Representing, Manipulating and Optimizing Reversible Circuits

Jacques Carette Amr Sabry

McMaster University

Indiana University

June 11, 2015

Reversible Computing

The "obvious" intersection between quantum computing and programming languages is reversible computing.

Representing Reversible Circuits

truth table, matrix, reed muller expansion, product of cycles, decision diagram, etc.

[any easy way to reproduce Figure 4 on p.7 of Saeedi and Markov? —JC]

[important remark: these are all *Boolean* circuits! —JC]

Most important part: reversible circuits are equivalent to permutations.

A (Foundational) Syntactic Theory

Ideally, want a notation that

1. is easy to write by programmers
2. is easy to mechanically manipulate
3. can be reasoned about
4. can be optimized.

A (Foundational) Syntactic Theory

Ideally, want a notation that

1. is easy to write by programmers
2. is easy to mechanically manipulate
3. can be reasoned about
4. can be optimized.

Start with a *foundational* syntactic theory on our way there:

1. easy to explain
2. clear operational rules
3. fully justified by the semantics
4. sound and complete reasoning
5. sound and complete methods of optimization

Starting Point

Typed isomorphisms. First, a universe of (finite) types

$$
\begin{array}{l}
\mathsf{data}\ \mathsf{U} : \mathsf{Set}\ \mathsf{where} \\
\quad \mathsf{ZERO} \ : \mathsf{U} \\
\quad \mathsf{ONE} \ \ \ : \mathsf{U} \\
\quad \mathsf{PLUS} \ \ : \mathsf{U} \to \mathsf{U} \to \mathsf{U} \\
\quad \mathsf{TIMES} : \mathsf{U} \to \mathsf{U} \to \mathsf{U}
\end{array}
$$

and its interpretation

$$
\begin{array}{ll}
[\![\_]\!] : \mathsf{U} \to \mathsf{Set} & \\
[\![\ \mathsf{ZERO}\ ]\!] & = \bot \\
[\![\ \mathsf{ONE}\ ]\!] & = \top \\
[\![\ \mathsf{PLUS}\ t_1\ t_2\ ]\!] & = [\![\ t_1\ ]\!] \uplus [\![\ t_2\ ]\!] \\
[\![\ \mathsf{TIMES}\ t_1\ t_2\ ]\!] & = [\![\ t_1\ ]\!] \times [\![\ t_2\ ]\!]
\end{array}
$$

Equivalences and semirings

If we denote type equivalence by $\simeq$, then we can prove that

Theorem 1.

The collection of all types (Set) forms a commutative semiring (up to $\simeq$).

Equivalences and semirings

If we denote type equivalence by $\simeq$, then we can prove that

Theorem 1.

The collection of all types (Set) forms a commutative semiring (up to $\simeq$).

We also get

Theorem 2.

If $A \simeq \mathsf{Fin}\, m$, $B \simeq \mathsf{Fin}\, n$ and $A \simeq B$ then $m \equiv n$.

(whose *constructive* proof is quite subtle).

Theorem 3.

If $A \simeq \mathsf{Fin}\, m$ and $B \simeq \mathsf{Fin}\, n$, then the type of all equivalences $A \simeq B$ is equivalent to the type of all permutations $\mathsf{Perm}\, n$.

Equivalences and semirings II

Semiring structures abound. We can define them on:

1. equivalences (disjoint union and cartesian product)
2. permutations (disjoint union and tensor product)

Equivalences and semirings II

Semiring structures abound. We can define them on:

1. equivalences (disjoint union and cartesian product)
2. permutations (disjoint union and tensor product)

The point, of course, is that they are related:

Theorem 4.

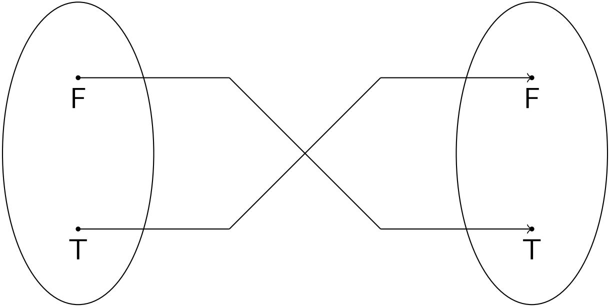
The equivalence of Theorem 3 is an isomorphism between the semirings of equivalences of finite types, and of permutations.

A Calculus of Permutations

First conclusion: it might be useful to *reify* a certain set of equivalences as combinators. We choose the fundamental "proof rules" of semirings:

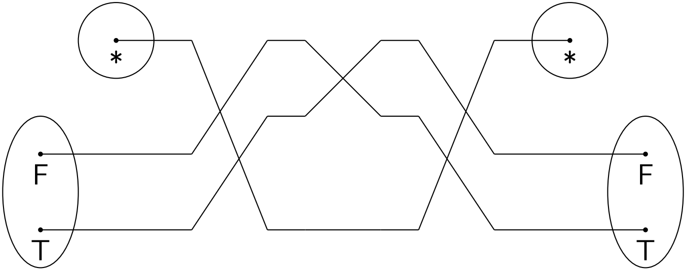
```
data _⟷_ : U → U → Set where
  unite₊  : {t : U} → PLUS ZERO t ⟷ t
  uniti₊  : {t : U} → t ⟷ PLUS ZERO t
  swap₊   : {t₁ t₂ : U} → PLUS t₁ t₂ ⟷ PLUS t₂ t₁
  assocl₊ : {t₁ t₂ t₃ : U} → PLUS t₁ (PLUS t₂ t₃) ⟷ PLUS (PLUS t₁ t₂) t₃
  assocr₊ : {t₁ t₂ t₃ : U} → PLUS (PLUS t₁ t₂) t₃ ⟷ PLUS t₁ (PLUS t₂ 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)
```

Example Circuit: Simple Negation



$$n_1 : \mathsf{BOOL} \longleftrightarrow \mathsf{BOOL}$$
$$n_1 = \mathsf{swap}_+$$

Example Circuit: Not So Simple Negation



$$
\begin{aligned}
n_2 &: \mathsf{BOOL} \longleftrightarrow \mathsf{BOOL} \\
n_2 &= \; \mathsf{uniti}\star \odot \\
&\quad\; \mathsf{swap}\star \odot \\
&\quad\; (\mathsf{swap}_+ \otimes \mathsf{id}\!\longleftrightarrow) \odot \\
&\quad\; \mathsf{swap}\star \odot \\
&\quad\; \mathsf{unite}\star
\end{aligned}
$$

Reasoning about Example Circuits

Algebraic manipulation of one circuit to the other:

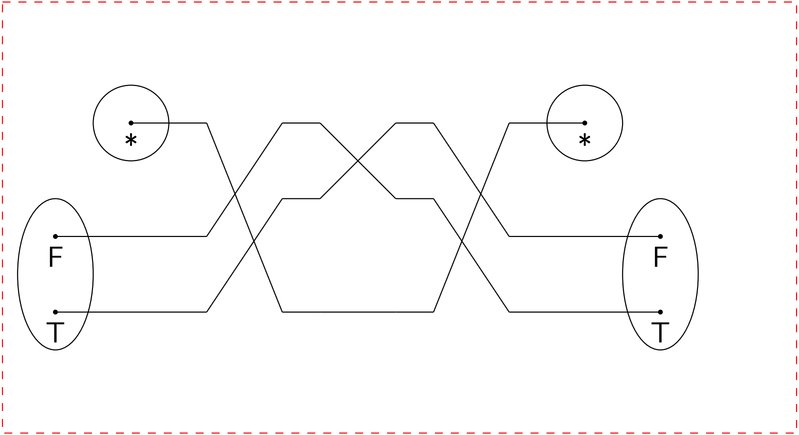
$$
\begin{array}{l}
\mathsf{negEx} : \mathsf{n}_2 \Leftrightarrow \mathsf{n}_1 \\
\mathsf{negEx} = \mathsf{uniti}\star \odot (\mathsf{swap}\star \odot ((\mathsf{swap}_+ \otimes \mathsf{id}\longleftrightarrow) \odot (\mathsf{swap}\star \odot \mathsf{unite}\star))) \\
\quad\quad \Leftrightarrow\langle\ \mathsf{id}\Leftrightarrow \boxdot \mathsf{assoc}\odot\mathsf{l}\ \rangle \\
\quad \mathsf{uniti}\star \odot ((\mathsf{swap}\star \odot (\mathsf{swap}_+ \otimes \mathsf{id}\longleftrightarrow)) \odot (\mathsf{swap}\star \odot \mathsf{unite}\star)) \\
\quad\quad \Leftrightarrow\langle\ \mathsf{id}\Leftrightarrow \boxdot (\mathsf{swapl}\star\Leftrightarrow \boxdot \mathsf{id}\Leftrightarrow)\ \rangle \\
\quad \mathsf{uniti}\star \odot (((\mathsf{id}\longleftrightarrow \otimes \mathsf{swap}_+) \odot \mathsf{swap}\star) \odot (\mathsf{swap}\star \odot \mathsf{unite}\star)) \\
\quad\quad \Leftrightarrow\langle\ \mathsf{id}\Leftrightarrow \boxdot \mathsf{assoc}\odot\mathsf{r}\ \rangle \\
\quad \mathsf{uniti}\star \odot ((\mathsf{id}\longleftrightarrow \otimes \mathsf{swap}_+) \odot (\mathsf{swap}\star \odot (\mathsf{swap}\star \odot \mathsf{unite}\star))) \\
\quad\quad \Leftrightarrow\langle\ \mathsf{id}\Leftrightarrow \boxdot (\mathsf{id}\Leftrightarrow \boxdot \mathsf{assoc}\odot\mathsf{l})\ \rangle \\
\quad \mathsf{uniti}\star \odot ((\mathsf{id}\longleftrightarrow \otimes \mathsf{swap}_+) \odot ((\mathsf{swap}\star \odot \mathsf{swap}\star) \odot \mathsf{unite}\star)) \\
\quad\quad \Leftrightarrow\langle\ \mathsf{id}\Leftrightarrow \boxdot (\mathsf{id}\Leftrightarrow \boxdot (\mathsf{linv}\odot\mathsf{l} \boxdot \mathsf{id}\Leftrightarrow))\ \rangle \\
\quad \mathsf{uniti}\star \odot ((\mathsf{id}\longleftrightarrow \otimes \mathsf{swap}_+) \odot (\mathsf{id}\longleftrightarrow \odot \mathsf{unite}\star)) \\
\quad\quad \Leftrightarrow\langle\ \mathsf{id}\Leftrightarrow \boxdot (\mathsf{id}\Leftrightarrow \boxdot \mathsf{idl}\odot\mathsf{l})\ \rangle \\
\quad \mathsf{uniti}\star \odot ((\mathsf{id}\longleftrightarrow \otimes \mathsf{swap}_+) \odot \mathsf{unite}\star) \\
\quad\quad \Leftrightarrow\langle\ \mathsf{assoc}\odot\mathsf{l}\ \rangle \\
\quad (\mathsf{uniti}\star \odot (\mathsf{id}\longleftrightarrow \otimes \mathsf{swap}_+)) \odot \mathsf{unite}\star \\
\quad\quad \Leftrightarrow\langle\ \mathsf{unitil}\star\Leftrightarrow \boxdot \mathsf{id}\Leftrightarrow\ \rangle \\
\quad (\mathsf{swap}_+ \odot \mathsf{uniti}\star) \odot \mathsf{unite}\star
\end{array}
$$

Reasoning about Example Circuits

foo

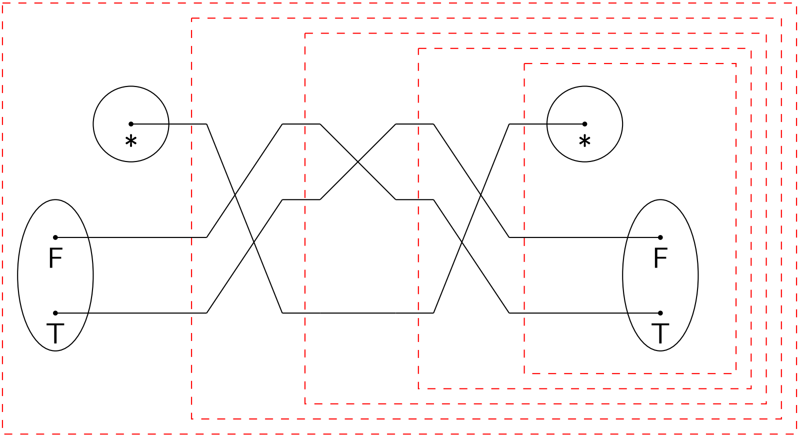
Visually

Original circuit:



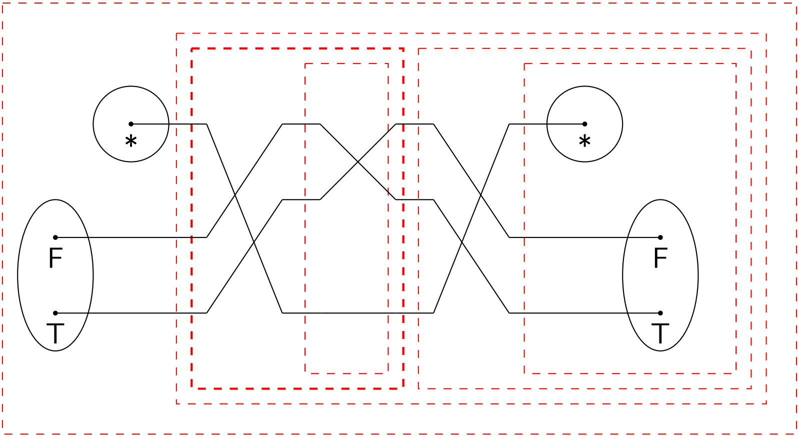
Visually

Making grouping explicit:



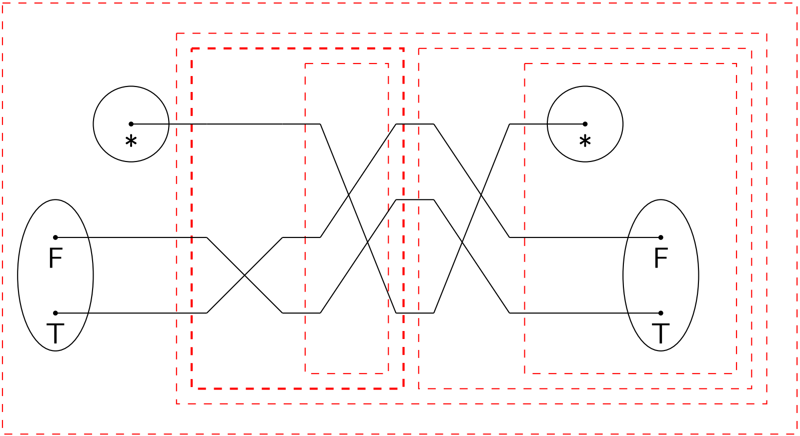
Visually

By associativity:



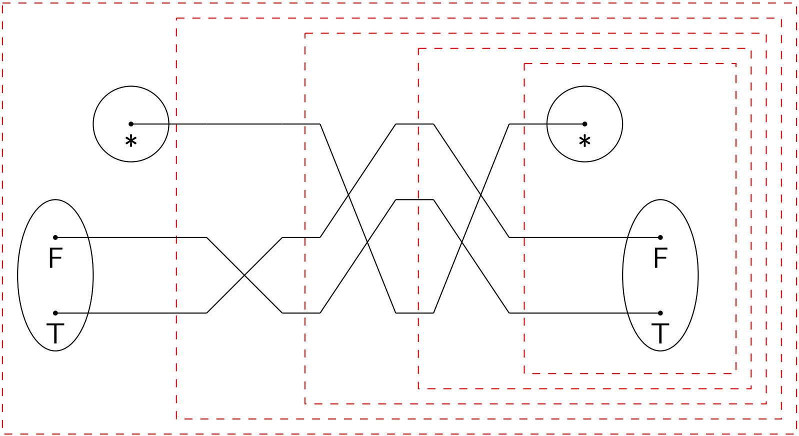
Visually

By pre-post-swap:



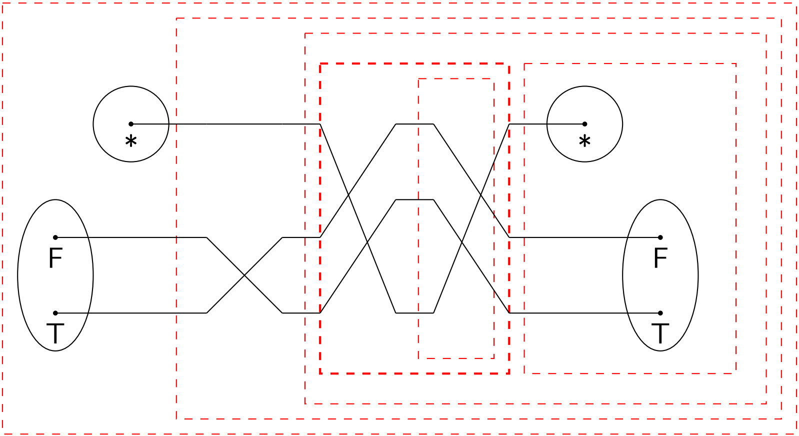
Visually

By associativity:



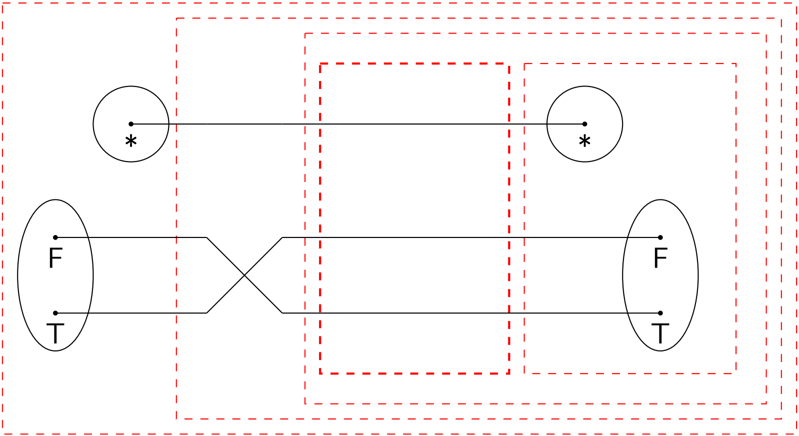
Visually

By associativity:



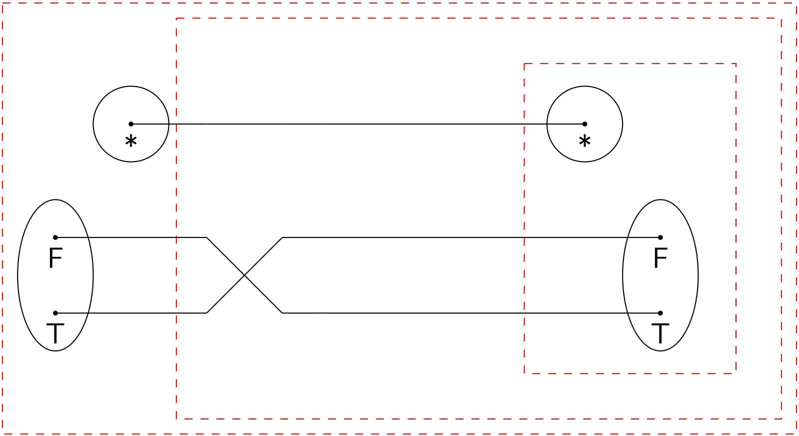
Visually

By swap-swap:



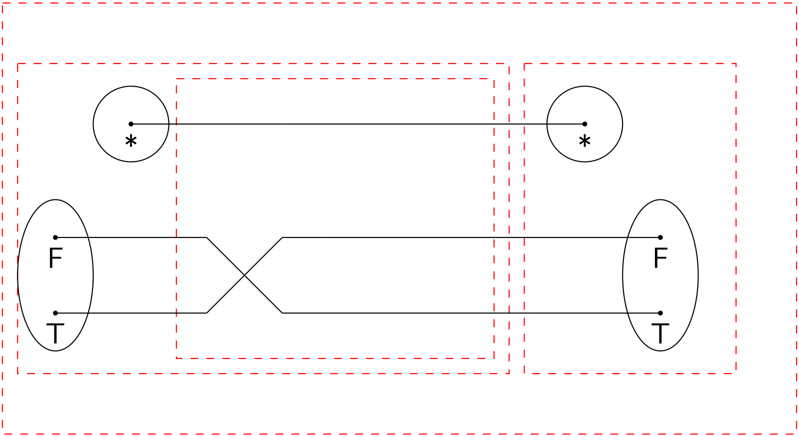
Visually

By id-compose-left:



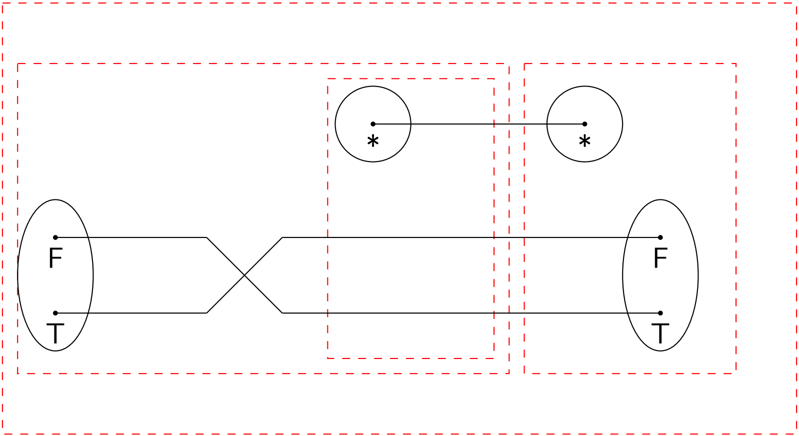
Visually

By associativity:



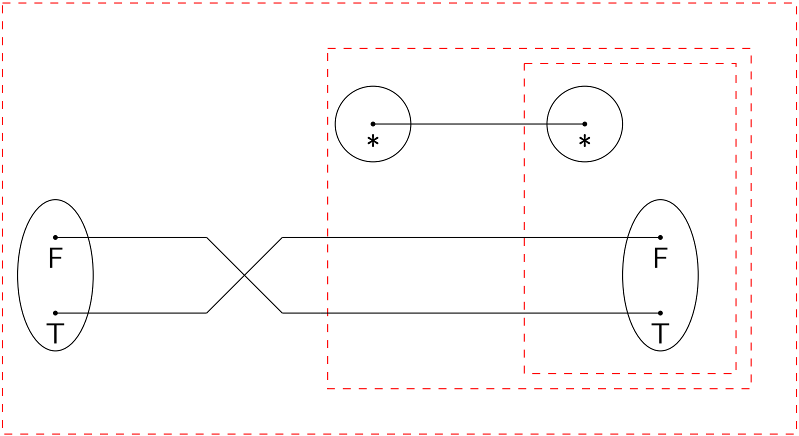
Visually

By swap-unit:



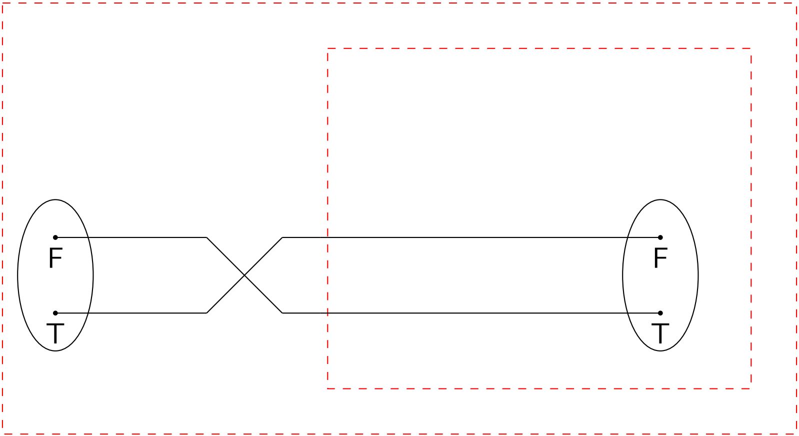
Visually

By associativity:



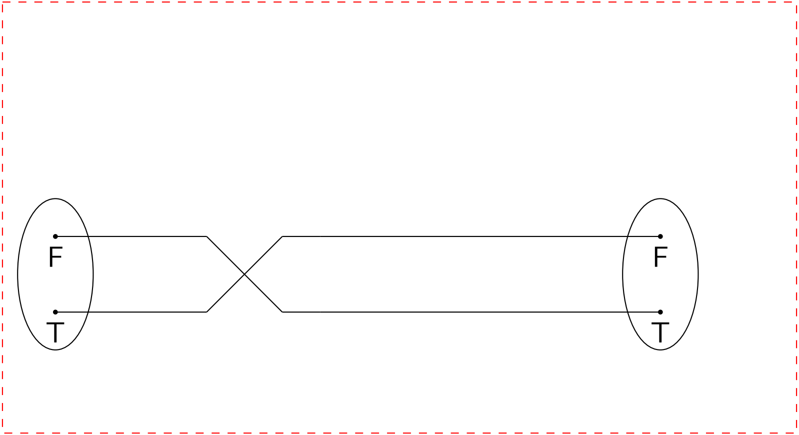
Visually

By unit-unit:



Visually

By id-unit-right:



Questions

- We don't want an ad hoc notation with ad hoc rewriting rules
- Notions of soundness; completeness; canonicity in some sense; what can we say?

1-paths vs. 2-paths

1-paths are between isomorphic types, e.g., $A * B$ and $B * A$. List them all.

1-paths vs. 2-paths

2-paths are between 1-paths, e.g.,
%endcode

1-paths vs. 2-paths