Computing with Semirings and Weak Rig Groupoids

Jacques Carette   Amr Sabry

McMaster University

Indiana University

7 April 2016

Simple Curry-Howard

Constructors

Logic	Types
true	$\top$
false	$\bot$
$\wedge$	$\times$
$\vee$	$\uplus$
$\Rightarrow$	$\rightarrow$
$\forall$	$\prod$
$\exists$	$\sum$

Simple Curry-Howard

Constructors

Logic	Types
true	$\top$
false	$\bot$
$\wedge$	$\times$
$\vee$	$\uplus$
$\Rightarrow$	$\rightarrow$
$\forall$	$\prod$
$\exists$	$\sum$

Digging deeper:

Logical Implication	Inhabiting Term
$A \wedge B \Rightarrow B \wedge A$	$\lambda\{(a, b) \rightarrow (b, a)\}$
$true \wedge A \Rightarrow A$	$\lambda\{(tt, a) \rightarrow a\}$
$A \Rightarrow true \wedge A$	$\lambda a \rightarrow tt, a$
$false \vee A \Rightarrow A$	$\bot e_+ : \{A : \mathsf{Set}\} \rightarrow \bot \uplus A \rightarrow A$ $\bot e_+ \ (\mathsf{inj}_1 \ ())$ $\bot e_+ \ (\mathsf{inj}_2 \ y) = y$
$A \Rightarrow false \vee A$	$\bot i_+ : \{A : \mathsf{Set}\} \rightarrow A \rightarrow \bot \uplus A$ $\bot i_+ \ a = \mathsf{inj}_2 \ a$

Simple Curry-Howard

Constructors

Logic	Types
true	$\top$
false	$\bot$
$\wedge$	$\times$
$\vee$	$\uplus$
$\Rightarrow$	$\rightarrow$
$\forall$	$\prod$
$\exists$	$\sum$

Digging deeper:

Logical Implication	Inhabiting Term
$A \wedge B \Rightarrow B \wedge A$	$\lambda\{(a,b) \rightarrow (b,a)\}$
$true \wedge A \Rightarrow A$	$\lambda\{(tt,a) \rightarrow a\}$
$A \Rightarrow true \wedge A$	$\lambda a \rightarrow tt, a$
$false \vee A \Rightarrow A$	$\bot e_+ : \{A : \mathsf{Set}\} \rightarrow \bot \uplus A \rightarrow A$ $\bot e_+ \; (\mathsf{inj}_1 \; ())$ $\bot e_+ \; (\mathsf{inj}_2 \; y) = y$
$A \Rightarrow false \vee A$	$\bot i_+ : \{A : \mathsf{Set}\} \rightarrow A \rightarrow \bot \uplus A$ $\bot i_+ \; a = \mathsf{inj}_2 \; a$

Logical Equivalence	Type Isomorphism
$A \wedge B \Leftrightarrow B \wedge A$	$A \times B \simeq B \times A$
$true \wedge A \Leftrightarrow A$	$\top \times A \simeq A$
$false \vee A \Leftrightarrow A$	$\bot \uplus A \simeq A$

Classical Curry-Howard: Inhabitation

Logical Equivalence	Type non-Isomorphism
$A \vee A \Leftrightarrow A$	$A \uplus A \not\simeq A$
$A \wedge A \Leftrightarrow A$	$A \times A \not\simeq A$
$true \vee A \Leftrightarrow true$	$\top \uplus A \not\simeq \top$

There *are* functions which witness inhabitation in each direction, but these forget information, i.e. are not inverses.

Classical Curry-Howard: Inhabitation

Logical Equivalence	Type non-Isomorphism
$A \vee A \Leftrightarrow A$	$A \uplus A \not\simeq A$
$A \wedge A \Leftrightarrow A$	$A \times A \not\simeq A$
$\mathit{true} \vee A \Leftrightarrow \mathit{true}$	$\top \uplus A \not\simeq \top$

There *are* functions which witness inhabitation in each direction, but these forget information, i.e. are not inverses.

Desideratum: only isomorphisms in the right column.
Motivation and inspiration:

- Reversible computing; Bidirectional computing; Quantum computing.
- Conservation of resources (including information)
- Homotopy Type Theory.

Classical Curry-Howard: Inhabitation

Logical Equivalence	Type non-Isomorphism
$A \vee A \Leftrightarrow A$	$A \uplus A \not\simeq A$
$A \wedge A \Leftrightarrow A$	$A \times A \not\simeq A$
$\mathit{true} \vee A \Leftrightarrow \mathit{true}$	$\top \uplus A \not\simeq \top$

There *are* functions which witness inhabitation in each direction, but these forget information, i.e. are not inverses.

Desideratum: only isomorphisms in the right column.
Motivation and inspiration:

- Reversible computing; Bidirectional computing; Quantum computing.
- Conservation of resources (including information)
- Homotopy Type Theory.

Q: What would the table look like then?

Type isomorphisms I

Definition 1 (Homotopy).

Two functions $f, g : A \to B$ are *homotopic*, written $f \sim g$, if $\forall x : A. f(x) = g(x)$.

```
_∼_ : ∀ {A : Set} {P : A → Set} → (f g : (x : A) → P x) → Set
_∼_ {A} f g = (x : A) → f x ≡ g x
```

Type isomorphisms I

Definition 1 (Homotopy).

Two functions $f, g : A \to B$ are *homotopic*, written $f \sim g$, if $\forall x : A . f(x) = g(x)$.

```
_∼_ : ∀ {A : Set} {P : A → Set} → (f g : (x : A) → P x) → Set
_∼_ {A} f g = (x : A) → f x ≡ g x
```

Definition 2 (Quasi-inverse).

For a function $f : A \to B$, a *quasi-inverse* of f is a triple (g, α, β), consisting of a function $g : B \to A$ and two homotopies $\alpha : f \circ g \sim \mathrm{id}_B$ and $\beta : g \circ f \sim \mathrm{id}_A$.

```
record isqinv {A : Set} {B : Set} (f : A → B) : Set where
  constructor qinv
  field
    g : B → A
    α : (f ∘ g) ∼ id
    β : (g ∘ f) ∼ id
```

Type isomorphisms II

Definition 3 (Equivalence of types).

Two types A and B are equivalent $A \simeq B$ if there exists a function $f : A \to B$ together with a quasi-inverse for f.

```
_≃_ : Set → Set → Set
A ≃ B = Σ (A → B) isqinv
```

Type isomorphisms II

Definition 3 (Equivalence of types).

Two types A and B are equivalent $A \simeq B$ if there exists a function $f : A \to B$ together with a quasi-inverse for f.

```
_≃_ : Set → Set → Set
A ≃ B = Σ (A → B) isqinv
```

(Tricky) Example

```
left0e : {A : Set} → ⊥ × A → ⊥
left0e = proj₁
left0i : {A : Set} → ⊥ → ⊥ × A
left0i = ⊥-elim

left0ei : {A : Set} → (left0e {A} ∘ left0i) ∼ id
left0ei ()
left0ie : {A : Set} → (left0i {A} ∘ left0e) ∼ id
left0ie (() , _)
```

```
⊥≃ : {A : Set} → (⊥ × A) ≃ ⊥
⊥≃ = left0e ,
     qinv left0i left0ei left0ie
```

Type isomorphisms III

Type isomorphisms
(Sound and complete)

$$\begin{array}{rcl} \bot \uplus A & \simeq & A \\ A \uplus B & \simeq & B \uplus A \\ A \uplus (B \uplus C) & \simeq & (A \uplus B) \uplus C \end{array}$$

$$\begin{array}{rcl} \top \times A & \simeq & A \\ A \times B & \simeq & B \times A \\ A \times (B \times C) & \simeq & (A \times B) \times C \end{array}$$

$$\begin{array}{rcl} \bot \times A & \simeq & \bot \\ (A \uplus B) \times C & \simeq & (A \times C) \uplus (B \times C) \end{array}$$

Type isomorphisms III

Type isomorphisms
(Sound and complete)

$$\begin{array}{rcl}
\bot \uplus A & \simeq & A \\
A \uplus B & \simeq & B \uplus A \\
A \uplus (B \uplus C) & \simeq & (A \uplus B) \uplus C \\
\\
\top \times A & \simeq & A \\
A \times B & \simeq & B \times A \\
A \times (B \times C) & \simeq & (A \times B) \times C \\
\\
\bot \times A & \simeq & \bot \\
(A \uplus B) \times C & \simeq & (A \times C) \uplus (B \times C)
\end{array}$$

Semiring Equalities
(Sound and complete)

$$\begin{array}{rcl}
0 + a & = & a \\
a + b & = & b + a \\
a + (b + c) & = & (a + b) + c \\
\\
1 \cdot a & = & a \\
a \cdot b & = & b \cdot a \\
a \cdot (b \cdot c) & = & (a \cdot b) \cdot c \\
\\
0 \cdot a & = & 0 \\
(a + b) \cdot c & = & (a \cdot c) + (b \cdot c)
\end{array}$$

Observations

1. the inhabitants of type isomorphisms are exactly the proof terms of basic commutative semiring identities

Observations

1. the inhabitants of type isomorphisms are exactly the proof terms of basic commutative semiring identities
2. the proof terms for a commutative semiring make a nice base programming language

Observations

1. the inhabitants of type isomorphisms are exactly the proof terms of basic commutative semiring identities

2. the proof terms for a commutative semiring make a nice base programming language

3. both are missing combinators
   - how would I prove $(0 \uplus A) \times (B \times C) \simeq (A \times B) \times C$?

Observations

1. the inhabitants of type isomorphisms are exactly the proof terms of basic commutative semiring identities
2. the proof terms for a commutative semiring make a nice base programming language
3. both are missing combinators
   - how would I prove $(0 \uplus A) \times (B \times C) \simeq (A \times B) \times C$?
4. $A \ x \ A \ \simeq \ A \ x \ A$ has two proofs, which are not equal
   - identity
   - swap (commutativity)

Observations

1. the inhabitants of type isomorphisms are exactly the proof terms of basic commutative semiring identities
2. the proof terms for a commutative semiring make a nice base programming language
3. both are missing combinators
   - how would I prove $(0 \uplus A) \times (B \times C) \simeq (A \times B) \times C$?
4. $A \; x \; A \;\simeq\; A \; x \; A$ has two proofs, which are not equal
   - identity
   - swap (commutativity)

Let's make a PL out of semiring identities / type isomorphisms!

The Π Language: reifying isomorphisms

$$
\begin{array}{rrcll}
id : & \tau & \leftrightarrow & \tau & : id \\
identl_+ : & 0 + \tau & \leftrightarrow & \tau & : identr_+ \\
swap_+ : & \tau_1 + \tau_2 & \leftrightarrow & \tau_2 + \tau_1 & : swap_+ \\
assocl_+ : & \tau_1 + (\tau_2 + \tau_3) & \leftrightarrow & (\tau_1 + \tau_2) + \tau_3 & : assocr_+ \\
identl_* : & 1 * \tau & \leftrightarrow & \tau & : identr_* \\
swap_* : & \tau_1 * \tau_2 & \leftrightarrow & \tau_2 * \tau_1 & : swap_* \\
assocl_* : & \tau_1 * (\tau_2 * \tau_3) & \leftrightarrow & (\tau_1 * \tau_2) * \tau_3 & : assocr_* \\
dist_0 : & 0 * \tau & \leftrightarrow & 0 & : factorl_0 \\
dist : & (\tau_1 + \tau_2) * \tau_3 & \leftrightarrow & (\tau_1 * \tau_3) + (\tau_2 * \tau_3) & : factor
\end{array}
$$

$$
\frac{\vdash c_1 : \tau_1 \leftrightarrow \tau_2 \quad \vdash c_2 : \tau_2 \leftrightarrow \tau_3}{\vdash c_1 \odot c_2 : \tau_1 \leftrightarrow \tau_3}
\qquad
\frac{\vdash c_1 : \tau_1 \leftrightarrow \tau_2 \quad \vdash c_2 : \tau_3 \leftrightarrow \tau_4}{\vdash c_1 \oplus c_2 : \tau_1 + \tau_3 \leftrightarrow \tau_2 + \tau_4}
$$

$$
\frac{\vdash c_1 : \tau_1 \leftrightarrow \tau_2 \quad \vdash c_2 : \tau_3 \leftrightarrow \tau_4}{\vdash c_1 \otimes c_2 : \tau_1 * \tau_3 \leftrightarrow \tau_2 * \tau_4}
$$

Example

One possible program that corresponds to the type isomorphism:

$$
\begin{array}{rl}
 & (1+1) * ((1+1) * b) \\
= & (1+1) * ((1 * b) + (1 * b)) \\
= & (1+1) * (b+b) \\
= & (1 * (b+b)) + (1 * (b+b)) \\
= & (b+b) + (b+b)
\end{array}
$$

is:

$$
(id \otimes (dist \odot (identl_* \oplus identl_*))) \odot (dist \odot (identl_* \oplus identl_*))
$$

Syntactically

Start with a universe of (finite) types

```
data U : Set where
  ZERO  : U
  ONE   : U
  PLUS  : U → U → U
  TIMES : U → U → U
```

and its interpretation

```
⟦_⟧ : U → Set
⟦ ZERO ⟧        = ⊥
⟦ ONE ⟧         = ⊤
⟦ PLUS t₁ t₂ ⟧  = ⟦ t₁ ⟧ ⊎ ⟦ t₂ ⟧
⟦ TIMES t₁ t₂ ⟧ = ⟦ t₁ ⟧ × ⟦ t₂ ⟧
```

For notational brevity, we will denote these $0, 1, +$, and $*$ respectively.

and the terms and combinators:

```
data _↔_ : U → U → Set where
unite₊  : {t : U} → PLUS ZERO t ↔ t
uniti₊  : {t : U} → t ↔ PLUS ZERO t
swap₊   : {t₁ t₂ : U} → PLUS t₁ t₂ ↔ PLUS t₂ t₁
assocl₊ : {t₁ t₂ t₃ : U} → PLUS t₁ (PLUS t₂ t₃) ↔ PLUS (PLUS t₁ t₂) t₃
assocr₊ : {t₁ t₂ t₃ : U} → PLUS (PLUS t₁ t₂) t₃ ↔ PLUS t₁ (PLUS t₂ t₃)
unite⋆  : {t : U} → TIMES ONE t ↔ t
uniti⋆  : {t : U} → t ↔ TIMES ONE t
swap⋆   : {t₁ t₂ : U} → TIMES t₁ t₂ ↔ TIMES t₂ t₁
assocl⋆ : {t₁ t₂ t₃ : U} → TIMES t₁ (TIMES t₂ t₃) ↔ TIMES (TIMES t₁ t₂) t₃
assocr⋆ : {t₁ t₂ t₃ : U} → TIMES (TIMES t₁ t₂) t₃ ↔ TIMES t₁ (TIMES t₂ t₃)
absorbr  : {t : U} → TIMES ZERO t ↔ ZERO
absorbl : {t : U} → TIMES t ZERO ↔ ZERO
factorzr : {t : U} → ZERO ↔ TIMES t ZERO
factorzl : {t : U} → ZERO ↔ TIMES ZERO t
dist    : {t₁ t₂ t₃ : U} → TIMES (PLUS t₁ t₂) t₃ ↔ PLUS (TIMES t₁ t₃) (TIMES t₂ t₃)
factor  : {t₁ t₂ t₃ : U} → PLUS (TIMES t₁ t₃) (TIMES t₂ t₃) ↔ TIMES (PLUS t₁ t₂) t₃
id↔  : {t : U} → t ↔ t
_⊙_   : {t₁ t₂ t₃ : U}  → (t₁ ↔ t₂) → (t₂ ↔ t₃) → (t₁ ↔ t₃)
_⊕_   : {t₁ t₂ t₃ t₄ : U} → (t₁ ↔ t₃) → (t₂ ↔ t₄) → (PLUS t₁ t₂ ↔ PLUS t₃ t₄)
_⊗_   : {t₁ t₂ t₃ t₄ : U} → (t₁ ↔ t₃) → (t₂ ↔ t₄) → TIMES t₁ t₂ ↔ TIMES t₃ t₄
```

Semantics I: Operational

It really is a PL:

$$
\begin{array}{l}
\mathsf{eval} \ : \{t_1 \ t_2 : \mathsf{U}\} \to (t_1 \leftrightarrow t_2) \to [\![\ t_1\ ]\!] \to [\![\ t_2\ ]\!] \\
\mathsf{eval}\ \mathsf{unite}_+\ (\mathsf{inj}_1\ ()) \\
\mathsf{eval}\ \mathsf{unite}_+\ (\mathsf{inj}_2\ y) = y \\
\mathsf{eval}\ \mathsf{uniti}_+\ x = \mathsf{inj}_2\ x \\
\mathsf{eval}\ \mathsf{swap}_+\ (\mathsf{inj}_1\ x) = \mathsf{inj}_2\ x \\
\mathsf{eval}\ \mathsf{swap}_+\ (\mathsf{inj}_2\ y) = \mathsf{inj}_1\ y
\end{array}
$$

...

Semantics I: Operational

It really is a PL:

$$\mathsf{eval}\ :\ \{t_1\ t_2 : \mathsf{U}\} \to (t_1 \leftrightarrow t_2) \to [\![\ t_1\ ]\!] \to [\![\ t_2\ ]\!]$$
$$\mathsf{eval}\ \mathsf{unite}_+\ (\mathsf{inj}_1\ ())$$
$$\mathsf{eval}\ \mathsf{unite}_+\ (\mathsf{inj}_2\ y) = y$$
$$\mathsf{eval}\ \mathsf{uniti}_+\ x = \mathsf{inj}_2\ x$$
$$\mathsf{eval}\ \mathsf{swap}_+\ (\mathsf{inj}_1\ x) = \mathsf{inj}_2\ x$$
$$\mathsf{eval}\ \mathsf{swap}_+\ (\mathsf{inj}_2\ y) = \mathsf{inj}_1\ y$$

...

And it can also be run backwards:

$$\mathsf{evalB}\ :\ \{t_1\ t_2 : \mathsf{U}\} \to (t_1 \leftrightarrow t_2) \to [\![\ t_2\ ]\!] \to [\![\ t_1\ ]\!]$$

...

$$\mathsf{evalB}\ (c_0 \odot c_1)\ x = \mathsf{evalB}\ c_0\ (\mathsf{evalB}\ c_1\ x)$$
$$\mathsf{evalB}\ (c_0 \oplus c_1)\ (\mathsf{inj}_1\ x) = \mathsf{inj}_1\ (\mathsf{evalB}\ c_0\ x)$$
$$\mathsf{evalB}\ (c_0 \oplus c_1)\ (\mathsf{inj}_2\ y) = \mathsf{inj}_2\ (\mathsf{evalB}\ c_1\ y)$$
$$\mathsf{evalB}\ (c_0 \otimes c_1)\ (x\ ,\ y) = \mathsf{evalB}\ c_0\ x\ ,\ \mathsf{evalB}\ c_1\ y$$

Good properties I

Forward and backward are inverses of each other:

$\mathsf{fwd{\circ}bwd{\sim}id} : \forall\ \{t_1\ t_2\} \to (c : t_1 \leftrightarrow t_2) \to (\mathsf{eval}\ c \circ \mathsf{evalB}\ c) \sim \mathsf{id}$

$\mathsf{bwd{\circ}fwd{\sim}id} : \forall\ \{t_1\ t_2\} \to (c : t_1 \leftrightarrow t_2) \to (\mathsf{evalB}\ c \circ \mathsf{eval}\ c) \sim \mathsf{id}$

We have a syntactic reverse operator:

$! : \{t_1\ t_2 : \mathsf{U}\} \to (t_1 \leftrightarrow t_2) \to (t_2 \leftrightarrow t_1)$

$!\ \mathsf{unite}_+ \quad = \mathsf{uniti}_+$

$!\ \mathsf{uniti}_+ \quad = \mathsf{unite}_+$

$!\ \mathsf{swap}_+ \quad = \mathsf{swap}_+$

$!\ \mathsf{assocl}_+ \quad = \mathsf{assocr}_+$

$!\ \mathsf{assocr}_+ \quad = \mathsf{assocl}_+$

...

which behaves as expected:

$!{\sim}\mathsf{B} : \forall\ \{t_1\ t_2\} \to (c : t_1 \leftrightarrow t_2) \to \mathsf{eval}\ (!\ c) \sim \mathsf{evalB}\ c$

Good properties II

Every syntactic combinator induces a type equivalence:
c2equiv : $\{t_1\ t_2 : \mathsf{U}\} \to (c : t_1 \leftrightarrow t_2) \to [\![\ t_1\ ]\!] \simeq [\![\ t_2\ ]\!]$

which also behaves nicely:

lemma0 : $\{t_1\ t_2 : \mathsf{U}\} \to (c : t_1 \leftrightarrow t_2) \to$
eval $c \sim \mathsf{proj}_1$ (c2equiv c)

lemma1 : $\{t_1\ t_2 : \mathsf{U}\} \to (c : t_1 \leftrightarrow t_2) \to$
evalB $c \sim \mathsf{proj}_1$ (sym≃ (c2equiv c))

Good properties II

Every syntactic combinator induces a type equivalence:

$\text{c2equiv} : \{t_1\ t_2 : \mathsf{U}\} \to (c : t_1 \leftrightarrow t_2) \to [\![\ t_1\ ]\!] \simeq [\![\ t_2\ ]\!]$

which also behaves nicely:

$$\begin{aligned}\text{lemma0} :\ & \{t_1\ t_2 : \mathsf{U}\} \to (c : t_1 \leftrightarrow t_2) \to \\ & \text{eval}\ c \sim \text{proj}_1\ (\text{c2equiv}\ c)\end{aligned}$$

$$\begin{aligned}\text{lemma1} :\ & \{t_1\ t_2 : \mathsf{U}\} \to (c : t_1 \leftrightarrow t_2) \to \\ & \text{evalB}\ c \sim \text{proj}_1\ (\text{sym}{\simeq}\ (\text{c2equiv}\ c))\end{aligned}$$

A foundational PL design method:

1. figure out a good denotational semantics,
2. derive a syntax for it,
3. then make sure you can implement it!

Semantics II: $\mathbb{B}$ as a Model

A simple and adequate model: finite sets and bijections (and its decategorification):

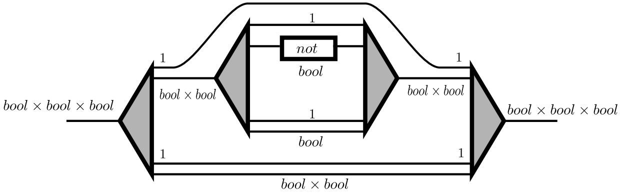
- A type is interpreted as a finite set (i.e., a natural number)
- 0 is the empty set (the number 0)
- 1 is a singleton set (the number 1)
- $+$ is disjoint union (addition of natural numbers)
- $\times$ is cartesian product (multiplication of natural numbers)
- Combinators are permutations (algebraic identities of natural numbers)

Fredkin and Toffoli Gates

- Universal gates for reversible combinational circuits
- Toffoli gate:

  BOOL BOOL^2 : U
  BOOL = PLUS ONE ONE
  BOOL^2 = TIMES BOOL BOOL

  Toffoli : TIMES BOOL $\text{BOOL}^2 \leftrightarrow$ TIMES BOOL BOOL^2
  Toffoli = dist $\odot$ (id$\leftrightarrow$ $\oplus$ (id$\leftrightarrow$ $\otimes$ cnot)) $\odot$ factor
  where
  cnot : $\text{BOOL}^2 \leftrightarrow \text{BOOL}^2$
  cnot = dist $\odot$ (id$\leftrightarrow$ $\oplus$ (id$\leftrightarrow$ $\otimes$ swap_+)) $\odot$ factor



Proofs

Now we can write down the two proofs that $A + A = A + A$:

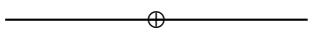
$\mathrm{pf}_1\pi\ \mathrm{pf}_2\pi : \{A : \mathrm{U}\} \rightarrow \mathrm{PLUS}\ A\ A \leftrightarrow \mathrm{PLUS}\ A\ A$
$\mathrm{pf}_1\pi = \mathrm{id}{\leftrightarrow}$
$\mathrm{pf}_2\pi = \mathrm{swap}_+$

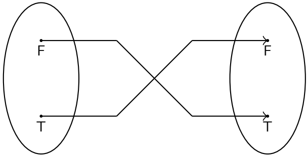
But also other proofs of semiring equalities:

$\mathrm{pf}_3\pi\ \mathrm{pf}_4\pi : \{A\ B : \mathrm{U}\} \rightarrow \mathrm{PLUS}\ (\mathrm{PLUS}\ A\ \mathrm{ZERO})\ B \leftrightarrow \mathrm{PLUS}\ A\ B$
$\mathrm{pf}_3\pi = (\mathrm{swap}_+ \oplus \mathrm{id}{\leftrightarrow}) \odot (\mathrm{unite}_+ \oplus \mathrm{id}{\leftrightarrow})$
$\mathrm{pf}_4\pi = \mathrm{assocr}_+ \odot (\mathrm{id}{\leftrightarrow} \oplus \mathrm{unite}_+)$

Claim: $\mathrm{pf}_1\pi$ and $\mathrm{pf}_2\pi$ are different, while $\mathrm{pf}_3\pi$ and $\mathrm{pf}_4\pi$ are equivalent.

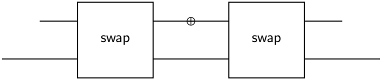
Visual example: Simple Negation

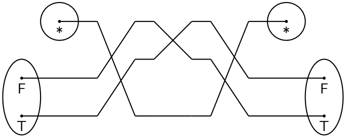




$n_1 : \mathrm{BOOL} \leftrightarrow \mathrm{BOOL}$
$n_1 = \mathrm{swap}_+$

Visual example: Not So Simple Negation

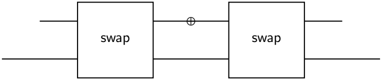


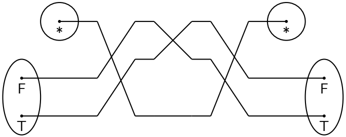


$n_2 : \text{BOOL} \leftrightarrow \text{BOOL}$

$n_2 = \;\text{uniti}\star \odot$
$\quad\;\text{swap}\star \odot$
$\quad\;(\text{swap}_+ \otimes \text{id}\leftrightarrow) \odot$
$\quad\;\text{swap}\star \odot$
$\quad\;\text{unite}\star$

Visual example: Not So Simple Negation





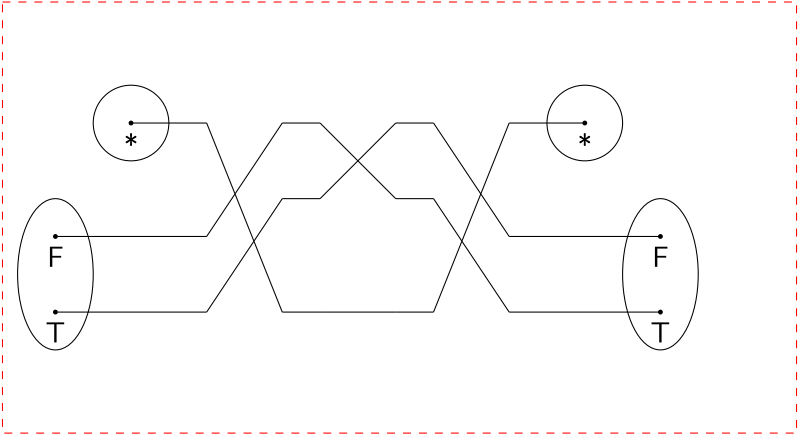
$\mathsf{n}_2 : \mathsf{BOOL} \leftrightarrow \mathsf{BOOL}$

$\mathsf{n}_2 = \ \mathsf{uniti}\star \odot$
$\quad \mathsf{swap}\star \odot$
$\quad (\mathsf{swap}_+ \otimes \mathsf{id}\!\leftrightarrow) \odot$
$\quad \mathsf{swap}\star \odot$
$\quad \mathsf{unite}\star$

Question: are these equivalent combinators?

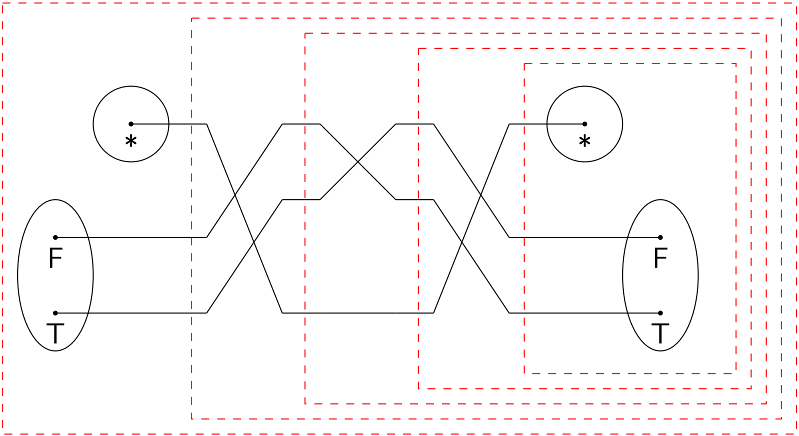
Visual “proof”

Original circuit:



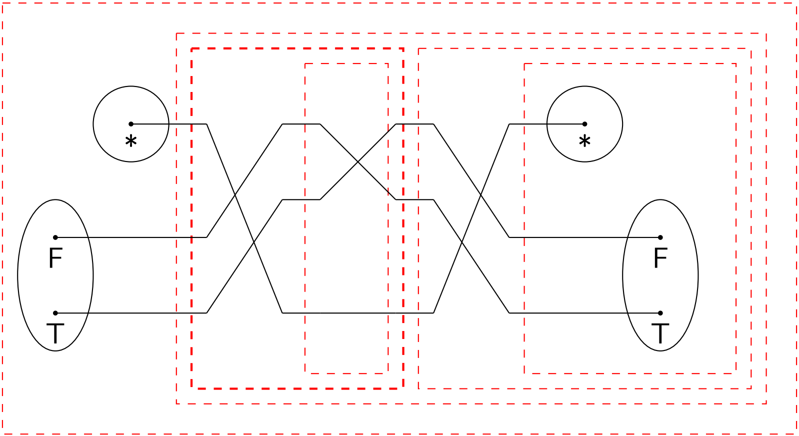
Visual "proof"

Making grouping explicit:



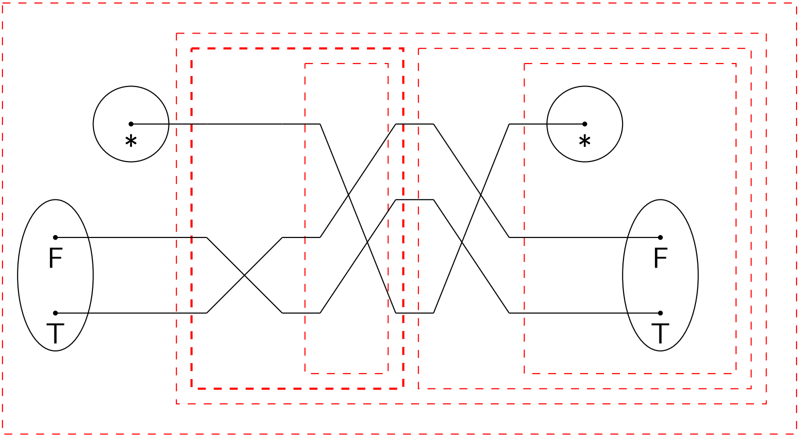
Visual "proof"

By associativity:



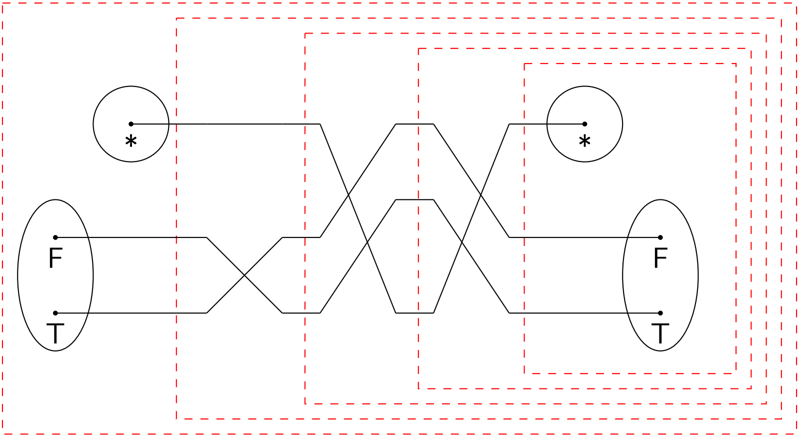
Visual "proof"

By pre-post-swap:



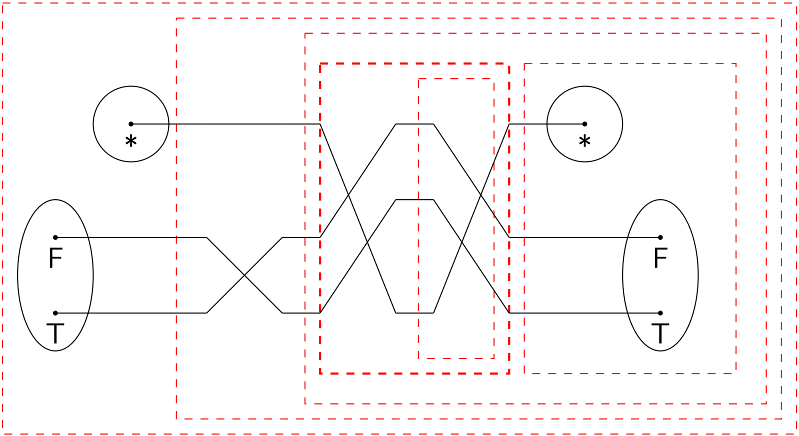
Visual "proof"

By associativity:



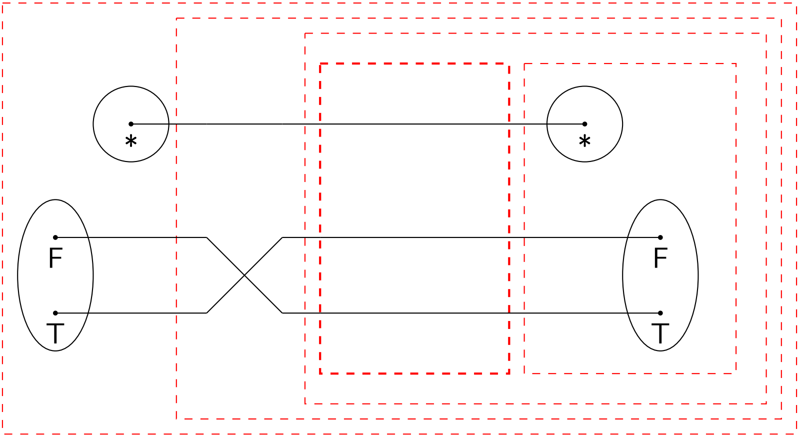
Visual “proof”

By associativity:



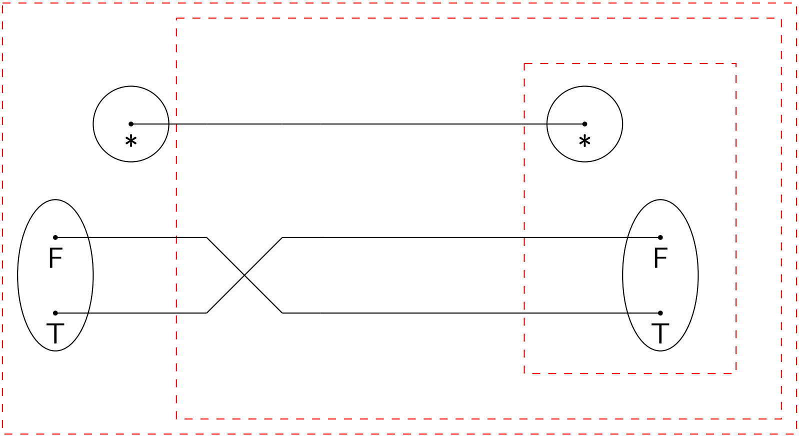
Visual "proof"

By swap-swap:



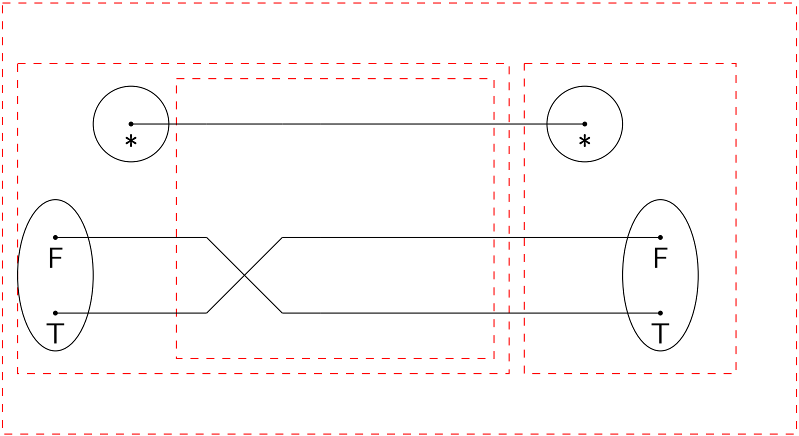
Visual "proof"

By id-compose-left:



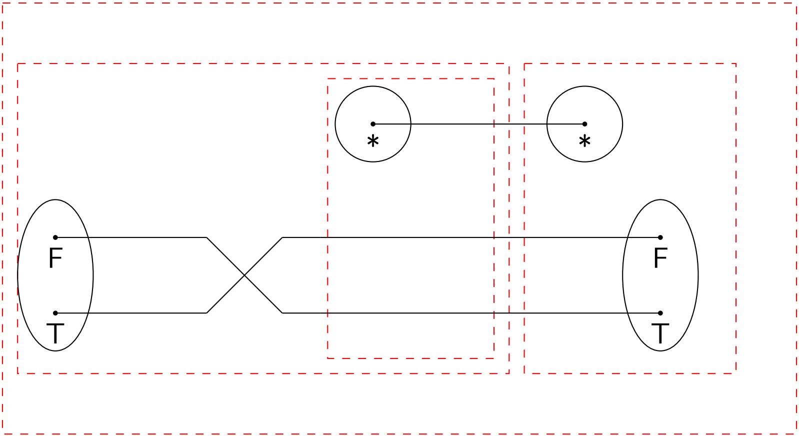
Visual "proof"

By associativity:



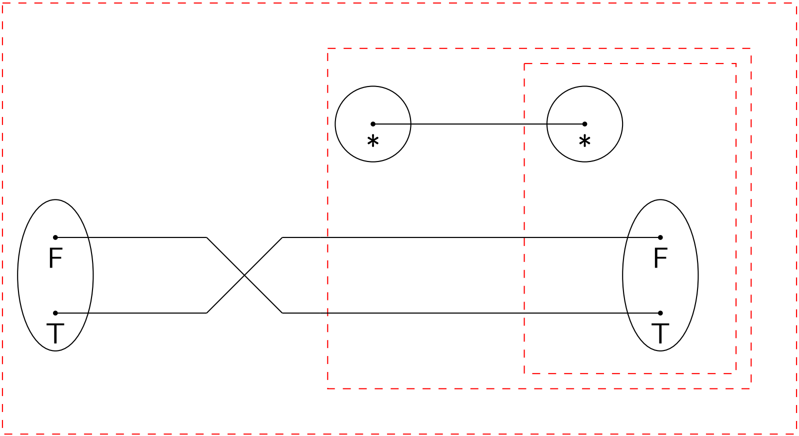
Visual “proof”

By swap-unit:



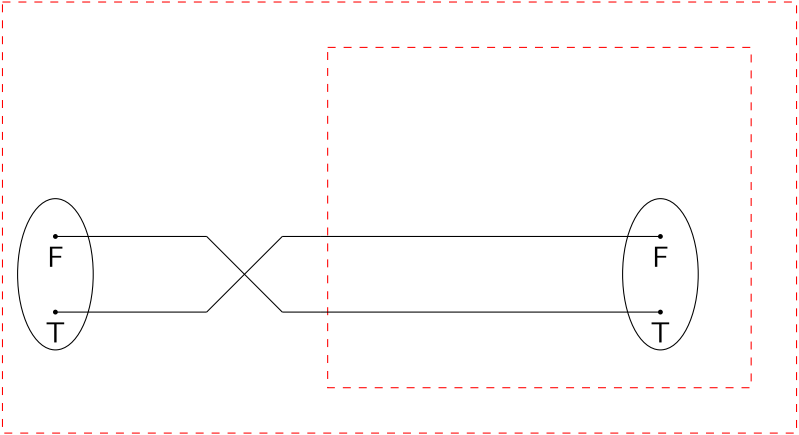
Visual "proof"

By associativity:



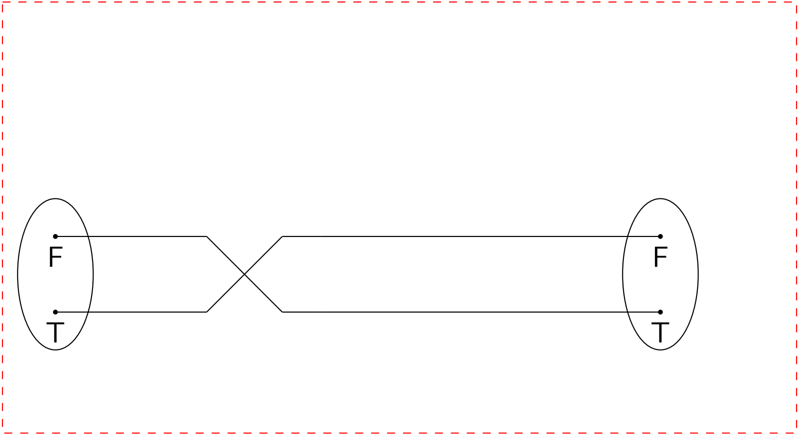
Visual “proof”

By unit-unit:



Visual "proof"

By id-unit-right:



Algebraic reasoning

```
negEx : n₂ ⇔ n₁
negEx = uniti⋆ ⊙ (swap⋆ ⊙ ((swap₊ ⊗ id⟷) ⊙ (swap⋆ ⊙ unite⋆)))
    ⇔⟨ id⇔ ⊡ assoc⊙l ⟩
  uniti⋆ ⊙ ((swap⋆ ⊙ (swap₊ ⊗ id⟷)) ⊙ (swap⋆ ⊙ unite⋆))
    ⇔⟨ id⇔ ⊡ (swapl⋆⇔ ⊡ id⇔) ⟩
  uniti⋆ ⊙ (((id⟷ ⊗ swap₊) ⊙ swap⋆) ⊙ (swap⋆ ⊙ unite⋆))
    ⇔⟨ id⇔ ⊡ assoc⊙r ⟩
  uniti⋆ ⊙ ((id⟷ ⊗ swap₊) ⊙ (swap⋆ ⊙ (swap⋆ ⊙ unite⋆)))
    ⇔⟨ id⇔ ⊡ (id⇔ ⊡ assoc⊙l) ⟩
  uniti⋆ ⊙ ((id⟷ ⊗ swap₊) ⊙ ((swap⋆ ⊙ swap⋆) ⊙ unite⋆))
    ⇔⟨ id⇔ ⊡ (id⇔ ⊡ (linv⊙l ⊡ id⇔)) ⟩
  uniti⋆ ⊙ ((id⟷ ⊗ swap₊) ⊙ (id⟷ ⊙ unite⋆))
    ⇔⟨ id⇔ ⊡ (id⇔ ⊡ idl⊙l) ⟩
  uniti⋆ ⊙ ((id⟷ ⊗ swap₊) ⊙ unite⋆)
    ⇔⟨ assoc⊙l ⟩
  (uniti⋆ ⊙ (id⟷ ⊗ swap₊)) ⊙ unite⋆
    ⇔⟨ unitil⋆⇔ ⊡ id⇔ ⟩
  (swap₊ ⊙ uniti⋆) ⊙ unite⋆
    ⇔⟨ assoc⊙r ⟩
  swap₊ ⊙ (uniti⋆ ⊙ unite⋆)
    ⇔⟨ id⇔ ⊡ linv⊙l ⟩
  swap₊ ⊙ id⟷
    ⇔⟨ idr⊙l ⟩
  swap₊ □
```

Equivalence of Equivalences

Definition 4 (Equivalence of equivalences).

Two equivalences $eq_1, eq_2 : A \simeq B$ are themselves equivalent, written $eq_2 \approxeq eq_2$, if the forward and backward functions underlying eq_1 and eq_2 are homotopic.

```
record _≋_ {A B : Set} (eq₁ eq₂ : A ≃ B) : Set where
  constructor eq
  open isqinv
  field
    f≡ : proj₁ eq₁ ∼ proj₁ eq₂
    g≡ : g (proj₂ eq₁) ∼ g (proj₂ eq₂)
```

Categorification

Theorem 5.

Taking Set as objects, $\simeq$ for homomorphisms and $\approx$ as equivalence of $\simeq$ gives a Groupoid.

Categorification

Theorem 5.

Taking Set as objects, $\simeq$ for homomorphisms and $\approx$ as equivalence of $\simeq$ gives a Groupoid.

- identity equivalence exists $id_{\simeq}$
- equivalences compose $\boxdot$
- Coherence Laws require the identity equivalence to be a left and right unit for composition and composition to be associative

$$
\begin{array}{rcl}
id_{\simeq} \boxdot E & \approx & E \\
E \boxdot id_{\simeq} & \approx & E \\
E_1 \boxdot (E_2 \boxdot E_3) & \approx & (E_1 \boxdot E_2) \boxdot E_3
\end{array}
$$

Categorification

Theorem 5.

Taking Set as objects, $\simeq$ for homomorphisms and $\approx$ as equivalence of $\simeq$ gives a Groupoid.

- identity equivalence exists $id_{\simeq}$
- equivalences compose $\boxdot$
- Coherence Laws require the identity equivalence to be a left and right unit for composition and composition to be associative

$$
\begin{array}{rcl}
id_{\simeq} \boxdot E & \approx & E \\
E \boxdot id_{\simeq} & \approx & E \\
E_1 \boxdot (E_2 \boxdot E_3) & \approx & (E_1 \boxdot E_2) \boxdot E_3
\end{array}
$$

And that is the source of our next level combinators

Categorification II

The category of finite types and equivalences is much richer than a plain category:

- every morphism has an inverse; it is a groupoid
- $\oplus, \bot$ is a commutative monoid; category is symmetric monoidal
- $\otimes, \top$ is another commutative monoid; category is symmetric bi-monoidal
- the multiplicative monoid distributes over the additive one; it is a rig groupoid
- equivalence of morphisms is up to $\approx$; category is a weak rig groupoid
- in fact, a weak rig groupoid bicategory !

Categorification III

Each coherence law is an equivalence of equivalences, which is an equivalence of combinational circuits, which must be an equivalence of Π programs.

- The full set of coherence laws is huge ($56 + 1$ laws)
- Add them to Π, level 2
- Double them to give syntactic reversibility ($2 \cdot 56 + 1 = 113$ laws)
- Can write Π programs (now level 1) and can write level 2 programs that manipulate level 1 programs in semantically-preserving ways!

Π level 2

Let $c_1 : t_1 \leftrightarrow t_2$, $c_2 : t_2 \leftrightarrow t_3$, and $c_3 : t_3 \leftrightarrow t_4$:

$$c_1 \odot (c_2 \odot c_3) \Leftrightarrow (c_1 \odot c_2) \odot c_3$$
$$(c_1 \oplus (c_2 \oplus c_3)) \odot \mathit{assocl}_+ \Leftrightarrow \mathit{assocl}_+ \odot ((c_1 \oplus c_2) \oplus c_3)$$
$$(c_1 \otimes (c_2 \otimes c_3)) \odot \mathit{assocl}_* \Leftrightarrow \mathit{assocl}_* \odot ((c_1 \otimes c_2) \otimes c_3)$$
$$((c_1 \oplus c_2) \oplus c_3) \odot \mathit{assocr}_+ \Leftrightarrow \mathit{assocr}_+ \odot (c_1 \oplus (c_2 \oplus c_3))$$
$$((c_1 \otimes c_2) \otimes c_3) \odot \mathit{assocr}_* \Leftrightarrow \mathit{assocr}_* \odot (c_1 \otimes (c_2 \otimes c_3))$$
$$\mathit{assocr}_+ \odot \mathit{assocr}_+ \Leftrightarrow ((\mathit{assocr}_+ \oplus \mathit{id}) \odot \mathit{assocr}_+) \odot (\mathit{id} \oplus \mathit{assocr}_+)$$
$$\mathit{assocr}_* \odot \mathit{assocr}_* \Leftrightarrow ((\mathit{assocr}_* \otimes \mathit{id}) \odot \mathit{assocr}_*) \odot (\mathit{id} \otimes \mathit{assocr}_*)$$

$$((a \oplus b) \otimes c) \odot \mathit{dist} \Leftrightarrow \mathit{dist} \odot ((a \otimes c) \oplus (b \otimes c))$$
$$(a \otimes (b \oplus c)) \odot \mathit{distl} \Leftrightarrow \mathit{distl} \odot ((a \otimes b) \oplus (a \otimes c))$$
$$((a \otimes c) \oplus (b \otimes c)) \odot \mathit{factor} \Leftrightarrow \mathit{factor} \odot ((a \oplus b) \otimes c)$$
$$((a \otimes b) \oplus (a \otimes c)) \odot \mathit{factorl} \Leftrightarrow \mathit{factorl} \odot (a \otimes (b \oplus c))$$

Let $c, c_1, c_2, c_3 : t_1 \leftrightarrow t_2$ and $c', c'' : t_3 \leftrightarrow t_4$:

$$\mathit{id} \odot c \Leftrightarrow c \qquad c \odot \mathit{id} \Leftrightarrow c \qquad c \odot !c \Leftrightarrow \mathit{id} \qquad !c \odot c \Leftrightarrow \mathit{id}$$
$$c \Leftrightarrow c \qquad \dfrac{c_1 \Leftrightarrow c_2 \quad c_2 \Leftrightarrow c_3}{c_1 \Leftrightarrow c_3} \qquad \dfrac{c_1 \Leftrightarrow c' \quad c_2 \Leftrightarrow c''}{c_1 \odot c_2 \Leftrightarrow c' \odot c''}$$

Let $c_0 : 0 \leftrightarrow 0$, $c_1 : 1 \leftrightarrow 1$, and $c : t_1 \leftrightarrow t_2$:

$$\mathit{identl}_+ \odot c \Leftrightarrow (c_0 \oplus c) \odot \mathit{identl}_+ \qquad \mathit{identr}_+ \odot (c_0 \oplus c) \Leftrightarrow c \odot \mathit{identr}_+$$
$$\mathit{unite}_+ r \odot c \Leftrightarrow (c \oplus c_0) \odot \mathit{unite}_+ r \qquad \mathit{uniti}_+ r \odot (c \oplus c_0) \Leftrightarrow c \odot \mathit{uniti}_+ r$$
$$\mathit{identl}_* \odot c \Leftrightarrow (c_1 \otimes c) \odot \mathit{identl}_* \qquad \mathit{identr}_* \odot (c_1 \otimes c) \Leftrightarrow c \odot \mathit{identr}_+$$
$$\mathit{unite}_* r \odot c \Leftrightarrow (c \otimes c_1) \odot \mathit{unite}_* r \qquad \mathit{uniti}_* r \odot (c \otimes c_1) \Leftrightarrow c \odot \mathit{uniti}_* r$$
$$\mathit{identl}_* \Leftrightarrow \mathit{distl} \odot (\mathit{identl}_* \oplus \mathit{identl}_*)$$
$$\mathit{identl}_+ \Leftrightarrow \mathit{swap}_+ \odot \mathit{uniti}_+ r \qquad \mathit{identl}_* \Leftrightarrow \mathit{swap}_* \odot \mathit{uniti}_* r$$
$$(\mathit{id} \otimes \mathit{swap}_+) \odot \mathit{distl} \quad \Leftrightarrow \quad \mathit{distl} \odot \mathit{swap}_+$$
$$\mathit{dist} \odot (\mathit{swap}_* \oplus \mathit{swap}_*) \quad \Leftrightarrow \quad \mathit{swap}_* \odot \mathit{distl}$$

Let $c_1 : t_1 \leftrightarrow t_2$ and $c_2 : t_3 \leftrightarrow t_4$:

$$\mathit{swap}_+ \odot (c_1 \oplus c_2) \Leftrightarrow (c_2 \oplus c_1) \odot \mathit{swap}_+ \qquad \mathit{swap}_* \odot (c_1 \otimes c_2) \Leftrightarrow (c_2 \otimes c_1) \odot \mathit{swap}_*$$
$$(\mathit{assocr}_+ \odot \mathit{swap}_+) \odot \mathit{assocr}_+ \Leftrightarrow ((\mathit{swap}_+ \oplus \mathit{id}) \odot \mathit{assocr}_+) \odot (\mathit{id} \oplus \mathit{swap}_+)$$
$$(\mathit{assocl}_+ \odot \mathit{swap}_+) \odot \mathit{assocl}_+ \Leftrightarrow ((\mathit{id} \oplus \mathit{swap}_+) \odot \mathit{assocl}_+) \odot (\mathit{swap}_+ \oplus \mathit{id})$$
$$(\mathit{assocr}_* \odot \mathit{swap}_*) \odot \mathit{assocr}_* \Leftrightarrow ((\mathit{swap}_* \otimes \mathit{id}) \odot \mathit{assocr}_*) \odot (\mathit{id} \otimes \mathit{swap}_*)$$
$$(\mathit{assocl}_* \odot \mathit{swap}_*) \odot \mathit{assocl}_* \Leftrightarrow ((\mathit{id} \otimes \mathit{swap}_*) \odot \mathit{assocl}_*) \odot (\mathit{swap}_* \otimes \mathit{id})$$

Let $c_1 : t_1 \leftrightarrow t_2$, $c_2 : t_3 \leftrightarrow t_4$, $c_3 : t_1 \leftrightarrow t_2$, and $c_4 : t_3 \leftrightarrow t_4$.
Let $a_1 : t_5 \leftrightarrow t_1$, $a_2 : t_6 \leftrightarrow t_2$, $a_3 : t_1 \leftrightarrow t_3$, and $a_4 : t_2 \leftrightarrow t_4$.

$$\dfrac{c_1 \Leftrightarrow c_3 \quad c_2 \Leftrightarrow c_4}{c_1 \oplus c_2 \Leftrightarrow c_3 \oplus c_4} \qquad \dfrac{c_1 \Leftrightarrow c_3 \quad c_2 \Leftrightarrow c_4}{c_1 \otimes c_2 \Leftrightarrow c_3 \otimes c_4}$$
$$\mathit{id} \oplus \mathit{id} \Leftrightarrow \mathit{id} \qquad \mathit{id} \otimes \mathit{id} \Leftrightarrow \mathit{id}$$
$$(a_1 \odot a_3) \oplus (a_2 \odot a_4) \Leftrightarrow (a_1 \oplus a_2) \odot (a_3 \oplus a_4)$$
$$(a_1 \odot a_3) \otimes (a_2 \odot a_4) \Leftrightarrow (a_1 \otimes a_2) \odot (a_3 \otimes a_4)$$
$$\mathit{unite}_+ r \oplus \mathit{id} \quad \Leftrightarrow \quad \mathit{assocr}_+ \odot (\mathit{id} \oplus \mathit{identl}_+)$$
$$\mathit{unite}_* r \otimes \mathit{id} \quad \Leftrightarrow \quad \mathit{assocr}_* \odot (\mathit{id} \otimes \mathit{identl}_*)$$

Let $c : t_1 \leftrightarrow t_2$:

$$(c \otimes \mathit{id}) \odot \mathit{absorbl} \Leftrightarrow \mathit{absorbl} \odot \mathit{id} \qquad (\mathit{id} \otimes c) \odot \mathit{absorbr} \Leftrightarrow \mathit{absorbr} \odot \mathit{id}$$
$$\mathit{id} \odot \mathit{factorl}_0 \Leftrightarrow \mathit{factorl}_0 \odot (\mathit{id} \otimes c) \qquad \mathit{id} \odot \mathit{factorzr} \Leftrightarrow \mathit{factorzr} \odot (c \otimes \mathit{id})$$
$$\mathit{absorbr} \Leftrightarrow \mathit{absorbl}$$
$$\mathit{absorbr} \Leftrightarrow (\mathit{distl} \odot (\mathit{absorbr} \oplus \mathit{absorbr})) \odot \mathit{identl}_+$$
$$\mathit{unite}_* r \Leftrightarrow \mathit{absorbr} \qquad \mathit{absorbl} \Leftrightarrow \mathit{swap}_* \odot \mathit{absorbr}$$
$$\mathit{absorbr} \Leftrightarrow (\mathit{assocl}_* \odot (\mathit{absorbr} \otimes \mathit{id})) \odot \mathit{absorbr}$$
$$(\mathit{id} \otimes \mathit{absorbr}) \odot \mathit{absorbl} \Leftrightarrow (\mathit{assocl}_* \odot (\mathit{absorbl} \otimes \mathit{id})) \odot \mathit{absorbr}$$
$$\mathit{id} \otimes \mathit{identl}_+ \Leftrightarrow (\mathit{distl} \odot (\mathit{absorbl} \oplus \mathit{id})) \odot \mathit{identl}_+$$
$$((\mathit{assocl}_+ \otimes \mathit{id}) \odot \mathit{dist}) \odot (\mathit{dist} \oplus \mathit{id}) \Leftrightarrow (\mathit{dist} \odot (\mathit{id} \oplus \mathit{dist})) \odot \mathit{assocl}_+$$
$$\mathit{assocl}_* \odot \mathit{distl} \Leftrightarrow ((\mathit{id} \otimes \mathit{distl}) \odot \mathit{distl}) \odot (\mathit{assocl}_* \oplus \mathit{assocl}_*)$$
$$\begin{aligned}(\mathit{distl} \odot (\mathit{dist} \oplus \mathit{dist})) \odot \mathit{assocl}_+ \quad \Leftrightarrow \quad & \mathit{dist} \odot (\mathit{distl} \oplus \mathit{distl}) \odot \mathit{assocl}_+ \odot \\ & (\mathit{assocr}_+ \oplus \mathit{id}) \odot ((\mathit{id} \oplus \mathit{swap}_+) \oplus \mathit{id}) \odot \\ & (\mathit{assocl}_+ \oplus \mathit{id})\end{aligned}$$

Revised Curry-Howard (here)

Rig of natural numbers with 0, 1, +, and * (1+1)*(1+0)	Syntax of finite types $(\top \uplus \top) \times (\top \uplus \bot)$
Semiring identities $a + 0 = a$ Proofs	Type isomorphisms $A \uplus \bot \simeq A$ Programs (reversible)
Rig category $\Rightarrow$ Coherence laws	Programs $\Rightarrow$ Program equivalences and transformation

Revised Curry-Howard (speculation)

Algebra	Syntactic classifier "type"
Equations	Isomorphisms
Proofs	Programs (reversible)
Categorification $\Rightarrow$ Coherence laws	Programs $\Rightarrow$ Program equivalences and transformation