

The IFF Type Namespace

Robert E. Kent

December 18, 2007

Contents

1	The Type Kernel	2
1.1	Introduction	2
1.2	Type Sets	9
1.3	Type Functions	21
1.3.1	Function Morphisms	35
1.4	Type Spans	40
1.4.1	Type Endospans	55
1.4.2	Span Morphisms	57
1.5	Type Predicates	65
1.6	Type Relations	73
1.6.1	Type Endorelations	85

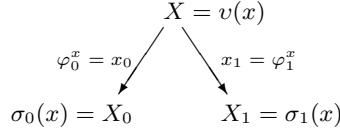


Figure 5: Span

1.4 Type Spans

Basics. A *span* is an object in the comma category

$$\text{Set} \xleftarrow{\pi_0} (\Delta \downarrow 1) \xrightarrow{\pi_1} \text{Set}^2$$

over the functorial ospan $\Delta : \text{Set} \rightarrow \text{Set}^2 \leftarrow \text{Set}^2 : 1$ with constant functor $\Delta : \text{Set} \rightarrow \text{Set}^2$. More specifically, a span is a triple $(X, (X_0, X_1), (x_0, x_1))$ consisting of a vertex set X , a set pair (X_0, X_1) , and a function pair $(x_0, x_1) : \Delta(X) \rightarrow (X_0, X_1)$. The first definition below expresses these observations; however, for simplicity, we use the second definition below, which is isomorphic.

$$\begin{aligned}
\widehat{\text{span}} &= \{(X, (X_0, X_1), (x_0, x_1)) \mid \\
&\quad X, X_0, X_1 \in \text{set}, x_0, x_1 \in \text{ftn}, \\
&\quad \partial_0(x_0) = X, \partial_1(x_0) = X_0, \partial_0(x_1) = X, \partial_1(x_1) = X_1\}, \text{ or} \\
\text{span} &= \{(x_0, x_1) \mid x_0, x_1 \in \text{ftn}, \partial_0(x_0) = \partial_0(x_1)\} \subseteq \text{ftn} \times \text{ftn}.
\end{aligned}$$

The map $\widehat{\text{span}} \rightarrow \text{span}$ is just projection; the map $\text{span} \rightarrow \widehat{\text{span}}$ is defined by $(x_0, x_1) \mapsto (\partial_0(x_0) = \partial_0(x_1), (\partial_1(x_0), \partial_1(x_1)), (x_0, x_1))$. The spans that are axiomatized here are concrete. They can be referenced as follows.

```

(type.ftn:function ?x0)
(type.ftn:function ?x1)
(= (type.ftn:source ?x0) (type.ftn:source ?x1))
(type.ftn:function ?x)
(= ?x (type.lim.prd2.obj:pairing [?x0 ?x1]))

```

We denote a span as $x = (x_0 : X_0 \leftarrow X \rightarrow X_1 : x_1)$, picture it as in Figure 5 and reference the sets X , X_0 and X_1 as *vertex*, *domain* and *codomain*. The notion of span here is the same as the notion of span in the type limit namespace. Additional functionality is specified here.

```

(iff:set span) (= span type.lim.prd2.obj:span)
(forall ((span ?x)
  (exists ((type.ftn:function ?x0) (type.ftn:function ?x1)
    (= ?x [?x0 ?x1])))
(forall ((type.ftn:function ?x0) (type.ftn:function ?x1)
  (<=> (span [?x0 ?x1])
    (= (type.ftn:source ?x0) (type.ftn:source ?x1))))

(iff:function function0) (= function0 type.lim.prd2.obj:function0)
(= (iff:source function0) span)
(= (iff:target function0) type.ftn:function)
(forall ((type.ftn:function ?x0) (type.ftn:function ?x1) (span [?x0 ?x1]))
  (= (function0 [?x0 ?x1]) ?x1))

(iff:function function1) (= function1 type.lim.prd2.obj:function1)

```

```

(= (iff:source function1) span)
(= (iff:target function1) type.ftn:function)
(forall ((type.ftn:function ?x0) (type.ftn:function ?x1) (span [?x0 ?x1]))
  (= (function1 [?x0 ?x1] ?x1))

(iff:function set) (iff:function vertex) (= vertex set) (= set type.lim.prd2.obj:set)
(= (iff:source set) span)
(= (iff:target set) type.set:set)
(forall ((span ?x))
  (and (= (set ?x) (type.ftn:source (function0 ?x)))
        (= (set ?x) (type.ftn:source (function1 ?x)))))

(iff:function set0) (= set0 type.lim.prd2.obj:set0)
(= (iff:source set0) span)
(= (iff:target set0) type.set:set)
(forall ((span ?x))
  (= (set0 ?x) (type.ftn:target (function0 ?x))))

(iff:function set1) (= set1 type.lim.prd2.obj:set1)
(= (iff:source set1) span)
(= (iff:target set1) type.set:set)
(forall ((span ?x))
  (= (set1 ?x) (type.ftn:target (function1 ?x))))

(iff:function set-pair) (= set-pair type.lim.prd2.obj:set-pair)
(= (iff:source set-pair) span)
(= (iff:target set-pair) type.dgm.pr.obj:set-pair)
(forall ((span ?x))
  (and (= (type.dgm.pr.obj:set0 (set-pair ?x)) (set0 ?x))
        (= (type.dgm.pr.obj:set1 (set-pair ?x)) (set1 ?x))))

(iff:function function-pair) (= function-pair type.lim.prd2.obj:function-pair)
(= (iff:source function-pair) span)
(= (iff:target function-pair) type.dgm.pr.mor:function-pair)
(forall ((span ?x))
  (and (= (type.dgm.pr.mor:source (function-pair ?x)) (type.dgm.pr.obj:constant (set ?x)))
        (= (type.dgm.pr.mor:target (function-pair ?x)) (set-pair ?x))
        (= (function-pair ?x) ?x)))

```

There is a binary *abridgment* relationship \preceq between pairs of type spans. One (smaller) type span $x = (x_0 : X_0 \leftarrow X \rightarrow X_1 : x_1)$ is the abridgment of another (larger) type span $y = (y_0 : Y_0 \leftarrow Y \rightarrow Y_1 : y_1)$, $x \preceq y$, when (1) the domain (codomain) of x is a subset of the domain (codomain) of y , $X_0 \subseteq Y_0$ ($Y_0 \subseteq Y_1$) (hence, the set-pair product of x is a subset of the set-pair product of y , $X_0 \times X_1 \subseteq Y_0 \times Y_1$), (2) the vertex of x is a subset of the vertex of y , $X \subseteq Y$, and (3) the function of x is the optimal restriction of the function of y , $\langle x \rangle \sqsubseteq \langle y \rangle$. We name the components of an abridgment relationship. The abridgment span is a partial order (reflexive, antisymmetric and transitive).

$$\begin{array}{ccc}
X & \xrightarrow{\langle x \rangle} & X_0 \times X_1 \\
\text{incl}_{X,Y} \downarrow & & \downarrow \text{incl}_{X_0 \times X_1, Y_0 \times Y_1} \\
Y & \xrightarrow{\langle y \rangle} & Y_0 \times Y_1
\end{array}$$

```
(iff:set abridgment-relation)
```

```

(forall ((abridgment-relation ?xy))
  (exists ((span ?x) (span ?y))
    (= (?xy [?x ?y])))
(forall ((span ?x) (span ?y))
  (<=> (abridgment ?x ?y)
    (and (type.set:subordinate [(set0 ?x) (set0 ?y)])
      (type.set:subordinate [(set1 ?x) (set1 ?y)])
      (type.set:subordinate [(vertex ?x) (vertex ?y)])
      (type.ftn:optimal-restriction-relation [(function ?x) (function ?y)]))))

(iff:function smaller)
(= (iff:source smaller) abridgment-relation)
(= (iff:target smaller) span)
(forall ((span ?x) (span ?y) (abridgment-relation [?x ?y]))
  (= (smaller [?x ?y]) ?x))

(iff:function larger)
(= (iff:source larger) abridgment-relation)
(= (iff:target larger) span)
(forall ((span ?x) (span ?y) (abridgment-relation [?x ?y]))
  (= (larger [?x ?y]) ?y))

(forall ((span ?x))
  (abridgment-relation [?x ?x]))
(forall ((span ?x) ?y (span ?y))
  (=> (and (abridgment-relation [?x ?y]) (abridgment-relation [?y ?x]))
    (= ?x ?y)))
(forall ((span ?x) (span ?y) (span ?z))
  (=> (and (abridgment-relation [?x ?y]) (abridgment-relation [?y ?z]))
    (abridgment-relation [?x ?z])))

```

Abridgment between two spans implies that the component functions satisfy restriction, $x_0 \sqsubseteq y_0$ and $x_1 \sqsubseteq y_1$.

```

(forall ((span ?x) (span ?y) (abridgment-relation [?x ?y]))
  (and (type.ftn:restriction [(function0 ?x) (function0 ?y)])
    (type.ftn:restriction [(function1 ?x) (function1 ?y)])))

```

Category Theory. Two type spans x and y are composable, and form a *composable pair*, when the codomain of the first is equal to the domain of the second $\sigma_1(x) = \sigma_0(y)$. We name the extent and component factors of the composable endorelation.

```

(iff:set composable-pair)
(forall ((composable-pair ?xy))
  (exists ((span ?x) (span ?y))
    (= ?xy [?x ?y])))
(forall ((span ?x) (span ?y))
  (<=> (composable-pair [?x ?y])
    (= (set1 ?x) (set0 ?y))))

(iff:function factor0)
(= (iff:source factor0) composable-pair)
(= (iff:target factor0) span)
(forall ((span ?x) (span ?y) (composable-pair [?x ?y]))
  (= (factor0 [?x ?y]) ?x))

(iff:function factor1)

```

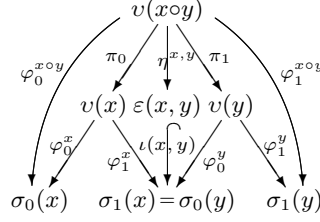


Figure 6: Composition of Spans

```
(= (iff:source factor1) composable-pair)
(= (iff:target factor1) span)
(forall ((span ?x) (span ?y) (composable-pair [?x ?y]))
  (= (factor1 [?x ?y]) ?y))
```

For any composable pair of spans (x, y) , there is an associated opspan

$$\overbrace{v(x) \xrightarrow{\varphi_1^x} \sigma_1(x) = \sigma_0(y) \xleftarrow{\varphi_0^y} v(y)}^{\Upsilon(x, y)}$$

```
(iff:function opspan)
(= (iff:source opspan) composable-pair)
(= (iff:target opspan) type.dgm.ospn.obj:opspan)
(forall ((span ?x) (span ?y) (composable-pair [?x ?y]))
  (and (= (type.dgm.ospn.obj:function0 (opspan [?x ?y])) (function1 ?x))
        (= (type.dgm.ospn.obj:function1 (opspan [?x ?y])) (function0 ?y))))
```

The *composition* of a composable pair of type spans

$$\overbrace{\sigma_0(x) \xleftarrow{\varphi_0^x} v(x) \xrightarrow{\varphi_1^x} \sigma_1(x)}^x \quad \text{and} \quad \overbrace{\sigma_0(y) \xleftarrow{\varphi_0^y} v(y) \xrightarrow{\varphi_1^y} \sigma_1(y)}^y$$

is a type span

$$\overbrace{\sigma_0(x) \xleftarrow{\varphi_0^{x \circ y}} v(x \circ y) \xrightarrow{\varphi_1^{x \circ y}} \sigma_1(y)}^{x \circ y}$$

as pictured in Figure 6. The domain of the composite is the domain of the first factor, the codomain of the composite is the codomain of the second factor, the vertex is the pullback of the associated opspan, and the component functions are composites of the pullback projections and component functions:

$$\begin{aligned} v(x \circ y) &= \lim \Upsilon(x, y) \\ \varphi_0^{x \circ y} &= \pi_0^{\Upsilon(x, y)} \cdot \varphi_0^x \\ \varphi_1^{x \circ y} &= \pi_1^{\Upsilon(x, y)} \cdot \varphi_1^y \end{aligned}$$

```
(iff:function composition)
(= (iff:source composition) composable-pair)
(= (iff:target composition) span)
(forall ((span ?x) (span ?y) (composable-pair [?x ?y]))
  (and (= (set0 (composition [?x ?y])) (set0 ?x))
        (= (set1 (composition [?x ?y])) (set1 ?y))))
```

```

(= (vertex (composition [?x ?y])) (type.lim.pbk.obj:pullback (opspan [?x ?y])))
(= (function0 (composition [?x ?y]))
   (type.ftn:composition
    [(type.lim.pbk.obj:projection0 (opspan [?x ?y])) (function0 ?x)]))
(= (function1 (composition [?x ?y]))
   (type.ftn:composition
    [(type.lim.pbk.obj:projection1 (opspan [?x ?y])) (function1 ?y)])))

```

Currently, pullbacks in the type namespace are concrete. Hence, the extent of the composition is $v(x \circ y) = \{(a, b) \mid a \in v(x), b \in v(y), \varphi_1^x(a) = \varphi_0^y(b)\}$, and the function components are $\varphi_0^{x \circ y}(a, b) = \varphi_0^x(a)$ and $\varphi_1^{x \circ y}(a, b) = \varphi_1^y(b)$.

```

(forall ((span ?x) (span ?y) (composable-pair [?x ?y]))
  (and (forall ((vertex (composition [?x ?y])) ?ab)
    (exists ((vertex ?x) ?a) ((vertex ?y) ?b)
      (= ?ab [?a ?b])))
    (forall ((vertex ?x) ?a) ((vertex ?y) ?b)
      (<=> ((vertex (composition [?x ?y])) [?a ?b])
          (= ((function1 ?x) ?a) ((function0 ?y) ?b))))))

(forall ((span ?x) (span ?y) (composable-pair [?x ?y])
  ((vertex ?x) ?a) ((vertex ?y) ?b) ((vertex (composition [?x ?y])) [?a ?b]))
  (and (= ((function0 (composition [?x ?y])) [?a ?b]) ((function0 ?x) ?a))
    (= ((function1 (composition [?x ?y])) [?a ?b]) ((function1 ?y) ?b))))

```

Composition is associative up to isomorphism.

```

(forall ((span ?x) (span ?y) (span ?z)
  (composable-pair [?x ?y]) (composable-pair ?y ?z))
  (isomorphism [(composition [?x (composition [?y ?z])]
    (composition [(composition [?x ?y]) ?z]))])

```

Composition is surjective (see identity properties below).

```

(forall ((span ?z)
  (exists ((span ?x) (span ?y) (composable-pair [?x ?y]))
    (= (composition [?x ?y]) ?z)))

```

For every type set X , there is an associated *identity* type span

$$\overbrace{X \xleftarrow{1_X} X \xrightarrow{1_X} X}^{1_X}$$

with X as domain and codomain. Its vertex is X and its component functions are $1_X : X \rightarrow X$.

```

(iff:function identity)
(= (iff:source identity) type.set:set)
(= (iff:target identity) span)
(forall ((type.set:set ?X)
  (and (= (set0 (identity ?X)) ?X)
    (= (set1 (identity ?X)) ?X)
    (= (vertex (identity ?X)) ?X)
    (= (function0 (identity ?X)) (type.ftn:identity ?X))
    (= (function1 (identity ?X)) (type.ftn:identity ?X))))

```

The identity satisfies up to isomorphism two unit laws with respect to composition.

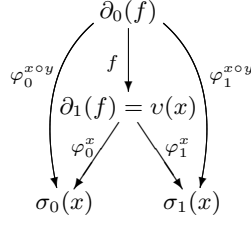


Figure 7: Combination of Function and Span

```
(forall ((span ?x))
  (and (isomorphism [(composition [(identity (set0 ?x)) ?x]) ?x])
    (isomorphism [?x (composition [?x (identity (set1 ?x))])]))))
```

Identity is injective; hence, sets can be regarded as special spans that satisfy the unit laws.

```
(forall ((type.set:set ?X0) (type.set:set ?X1)
  (= (identity ?X0) (identity ?X1)))
  (= ?X0 ?X1))
```

A type function f and a type span x are combinable, and form a *combinable pair*, when the functions $(f, \langle x \rangle)$ are composable, and form a composable pair of functions; that is, when the target of the function f is equal to the vertex of the span x , $\partial_1(f) = v(x)$. We name the extent and component factors of the combinable endorelation.

```
(iff:set combinable-pair)
(forall ((combinable-pair ?fx))
  (exists ((type.ftn:function ?f) (span ?x))
    (= ?fx [?f ?x])))
(forall ((type.ftn:function ?f) (span ?x))
  (<=> (combinable-pair [?f ?x])
    (type.ftn:composable-pair [?f (function ?x)])))
(forall ((type.ftn:function ?f) (span ?x))
  (<=> (combinable-pair [?f ?x])
    (= (type.ftn:target ?f) (vertex ?x))))

(iff:function component0)
(= (iff:source component0) combinable-pair)
(= (iff:target component0) type.ftn:function)
(forall ((type.ftn:function ?f) (span ?x) (combinable-pair [?f ?x]))
  (= (component0 [?f ?x]) ?f))

(iff:function component1)
(= (iff:source component1) combinable-pair)
(= (iff:target component1) span)
(forall ((type.ftn:function ?f) (span ?x) (combinable-pair [?f ?x]))
  (= (component1 [?f ?x]) ?x))
```

The *combination* of a combinable pair

$$\overbrace{\partial_0(f) \xrightarrow{f} \partial_1(f)}^f \quad \text{and} \quad \overbrace{\sigma_0(x) \xleftarrow{\varphi_0^x} v(x) \xrightarrow{\varphi_1^x} \sigma_1(x)}^x$$

is a type span

$$\overbrace{\sigma_0(x) \xleftarrow{\varphi_0^{f \circ x}} v(f \circ x) \xrightarrow{\varphi_1^{f \circ x}} \sigma_1(y)}^{f \circ x}$$

as pictured in Figure 7. The function of the combination is the function composition $\langle f \diamond x \rangle = f \cdot \langle x \rangle$. The domain (codomain) of the combination is the domain (codomain) of the span, the vertex of the combination is the source of the function, and the component functions of the combination are the composites of the function and component functions of the span:

$$\begin{aligned}\sigma_0(f \diamond x) &= \sigma_0(x) \\ \sigma_1(f \diamond x) &= \sigma_1(x) \\ v(f \diamond x) &= \partial_0(f) \\ \varphi_0^{f \diamond x} &= f \cdot \varphi_0^x \\ \varphi_1^{f \diamond x} &= f \cdot \varphi_1^x.\end{aligned}$$

```
(iff:function combination)
(= (iff:source combination) combinable-pair)
(= (iff:target combination) span)
(forall ((type.ftn:function ?f) (span ?x) (combinable-pair [?f ?x]))
  (and (= (function (combination [?f ?x])) (type.ftn:composition [?f (function ?x)]))
    (= (set0 (combination [?f ?x])) (set0 ?x))
    (= (set1 (combination [?f ?x])) (set1 ?x))
    (= (vertex (combination [?f ?x])) (type.ftn:source ?f))
    (= (function0 (combination [?f ?x])) (type.ftn:composition [?f (function0 ?x)]))
    (= (function1 (combination [?f ?x])) (type.ftn:composition [?f (function1 ?x)]))))
```

Combination satisfies a mixed associative law: $f_2 \diamond (f_1 \diamond x) = (f_2 \cdot f_1) \diamond x$ for composable pairs (f_2, f_1) and combinable pairs (f_1, x) , since $\langle f_2 \diamond (f_1 \diamond x) \rangle = f_2 \cdot \langle f_1 \diamond x \rangle = f_2 \cdot (f_1 \cdot \langle x \rangle) = (f_2 \cdot f_1) \cdot \langle x \rangle = \langle (f_2 \cdot f_1) \diamond x \rangle$.

```
(forall ((type.ftn:function ?f2) (type.ftn:function ?f1) (span ?x)
  (composable-pair [?f2 ?f1]) (combinable-pair [?f1 ?x]))
  (= (combination [?f2 (combination [?f1 ?x])]
    (combination [(type.ftn:composition [?f2 ?f1]) ?x])))
```

Factorization. There are special kinds of type spans. A type span $x = (x_0 : X_0 \leftarrow X \rightarrow X_1 : x_1)$ is an *injection* (*surjection*, *bijection*) when its function $\langle x \rangle : X \rightarrow X_0 \times X_1$ is an injection (surjection, bijection). Injections are essentially relational spans, and bijections are essentially binary product spans of set pairs.

```
(iff:set injection)
(forall ((injection ?x) (span ?x))
  (forall ((span ?x))
    (<=> (injection ?x)
      (type.ftn:injection (function ?x))))
(forall ((span ?x))
  (<=> (injection ?x)
    (exists (?r (type.rel:relation ?r))
      (= ?x (type.rel:span ?r)))))

(iff:set surjection)
(forall ((surjection ?x) (span ?x))
  (forall ((span ?x))
    (<=> (surjection ?x)
      (type.ftn:surjection (function ?x))))

(iff:set bijection)
(forall ((bijection ?x) (span ?x))
  (forall ((span ?x))
```

```

(<=> (bijection ?x)
      (type.ftn:bijection (function ?x)))
(forall ((span ?x))
  (<=> (bijection ?x)
        (exists (?X01 (type.dgm.pr.obj:set-pair ?X01))
          (isomorphism [?x (type.lim.prd2.obj:span ?X01)]))))

```

A span is a bijection when it is both an injection and a surjection.

```

(forall ((span ?x))
  (<=> (bijection ?x)
        (and (injection ?x) (surjection ?x))))

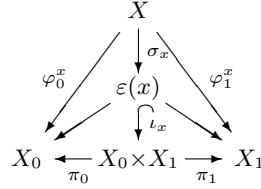
```

Injections, surjections and bijections are closed under combination with function injections, function surjections and function bijections.

```

(forall ((type.ftn:injection ?f) (injection ?x) (combinable-pair [?f ?x]))
  (injection (combination [?f ?x])))
(forall ((type.ftn:surjection ?f) (surjection ?x) (combinable-pair [?f ?x]))
  (surjection (combination [?f ?x])))
(forall ((type.ftn:bijection ?f) (bijection ?x) (combinable-pair [?f ?x]))
  (bijection (combination [?f ?x])))

```



For any type span $x = (x_0 : X_0 \leftarrow X \rightarrow X_1 : x_1)$, there is a *extent* (or *range*) type set $\varepsilon(x) = \text{ext}(x) \subseteq X_0 \times X_1$, defined as the range of the function $\langle x \rangle : X \rightarrow X_0 \times X_1$.

```

(iff:function extent) (iff:function range) (= range extent)
(= (iff:source range) span)
(= (iff:target range) type.set:set)
(= (range ?x) (type.ftn:range (function ?x)))

```

Pointwise this is $\varepsilon(x) = \{(a_0, a_1) \in X_0 \times X_1 \mid \exists a \in X \langle x \rangle(a) = (a_0, a_1)\}$.

```

(forall ((span ?x))
  (subset-relation [(range ?x) (type.lim.prd2.obj:product (set-pair ?x)])))
(forall ((span ?x) ((set0 ?x) ?a0) ((set1 ?x) ?a1))
  (<=> ((range ?x) [?a0 ?a1])
        (exists ((vertex ?x) ?a)
          (= [?a0 ?a1] ((function ?x) ?a))))

```

For any type span $x = (x_0 : X_0 \leftarrow X \rightarrow X_1 : x_1)$, the *injective factor* $\iota_x : \varepsilon(x) \rightarrow X_0 \times X_1$ (*surjective factor* $\sigma_x : X \rightarrow \varepsilon(x)$) is the injective factor (surjective factor) of the function $\langle x \rangle : X \rightarrow X_0 \times X_1$.

```

(iff:function injective-factor)
(= (iff:source injective-factor) span)
(= (iff:target injective-factor) type.ftn:function)
(forall ((span ?x))
  (and (= (type.ftn:source (injective-factor ?x)) (range ?x))
        (= (type.ftn:target (injective-factor ?x))
           (type.lim.prd2.obj:product (set-pair ?x)))))

```

```

      (type.lim.prd2.obj:product (set-pair ?x))
      (type.ftn:injection (injective-factor ?x))
      (= (injective-factor ?x) (type.ftn:injective-factor (function ?x))))

(iff:function surjective-factor)
(= (iff:source surjective-factor) span)
(= (iff:target surjective-factor) type.ftn:function)
(forall ((span ?x)
  (and (= (type.ftn:source (surjective-factor ?x)) (source ?x))
        (= (type.ftn:target (surjective-factor ?x)) (range ?x))
        (type.ftn:surjection (surjective-factor ?x))
        (= (surjective-factor ?x) (type.ftn:surjective-factor (function ?x)))))

```

Pointwise, the injective factor is the inclusion of the range into the product of the set pair of x , and the surjective factor is the restriction of the function $\langle x \rangle$ to the range.

```

(forall ((span ?x) ((set0 ?x) ?a0) ((set1 ?x) ?a1) ((range ?x) [?a0 ?a1]))
  (= ((injective-factor ?x) [?a0 ?a1]) [?a0 ?a1]))

(forall ((span ?x) ((vertex ?x) ?a))
  (= ((surjective-factor ?x) ?a) ((function ?x) ?a)))

```

The function $\langle x \rangle : X \rightarrow X_0 \times X_1$ is the composition of its surjective factor and injective factor: $\langle x \rangle = \sigma_x \cdot \iota_x : X \rightarrow \varepsilon(x) \rightarrow X_0 \times X_1$.

```

(forall ((span ?x)
  (= (function ?x)
    (type.ftn:composition [(surjective-factor ?x) (injective-factor ?x)])))

```

The (surjective-factor, injective-factor) pair forms a surjection-injection “left factorization system” for type sets, functions and type spans; that is, if the function $\langle x \rangle : X \rightarrow X_0 \times X_1$ of a type span $x = (x_0 : X_0 \leftarrow X \rightarrow X_1 : x_1)$ is the composition of a surjection with an injection, $\langle x \rangle = e \cdot m : X \rightarrow Z \rightarrow X_0 \times X_1$, then there is a (unique) “diagonal” bijection $d : \varepsilon(x) \rightarrow Z$, such that $\sigma_x \cdot d = e$ and $d \cdot m = \iota_x$.

$$\begin{array}{ccc}
 X & \xrightarrow{\sigma_x} & \varepsilon(x) \\
 e \downarrow & \swarrow d & \downarrow \iota_x \\
 Z & \xrightarrow{m} & X_0 \times X_1
 \end{array}$$

```

(forall ((span ?x) (type.ftn:surjection ?e) (type.ftn:injection ?m)
  (= (function ?x) (type.ftn:composition [?e ?m])))
  (and (exists ((type.ftn:bijection ?d)
    (= (type.ftn:source ?d) (range ?x))
    (= (type.ftn:target ?d) (type.ftn:target ?e)))
    (and (= (type.ftn:composition [(surjective-factor ?x) ?d]) ?e)
          (= (type.ftn:composition [?d ?m]) (injective-factor ?x))))
    (forall ((bijection ?d1) (bijection ?d2)
      (= (type.ftn:source ?d1) (range ?x))
      (= (type.ftn:source ?d2) (range ?x))
      (= (type.ftn:target ?d1) (type.ftn:target ?e))
      (= (type.ftn:target ?d2) (type.ftn:target ?e))
      (= (type.ftn:composition [(surjective-factor ?x) ?d1]) ?e)

```

```

      (= (type.ftn:composition [(surjective-factor ?x) ?d2]) ?e)
      (= (type.ftn:composition [?d1 ?m]) (injective-factor ?x))
      (= (type.ftn:composition [?d2 ?m]) (injective-factor ?x)))
    (= ?d1 ?d2)))

```

A surjection is a span whose injective factor is a bijection. An injection is a span whose surjective factor is a bijection. An injection is a span that is the embedding of some relation.

```

(forall ((span ?x))
  (<=> (surjection ?x)
    (type.ftn:bijection (injective-factor ?x))))

(forall ((span ?x))
  (<=> (injection ?x)
    (type.ftn:bijection (surjective-factor ?x))))

(forall ((span ?x))
  (<=> (injection ?x)
    (type.spn.mor:isomorphism [?x (type.rel:span (relation ?x))])))

(forall ((span ?x))
  (<=> (injection ?x)
    (exists (?r (type.rel:relation ?r))
      (type.spn.mor:isomorphism [?x (type.rel:span ?r)]))))

```

Let $x = (x_0 : X_0 \leftarrow X \rightarrow X_1 : x_1)$ be a type span. The *kernel* (*coimage*, *coequalizer*) of x is the kernel (coimage, coequalizer) of the function $\langle x \rangle : X \rightarrow X_0 \times X_1$ ⁹.

```

(iff:function kernel)
(= (iff:source kernel) span)
(= (iff:target kernel) type.rel:equivalence-relation)
(forall ((span ?x))
  (and (= (type.rel:set (kernel ?x)) (vertex ?x))
    (= (kernel ?x) (type.ftn:kernel (function ?x)))))

(iff:function coimage)
(= (iff:source coimage) span)
(= (iff:target coimage) type.set:set)
(forall ((span ?x))
  (= (coimage ?x) (type.ftn:coimage (function ?x))))

(iff:function coequalizer)
(= (iff:source coequalizer) span)
(= (iff:target coequalizer) type.ftn:function)
(forall ((span ?f))
  (and (= (type.ftn:source (coequalizer ?x)) (vertex ?x))
    (= (type.ftn:target (coequalizer ?x)) (coimage ?x))
    (type.ftn:surjection (coequalizer ?x))
    (= (coequalizer ?x) (type.ftn:coequalizer (function ?x)))))

```

⁹Pointwise, (1) the kernel of x is the equivalence relation $\ker_x = \equiv_x$ on the vertex set X defined by $a \equiv_x b$ iff $\langle x \rangle(a) = (x_0(a), x_1(a)) = (x_0(b), x_1(b)) = \langle x \rangle(b)$ for all vertex pairs $a, b \in X$ (hence, the kernel of the span x is the intersection $\equiv_x = \equiv_{x_0} \cap \equiv_{x_1}$ of the kernels of the component functions x_0 and x_1); (2) the coimage of x is the quotient of the kernel $\text{coim}_x = X/\equiv_x = \{ [a]_{\equiv_x} \mid a \in X \}$, where $[a]_x = \{ b \in X \mid \langle x \rangle(b) = \langle x \rangle(a) \}$ is the equivalence class of a with respect to the kernel of x ; and (3) the coequalizer of x is the canon of the kernel $\text{coeq}_x = [-]_x = [-]_{\equiv_x} : X \rightarrow X/\equiv_x$.

The function $\langle x \rangle : X \rightarrow X_0 \times X_1$ of any type span $x = (x_0 : X_0 \leftarrow X \rightarrow X_1 : x_1)$ respects its kernel $\pi_0^{\equiv_x} \cdot \langle x \rangle = \pi_1^{\equiv_x} \cdot \langle x \rangle$, and hence factors through its coimage $\langle x \rangle = [-]_x \cdot \wr_x$ for some (injective) span $\text{apply}_x = \wr_x : \text{coim}_x \rightarrow X_0 \times X_1$. This factor, called the *coapplication* of x , is defined by $\wr_f([a]_{\equiv_x}) = \langle x \rangle(a) = (x_0(a), x_1(a))$ for all $a \in X$.

$$\begin{array}{ccc} \text{ext}(\equiv_x) & \begin{array}{c} \xrightarrow{\pi_0} \\ \xrightarrow{\pi_1} \end{array} & X & \xrightarrow{\langle x \rangle} & X_0 \times X_1 \\ & & \searrow [-]_x & & \nearrow \wr_x \\ & & & & X/\equiv_x \end{array}$$

```
(iff:function coapplication)
(= (iff:source coapplication) span)
(= (iff:target coapplication) type.ftn:function)
(forall ((span ?x))
  (and (= (type.ftn:source (coapplication ?x)) (coimage ?x))
        (= (type.ftn:target (coapplication ?x)) (type.lim.prd2.obj:product (set-pair ?x)))
        (type.ftn:injection (coapplication ?x))
        (= (function ?x) (type.ftn:composition [(coequalizer ?x) (coapplication ?x)]))))
```

The (coequalizer, coapplication) pair forms a surjection-injection “left factorization system” for type sets, type functions and type spans; that is, if the function $\langle x \rangle : X \rightarrow X_0 \times X_1$ of a type span $x = (x_0 : X_0 \leftarrow X \rightarrow X_1 : x_1)$ is the composition of a surjection with an injection, $\langle x \rangle = e \cdot m : X \rightarrow Z \rightarrow X_0 \times X_1$, then there is a (unique) “diagonal” bijection $d : X/\equiv_f \rightarrow Z$ such that $[-]_x \cdot d = e$ and $d \cdot m = \wr_x$.

$$\begin{array}{ccc} X & \xrightarrow{[-]_f} & X/\equiv_f \\ e \downarrow & \swarrow d & \downarrow \wr_f \\ Z & \xrightarrow{m} & X_0 \times X_1 \end{array}$$

This implies that the coimage is naturally isomorphic to the range (image), $\text{coim}_x \cong \text{rng}_x$; specifically, for any source element of $a \in X$, the equivalence class $[a]_{\equiv_x} \in \text{coim}_x$ corresponds to the image element $\langle x \rangle(a) = (x_0(a), x_1(a)) \in \text{rng}_x$.

Conversion. Any type span

$$\overbrace{\sigma_0(x) \xleftarrow{\varphi_0^x} v(x) \xrightarrow{\varphi_1^x} \sigma_1(x)}^x$$

has an *opposite* type span

$$\overbrace{\sigma_1(x) \xleftarrow{\varphi_1^x} v(x) \xrightarrow{\varphi_0^x} \sigma_0(x)}^{x^\infty}$$

```
(iff:function opposite)
(= (iff:source opposite) span)
(= (iff:target opposite) span)
```

```

(forall ((span ?x))
  (and (= (set0 (opposite ?x)) (set1 ?x))
        (= (set1 (opposite ?x)) (set0 ?x))
        (= (vertex (opposite ?x)) (vertex ?x))
        (= (function0 (opposite ?x)) (function1 ?x))
        (= (function1 (opposite ?x)) (function0 ?x))))

```

The opposite is an pseudo-involution: $r^{\alpha\alpha} = r$, $(r \circ s)^{\alpha} \cong s^{\alpha} \circ r^{\alpha}$, and $1_X^{\alpha} = 1_X$.

```

(forall ((span ?x))
  (= (opposite (opposite ?x)) ?x))

(forall ((span ?x) (span ?y) (composable-pair [?x ?y]))
  (isomorphism [(opposite (composition [?x ?y]))
                (composition [(opposite ?y) (opposite ?x)])]))

(forall ((type.set:set ?X))
  (= (opposite (identity ?X)) (identity ?X)))

```

Any type span $x = (X_0 \xleftarrow{x_0} X \xrightarrow{x_1} X_1)$ has an associated *function* $\langle x \rangle = \langle x_0, x_1 \rangle : X \rightarrow X_0 \times X_1$, which is the product pairing of the function pair.

```

(iff:function function)
(= (iff:source function) span)
(= (iff:target function) type.ftn:function)
(forall ((span ?x))
  (and (= (type.ftn:source (function ?x)) (vertex ?x))
        (= (type.ftn:target (function ?x)) (type.lim.prd2.obj:product (set-pair ?x)))
        (= (function ?x) (type.lim.prd2.obj:pairing ?x))))

```

Any type span $x = (X_0 \xleftarrow{x_0} X \xrightarrow{x_1} X_1)$ has an associated *relation* $\text{rel}(x) = \tilde{x} : X_0 \rightarrow X_0$, whose set pair is the set pair of the span, whose extent is the extent of the span, and whose injection is the injective factor of the span.

```

(iff:function relation)
(= (iff:source relation) span)
(= (iff:target relation) type.rel:relation)
(forall ((span ?x))
  (and (= (type.rel:set0 (relation ?x)) (set0 ?x))
        (= (type.rel:set1 (relation ?x)) (set1 ?x))
        (= (type.rel:extent (relation ?x)) (extent ?x))
        (= (type.rel:function (relation ?x)) (injective-factor ?x))))

```

The relation of a span composition is the relation composition of the spans $\text{rel}(x \circ y) = \text{rel}(x) \circ \text{rel}(y)$. The relation of the span identity is the relation identity $\text{rel}(1_X) = 1_X$. The relation of the opposite of a span is the opposite of the relation of the span $\text{rel}(x^{\alpha}) = \text{rel}(x)^{\alpha}$.

```

(forall ((span ?x) (span ?y) (composable-pair [?x ?y]))
  (= (relation (composition [?x ?y]))
     (type.rel:composition [(relation ?x) (relation ?y)])))

(forall ((type.set:set ?X))
  (= (relation (identity ?X)) (type.rel:identity ?X)))

(forall ((span ?x))
  (= (relation (opposite ?x)) (type.rel:opposite (relation ?x))))

```

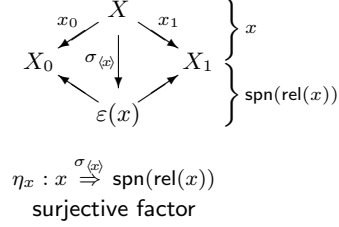


Figure 8: Unit

Any type span $x = (X_0 \xleftarrow{x_0} X \xrightarrow{x_1} X_1)$ is linked to the span of its relation by the *unit* span 2-cell $\eta_x : x \xrightarrow{\sigma(x)} \text{spn}(\text{rel}(x))$, where the function $\sigma(x) : X \rightarrow \varepsilon(x)$ is the surjective factor of x (Figure 8).

```

(iff:function unit)
(= (iff:source unit) span)
(= (iff:target unit) type.spn.mor:2-cell)
(forall ((span ?x))
  (and (= (type.spn.mor:source (unit ?r)) ?x)
        (= (type.spn.mor:target (unit ?r)) (type.rel:span (relation ?x)))
        (= (type.spn.mor:function (unit ?r)) (surjective-factor ?x))))

```

For any two spans $y = (X_0 \xleftarrow{y_0} Y \xrightarrow{y_1} X_1)$ and $z = (X_0 \xleftarrow{z_0} Z \xrightarrow{z_1} X_1)$ that share common domain and codomain sets, there is a fiber product span $y \times z = (X_0 \leftarrow Y \times_{X_0 \times X_1} Z \rightarrow X_1)$ whose vertex and projection functions are defined in terms of the pullback of the pairing functions. The pullback projections are the functions of 2-cells $\pi_0 : y \times z \xrightarrow{\pi_0} y$ and $\pi_1 : y \times z \xrightarrow{\pi_1} z$, showing that the fiber product belongs to each component: $y \times z \in y$ and $y \times z \in z$. Any other span that belongs to y and z also belongs to the fiber product $y \times z$.

```

(forall ((span ?y) (span ?z) (= (set-pair ?y) (set-pair ?z)))
  (and (= (set (product [?y ?z])) (type.lim.pbk.obj:pullback [(function ?y) (function ?z)]))
        (= (function0 (product [?y ?z]))
            (type.ftn:composition
              [(type.lim.pbk.obj:projection0 [(function ?y) (function ?z)] (function0 ?y)]))
          (= (function1 (product [?y ?z]))
              (type.ftn:composition
                [(type.lim.pbk.obj:projection1 [(function ?y) (function ?z)] (function1 ?z)]))))))
  (forall ((span ?w) (= (set-pair ?w) (set-pair ?y)))
    (=> (and (type.spn.mor:belonging [?w ?y]) (type.spn.mor:belonging [?w ?z]))
         (type.spn.mor:belonging [?w (product [?y ?z]))])))

```

Instances. For any set pair (X_0, X_1) , the empty set and counique functions form an *initial* (relational) span $0_{X_0, X_1} = (X_0 \xleftarrow{\hat{i}_{X_0}} \emptyset \xrightarrow{\hat{i}_{X_1}} X_1)$ with $\langle 0_{X_0, X_1} \rangle =$

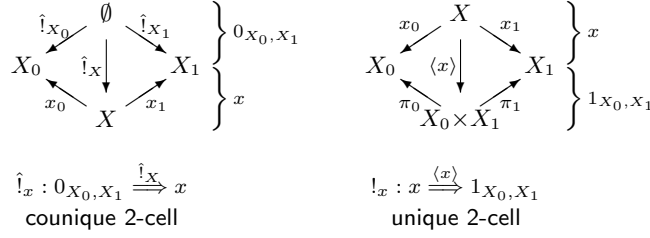


Figure 9: Counique and Unique Span 2-Cells

$0_{X_0 \times X_1}$, and the binary product and projections form a *terminal* (relational) span $1_{X_0, X_1} = (X_0 \xleftarrow{\pi_0} X_0 \times X_1 \xrightarrow{\pi_1} X_1)$ with $\langle 1_{X_0, X_1} \rangle = 1_{X_0 \times X_1}$.

```
(iff:function initial)
(= (iff:source initial) type.dgm.pr.obj:set-pair)
(= (iff:target initial) span)
(forall ((type.dgm.pr.obj:set-pair ?X01))
  (and (= (function0 (initial ?X01)) (type.set:counique (type.dgm.pr.obj:set0 ?X01)))
        (= (function1 (initial ?X01)) (type.set:counique (type.dgm.pr.obj:set1 ?X01)))
        (= (vertex (initial ?X01)) type.set:zero)
        (= (set0 (initial ?X01)) (type.dgm.pr.obj:set0 ?X01))
        (= (set1 (initial ?X01)) (type.dgm.pr.obj:set1 ?X01))
        (= (set-pair (initial ?X01)) ?X01)))

(iff:function terminal)
(= (iff:source terminal) type.dgm.pr.obj:set-pair)
(= (iff:target terminal) span)
(forall ((type.dgm.pr.obj:set-pair ?X01))
  (and (= (function0 (terminal ?X01)) (type.lim.prd2.obj:projection0 ?X01))
        (= (function1 (terminal ?X01)) (type.lim.prd2.obj:projection1 ?X01))
        (= (vertex (terminal ?X01)) (type.lim.prd2.obj:product ?X01))
        (= (set0 (terminal ?X01)) (type.dgm.pr.obj:set0 ?X01))
        (= (set1 (terminal ?X01)) (type.dgm.pr.obj:set1 ?X01))
        (= (set-pair (terminal ?X01)) ?X01)))
```

Let $x = (X_0 \xleftarrow{x_0} X \xrightarrow{x_1} X_1)$ be any type span.

- There is a *counique* type span 2-cell $\hat{!}_x : 0_{X_0, X_1} \rightrightarrows x$ (left side of Figure 9), whose source is the initial span $0_{X_0, X_1}$, whose target is x and whose function is $\hat{!}_X : \emptyset \rightarrow X$. This is the unique type span 2-cell from $0_{X_0, X_1}$ to x . Hence, $0_{X_0, X_1}$ is the initial object in the slice category $\text{Set}/(X_0, X_1)$. Here, $\langle \hat{!}_x \rangle = \hat{!}_{\langle x \rangle}$.
- There is a *unique* type span 2-cell $!_x : x \rightrightarrows 1_{X_0, X_1}$ (right side of Figure 9), whose source is x , whose target is the terminal span $1_{X_0, X_1}$ and whose function is $\langle x \rangle$. This is the unique type span 2-cell from x to $1_{X_0, X_1}$. Hence, $1_{X_0, X_1}$ is the terminal object in the slice category $\text{Set}/(X_0, X_1)$. Here, $\langle !_x \rangle = !_{\langle x \rangle}$. Note: $0_{X_0, X_1}$ is the (span of the) bottom relation in the fiber over (X_0, X_1) , and $1_{X_0, X_1}$ is the (span of the) top relation in the fiber over (X_0, X_1) .

```

(iff:function counique)
(= (iff:source counique) type.spn:span)
(= (iff:target counique) type.spn.mor:2-cell)
(forall ((span ?x))
  (and (= (type.spn.mor:source (counique ?x)) (initial (set-pair ?x)))
        (= (type.spn.mor:target (counique ?x)) ?x)
        (= (type.spn.mor:function (counique ?x)) (type.set:counique (vertex ?x))))
    (forall ((type.spn.mor:2-cell ?a)
              (= (type.spn.mor:source ?a) (initial (set-pair ?x)))
              (= (type.spn.mor:target ?a) ?x))
      (= ?a (counique ?x))))))

(iff:function unique)
(= (iff:source unique) span)
(= (iff:target unique) type.spn.mor:2-cell)
(forall ((span ?x))
  (and (= (type.spn.mor:source (unique ?x)) ?x)
        (= (type.spn.mor:target (unique ?x)) (terminal (set-pair ?x)))
        (= (type.spn.mor:function (unique ?x)) (function ?x)))
    (forall ((type.spn.mor:2-cell ?a)
              (= (type.spn.mor:source ?a) ?x)
              (= (type.spn.mor:target ?a) (terminal (set-pair ?x))))
      (= ?a (unique ?x))))))

```

1.4.1 Type Endospans

Basics. The set of all type *endospans* is an IFF set. Type endospans are special type spans, whose domain and codomain sets are identical.

```
(iff:set endospan)
(forall ((endospan ?x)) (span ?x))
(forall ((span ?x))
  (<=> (endospan ?x)
    (= (set0 ?x) (set1 ?x))))
```

There is a common component type *set*.

```
(iff:function set)
(= (iff:source set) endospan)
(= (iff:target set) type.set:set)
(forall (?x (endospan ?x))
  (and (= (set ?x) (set0 ?x))
    (= (set ?x) (set1 ?x))))
```

Order Theory. We have predicates to express the fact that a type endospan is *reflexive*, *transitive* or *symmetric*. These are predicates (adjectives) that refer to (modify) endospans. Let $x = (X \xleftarrow{\partial_0} Y \xrightarrow{\partial_1} X)$ be a type endospan.

- x is *reflexive* when $1_X \in x$; that is, when the identity endospan belongs to x ; that is, when there is a span 2-cell $1_X \xrightarrow{\iota} x$ from 1_X to x . that is, when there exists a proof function $\iota : X \rightarrow Y$ such that $\iota \cdot \partial_0 = 1_X$ and $\iota \cdot \partial_1 = 1_X$.
- x is *transitive* when $x \circ x \in x$; that is, when the composition belongs to x ; that is, when there is a span 2-cell $x \circ x \xrightarrow{\mu} x$ from $x \circ x$ to x . that is, when there exists a proof function $\mu : Y \times_X Y \rightarrow Y$ such that $\mu \cdot \partial_0 = \pi_0 \cdot \partial_0$ and $\mu \cdot \partial_1 = \pi_1 \cdot \partial_1$.
- x is *symmetric* when $x^\circ \in x$; that is, the transpose belongs to x . Since the relation conversion preserves transpose, $\text{rel}(x)$ is symmetric; that is, the relation of a symmetric span is a symmetric relation $\text{rel}(x)^\circ = \text{rel}(x^\circ) \subseteq \text{rel}(x)$.

```
(iff:set reflexive-span)
(forall ((reflexive-span ?x)) (endospan ?x))
(forall ((endospan ?x))
  (<=> (reflexive-span ?x)
    (type.spn.mor:belonging [(identity (set ?x)) ?x])))
(forall ((endospan ?x))
  (=> (reflexive-span ?x) (type.rel:reflexive-relation (relation ?x))))
```

```
(iff:set transitive-span)
(forall ((transitive-span ?x)) (endospan ?x))
(forall ((endospan ?x))
  (<=> (transitive-span ?x)
    (type.spn.mor:belonging [(composition [?x ?x]) ?x])))
(forall ((endospan ?x))
  (=> (transitive-span ?x) (type.rel:transitive-relation (relation ?x))))
```

```

(iff:set symmetric-span)
(forall ((symmetric-span ?x)) (endospan ?x))
(forall ((endospan ?x))
  (<=> (symmetric-span ?x)
    (type.spn.mor:belonging [(opposite ?x) ?x])))
(forall ((endospan ?x))
  (=> (symmetric-span ?x) (type.rel:symmetric-relation (relation ?x))))

```

We can declare special type endospans called proto-categories and equivalence spans. A *proto-category* $x = (X \xleftarrow{\partial_0} Y \xrightarrow{\partial_1} X)$ with object set X and morphism set Y , is a reflexive and transitive endospan. The proof function $\iota : X \rightarrow Y$ is a proto-identity function and the proof function $\mu : Y \times_X Y \rightarrow Y$ is a proto-composition function. Since belonging implies inclusion and the relation conversion preserves identity and composition, $\text{rel}(x)$ is reflexive and transitive; that is, the relation of a proto-category (preorder span) is a preorder relation. Any category is a proto-category that satisfies associative and unit laws. An *equivalence span* is a reflexive, symmetric and transitive span.

```

(iff:set proto-category)
(forall ((proto-category ?x)) (endospan ?x))
(forall ((endospan ?x))
  (<=> (proto-category ?x)
    (and (reflexive-span ?x) (transitive-span ?x))))
(forall ((endospan ?x))
  (=> (proto-category ?x) (type.rel:preorder (relation ?x))))

(iff:set equivalence-span)
(forall ((equivalence-span ?x)) (preorder ?x))
(forall ((preorder ?x))
  (<=> (equivalence-span ?x)
    (symmetric-span ?x)))
(forall ((endospan ?x))
  (=> (equivalence-span ?x) (type.rel:equivalence-relation (relation ?x))))

```

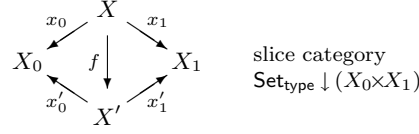


Figure 10: Span 2-Cell

1.4.2 Span Morphisms

Basics. A type span *morphism* is a morphism in the comma category

$$\mathbf{Set}_{\text{type}} \xleftarrow{\pi_0} (\Delta \downarrow 1) \xrightarrow{\pi_1} \mathbf{Set}_{\text{type}}^2$$

over the functorial ospan $\Delta : \mathbf{Set}_{\text{type}} \rightarrow \mathbf{Set}_{\text{type}}^2 \leftarrow \mathbf{Set}_{\text{type}}^2 : 1$ with constant functor $\Delta : \mathbf{Set}_{\text{type}} \rightarrow \mathbf{Set}_{\text{type}}^2$. More specifically, a type span morphism, from source span $x = (X, (X_0, X_1), (x_0, x_1))$ to target span $x' = (X', (X'_0, X'_1), (x'_0, x'_1))$, is a pair $(f, (f_0, f_1))$ consisting of a vertex function $f : X \rightarrow X'$ and a function pair $(f_0, f_1) : (X_0, X_1) \rightarrow (X'_0, X'_1)$, which satisfy the identity $f \cdot \langle x' \rangle = \langle x \rangle \cdot (f_0 \times f_1)$.

However, we do not need full generality here. Instead we restrict the definition by requiring the component functions (f_0, f_1) to be identities. A type span *2-cell* $\alpha : x \xrightarrow{f} x'$, from span $x = (X_0 \xrightarrow{x_0} X \xrightarrow{x_1} X_1)$ to span $x' = (X_0 \xrightarrow{x'_0} X' \xrightarrow{x'_1} X_1)$, is a vertex function $f : X \rightarrow X'$ satisfying the identity $f \cdot \langle x' \rangle = \langle x \rangle$; or equivalently, the identities $f \cdot x'_0 = x_0$ and $f \cdot x'_1 = x_1$ (Figure 10).

```
(iff:set 2-cell)

(iff:function source)
(= (iff:source source) 2-cell)
(= (iff:target source) type.spn:span)

(iff:function target)
(= (iff:source target) 2-cell)
(= (iff:target target) type.spn:span)

(iff:function function) (iff:function vertex) (= vertex function)
(= (iff:source function) 2-cell)
(= (iff:target function) type.ftn:function)

(forall ((2-cell ?a))
  (and (= (type.ftn:composition [(function ?a) (type.spn:function0 (target ?a))])
        (type.spn:function0 (source ?a)))
        (= (type.ftn:composition [(function ?a) (type.spn:function1 (target ?a))])
          (type.spn:function1 (source ?a)))))
```

For convenience of expression, we name the component sets of a type span morphism.

```
(iff:function set0)
(= (iff:source set0) 2-cell)
(= (iff:target set0) type.set:set)
(forall ((2-cell ?a))
```

```

      (and (= (set0 ?a) (type.spn:set0 (source ?a)))
            (= (set0 ?a) (type.spn:set0 (target ?a))))

      (iff:function set1)
      (= (iff:source set1) 2-cell)
      (= (iff:target set1) type.set:set)
      (forall ((2-cell ?a))
        (and (= (set1 ?a) (type.spn:set0 (source ?a)))
              (= (set1 ?a) (type.spn:set0 (target ?a))))))

```

For any two spans $x, x' \in \text{spn}_{\text{type}}$, x belongs to x' , denoted $x \in x'$, when there is a span 2-cell $\alpha : x \xrightarrow{f} x'$ from x to x' . Equivalently, x belongs to x' when the associated functions satisfy belonging: there exists a *proof* function $f \in \text{ftn}_{\text{type}}$ such that $f \cdot \langle x' \rangle = \langle x \rangle$.

$$\begin{array}{ccc}
 & \xrightarrow{f} & \\
 \langle x \rangle & \searrow & \swarrow \langle x' \rangle \\
 & X_0 \times X_1 &
 \end{array}$$

When x' is a span injection (equivalently, $\langle x' \rangle$ is a function injection), the proof function f is unique. We name the component *elements* of each belonging relationship. The belonging relation is a preorder (reflexive and transitive).

```

      (iff:set belonging)
      (forall ((belonging ?xy))
        (exists ((type.spn:span ?x) (type.spn:span ?y))
          (= ?xy [?x ?y])))
      (forall ((type.spn:span ?x) (type.spn:span ?y))
        (<=> (belonging [?x ?y])
              (exists ((2-cell ?a)
                        (and (= (source ?a) ?x)
                             (= (target ?a) ?y))))))

      (forall ((type.spn:span ?x) (type.spn:span ?y))
        (<=> (belonging [?x ?y])
              (type.ftn:belonging [(type.spn:function ?x) (type.spn:function ?y)])))

      (iff:function element0)
      (= (iff:source element0) belonging)
      (= (iff:target element0) type.spn:span)
      (forall ((type.spn:span ?x) (type.spn:span ?y) (belonging [?x ?y]))
        (= (element0 [?x ?y]) ?x))

      (iff:function element1)
      (= (iff:source element1) belonging)
      (= (iff:target element1) type.spn:span)
      (forall ((type.spn:span ?x) (type.spn:span ?y) (belonging [?x ?y]))
        (= (element1 [?x ?y]) ?y))

      (forall ((type.spn:span ?x) (type.spn:injection ?y) (belonging [?x ?y]))
        (forall ((2-cell ?a1) (2-cell ?a2))
          (=> (and (= ?x (source ?a1)) (= ?y (target ?a1))
                  (= ?x (source ?a2)) (= ?y (target ?a2)))
              (= ?a1 ?a2))))

      (forall ((type.spn:span ?x))

```

```

(belonging [?x ?x])
(forall ((type.spn:span ?x) (type.spn:span ?y) (type.spn:span ?z))
  (=> (and (belonging [?x ?y]) (belonging [?y ?z])
        (belonging [?x ?z])))

```

Two spans $x, x' \in \text{spn}_{\text{type}}$ are *equivalent*, $x \equiv x'$, when each belongs to the other, $x \sqsubseteq x'$ and $x' \sqsubseteq x$. The equivalence relation is an equivalence relation (reflexive, symmetric and transitive). Two equivalent spans are *isomorphic*, $x \cong x'$, when there exists a bijection $p \in \text{ftn}_{\text{type}}$ proving belonging. The isomorphism relation is an equivalence relation (reflexive, symmetric and transitive).

```

(iff:set equivalence)
(forall ((equivalence ?xy))
  (exists ((type.spn:span ?x) (type.spn:span ?x))
    (= ?xy [?x ?y])))
(forall ((type.spn:span ?x) (type.spn:span ?y))
  (<=> (equivalence [?x ?y])
        (and (belonging [?x ?y]) (belonging [?y ?x]))))

(forall ((type.spn:span ?x))
  (equivalence [?x ?x]))
(forall ((type.spn:span ?x) (type.spn:span ?y))
  (=> (equivalence [?x ?y]) (equivalence [?y ?x])))
(forall ((type.spn:span ?x1) (type.spn:span ?x2) (type.spn:span ?x3))
  (=> (and (equivalence [?x1 ?x2]) (equivalence [?x2 ?x3])
        (equivalence [?x1 ?x3])))

(iff:set isomorphism)
(forall ((isomorphism ?xy)) (equivalence [?xy]))
(forall ((type.spn:span ?x) (type.spn:span ?y))
  (<=> (isomorphism [?x ?y])
        (exists ((2-cell ?a) (= ?x (source ?a)) (= ?y (target ?a)))
          (type.spn:bijection (function ?a)))))

(forall ((type.spn:span ?x))
  (isomorphism [?x ?x]))
(forall ((type.spn:span ?x) (type.spn:span ?y))
  (=> (isomorphism [?x ?y]) (isomorphism [?y ?x])))
(forall ((type.spn:span ?x1) (type.spn:span ?x2) (type.spn:span ?x3))
  (=> (and (isomorphism [?x1 ?x2]) (isomorphism [?x2 ?x3])
        (isomorphism [?x1 ?x3])))

```

Conversion. A type span 2-cell $\alpha : x \xrightarrow{f} x'$ has an associated type function 2-cell $\langle \alpha \rangle : \langle x \rangle \xrightarrow{f} \langle x' \rangle$ between the functions of the source and target spans. The function of $\langle \alpha \rangle$, which is the function of α , $f : X \rightarrow X'$, commutes with source and target functions: $f \cdot \langle x' \rangle = \langle x \rangle$.

```

(iff:function function-2-cell)
(= (iff:source function-2-cell) 2-cell)
(= (iff:target function-2-cell) type.ftn.mor:2-cell)
(forall ((2-cell ?a))
  (and (= (type.ftn.mor:source (function-2-cell ?a)) (type.spn:function (source ?a)))
        (= (type.ftn.mor:target (function-2-cell ?a)) (type.spn:function (target ?a)))
        (= (type.ftn.mor:function (function-2-cell ?a)) (function ?a))))

```

Belonging implies inclusion: $x \in x'$ implies $\text{rel}(x) \subseteq \text{rel}(x')$. That is, for any two spans $x, x' \in \text{spn}_{\text{type}}$, if x belongs to x' , then $\text{rel}(x)$ is included in $\text{rel}(x')$.

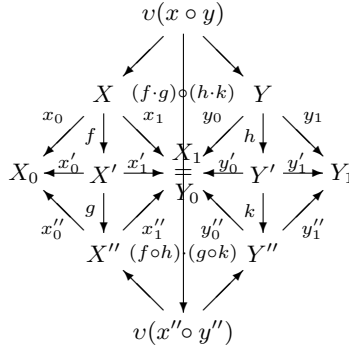
```

(forall ((type.spn:span ?x) (type.spn:span ?y))
  (=> (belonging [?x ?y])
    (type.rel:inclusion-relation [(type.spn:relation ?x) (type.spn:relation ?y)])))

```

Category Theory. Vertical and horizontal compositions satisfy the “inter-change” law

$$(\alpha \cdot \beta) \circ (\gamma \cdot \delta) \cong (\alpha \circ \beta) \cdot (\gamma \circ \delta).$$



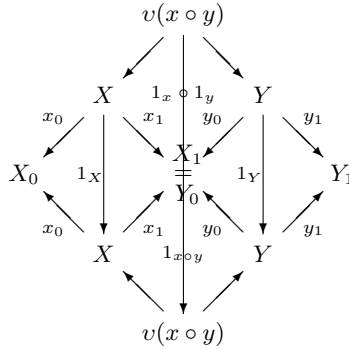
```

(forall ((2-cell ?a) (2-cell ?b) (2-cell ?c) (2-cell ?d)
  (type.spn.mor.vrt:composable-pair [?a ?b]) (type.spn.mor.vrt:composable-pair [?c ?d])
  (type.spn.mor.hrz:composable-pair [?a ?c]) (type.spn.mor.hrz:composable-pair [?b ?d]))
  (type.spn:isomorphism
    [(type.spn.mor.hrz:composition
      [(type.spn.mor.vrt:composition [?a ?b]) (type.spn.mor.vrt:composition [?c ?d])])
      (type.spn.mor.vrt:composition
        [(type.spn.mor.hrz:composition [?a ?c]) (type.spn.mor.hrz:composition [?b ?d])])])])

```

For a composable pair of spans $x = (X_0 \xleftarrow{x_0} X \xrightarrow{x_1} X_1)$ and $y = (Y_0 \xleftarrow{y_0} Y \xrightarrow{y_1} Y_1)$, vertical identity and horizontal composition satisfy the law

$$1_x \circ 1_y = 1_{x \circ y}.$$



```

(forall ((type.spn:span ?x) (type.spn:span ?y) (type.spn:composable-pair [?x ?y]))
  (= (type.spn.mor.hrz:composition [(type.spn.mor.vrt:identity ?x) (type.spn.mor.vrt:identity ?y)])
    (type.spn.mor.vrt:identity (type.spn:composition [?x ?y])))

```

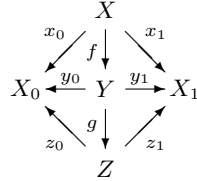
Vertical Aspect. A pair of type span 2-cells is *vertically composable* when the target span of the first is equal to the source span of the second. We name the vertical projection *factors* of vertically composable pairs.

```
(iff:set composable-pair)
(forall ((composable-pair ?ab))
  (exists ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b))
    (= ?ab [?a ?b])))
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b))
  (<=> (composable-pair [?a ?b])
    (= (type.spn.mor:target ?a) (type.spn.mor:source ?b))))

(iff:function vertical-factor0)
(= (iff:source vertical-factor0) composable-pair)
(= (iff:target vertical-factor0) type.spn.mor:2-cell)
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (composable-pair [?a ?b]))
  (= (vertical-factor0 [?a ?b]) ?a))

(iff:function vertical-factor1)
(= (iff:source vertical-factor1) composable-pair)
(= (iff:target vertical-factor1) type.spn.mor:2-cell)
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (composable-pair [?a ?b]))
  (= (vertical-factor1 [?a ?b]) ?b))
```

The *vertical composition* of two vertically composable type span 2-cells $\alpha : x \xrightarrow{f} y$ and $\beta : y \xrightarrow{g} z$ is a type span 2-cell $\alpha \cdot \beta : x \xrightarrow{f \cdot g} z$. The vertical source of the composite is the vertical source of the first component factor, the vertical target of the composite is the vertical target of the second component factor, and the function of the composite is the composite of the functions of the component factors.

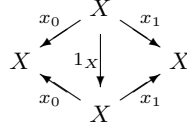


```
(iff:function composition)
(= (iff:source composition) composable-pair)
(= (iff:target composition) type.spn.mor:2-cell)
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (composable-pair [?a ?b]))
  (and (= (type.spn.mor:source (composition [?a ?b])) (type.spn.mor:source ?a))
    (= (type.spn.mor:target (composition [?a ?b])) (type.spn.mor:target ?b))
    (= (type.spn.mor:function (composition [?a ?b]))
      (type.ftn:composition [(type.spn.mor:function ?a) (type.spn.mor:function ?b)]))))))
```

Composition is associative. Any three vertically composable type span 2-cells $\alpha : x \xrightarrow{f} y$, $\beta : y \xrightarrow{g} z$ and $\gamma : z \xrightarrow{c} w$ satisfy the associative law $\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma$.

```
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (type.spn.mor:2-cell ?c)
  (composable-pair [?a ?b]) (composable-pair [?b ?c]))
  (= (composition [?a (composition [?b ?c])])
    (composition [(composition [?a ?b]) ?c])))
```

For every type span $x = (X_0 \xleftarrow{x_0} X \xrightarrow{x_1} X_1)$, there is a unique associated *vertical identity* type span 2-cell $1_x : x \overset{1_x}{\rightrightarrows} x$.



```

(iff:function identity)
(= (iff:source identity) type.spn:span)
(= (iff:target identity) type.spn.mor:2-cell)
(forall ((type.spn:span ?x))
  (and (= (type.spn.mor:source (identity ?x)) ?x)
        (= (type.spn.mor:target (identity ?x)) ?x)
        (= (type.spn.mor:function (identity ?x)) (type.ftn:identity (type.spn:vertex ?x)))))

```

Vertical identity satisfies two unit laws with respect to vertical composition: vertical composition with the vertical identity of the source (target) span of a span 2-cell $\alpha : x \xrightarrow{f} y$ returns that span 2-cell: $1_x \cdot \alpha = \alpha = \alpha \cdot 1_y$.

```

(forall ((type.spn.mor:2-cell ?a))
  (and (= (composition [(identity (type.spn.mor:source ?a)) ?a] ?a)
        (= ?a (composition [?a (identity (type.spn.mor:target ?a))])))

```

Horizontal Aspect. A pair of type span 2-cells is *horizontally composable* when the target set of the first is equal to the source set of the second. We name the horizontal projection *factors* of horizontally composable pairs.

```

(iff:set composable-pair)
(forall ((composable-pair ?ab))
  (exists ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b))
    (= ?ab [?a ?b])))
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b))
  (<=> (composable-pair [?a ?b])
        (= (type.spn.mor:set1 ?a) (type.spn.mor:set0 ?b))))

(iff:function factor0)
(= (iff:source factor0) composable-pair)
(= (iff:target factor0) type.spn.mor:2-cell)
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (composable-pair [?a ?b]))
  (= (factor0 [?a ?b]) ?a))

(iff:function factor1)
(= (iff:source factor1) composable-pair)
(= (iff:target factor1) type.spn.mor:2-cell)
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (composable-pair [?a ?b]))
  (= (factor1 [?a ?b]) ?b))

```

For any horizontally composable pair of span 2-cells $\alpha : x \xrightarrow{f} x'$ and $\beta : y \xrightarrow{g} y'$, the pair of source (target) spans is composable.

```

(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (composable ?a ?b))
  (and (type.spn:composable (type.spn.mor:source ?a) (type.spn.mor:source ?b))
        (type.spn:composable (type.spn.mor:target ?a) (type.spn.mor:target ?b))))

```

For any horizontally composable pair of span 2-cells $\alpha : x \xrightarrow{f} x'$ and $\beta : y \xrightarrow{g} y'$, there is an associated opspan morphism

$$\Upsilon(\alpha, \beta) \left\{ \begin{array}{c} \overbrace{\begin{array}{ccc} v(x) & \xrightarrow{\varphi_1^x} & \sigma_1(x) = \sigma_0(y) & \xleftarrow{\varphi_0^y} & v(y) \\ f \downarrow & & 1_{X_1} \neq 1_{Y_0} & & \downarrow g \\ v(x') & \xrightarrow{\varphi_1^{x'}} & \sigma_1(x') = \sigma_0(y') & \xleftarrow{\varphi_0^{y'}} & v(y') \end{array}}^{\Upsilon(x, y)} \\ \underbrace{\hspace{10em}}_{\Upsilon(x', y')} \end{array} \right.$$

```

(iff:function opspan-morphism)
(= (iff:source opspan-morphism) composable-pair)
(= (iff:target opspan-morphism) type.dgm.ospn.mor:opspan-morphism)
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (composable-pair [?a ?b]))
  (and (= (type.dgm.ospn.mor:source (opspan-morphism [?a ?b]))
    (type.spn:opspan [(type.spn.mor:source ?a) (type.spn.mor:source ?b)]))
    (= (type.dgm.ospn.mor:target (opspan-morphism [?a ?b]))
    (type.spn:opspan [(type.spn.mor:target ?a) (type.spn.mor:target ?b)]))
    (= (type.dgm.ospn.mor:function0 (opspan-morphism [?a ?b])) (type.spn.mor:function ?a))
    (= (type.dgm.ospn.mor:function1 (opspan-morphism [?a ?b])) (type.spn.mor:function ?b))))
(= (type.dgm.ospn.mor:opvertex (opspan-morphism [?a ?b]))
  (type.ftn:identity (type.spn.mor:set1 ?a)))
(= (type.dgm.ospn.mor:opvertex (opspan-morphism [?a ?b]))
  (type.ftn:identity (type.spn.mor:set0 ?b)))

```

The *horizontal composition* of two horizontally composable type span 2-cells $\alpha : x \xrightarrow{f} x'$ and $\beta : y \xrightarrow{g} y'$ is a type span 2-cell $\alpha \circ \beta : x \circ y \xrightarrow{\lim \Upsilon(\alpha, \beta)} y \circ y'$. The horizontal source of the composite is the composite of the source spans of the component factors, the horizontal target of the composite is the composite of the target spans of the component factors, and the function of the composite is the pullback of the opspan morphism of the horizontally composable pair.

$$\begin{array}{ccccc} & & v(x \circ y) = X \times_{X_1=Y_0} Y & & \\ & \swarrow y_0 & \downarrow f \circ g & \searrow y_1 & \\ & X & X_1 = Y_0 & Y & \\ & \swarrow x_0 & \downarrow f & \downarrow g & \searrow y_1 \\ X_0 & & X_1 & & Y_1 \\ & \swarrow x_0 & \downarrow f & \downarrow g & \searrow y_1 \\ & X' & X_1 = Y_0 & Y' & \\ & \swarrow x'_0 & \downarrow f & \downarrow g & \searrow y'_1 \\ & X' & X_1 = Y_0 & Y' & \\ & \swarrow y'_0 & \downarrow f & \downarrow g & \searrow y'_1 \\ & & v(x' \circ y') = X' \times_{X'_1=Y'_0} Y' & & \end{array}$$

```

(iff:function composition)
(= (iff:source composition) composable-pair)
(= (iff:target composition) type.spn.mor:2-cell)
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (composable-pair [?a ?b]))
  (and (= (type.spn.mor:source (composition [?a ?b]))
    (type.spn:composition [(type.spn.mor:source ?a) (type.spn.mor:source ?b)]))
    (= (type.spn.mor:target (composition [?a ?b]))
    (type.spn:composition [(type.spn.mor:target ?a) (type.spn.mor:target ?b)]))
    (= (type.spn.mor:function (composition [?a ?b]))
    (type.lim.pbk.mor:pullback (opspan-morphism [?a ?b])))))

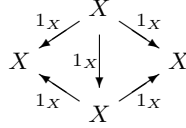
```

Composition is associative (up to natural isomorphism)¹⁰. Any three horizontally composable type span 2-cells $\alpha : x \xrightarrow{f} x'$, $\beta : y \xrightarrow{g} y'$ and $\gamma : z \xrightarrow{h} z'$ satisfy the associative law

$$\alpha \circ (\beta \circ \gamma) \cong (\alpha \circ \beta) \circ \gamma.$$

```
(forall ((type.spn.mor:2-cell ?a) (type.spn.mor:2-cell ?b) (type.spn.mor:2-cell ?c)
  (composable-pair [?a ?b]) (composable-pair [?b ?c]))
  (type.spn.mor:isomorphism
    [(composition [?a (composition [?b ?c])])
     (composition [(composition [?a ?b]) ?c])]))
```

For every type set X , there is a unique associated *horizontal identity* type span 2-cell $1_{1_X} : 1_X \xrightarrow{1_X} 1_X$. The horizontal source of the identity is the identity span $1_X = (X \xleftarrow{1_X} X \xrightarrow{1_X} X)$, the horizontal target of the identity is the identity span $1_X = (X \xleftarrow{1_X} X \xrightarrow{1_X} X)$, and the function of the identity is the identity function $1_X : X \rightarrow X$.



```
(iff:function identity)
(= (iff:source identity) type.set:set)
(= (iff:target identity) type.spn.mor:2-cell)
(forall ((type.set:set ?X))
  (and (= (type.spn.mor:source (identity ?X)) (type.spn:identity ?X))
        (= (type.spn.mor:target (identity ?X)) (type.spn:identity ?X))
        (= (type.spn.mor:function (identity ?x)) (type.ftn:identity ?X))))
```

Horizontal identity satisfies two unit laws with respect to horizontal composition: horizontal composition with the horizontal identity of the source (target) set of a span 2-cell $\alpha : x \xrightarrow{f} x'$, where $x = (X_0 \xleftarrow{x_0} X \xrightarrow{x_1} X_1)$ and $x' = (X_0 \xleftarrow{x'_0} X' \xrightarrow{x'_1} X_1)$, returns that span 2-cell (up to natural isomorphism):

$$1_{1_{X_0}} \circ \alpha \cong \alpha \cong \alpha \circ 1_{1_{X_1}}.$$

```
(forall ((type.spn.mor:2-cell ?a))
  (and (type.spn.mor:isomorphism
        [(composition [(identity (type.spn.mor:set0 ?a)) ?a]) ?a])
        (type.spn.mor:isomorphism
          [?a (composition [?a (identity (type.spn.mor:set1 ?a))])]))))
```

¹⁰The category Set_{type} gives rise to a span bicategory with 0-cells being type sets, 1-cells being type spans and 2-cells being type span 2-cells.

Our two standard references for the IFF are the books: *Sets for Mathematics* (2003) by F. William Lawvere and Robert Rosebrugh [1] and *Categories for the Working Mathematician* (1971) by Saunders Mac Lane [2].

References

- [1] F. William Lawvere and Robert Rosebrugh. *Sets for Mathematics*. Cambridge University Press, 2003.
- [2] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.