From Reversible Programming Languages to Univalent Universes and Back

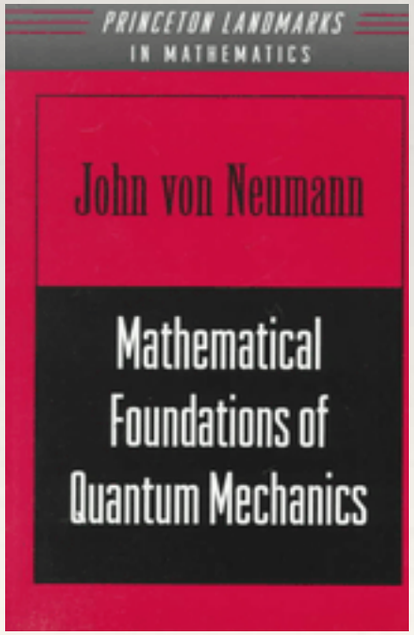
Jacques Carette (C), Chao-Hong Chen (C**), Vikraman Choudhury (C**), Robert Rose (M**), and Amr Sabry (C**)*
(*) McMaster University
(**) Indiana University)
(C) Computer Science
(M) Mathematics

MFPS XXXIII
Ljubljana
June 2017

Physics and Computation

Physics

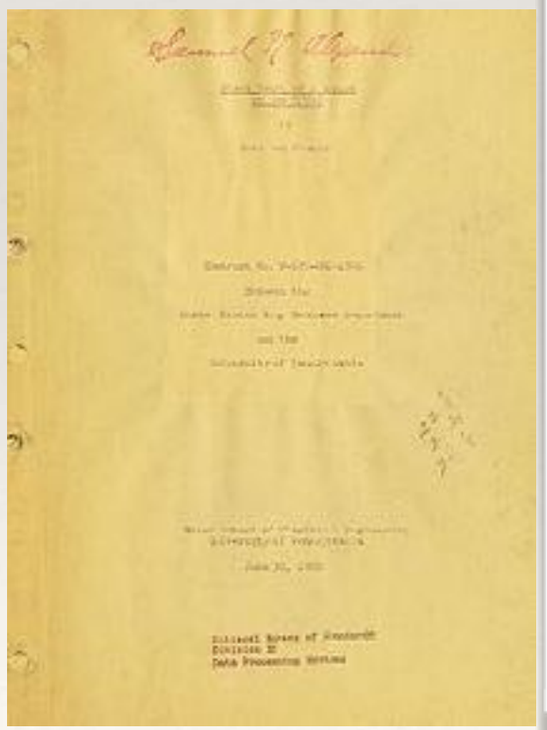
- Most accurate theory known to us is *Quantum Mechanics*



Computation

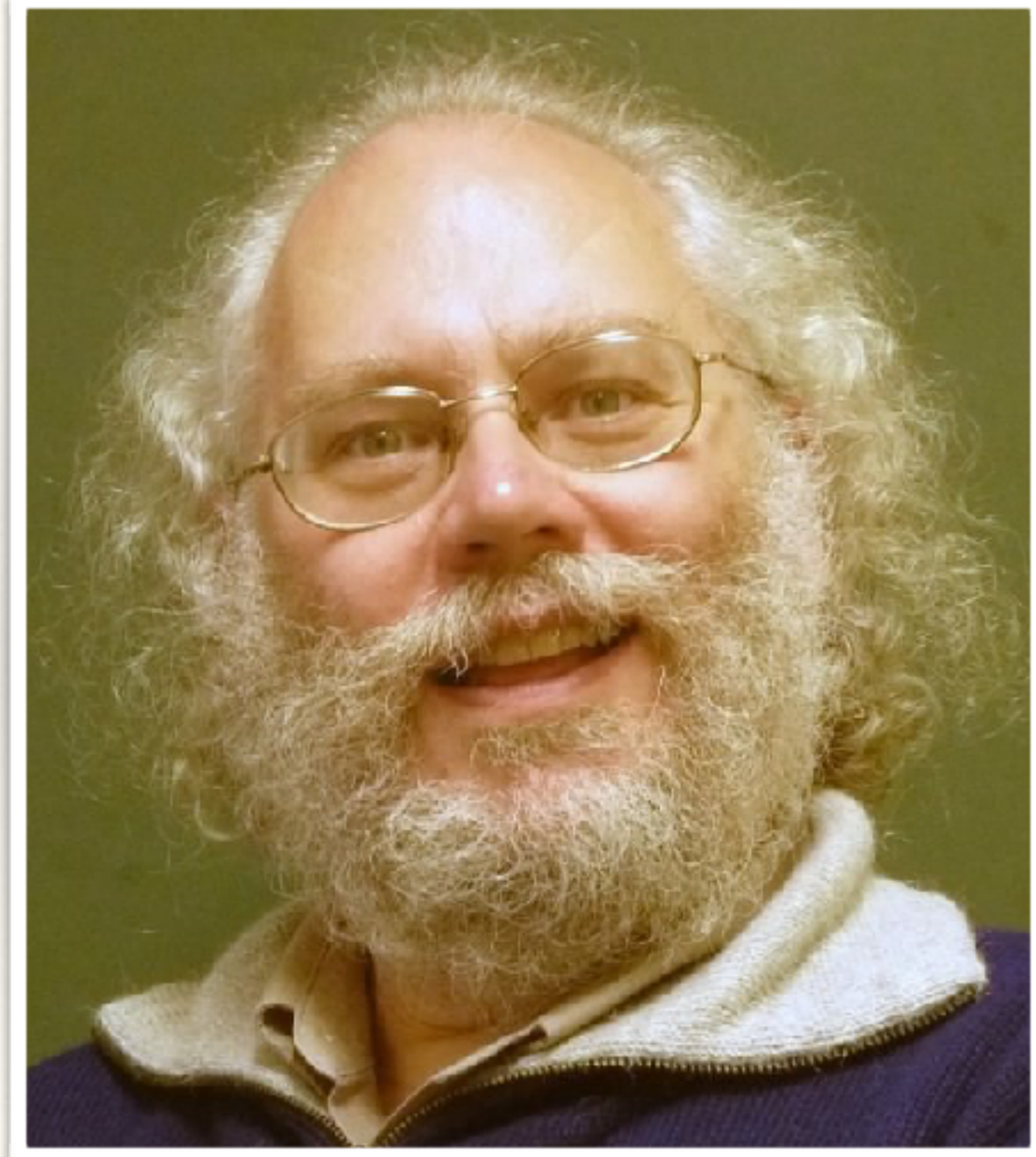
- Universal model of computation is based on the Von Neumann architecture
- Equivalent to Turing machines, the λ-calculus, and many other models

First Draft of a Report on the EDVAC
by John von Neumann,
Contract No. W-670-ORD-4926,
Between the United States Army Ordinance Department
and the University of Pennsylvania Moore School of Electrical Engineering
University of Pennsylvania
June 30, 1945



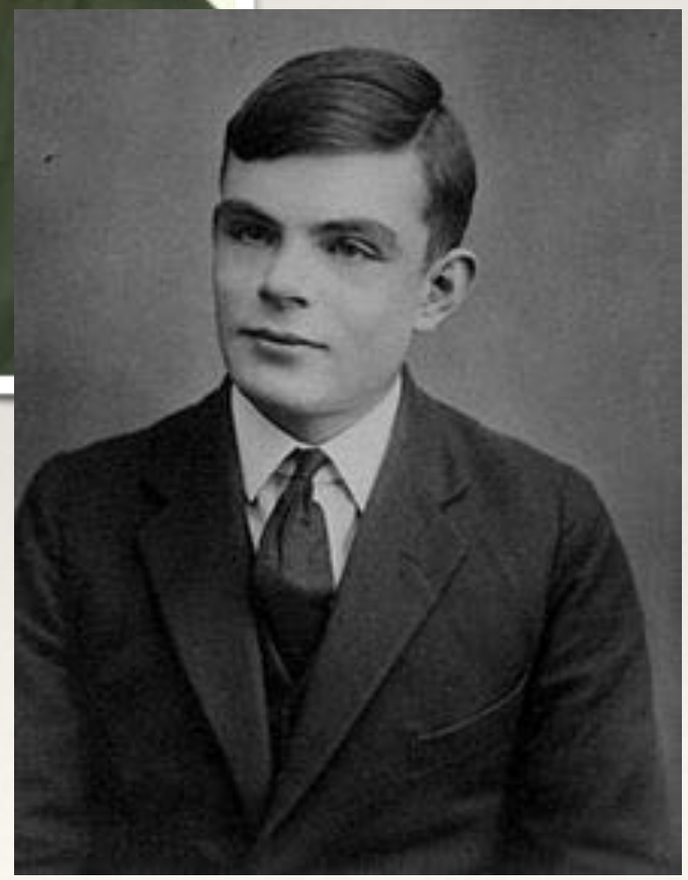
Physics Worldview

- Quantum mechanics implies the existence of an efficient (polynomial) algorithm for factoring integers (Shor)
- RSA is not secure.



Computer Science Worldview

- Extended Church-Turing Thesis: Everything that is computable in Nature is computable by a von Neumann machine, and is not exponentially faster.
- Despite intense interest and effort, no one knows how to factor integers efficiently in that model.
- RSA is secure.



Something is Wrong

- Either Shor's algorithm is not "natural". Textbook quantum mechanics is wrong;
- or, the von Neumann architecture is not universal. There are other "natural" computing models that are exponentially faster;
- or, computer scientists have not been clever enough to find an efficient factoring algorithm;
- or, everybody is wrong

Possibility I: Revise quantum mechanics

The mathematician's vision of an unlimited sequence of totally reliable operations is unlikely to be implementable in this real universe.

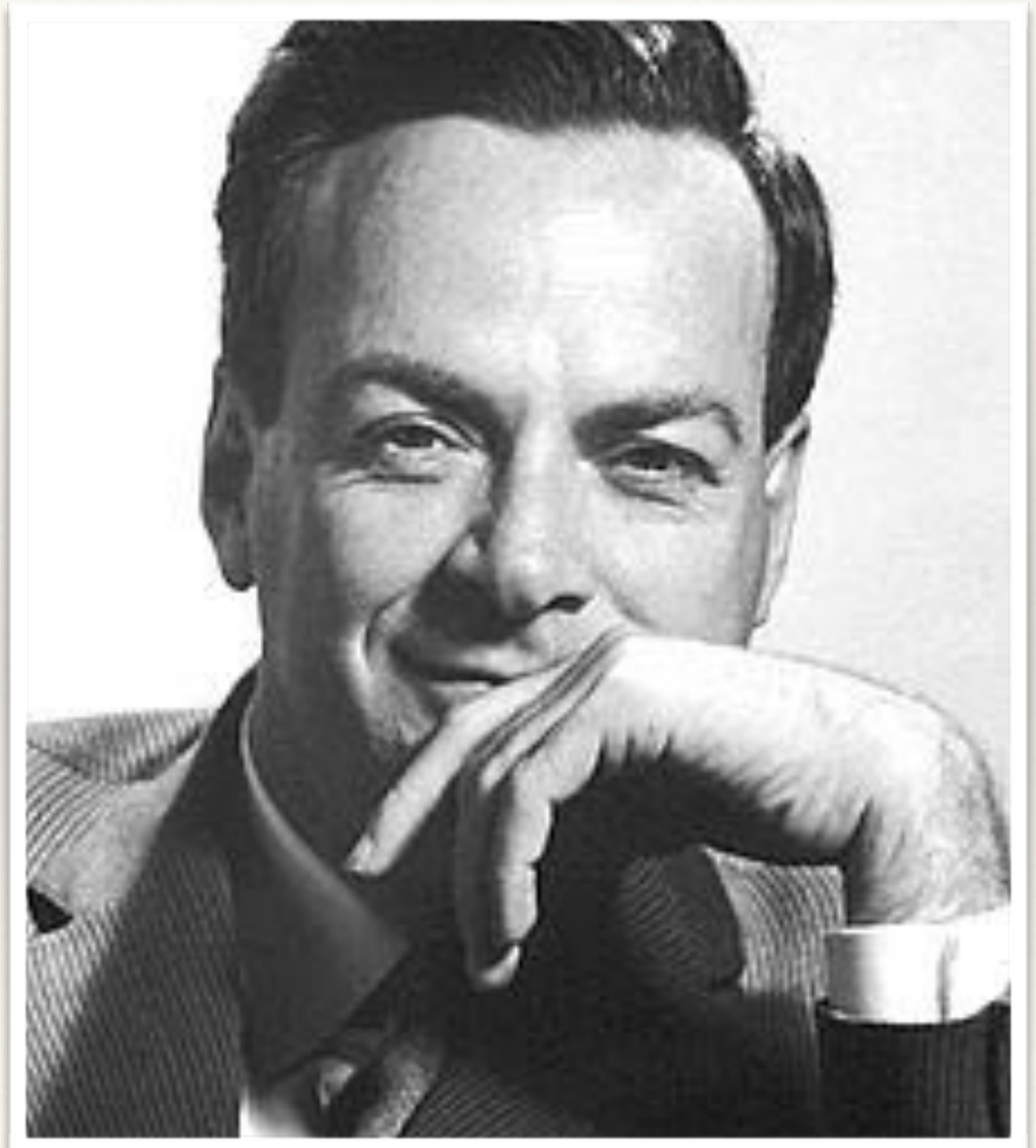
But the real world is unlikely to supply us with unlimited memory or unlimited Turing machine tapes. Therefore, continuum mathematics is not executable, and physical laws which invoke that can not really be satisfactory. They are references to illusionary procedures.

(Landauer 1996 and 1999)

Possibility I: Revise quantum mechanics

I want to talk about the possibility that there is to be an exact simulation, that the computer will do exactly the same as nature. If this is to be proved and the type of computer is as I've already explained, then it's going to be necessary that everything that happens in a finite volume of space and time would have to be exactly analyzable with a finite number of logical operations. The present theory of physics is not that way, apparently. It allows space to go down into infinitesimal distances, wavelengths to get infinitely great, terms to be summed in infinite order, and so forth; and therefore, if this proposition is right, physical law is wrong.

(Feynman 1981)



Possibility II: Revise Computer Science

Another thing that had been suggested early was that natural laws are reversible, but that computer rules are not. But this turned out to be false; the computer rules can be reversible, and it has been a very, very useful thing to notice and to discover that. This is a place where the relationship of physics and computation has turned itself the other way and told us something about the possibilities of computation. So this is an interesting subject because it tells us something about computer rules…

(Feynman 1981)

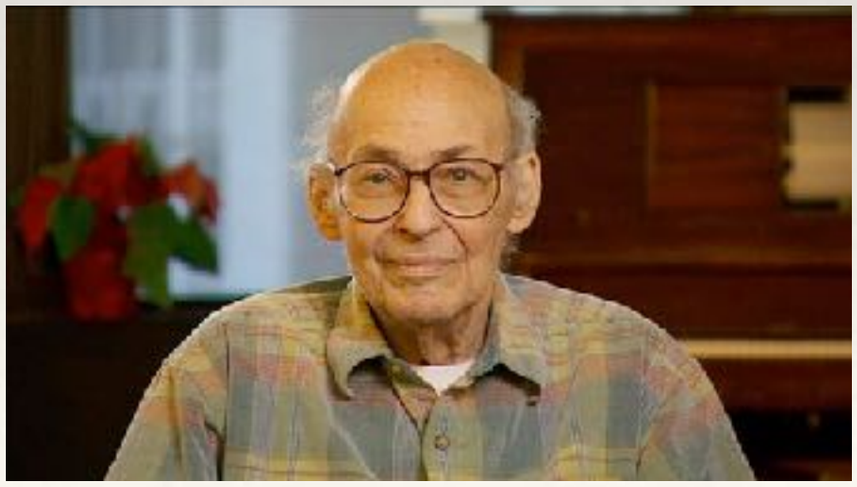
Possibility II: Revise Computer Science

Turing hoped that his abstracted-paper-tape model was so simple, so transparent and well defined, that it would not depend on any assumptions about physics that could conceivably be falsified, and therefore that it could become the basis of an abstract theory of computation that was independent of the underlying physics. 'He thought,' as Feynman once put it, 'that he understood paper.' But he was mistaken. Real, quantum-mechanical paper is wildly different from the abstract stuff that the Turing machine uses. The Turing machine is entirely classical, and does not allow for the possibility the paper might have different symbols written on it in different universes, and that those might interfere with one another.

(Deutsch 1985)

Possibility II: Revise Computer Science

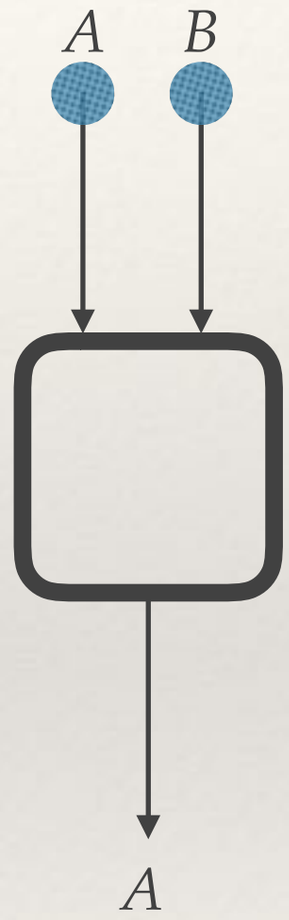
Ed Fredkin pursued the idea that information must be finite in density. One day, he announced that things must be even more simple than that. He said that he was going to assume that information itself is conserved. "You're out of you mind, Ed." I pronounced. "That's completely ridiculous. Nothing could happen in such a world. There couldn't even be logical gates. No decisions could ever be made." But when Fredkin gets one of his ideas, he's quite immune to objections like that; indeed, they fuel him with energy. Soon he went on to assume that information processing must also be reversible — and invented what's now called the Fredkin gate.



(Minsky 1999)

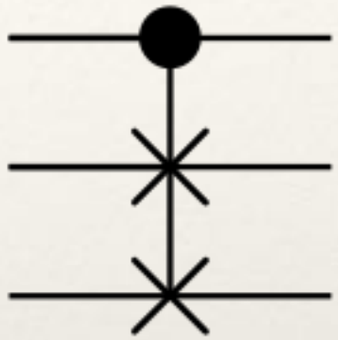
Conservation of Information

- None of the common foundational models of computation is based on conservation of information
  - Circuit model has AND, OR, etc gates that lose information;
  - Turing Machine allows one to overwrite a cell losing information;
  - λ-calculus allows functions that throw away their arguments losing information;
  - etc etc etc

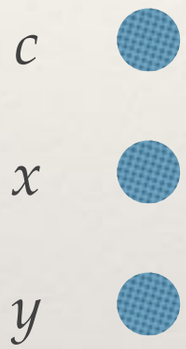


Laundauer's principle:
Erasure of information generates heat!!!

Fredkin Gate



c = *false*



C

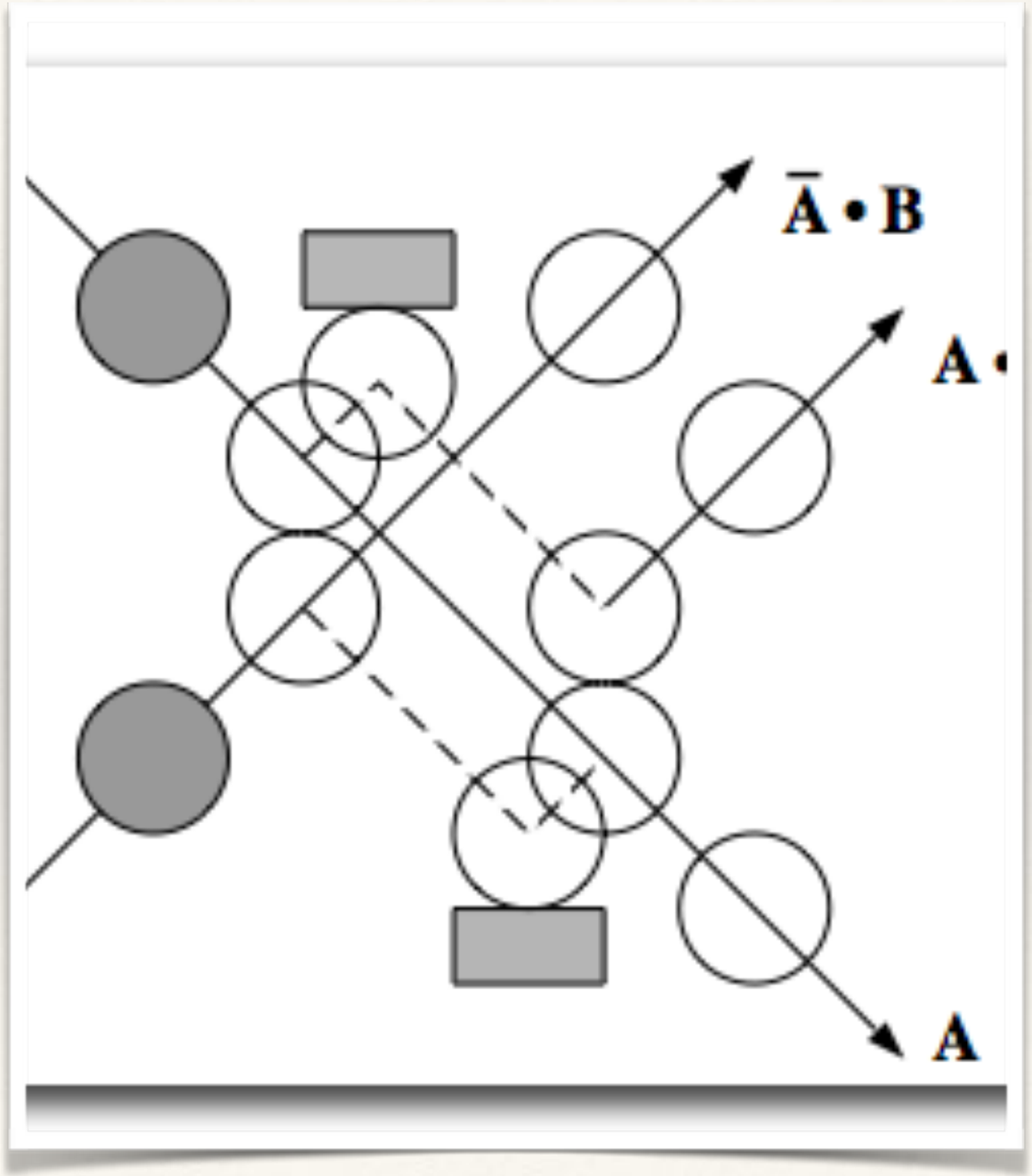
X

Y

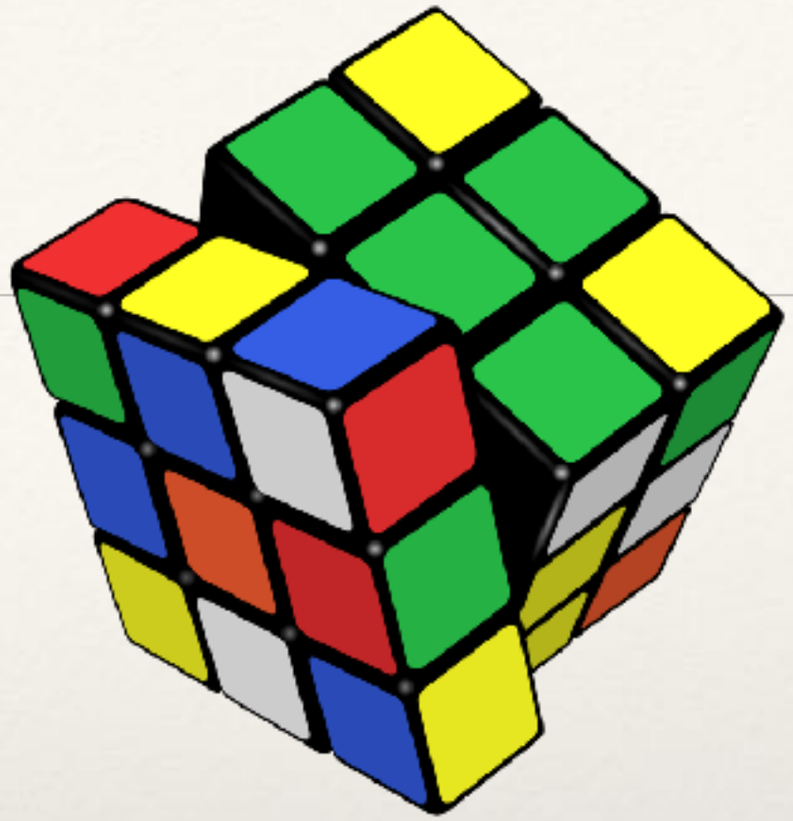
If y = *false*, then $Y = c$ AND x
If x = *false*, y = *true*, then y = NOT c
If y = *true*, then $X = c$ OR x

Conservation of Information

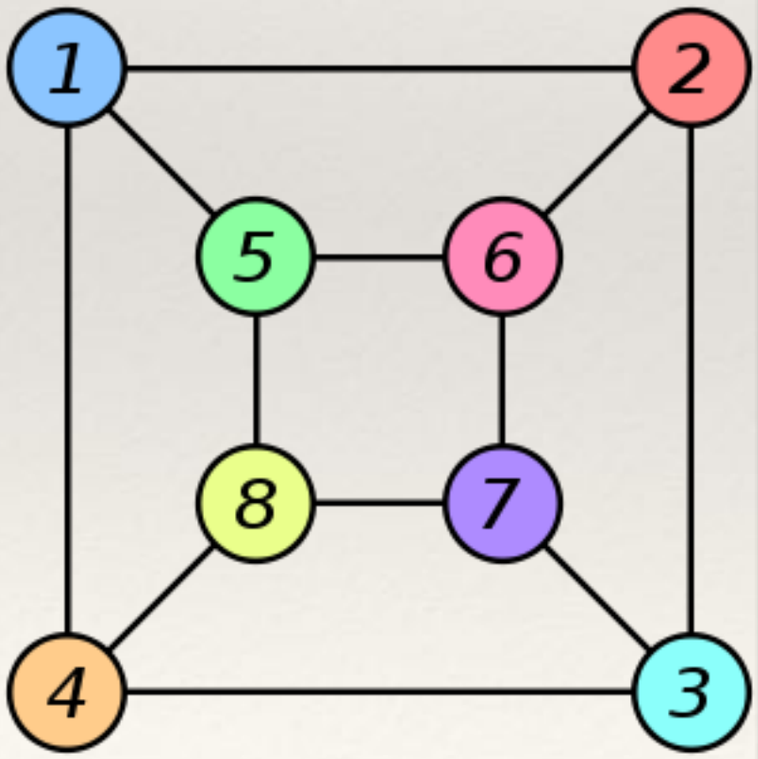
- Is it just a low-level concern, a curiosity, an esoteric model, or is it really foundational?
- We argue that, if taken seriously, it revolutionizes the theory and practice of computation, and even its logical foundations.
- This perspective has far reaching implications in many high-level applications

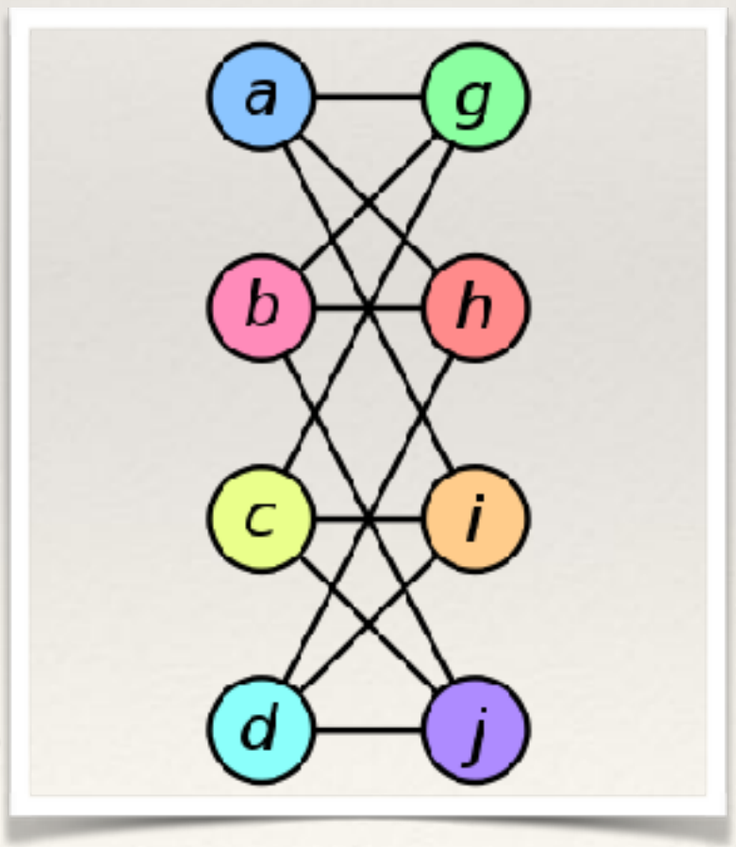


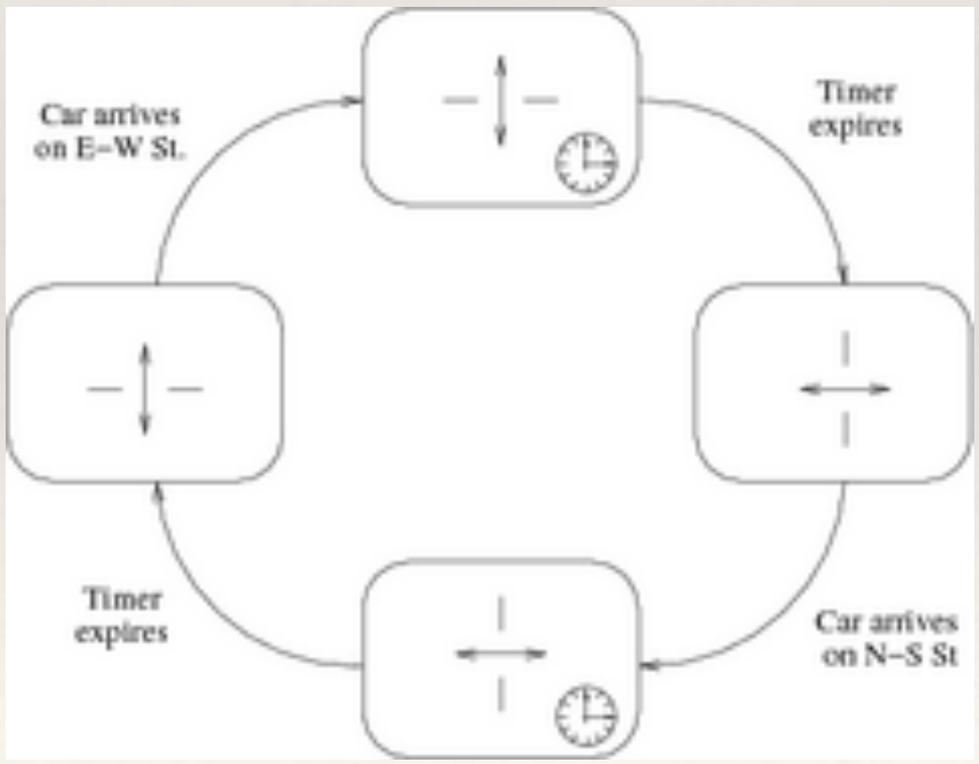
Key Idea



- All programs / proofs / deductions are isomorphisms / equivalences







From Irreversible to Reversible

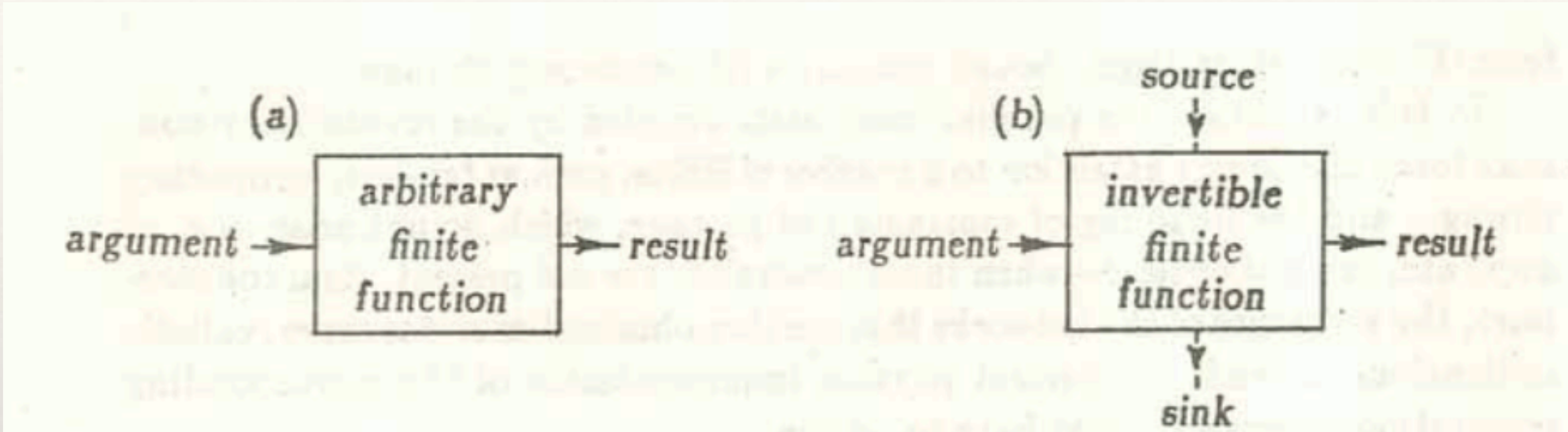
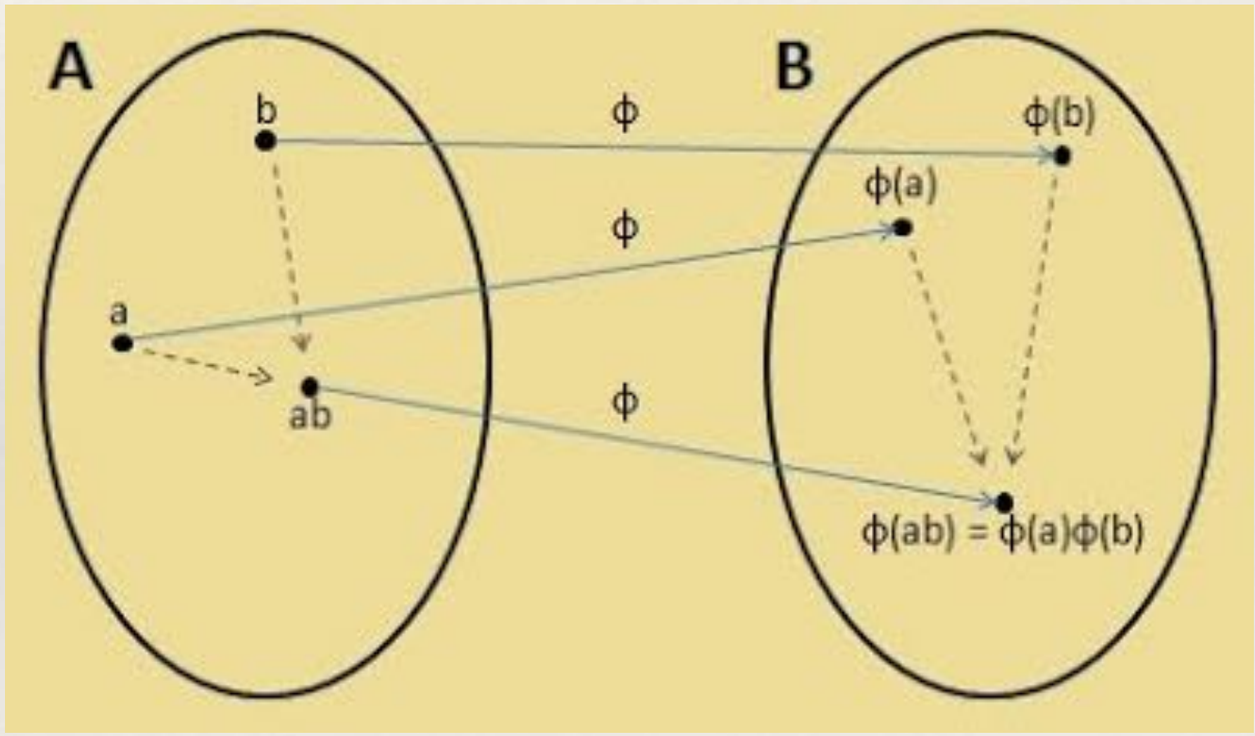


FIG. 4.2 *Any finite function (a) can be realized as an invertible finite function (b) having a number of auxiliary input lines which are fed with constants and a number of auxiliary output lines whose values are disregarded.*

(Toffoli 1980)

Type Isomorphisms

- Some problems are naturally reversible.
- Those that are not can be embedded in reversible problems.
- When it comes to writing programs to solve these problems, all is needed is a language for expressing equivalences.
- Technically a language that is sound and complete with respect to isomorphisms between types.



Sound and Complete Type Isomorphisms (for finite types)

$$
\begin{array}{rcll}
0 + b & \rightleftharpoons & b & \textit{identity for } + \\
b_1 + b_2 & \rightleftharpoons & b_2 + b_1 & \textit{commutativity for } + \\
b_1 + (b_2 + b_3) & \rightleftharpoons & (b_1 + b_2) + b_3 & \textit{associativity for } +
\end{array}
$$

$$
\begin{array}{rcll}
1 \times b & \rightleftharpoons & b & \textit{identity for } \times \\
b_1 \times b_2 & \rightleftharpoons & b_2 \times b_1 & \textit{commutativity for } \times \\
b_1 \times (b_2 \times b_3) & \rightleftharpoons & (b_1 \times b_2) \times b_3 & \textit{associativity for } \times
\end{array}
$$

$$
\begin{array}{rcll}
0 \times b & \rightleftharpoons & 0 & \textit{distribute over } 0 \\
(b_1 + b_2) \times b_3 & \rightleftharpoons & (b_1 \times b_3) + (b_2 \times b_3) & \textit{distribute over } +
\end{array}
$$

$$
\frac{}{b_1 \rightleftharpoons b_1} \quad \frac{b_1 \rightleftharpoons b_2}{b_2 \rightleftharpoons b_1} \quad \frac{b_1 \rightleftharpoons b_2 \quad b_2 \rightleftharpoons b_3}{b_1 \rightleftharpoons b_3}
$$

$$
\frac{b_1 \rightleftharpoons b_3 \quad b_2 \rightleftharpoons b_4}{(b_1 + b_2) \rightleftharpoons (b_3 + b_4)} \quad \frac{b_1 \rightleftharpoons b_3 \quad b_2 \rightleftharpoons b_4}{(b_1 \times b_2) \rightleftharpoons (b_3 \times b_4)}
$$

Name the Isomorphisms

$$
\begin{array}{rrcll}
\mathit{zeroe}: & 0+b & \rightleftharpoons & b & : \mathit{zeroi} \\
\mathit{swap}^{+}: & b_1+b_2 & \rightleftharpoons & b_2+b_1 & : \mathit{swap}^{+} \\
\mathit{assocl}^{+}: & b_1+(b_2+b_3) & \rightleftharpoons & (b_1+b_2)+b_3 & : \mathit{assocr}^{+} \\
 & & & & \\
\mathit{unite}: & 1\times b & \rightleftharpoons & b & : \mathit{uniti} \\
\mathit{swap}^{\times}: & b_1\times b_2 & \rightleftharpoons & b_2\times b_1 & : \mathit{swap}^{\times} \\
\mathit{assocl}^{\times}: & b_1\times(b_2\times b_3) & \rightleftharpoons & (b_1\times b_2)\times b_3 & : \mathit{assocr}^{\times} \\
 & & & & \\
\mathit{distrib}_0: & 0\times b & \rightleftharpoons & 0 & : \mathit{factor}_0 \\
\mathit{distrib}: & (b_1+b_2)\times b_3 & \rightleftharpoons & (b_1\times b_3)+(b_2\times b_3) & : \mathit{factor}
\end{array}
$$

$$
\frac{}{\mathit{id} : b \rightleftharpoons b}
\qquad
\frac{c : b_1 \rightleftharpoons b_2}{\mathit{sym}\ c : b_2 \rightleftharpoons b_1}
\qquad
\frac{c_1 : b_1 \rightleftharpoons b_2 \quad c_2 : b_2 \rightleftharpoons b_3}{(c_1 \fatsemi c_2) : b_1 \rightleftharpoons b_3}
$$

$$
\frac{c_1 : b_1 \rightleftharpoons b_3 \quad c_2 : b_2 \rightleftharpoons b_4}{(c_1 + c_2) : (b_1+b_2) \rightleftharpoons (b_3+b_4)}
\qquad
\frac{c_1 : b_1 \rightleftharpoons b_3 \quad c_2 : b_2 \rightleftharpoons b_4}{(c_1 \times c_2) : (b_1\times b_2) \rightleftharpoons (b_3\times b_4)}
$$

Example

The type isomorphism:

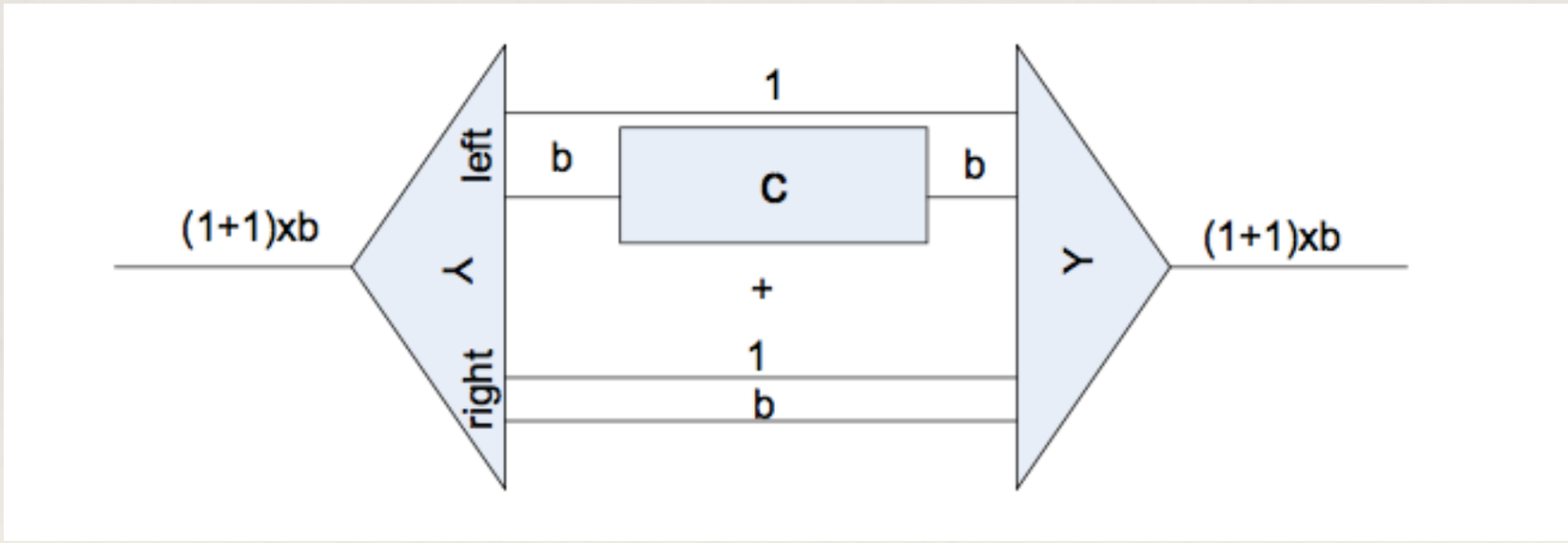
$$
\begin{aligned}
& (1+1)\times((1+1)\times b) \\
= \quad & (1+1)\times((1\times b)+(1\times b)) \\
= \quad & (1+1)\times(b+b) \\
= \quad & (1\times(b+b))+(1\times(b+b)) \\
= \quad & (b+b)+(b+b)
\end{aligned}
$$

corresponds to the program:

$$(id \times (distrib \fatsemi (unite \times unite))) \fatsemi (distrib \fatsemi (unite \times unite))$$

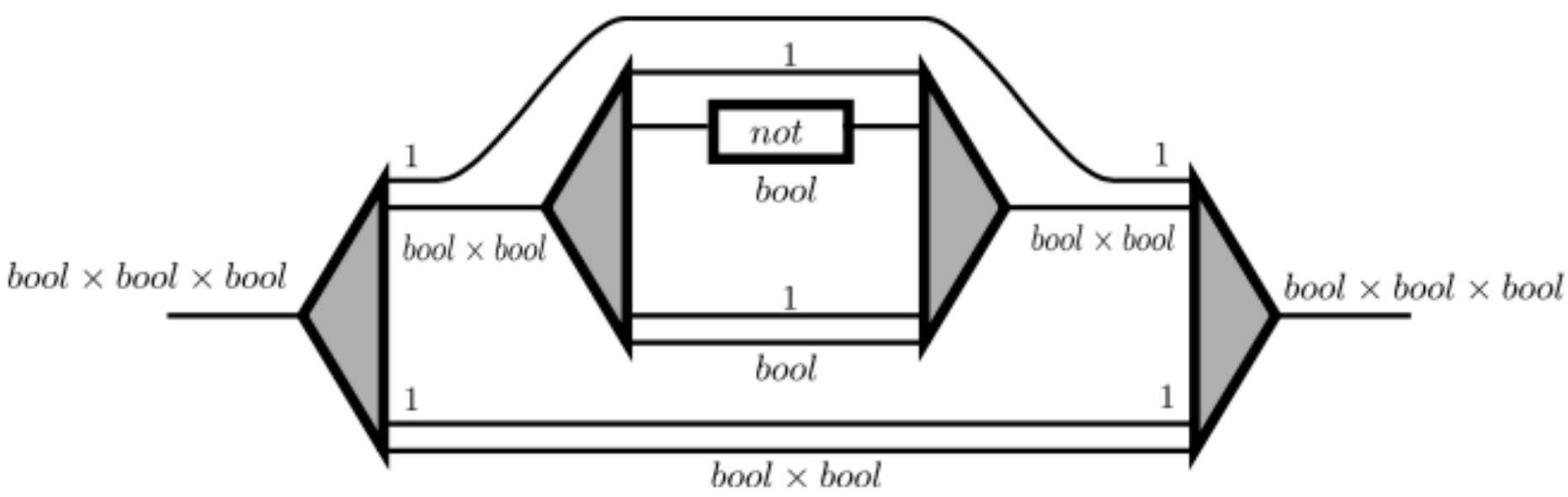
Conditionals

$$\mathbf{if}_c : bool \times b \rightleftharpoons bool \times b$$
$$\mathbf{if}_c = distrib \fatsemi ((id \times c) + id) \fatsemi factor$$



Toffoli and Fredkin Gates

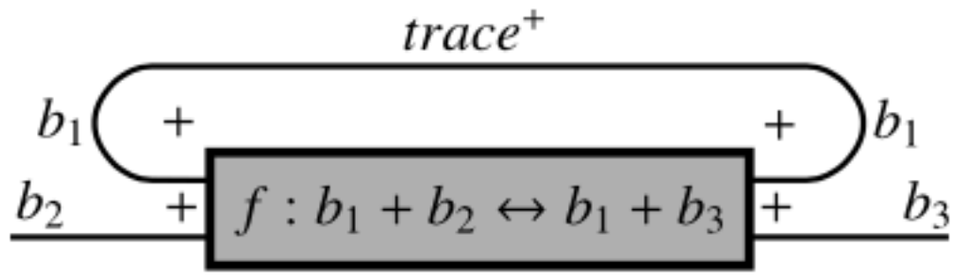
- Fredkin gate: $\mathbf{if}_{swap^{\times}}$
- Toffoli gate: $\mathbf{if}_{\mathbf{if}_{swap^{+}}}$



Recursion

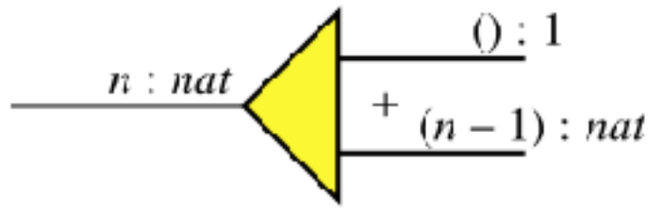
- Add recursive types (so that we can now define natural numbers, lists, trees, etc.)

- Add categorical trace (the categorical abstraction of feedback, looping, and recursion)

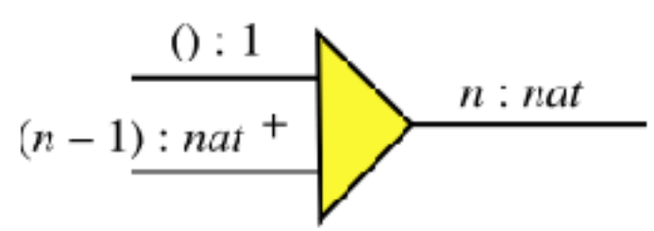


Numbers

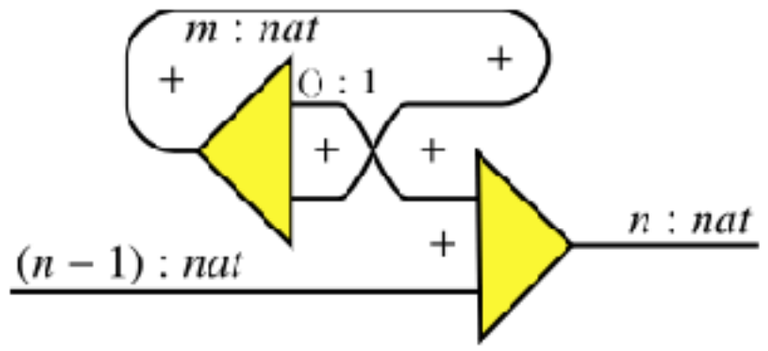
- Unfolding a natural number:



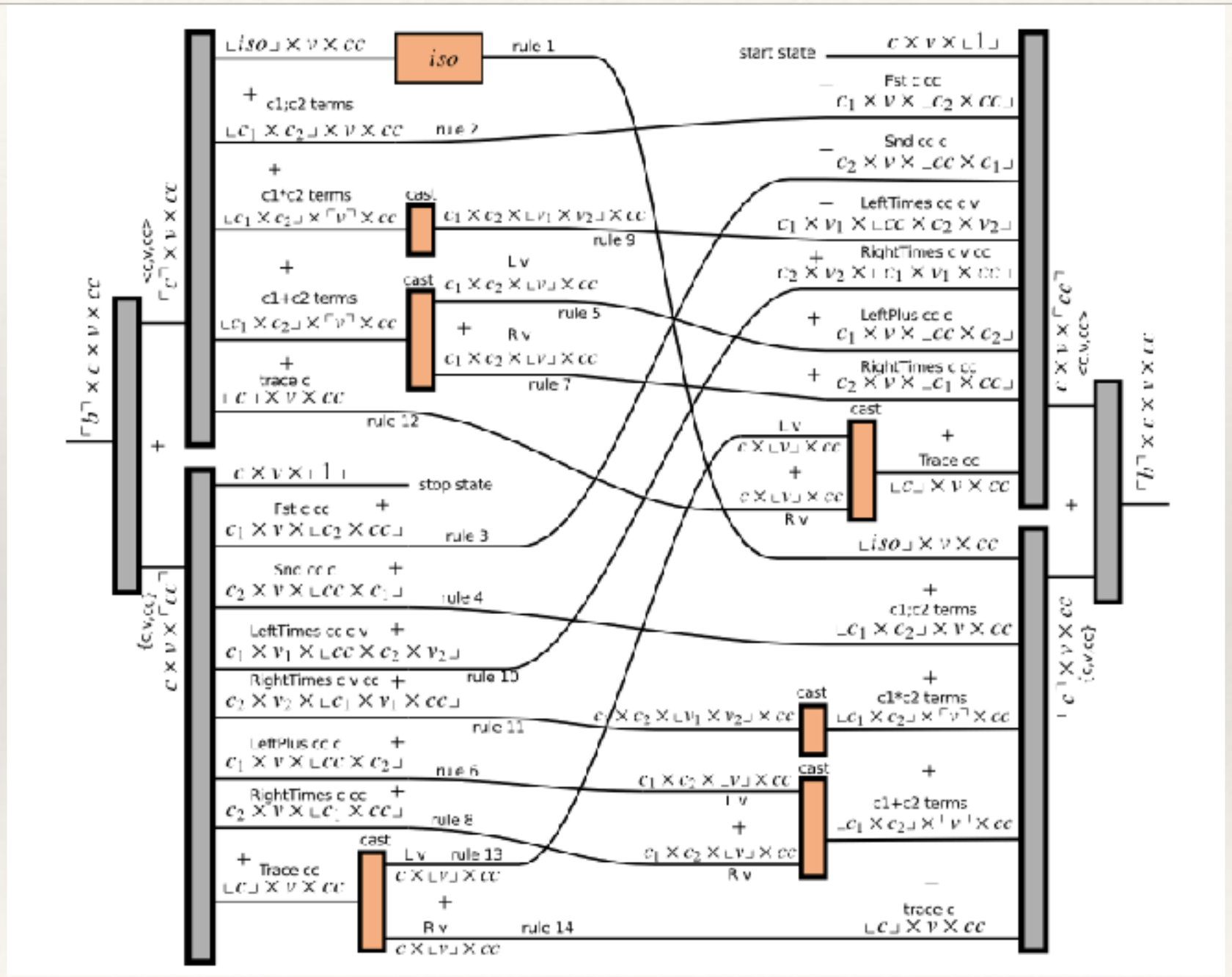
- Folding a natural number:



- add1/sub1 (partial isomorphisms)

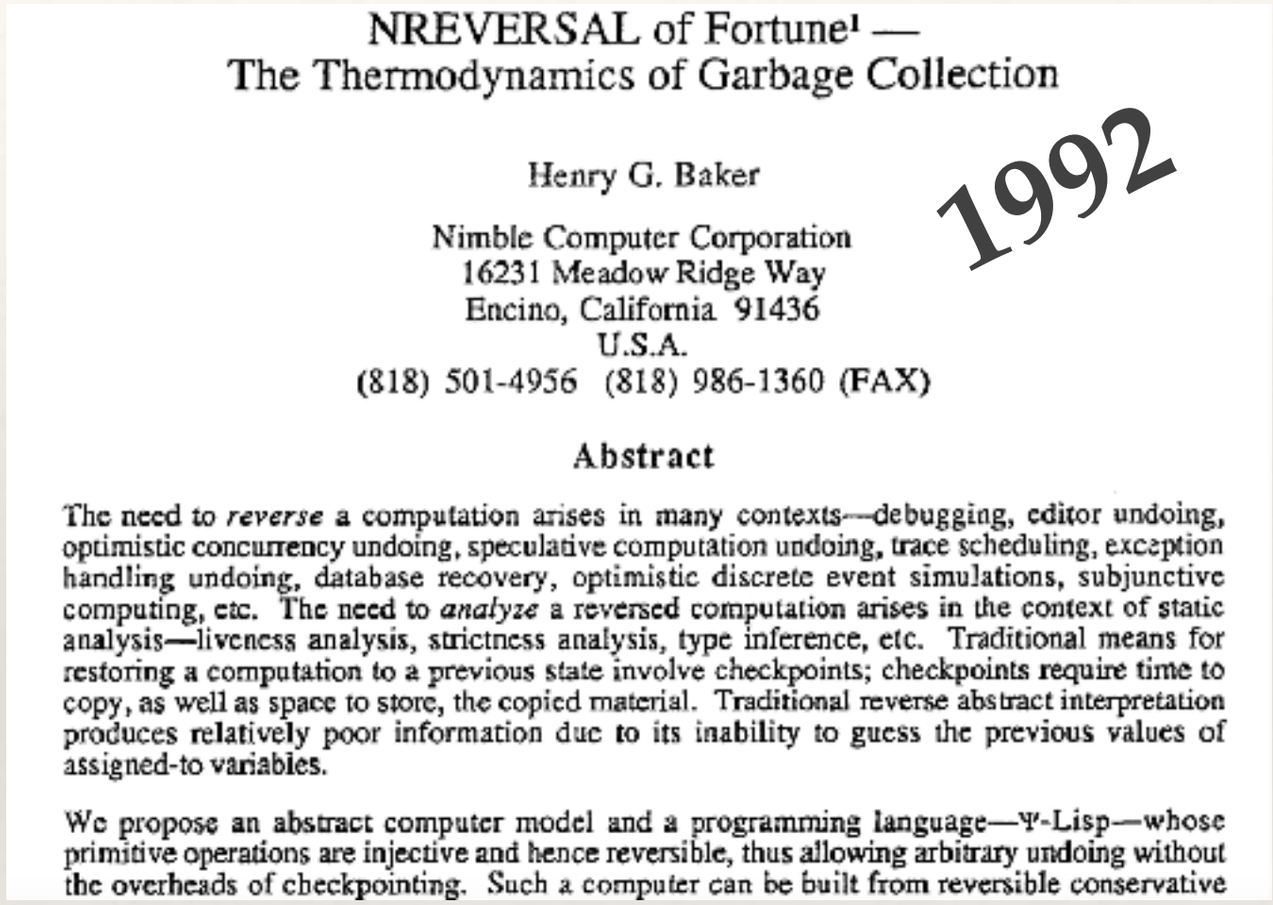


A meta-circular interpreter



Reversibility

- Many other reversible languages designed from different perspectives
- Special recognition to Henry Baker
- Many ad hoc constructions: database transactions, data provenance, checkpoints, logs, rollback recovery, version control systems, backtracking, continuations



NREVERSAL of Fortune[1] —
The Thermodynamics of Garbage Collection

1992

Henry G. Baker

Nimble Computer Corporation
16231 Meadow Ridge Way
Encino, California 91436
U.S.A.
(818) 501-4956 (818) 986-1360 (FAX)

Abstract

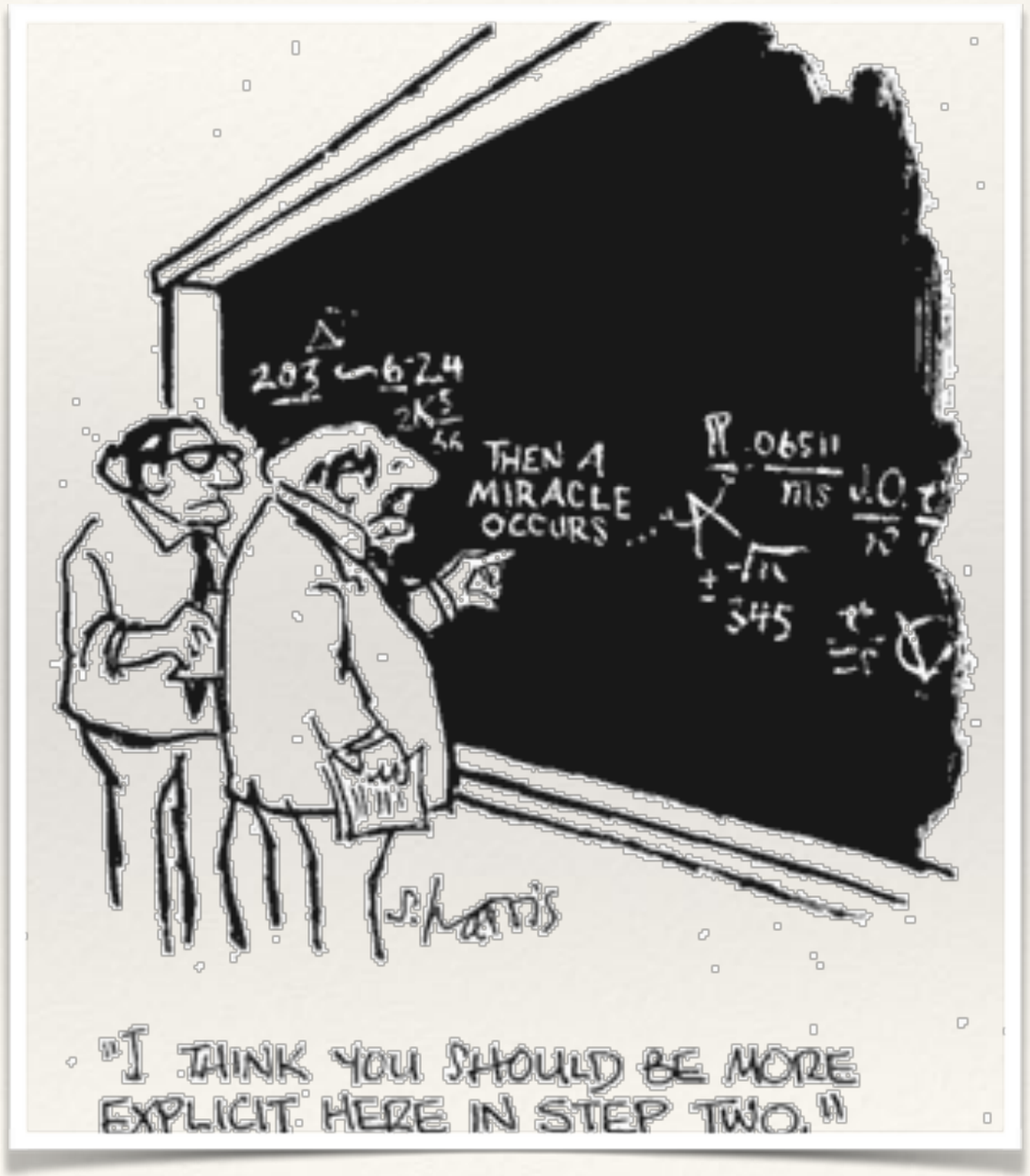
The need to *reverse* a computation arises in many contexts—debugging, editor undoing, optimistic concurrency undoing, speculative computation undoing, trace scheduling, exception handling undoing, database recovery, optimistic discrete event simulations, subjunctive computing, etc. The need to *analyze* a reversed computation arises in the context of static analysis—liveness analysis, strictness analysis, type inference, etc. Traditional means for restoring a computation to a previous state involve checkpoints; checkpoints require time to copy, as well as space to store, the copied material. Traditional reverse abstract interpretation produces relatively poor information due to its inability to guess the previous values of assigned-to variables.

We propose an abstract computer model and a programming language—Ψ-Lisp—whose primitive operations are injective and hence reversible, thus allowing arbitrary undoing without the overheads of checkpointing. Such a computer can be built from reversible conservative

Homotopy Type Theory

Equality

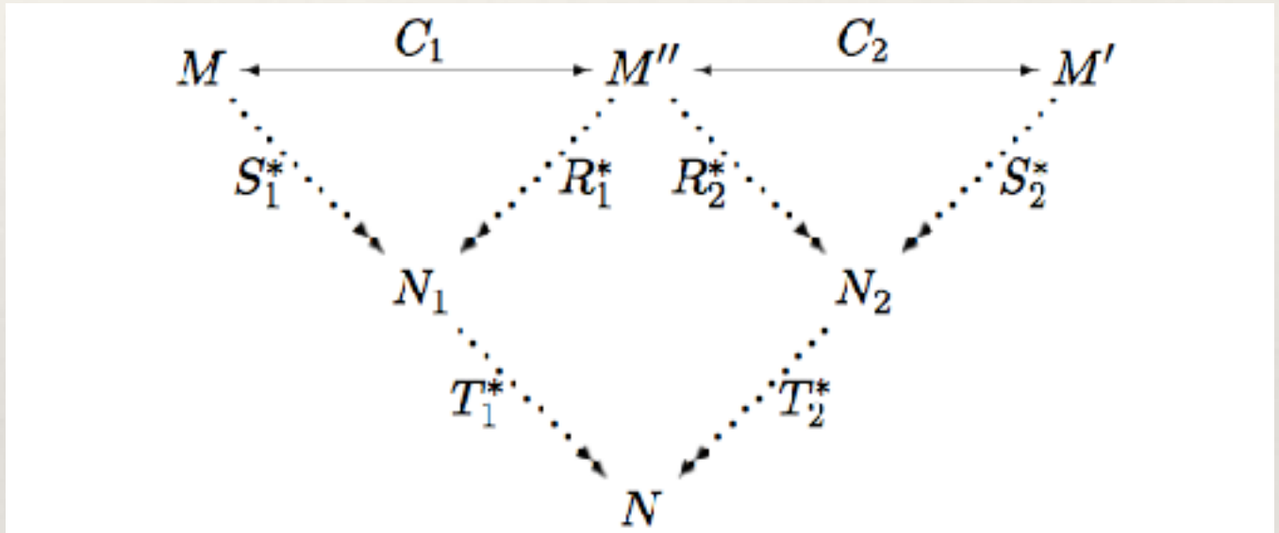
- In mathematics, physics, etc. we often use the equality sign without *explicit* justification.
- Sometimes this is innocent: *5+2 = 2+5* is has an *"obvious"* justification.
- What is the justification of = in the equation $E = mc^2$?
- Would we be justified in using = in $N = N \times N$?



Equality of Programs in a Computational Setting

In the λ-calculus $M = M'$ implies
$M \to^* N_1 \to^* N \to^{-1*} N_2 \to^{-1*} M'$

- An *operational* understanding of equality based on a notion of reduction
- Take compatible, reflexive, symmetric, and transitive closure of reduction to define equality
- Need Church-Rosser theorem to show that 1 is ***not equal*** to 2



Equality of Types in a Computational Setting

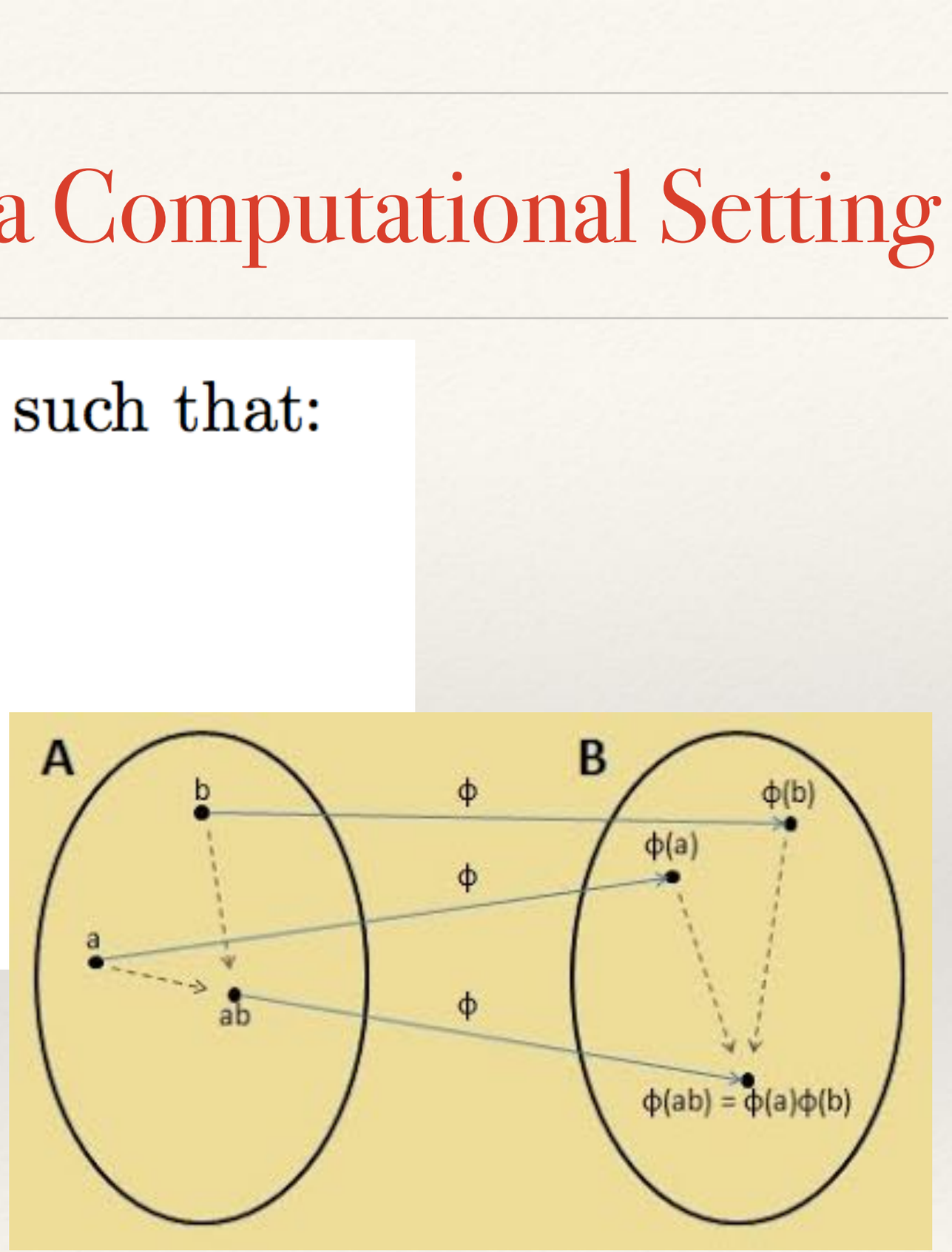
- Given an "interesting" language of types (polymorphism, dependent types, etc.)
- We have an *operational* understanding of type equality

$$\textsc{Type-reduction}\ \frac{}{(\Lambda X.\, e)\ [\tau] \rightarrow e\{\tau/X\}}$$

Equality of Types in a Computational Setting

$A \simeq B$ if exists f and g such that:

- $f : A \to B$
- $g : B \to A$
- $g \circ f = id_A$
- $f \circ g = id_B$



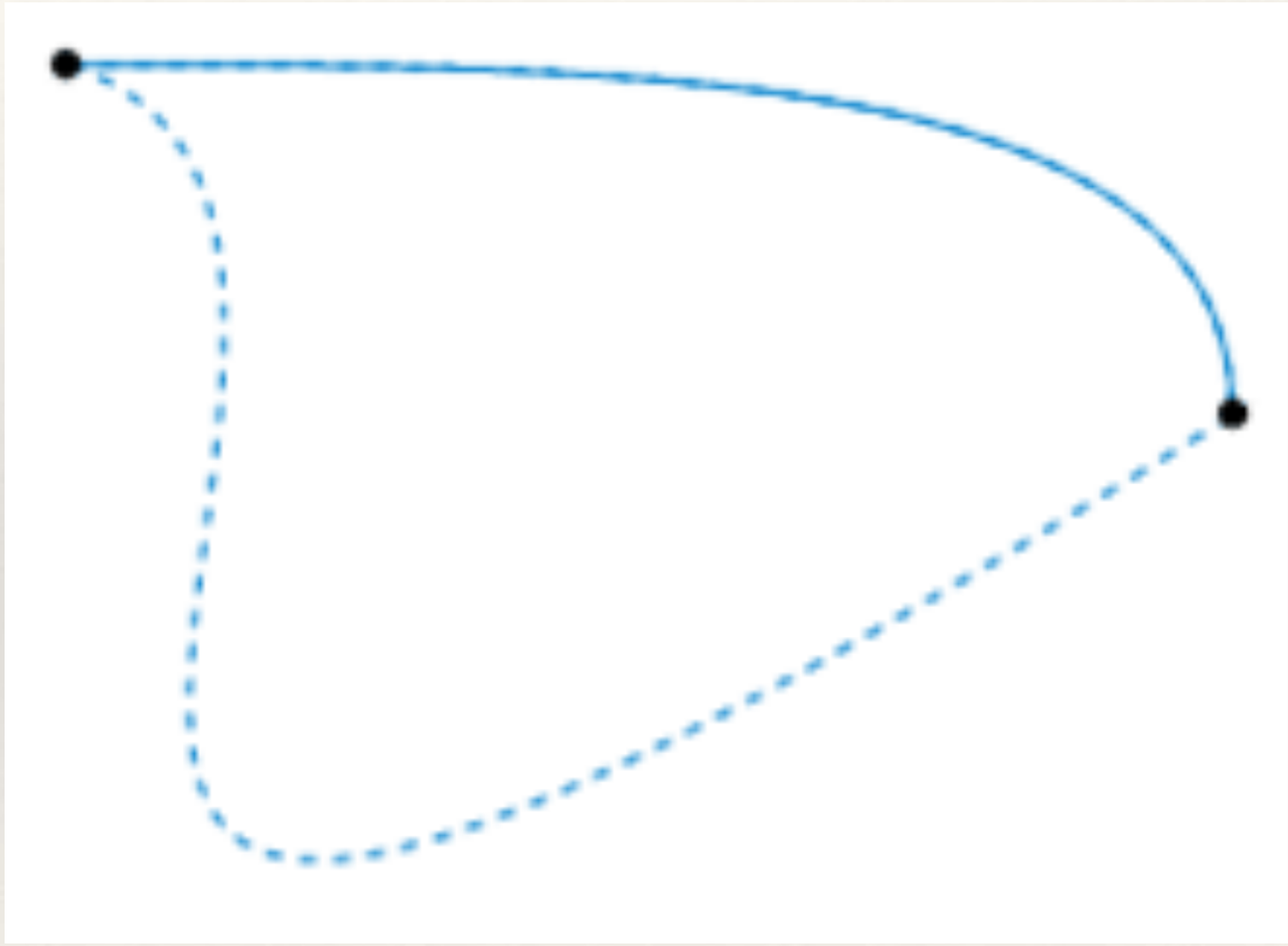
We also have an *extensional* understanding of type equality

Type Equality

- Given an operational model of how a type A **reduces** to a type B
- Given a denotational model of which types A and B are equivalent
- Want soundness and completeness results
- Guarantee that the two notions of equality are themselves "equivalent"
- Generalized "identity of indiscernibles"

Path Types and Equivalences

- In HoTT
- *Path types* capture the operational perspective on equality
- *Equivalences* capture the extensional perspective on equality



Univalence

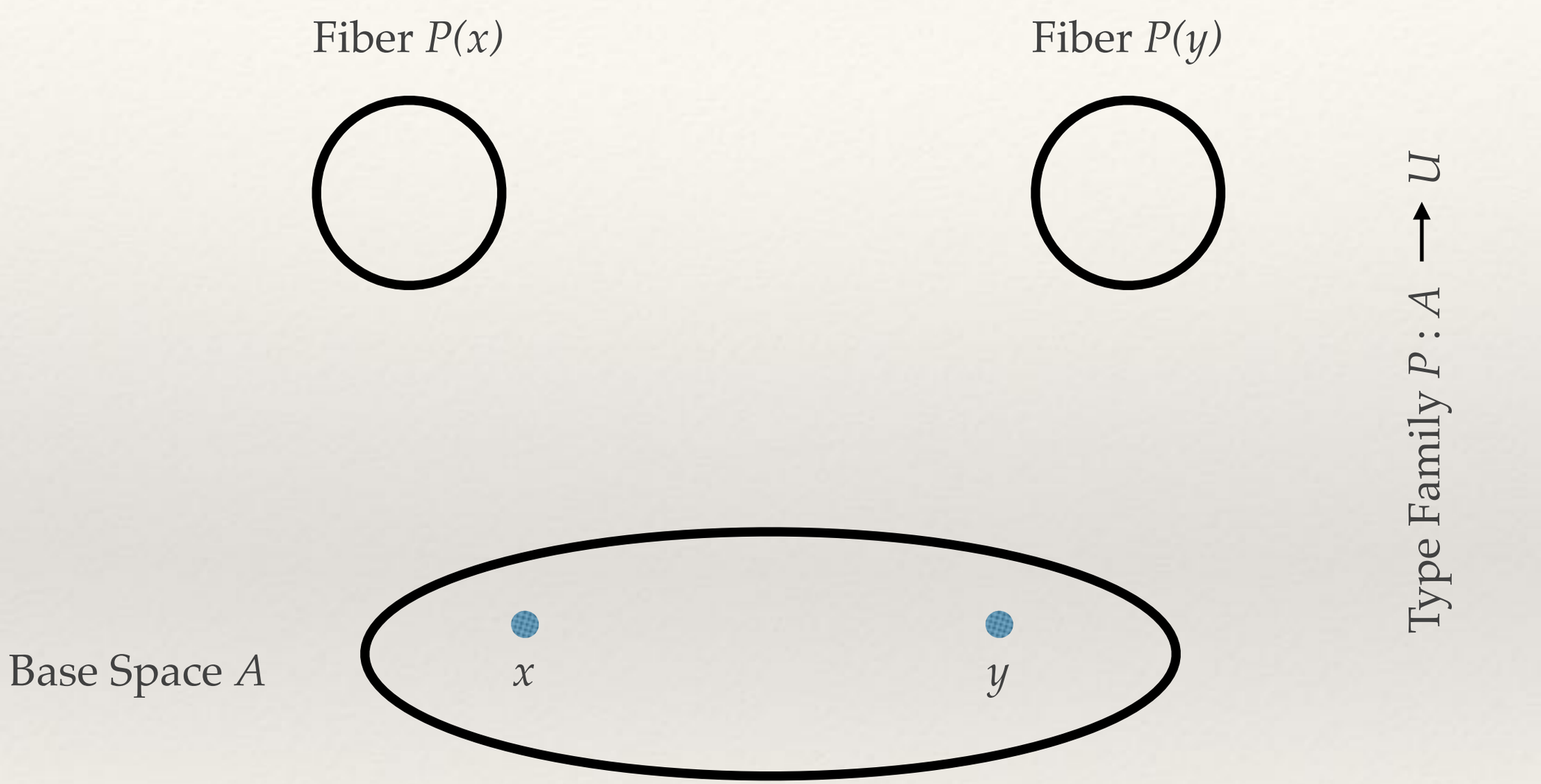
$$(A \equiv B) \simeq (A \simeq B)$$

where $A \equiv B$ is the type of paths

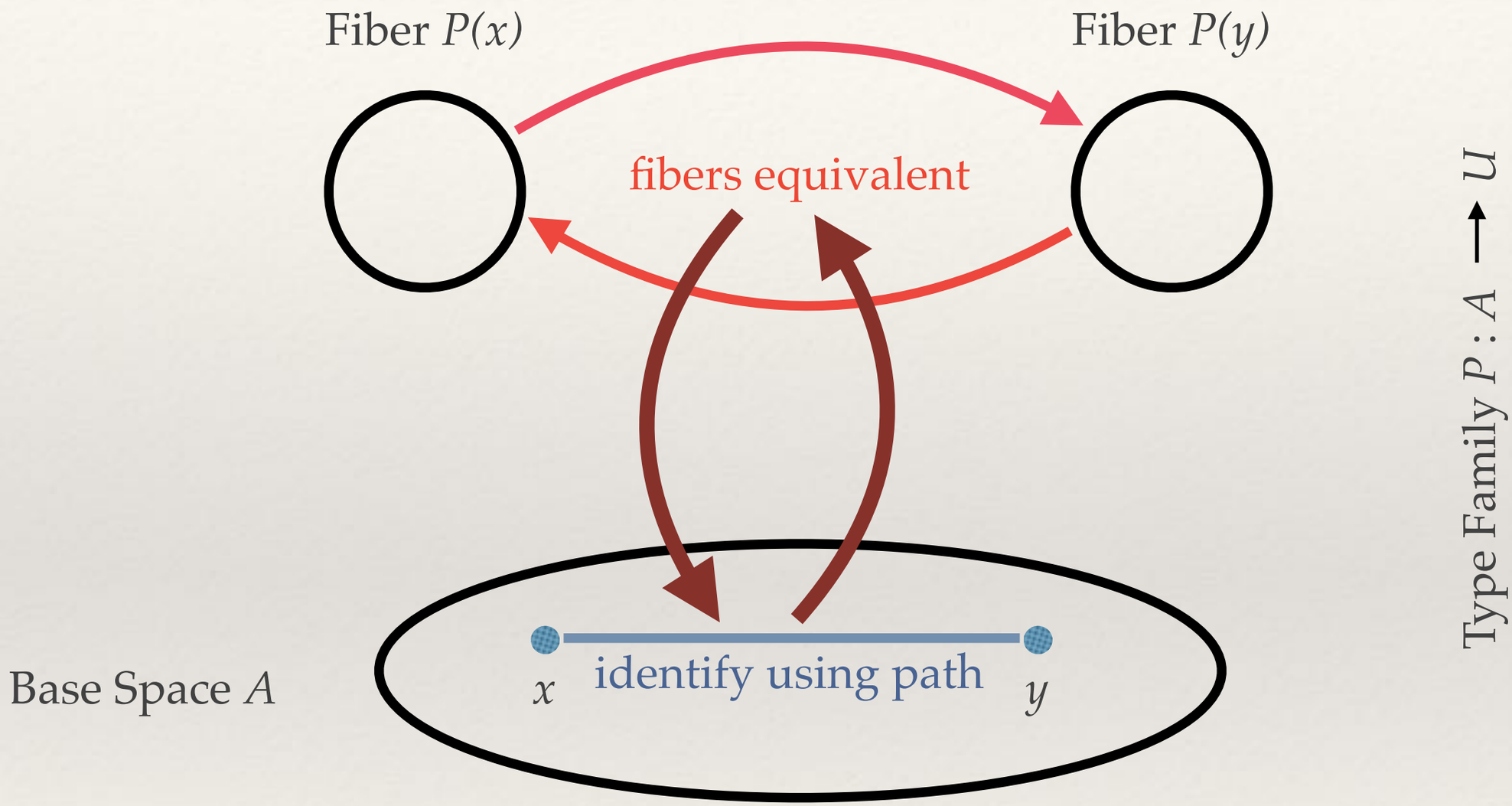
and $A \simeq B$ is the type of equivalences

A postulate / axiom in "book HoTT"

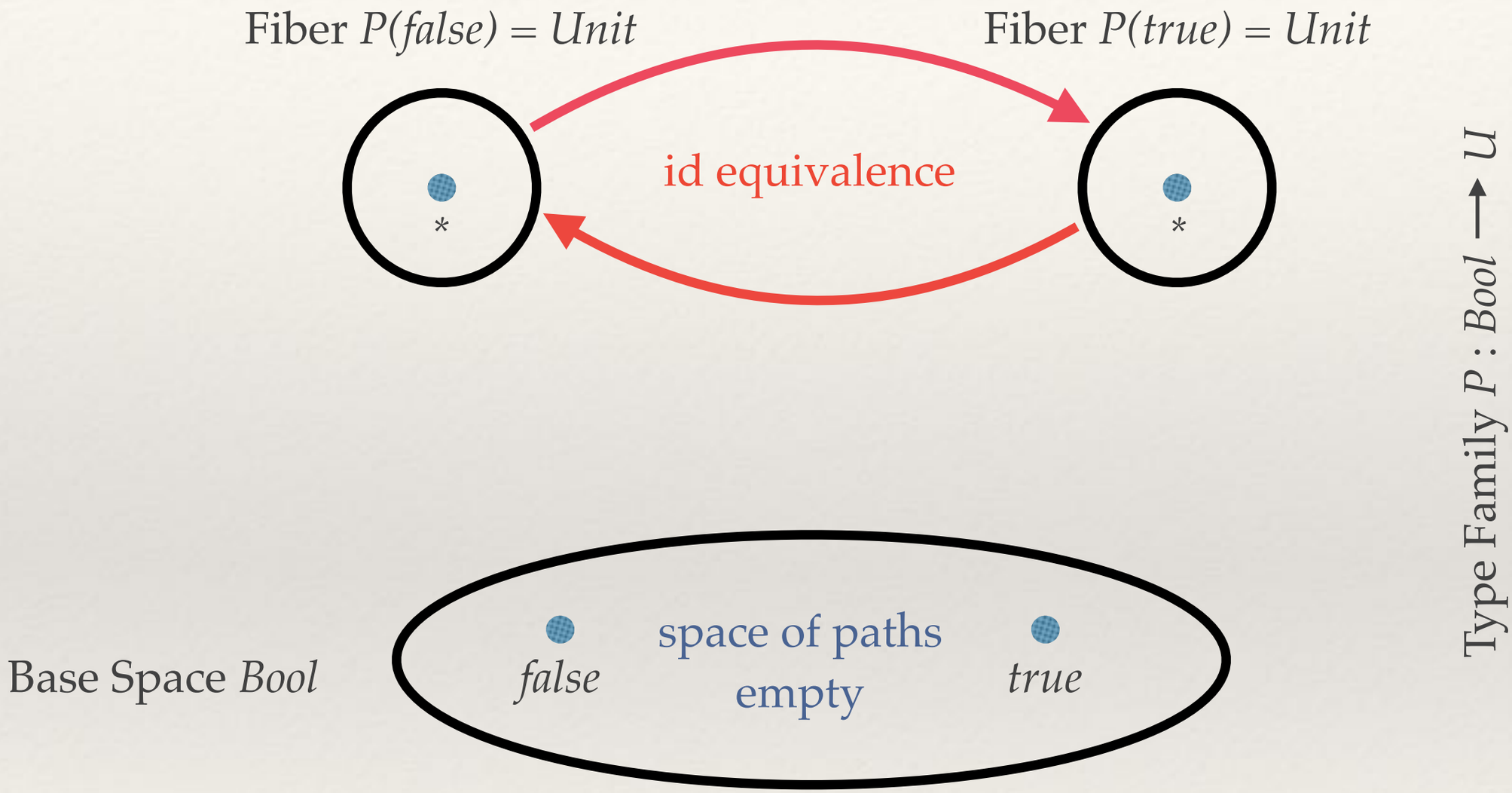
Fibrations



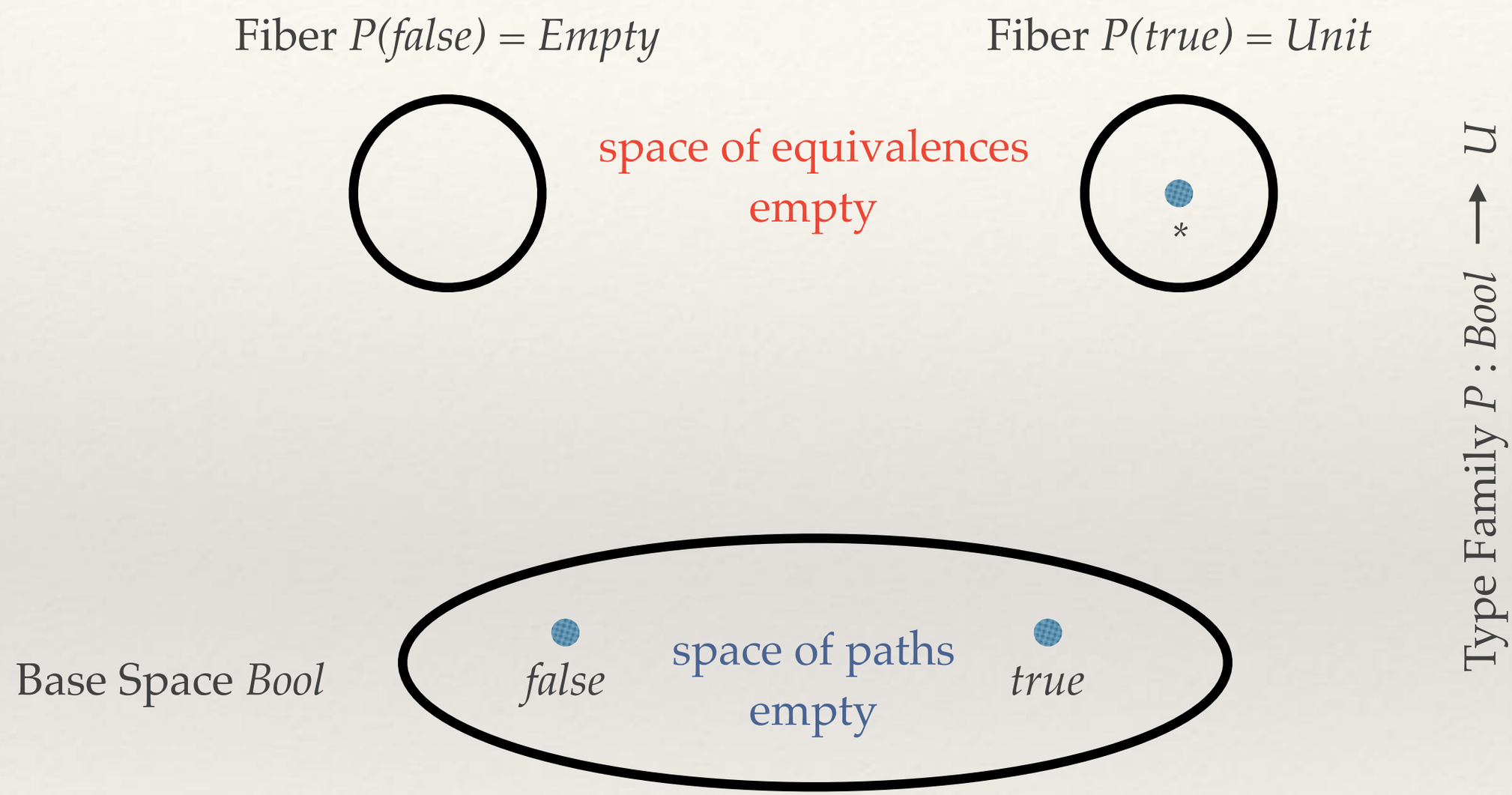
Univalent Fibrations



A Non-Univalent Fibration



A Univalent Fibration



Characterizing Univalent Fibrations

A given type X is <u>rarely</u> the base of a univalent fibration but we have the following theorem.

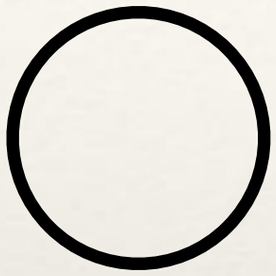
Theorem. *For any type F, the type:*

$$\Sigma A{:}U \, \| F \equiv A \|$$

is always the base of a univalent fibration.

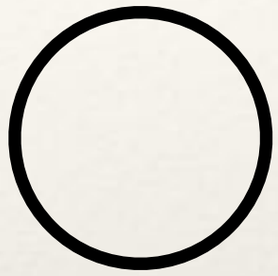
Choose F = *Bool*

Fiber $P(A, |\, p\, |) = A$



…

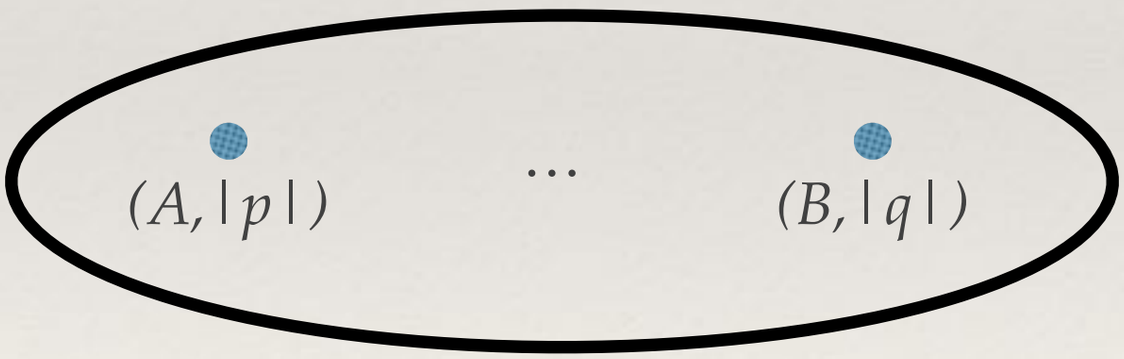
Fiber $P(B, |\, q\, |) = B$



Type Family $P : \Sigma_{(A:U)} \| Bool \equiv A \| \longrightarrow U$

Base Space

$\Sigma_{(A:U)} \| Bool \equiv A \|$



Questions

- What does this mean?
- What does $\Sigma_{(A:U)} \| Bool \equiv A \|$ look like?
- The "normal" thing to do is to explain the ingredients: first Σ-types, then propositional truncations $\|\_\|$, and finally identity types $A \equiv B$
- *That's about four chapters of the HoTT book!*

A Different Answer

- $\Sigma_{(A:\mathcal{U})} \| Bool \equiv A \|$ is the specification of a non-trivial but rather intuitive reversible programming language!
- Paths are reversible programs
- Univalence ensures every equivalence is expressible as a reversible program
- The space has non-trivial 2-paths!!! So we have a layer of reversible programs that manipulate other reversible programs in reversible ways