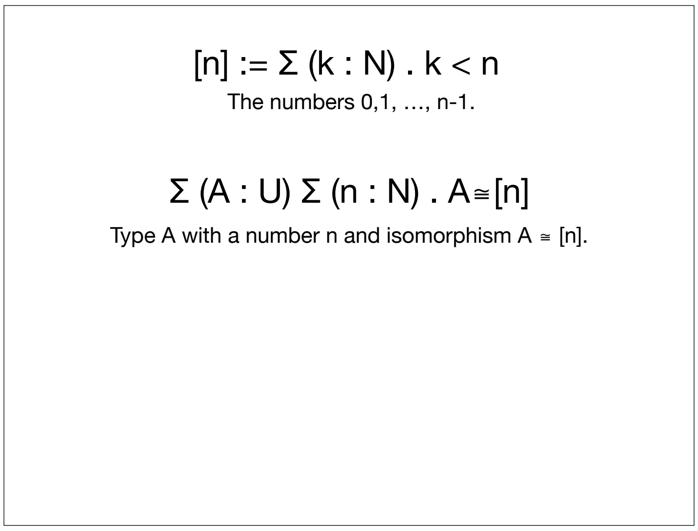


$[n] := \Sigma\, (k : N)\, .\, k < n$

The numbers 0,1, …, n-1.

First we can define the *standard finite types*, as a dependent type [n], which for each n gives the type of numbers below n. (To be quite formal, the elements are numbers with proofs that they are smaller than n.)

The type of all finite types, what should it be?



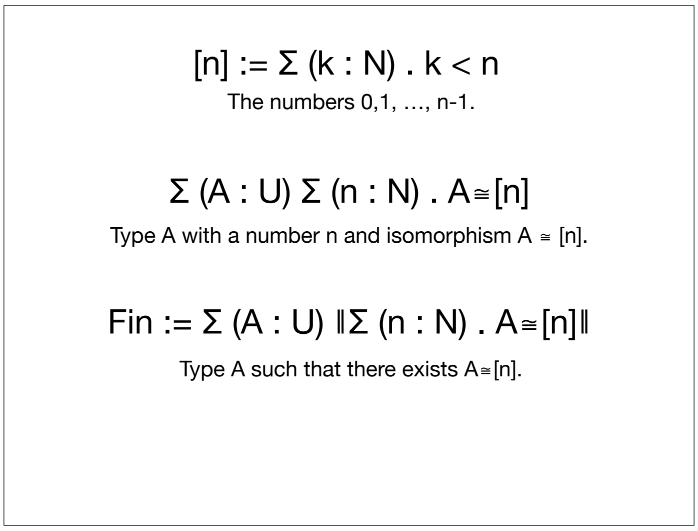
$$[n] := \Sigma\,(k : N)\,.\,k < n$$

The numbers 0,1, …, n-1.

$$\Sigma\,(A : U)\,\Sigma\,(n : N)\,.\,A \cong [n]$$

Type A with a number n and isomorphism $A \cong [n]$.

Here is a first attempt. A finite type is a type A together with an isomorphism to some standard finite type. However, we made the isomorphism explicit, so we actually have a *chosen* specific isomorphism, which amount to having a linear order on A. This is the type of linearly ordered finite types.



$$[n] := \Sigma\, (k : N)\, .\, k < n$$

The numbers 0,1, …, n-1.

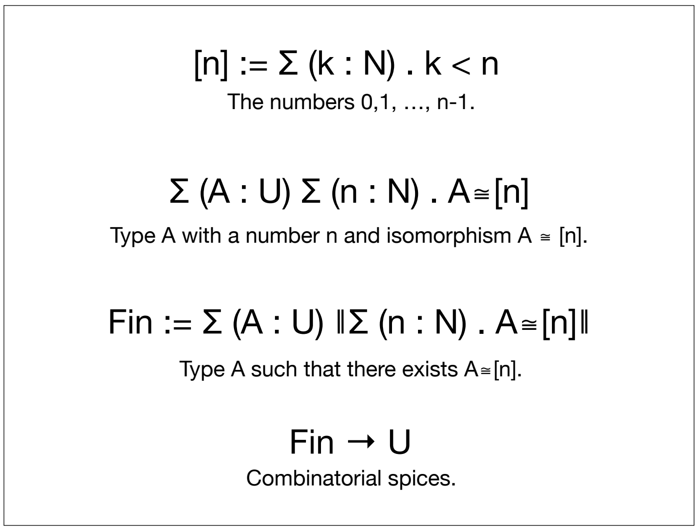
$$\Sigma\, (A : U)\, \Sigma\, (n : N)\, .\, A \cong [n]$$

Type A with a number n and isomorphism $A \cong [n]$.

$$\mathrm{Fin} := \Sigma\, (A : U)\, \|\Sigma\, (n : N)\, .\, A \cong [n]\|$$

Type A such that there exists $A \cong [n]$.

By truncating, we obtain the actual finite types.



$[n] := \Sigma\,(k : N)\;.\;k < n$

The numbers 0,1, …, n-1.

$\Sigma\,(A : U)\;\Sigma\,(n : N)\;.\;A \cong [n]$

Type A with a number n and isomorphism A ≅ [n].

$\mathrm{Fin} := \Sigma\,(A : U)\;\|\Sigma\,(n : N)\;.\;A \cong [n]\|$

Type A such that there exists A≅[n].

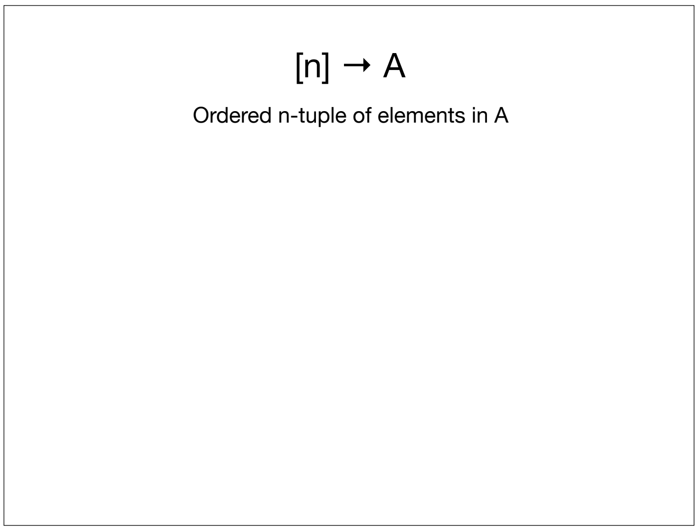
$\mathrm{Fin} \to U$

Combinatorial spices.

The combinatorial spices are maps from finite types to the universe.

Type theory and the univalence axiom take care of what is usually achieved by requiring that a spices be a *functor*. Things are getting *simpler*!

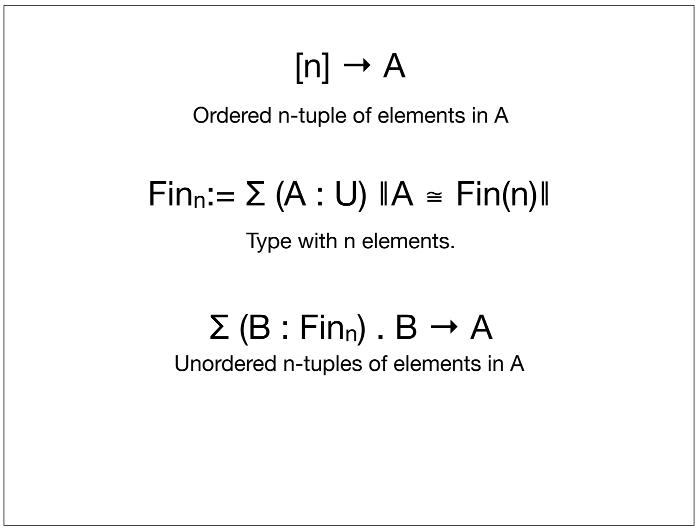
The ability to precisely distinguish between explicit constructions and abstract existence is paying off. And we can keep going.



$[n] \rightarrow A$

Ordered n-tuple of elements in A

For example, the n-tuples of elements in A are easy, they are just maps from [n] to A.



$[n] \to A$

Ordered n-tuple of elements in A

$\mathrm{Fin}_n := \Sigma\,(A : U)\ \|A \cong \mathrm{Fin}(n)\|$

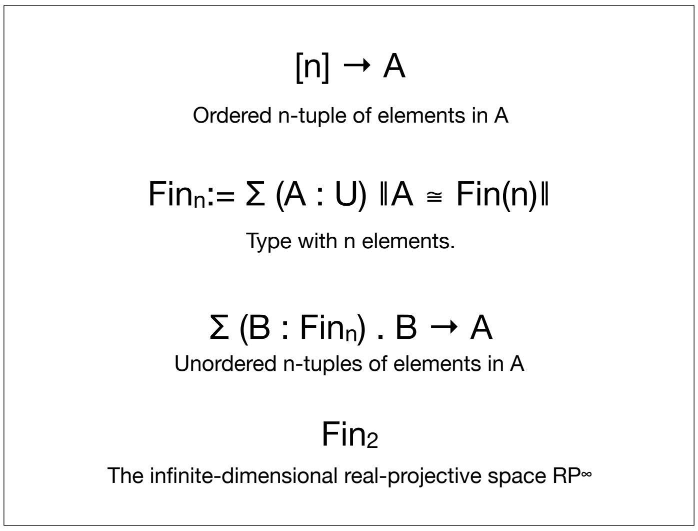
Type with n elements.

$\Sigma\,(B : \mathrm{Fin}_n)\ .\ B \to A$

Unordered n-tuples of elements in A

To get the unordered n-tuples of elements of A, we first define the type of all n-element types.

The unordered n-tuples of elements in A are given as a sum: an indexing type B of size n, and a map from B to A. (Note that we cannot use a *specific* B, as any particular ordering of that B would automatically order the tuple, as in the first line.)



$[n] \to A$

Ordered n-tuple of elements in A

$\mathrm{Fin}_n := \Sigma\,(A : U)\ \|A \cong \mathrm{Fin}(n)\|$

Type with n elements.

$\Sigma\,(B : \mathrm{Fin}_n)\ .\ B \to A$

Unordered n-tuples of elements in A

Fin_2

The infinite-dimensional real-projective space RP^∞

Here is an amazing fact, shown by Ulrich Buchholz and Egbert Rijke. The type of two-element sets is the infinite-dimensional real-projective space. If there are topologists in the audience, they might remark that this is just the classifying space of the symmetric group S_2. Indeed it is, and you can guess what Fin_n is going to be.

Thank you!