Type Language Colimits

lang.col

Table 1 lists the terminology for the sub-namespace of finite type language colimits.

Table 1: Terminology for the type language colimit namespace

	Class	Function	Other
lang.col		counique	initial
lang.col .coprd2	diagram = pair	language1 language2 variable-diagram entity-diagram relation-diagram	
	cocone	cocone-diagram opvertex opfirst opsecond variable-cocone entity-cocone relation-cocone	
		colimiting-cocone colimit = binary-coproduct injection1 injection2 comediator	
lang.col .endo	endorelation	language = base variable entity relation quotient canon comediator	matches-opspan matches respects
lang.col .coeq	diagram = parallel-pair	source target language-morphism1 language-morphism2 variable-diagram entity-diagram relation-diagram endorelation	
	cocone	cocone-diagram opvertex language-morphism variable-cocone entity-cocone relation-cocone	
		colimiting-cocone colimit = coequalizer canon comediator	
lang.col .psh	diagram = span	language1 language2 vertex first second pair opposite variable-diagram entity-diagram relation-diagram coequalizer-diagram	
	cocone	cocone-diagram opvertex opfirst opsecond binary-coproduct-cocone coequalizer-cocone variable-cocone entity-cocone relation-cocone	
		colimiting-cocone colimit = pushout injection1 injection2 comediator	

The Initial Type Language

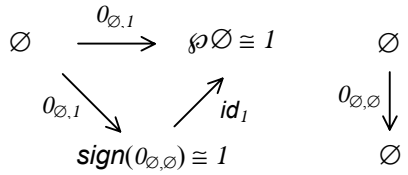


Figure 1: Initial Type Language

- There is a special type language $\mathbf{0} = \langle 0_{\emptyset,\emptyset}, 0_{\emptyset,1} \rangle$ (see Figure 1, where arrows denote functions) called the *initial type language*, which has no variables, no entity types and no relation types
  - the set of *entity* types $ent(\mathbf{0}) = \emptyset$,
  - the set of *relation* types $rel(\mathbf{0}) = \emptyset$, and
  - the set of *variables* $var(\mathbf{0}) = \emptyset$.

  It has the following empty signature, arity and reference functions:
  - *signature* function $\partial_{\mathbf{0}} = 0_{\emptyset,1} : \emptyset \to sign(0_{\emptyset,\emptyset}) \cong 1$,
  - *arity* function $\#_{\mathbf{0}} = 0_{\emptyset,1} : \emptyset \to \wp\emptyset \cong 1$, and
  - *reference* function $*_{\mathbf{0}} = id_{\emptyset} = 0_{\emptyset,\emptyset} : \emptyset \to \emptyset$.

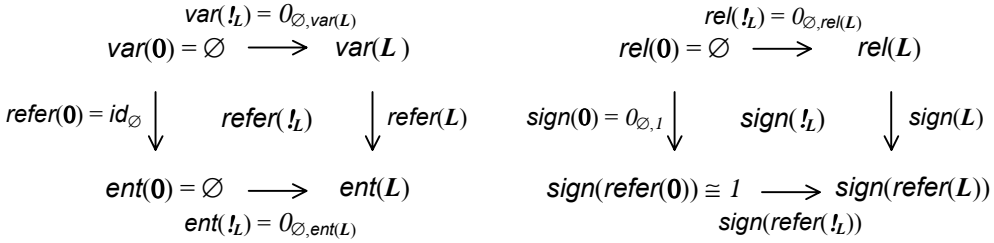


Figure 2: Counique Type Language Morphism

The initial type language has the property that for any type language $\boldsymbol{L} = \langle refer(\boldsymbol{L}), sign(\boldsymbol{L}) \rangle$ there is a *counique type language morphism* $!_{\boldsymbol{L}} = \langle refer(!_{\boldsymbol{L}}), sign(!_{\boldsymbol{L}}) \rangle : \mathbf{0} \to \boldsymbol{L}$, (see Figure 2, where arrows denote functions) the unique type language morphism from $\mathbf{0}$ to $\boldsymbol{L}$. The variable function of this type language morphism is the counique function (the empty function) from the initial (empty or null) set $\emptyset$ to the variable set $var(\boldsymbol{L})$. The entity type function of this type language morphism is the counique function (the empty function) from the initial (empty or null) set $\emptyset$ to the entity type set $ent(\boldsymbol{L})$. The relation type function of this type language morphism is the counique function (the empty function) from the initial (empty or null) set $\emptyset$ to the relation type set $rel(\boldsymbol{L})$.

- The empty *entity* type function is $ent(!_{\boldsymbol{L}}) = 0_{\emptyset,ent(\boldsymbol{L})} : \emptyset \to ent(\boldsymbol{L})$,
- The empty *relation* type function is $rel(!_{\boldsymbol{L}}) = 0_{\emptyset,rel(\boldsymbol{L})} : \emptyset \to rel(\boldsymbol{L})$, and
- The empty *variable* function is $var(!_{\boldsymbol{L}}) = 0_{\emptyset,var(\boldsymbol{L})} : \emptyset \to var(\boldsymbol{L})$.

The function $sign(refer(!_{\boldsymbol{L}})) : sign(refer(\mathbf{0})) \cong 1 \to var(\boldsymbol{L})$ maps the single (empty) signature in $sign(refer(\mathbf{0}))$ to the empty signature in $sign(refer(\boldsymbol{L}))$. The reference and signature quartets for the counique type language morphism are vacuous.

```
(1) (lang$language initial)
    (= (lang$variable initial) set.lim$initial)
    (= (lang$entity initial) set.lim$initial)
    (= (lang$relation initial) set.lim$initial)
    (= (lang$reference initial) (set.ftn$identity set.lim$initial))
    (= (lang$signature initial)
       (set.col$counique (set.ftn$signature (lang$reference initial))))

(2) (SET.FTN$function counique)
    (= (SET.FTN$source counique) lang$language)
    (= (SET.FTN$target counique) lang.mor$language-morphism)
    (= (SET.FTN$composition [counique lang.mor$source])
       ((SET.FTN$constant [lang$language set$set]) initial))
    (= (SET.FTN$composition [counique lang.mor$target])
       (SET.FTN$identity lang$language))
    (= (SET.FTN$composition [counique lang.mor$variable])
       (SET.FTN$composition [lang$variable set.col$counique]))
```

```
(= (SET.FTN$composition [counique lang.mor$entity])
   (SET.FTN$composition [lang$entity set.col$counique]))
(= (SET.FTN$composition [counique lang.mor$relation])
   (SET.FTN$composition [lang$relation set.col$counique]))
```

Binary Coproducts

`lang.col.coprd2`

A *binary coproduct* is a finite colimit in the category Language for a diagram of shape *two* = • •. Such a diagram (of type languages and type language morphisms) is called a *pair* of type languages.

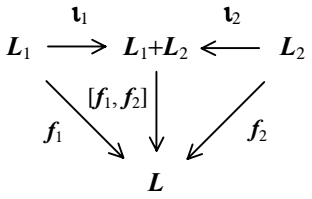


Figure 3a: Binary Coproduct Type Language, Cocone and Comediator – abstract version

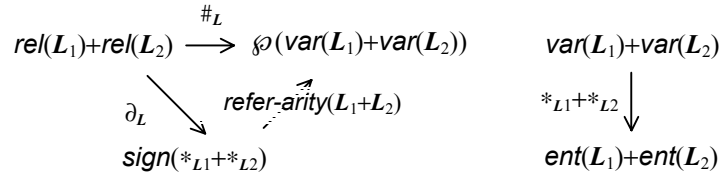


Figure 3b: Binary Coproduct Type Language – concrete version

- Given a pair of type languages $\boldsymbol{L}_1 = \langle refer(\boldsymbol{L}_1), sign(\boldsymbol{L}_1)\rangle$ and $\boldsymbol{L}_2 = \langle refer(\boldsymbol{L}_2), sign(\boldsymbol{L}_2)\rangle$, the *coproduct* or *sum* $\boldsymbol{L}_1 + \boldsymbol{L}_2$ (see Figure 3a where arrows denote type language morphisms, of Figure 3b where arrows denote functions) is the type language defined as follows:
  - The class of variables is the disjoint union $var(\boldsymbol{L}_1+\boldsymbol{L}_2) = var(\boldsymbol{L}_1)+var(\boldsymbol{L}_2)$. Concretely, variables of $\boldsymbol{L}_1 + \boldsymbol{L}_2$ are either pairs $(1, x_1)$ where $x_1 \in var(\boldsymbol{L}_1)$ or pairs $(2, x_2)$ where $x_2 \in var(\boldsymbol{L}_2)$.
  - The class of entity types is the disjoint union $ent(\boldsymbol{L}_1+\boldsymbol{L}_2) = ent(\boldsymbol{L}_1)+ent(\boldsymbol{L}_2)$. Concretely, entity types of $\boldsymbol{L}_1 + \boldsymbol{L}_2$ are either pairs $(1, \varepsilon_1)$ where $\varepsilon_1 \in ent(\boldsymbol{L}_1)$ or pairs $(2, \varepsilon_2)$ where $\varepsilon_2 \in ent(\boldsymbol{L}_2)$.
  - The class of relation types is the disjoint union $rel(\boldsymbol{L}_1+\boldsymbol{L}_2) = rel(\boldsymbol{L}_1)+rel(\boldsymbol{L}_2)$. Concretely, relation types of $\boldsymbol{L}_1 + \boldsymbol{L}_2$ are either pairs $(1, \rho_1)$ where $\rho_1 \in rel(\boldsymbol{L}_1)$ or pairs $(2, \rho_2)$ where $\rho_2 \in rel(\boldsymbol{L}_2)$.

  This results in the following presentations of the signature, arity and reference functions:
  - the *reference* function $*_{\boldsymbol{L}1+\boldsymbol{L}2} : var(\boldsymbol{L}_1)+ var(\boldsymbol{L}_2) \to ent(\boldsymbol{L}_1)+ ent(\boldsymbol{L}_2)$ is the sum $*_{\boldsymbol{L}1+\boldsymbol{L}2} = *_{\boldsymbol{L}1}+*_{\boldsymbol{L}2}$
  - the *arity* function $\#_{\boldsymbol{L}1+\boldsymbol{L}2} : rel(\boldsymbol{L}_1)+rel(\boldsymbol{L}_2) \to \wp(var(\boldsymbol{L}_1)+var(\boldsymbol{L}_2)) \to \wp(var(\boldsymbol{L}_1)+var(\boldsymbol{L}_2))$ is the composition $\#_{\boldsymbol{L}1+\boldsymbol{L}2} = (\#_{\boldsymbol{L}1}+\#_{\boldsymbol{L}2}) \cdot [\wp\iota_{var(\boldsymbol{L}1)}, \wp\iota_{var(\boldsymbol{L}2)}]$
  - the *signature* function $\partial_{\boldsymbol{L}1+\boldsymbol{L}2} : rel(\boldsymbol{L}_1)+rel(\boldsymbol{L}_2) \to sign(*_{\boldsymbol{L}1}+*_{\boldsymbol{L}2})$, where the signatures can viewed as signature pairs via the bijection $sign(*_{\boldsymbol{L}1}+*_{\boldsymbol{L}2}) \cong sign(*_{\boldsymbol{L}1} \cdot \iota_{ent(\boldsymbol{L}1)}) \times sign(*_{\boldsymbol{L}2} \cdot \iota_{ent(\boldsymbol{L}2)})$, has the 1st projection function $\partial_{\boldsymbol{L}1+\boldsymbol{L}2} \cdot \pi_1 : rel(\boldsymbol{L}_1)+rel(\boldsymbol{L}_2) \to sign(*_{\boldsymbol{L}1} \cdot \iota_{ent(\boldsymbol{L}1)})$ that maps a relation type $\rho_1 \in rel(\boldsymbol{L}_1)$ to the signature $\partial_{\boldsymbol{L}1}(\rho_1) \cdot \iota_{ent(\boldsymbol{L}1)}$, and likewise for the other projection.
- A *pair* (of type languages) is the appropriate base diagram for a binary coproduct. Each pair consists of a pair of type languages called *language1* and *language2*. We use either the generic term '`diagram`' or the specific term '`pair`' to denotes the *pair* class. Pairs are determined by their two component type languages.

```
(1) (SET$class diagram)
    (SET$class pair)
    (= pair diagram)

(2) (SET.FTN$function language1)
    (= (SET.FTN$source language1) diagram)
    (= (SET.FTN$target language1) lang$language)

(3) (SET.FTN$function language2)
    (= (SET.FTN$source language2) diagram)
    (= (SET.FTN$target language2) lang$language)

    (forall (?p (diagram ?p) ?q (diagram ?q))
        (=> (and (= (language1 ?p) (language1 ?q))
                 (= (language2 ?p) (language2 ?q)))
            (= ?p ?q)))
```

- There is a *variable pair* or *variable diagram* function, which maps a pair of type languages to the underlying pair of variable sets. Similarly, there is an *entity type pair* or *entity type diagram* function, which maps a pair of type languages to the underlying pair of entity type sets; and there is a *relation type pair* or *relation type diagram* function, which maps a pair of type languages to the underlying pair of relation type sets.

```
(4) (SET.FTN$function variable-diagram)
    (= (SET.FTN$source variable-diagram) diagram)
    (= (SET.FTN$target variable-diagram) set.col.coprd2$diagram)
    (= (SET.FTN$composition [variable-diagram set.col.coprd2$set1])
       (SET.FTN$composition [language1 lang$variable]))
    (= (SET.FTN$composition [variable-diagram set.col.coprd2$set2])
       (SET.FTN$composition [language2 lang$variable]))

(5) (SET.FTN$function entity-diagram)
    (= (SET.FTN$source entity-diagram) diagram)
    (= (SET.FTN$target entity-diagram) set.col.coprd2$diagram)
    (= (SET.FTN$composition [entity-diagram set.col.coprd2$set1])
       (SET.FTN$composition [language1 lang$entity]))
    (= (SET.FTN$composition [entity-diagram set.col.coprd2$set2])
       (SET.FTN$composition [language2 lang$entity]))

(6) (SET.FTN$function relation-diagram)
    (= (SET.FTN$source relation-diagram) diagram)
    (= (SET.FTN$target relation-diagram) set.col.coprd2$diagram)
    (= (SET.FTN$composition [relation-diagram set.col.coprd2$set1])
       (SET.FTN$composition [language1 lang$relation]))
    (= (SET.FTN$composition [relation-diagram set.col.coprd2$set2])
       (SET.FTN$composition [language2 lang$relation]))
```

- A *binary coproduct cocone* is the appropriate cocone for a binary coproduct. A coproduct cocone (see Figure 3, where arrows denote functions) consists of a pair of type language morphisms called *opfirst* and *opsecond*. These are required to have a common target type language called the *opvertex* of the cocone. Each binary coproduct cocone is under a pair of type languages.

```
(7) (SET$class cocone)

(8) (SET.FTN$function cocone-diagram)
    (= (SET.FTN$source cocone-diagram) cocone)
    (= (SET.FTN$target cocone-diagram) diagram)

(9) (SET.FTN$function opvertex)
    (= (SET.FTN$source opvertex) cocone)
    (= (SET.FTN$target opvertex) lang$language)

(10) (SET.FTN$function opfirst)
     (= (SET.FTN$source opfirst) cocone)
     (= (SET.FTN$target opfirst) lang.mor$language-morphism)
     (= (SET.FTN$composition [opfirst lang.mor$source])
        (SET.FTN$composition [cocone-diagram language1]))
     (= (SET.FTN$composition [opfirst lang.mor$target]) opvertex)

(11) (SET.FTN$function opsecond)
     (= (SET.FTN$source opsecond) cocone)
     (= (SET.FTN$target opsecond) lang.mor$language-morphism)
     (= (SET.FTN$composition [opsecond lang.mor$source])
        (SET.FTN$composition [cocone-diagram language2]))
     (= (SET.FTN$composition [opsecond lang.mor$target]) opvertex)
```

- There is a *variable cocone* function, which maps a binary coproduct cocone of type languages and type language morphisms to the underlying binary coproduct cocone of variable sets and variable functions. Similarly, there is an *entity type cocone* function, which maps a binary coproduct cocone of type languages and type language morphisms to the underlying binary coproduct cocone of entity type sets and entity type functions; and there is a *relation type cocone* function, which maps a binary coproduct cocone of type languages and type language morphisms to the underlying binary coproduct cocone of relation type sets and relation type functions.

```
(12) (SET.FTN$function variable-cocone)
     (= (SET.FTN$source variable-cocone) cocone)
     (= (SET.FTN$target variable-cocone) set.colim.coprd2$cocone)
     (= (SET.FTN$composition [variable-cocone set.colim.coprd2$cone-diagram])
        (SET.FTN$composition [cocone-diagram variable-diagram]))
     (= (SET.FTN$composition [variable-cocone set.colim.coprd2$opvertex])
        (SET.FTN$composition [opvertex lang$variable]))
     (= (SET.FTN$composition [variable-cocone set.colim.coprd2$opfirst])
        (SET.FTN$composition [opfirst lang.mor$variable]))
     (= (SET.FTN$composition [variable-cocone set.colim.coprd2$opsecond])
        (SET.FTN$composition [opsecond lang.mor$variable]))

(13) (SET.FTN$function entity-cocone)
     (= (SET.FTN$source entity-cocone) cocone)
     (= (SET.FTN$target entity-cocone) set.colim.coprd2$cocone)
     (= (SET.FTN$composition [entity-cocone set.colim.coprd2$cone-diagram])
        (SET.FTN$composition [cocone-diagram entity-diagram]))
     (= (SET.FTN$composition [entity-cocone set.colim.coprd2$opvertex])
        (SET.FTN$composition [opvertex lang$entity]))
     (= (SET.FTN$composition [entity-cocone set.colim.coprd2$opfirst])
        (SET.FTN$composition [opfirst lang.mor$entity]))
     (= (SET.FTN$composition [entity-cocone set.colim.coprd2$opsecond])
        (SET.FTN$composition [opsecond lang.mor$entity]))

(14) (SET.FTN$function relation-cocone)
     (= (SET.FTN$source relation-cocone) cocone)
     (= (SET.FTN$target relation-cocone) set.colim.coprd2$cocone)
     (= (SET.FTN$composition [relation-cocone set.colim.coprd2$cone-diagram])
        (SET.FTN$composition [cocone-diagram relation-diagram]))
     (= (SET.FTN$composition [relation-cocone set.colim.coprd2$opvertex])
        (SET.FTN$composition [opvertex lang$relation]))
     (= (SET.FTN$composition [relation-cocone set.colim.coprd2$opfirst])
        (SET.FTN$composition [opfirst lang.mor$relation]))
     (= (SET.FTN$composition [relation-cocone set.colim.coprd2$opsecond])
        (SET.FTN$composition [opsecond lang.mor$relation]))
```

- There is a class function ‘`colimiting-cone`’ that maps a pair (of type languages) to its binary coproduct (colimiting binary coproduct cocone). The totality of this function, along with the universality of the comediator type language morphism, implies that a binary coproduct exists for any pair of type languages. The opvertex of the colimiting binary coproduct cocone is a specific *binary coproduct* set. It comes equipped with two *injection* type language morphisms. The binary coproduct and injections are expressed both abstractly in their defining axioms and concretely in the last axiom. The last axiom implicitly ensures that the coproduct is specific.

```
(15) (SET.FTN$function colimiting-cocone)
     (= (SET.FTN$source colimiting-cocone) diagram)
     (= (SET.FTN$target colimiting-cocone) cocone)
     (= (SET.FTN$composition [colimiting-cocone cocone-diagram])
        (SET.FTN$identity diagram))

(16) (SET.FTN$function colimit)
     (SET.FTN$function binary-coproduct)
     (= binary-coproduct colimit)
     (= (SET.FTN$source colimit) diagram)
     (= (SET.FTN$target colimit) lang$language)
     (= colimit (SET.FTN$composition [colimiting-cocone opvertex]))

(17) (SET.FTN$function injection1)
     (= (SET.FTN$source injection1) diagram)
     (= (SET.FTN$target injection1) lang.mor$language-morphism)
     (= (SET.FTN$composition [injection1 lang.mor$source]) language1)
     (= (SET.FTN$composition [injection1 lang.mor$target]) colimit)
     (= injection1 (SET.FTN$composition [colimiting-cocone opfirst]))

(18) (SET.FTN$function injection2)
     (= (SET.FTN$source injection2) diagram)
     (= (SET.FTN$target injection2) lang.mor$language-morphism)
     (= (SET.FTN$composition [injection2 lang.mor$source]) language2)
     (= (SET.FTN$composition [injection2 lang.mor$target]) colimit)
```

```
     (= injection2 (SET.FTN$composition [colimiting-cocone opsecond]))

(19) (= (SET.FTN$composition [colimit lang$reference])
        (SET.FTN$composition [reference-pair set.col.coprd2.mor$coproduct]))
     (forall (?p (diagram ?p))
       (= (arity (colimit ?p))
          (set.ftn$composition
            [(set.col.coprd2.mor$colimit (arity-pair ?p))
             (set.col.coprd2$copairing
               [(set.ftn$power (set.col.coprd2$injection1 (variable-pair ?p)))
                (set.ftn$power (set.col.coprd2$injection2 (variable-pair ?p)))])])))
```

- The following three axioms are the necessary conditions that the variable, entity and relation functors preserve concrete colimits. These explicitly ensure that this colimit is specific – that its variable, entity type and relation type sets are exactly the disjoint unions of the corresponding sets of the pair of type languages. Also, these explicitly ensure that the two coproduct injection infomorphisms are specific – that their variable, entity type and relation type functions are exactly the coproduct injections of the variable, entity type and relation type set pairs of the pair of type languages.

```
(20) (= (SET.FTN$composition [colimiting-cocone variable-cocone])
        (SET.FTN$composition [variable-diagram set.col.coprd2$colimiting-cone]))
     (= (SET.FTN$composition [colimit lang$variable])
        (SET.FTN$composition [variable-diagram set.col.coprd2$colimit]))
     (= (SET.FTN$composition [injection1 lang.mor$variable])
        (SET.FTN$composition [variable-diagram set.col.coprd2$injection1]))
     (= (SET.FTN$composition [injection2 lang.mor$variable])
        (SET.FTN$composition [variable-diagram set.col.coprd2$projection2]))

(21) (= (SET.FTN$composition [colimiting-cocone entity-cocone])
        (SET.FTN$composition [entity-diagram set.col.coprd2$colimiting-cone]))
     (= (SET.FTN$composition [colimit lang$entity])
        (SET.FTN$composition [entity-diagram set.col.coprd2$colimit]))
     (= (SET.FTN$composition [injection1 lang.mor$entity])
        (SET.FTN$composition [entity-diagram set.col.coprd2$injection1]))
     (= (SET.FTN$composition [injection2 lang.mor$entity])
        (SET.FTN$composition [entity-diagram set.col.coprd2$projection2]))

(22) (= (SET.FTN$composition [colimiting-cocone relation-cocone])
        (SET.FTN$composition [relation-diagram set.col.coprd2$colimiting-cone]))
     (= (SET.FTN$composition [colimit lang$relation])
        (SET.FTN$composition [relation-diagram set.col.coprd2$colimit]))
     (= (SET.FTN$composition [injection1 lang.mor$relation])
        (SET.FTN$composition [relation-diagram set.col.coprd2$injection1]))
     (= (SET.FTN$composition [injection2 lang.mor$relation])
        (SET.FTN$composition [relation-diagram set.col.coprd2$projection2]))
```

- For any binary coproduct cocone, there is a *comediator* type language morphism

  $[\boldsymbol{f}_1, \boldsymbol{f}_2] = \langle \mathit{refer}([\boldsymbol{f}_1, \boldsymbol{f}_2]), \mathit{sign}([\boldsymbol{f}_1, \boldsymbol{f}_2]) \rangle : \boldsymbol{L}_1 + \boldsymbol{L}_2 \to \boldsymbol{L}$

  (see Figure 3, where arrows denote infomorphisms) from the binary coproduct of the underlying diagram (pair of type languages) to the opvertex of the cocone. This is the unique type language morphism, which commutes with the opfirst $\boldsymbol{f}_1$ and the opsecond $\boldsymbol{f}_2$. We define this by using the comediators of the underlying variable, entity and relation cocones. Existence and uniqueness represents the universality of the binary coproduct operator.

```
(23) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) cocone)
     (= (SET.FTN$target comediator) lang.mor$language-morphism)
     (= (SET.FTN$composition [comediator lang.mor$source])
        (SET.FTN$composition [cocone-diagram colimit]))
     (= (SET.FTN$composition [comediator lang.mor$target]) opvertex)
     (= (SET.FTN$composition [comediator lang.mor$variable])
        (SET.FTN$composition [variable-cocone set.col.coprd2$comediator]))
     (= (SET.FTN$composition [comediator lang.mor$entity])
        (SET.FTN$composition [entity-cocone set.col.coprd2$comediator]))
     (= (SET.FTN$composition [comediator lang.mor$relation])
        (SET.FTN$composition [relation-cocone set.col.coprd2$comediator]))
```

- It can be verified that the comediator is the unique type language morphism that makes the diagram in Figure 3a commutative.

```
(24) (forall (?s (cocone ?s))
         (= (comediator ?s)
            (the (?f (lang.mor$language-morphism ?f))
                (and (= (lang.mor$composition [(injection1 (cocone-diagram ?s)) ?f])
                        (opfirst ?s))
                     (= (lang.mor$composition [(injection2 (cocone-diagram ?s)) ?f])
                        (opsecond ?s))))))
```

Endorelations and Quotients

`lang.col.endo`

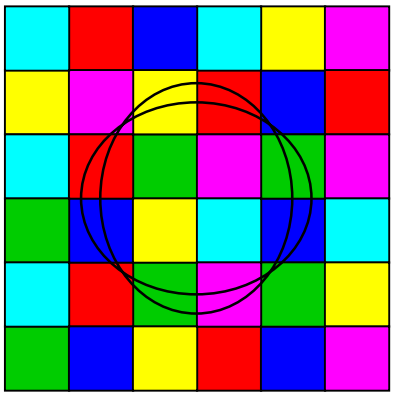


Figure 4: An endorelation *J* on a type language *L*

- Given a type language $\boldsymbol{L} = \langle \mathit{refer}(\boldsymbol{L}), \mathit{sign}(\boldsymbol{L}) \rangle$, a *type language endorelation* is a triple $\boldsymbol{J} = \langle \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{R} \rangle$ consisting of a binary *endorelation* $\boldsymbol{V}$ on variables $V \subseteq \mathit{var}(\boldsymbol{L}) \times \mathit{var}(\boldsymbol{L})$, a binary *endorelation* $\boldsymbol{E}$ on entity types $E \subseteq \mathit{ent}(\boldsymbol{L}) \times \mathit{ent}(\boldsymbol{L})$, and a binary *endorelation* $\boldsymbol{R}$ on relation types $R \subseteq \mathit{rel}(\boldsymbol{L}) \times \mathit{rel}(\boldsymbol{L})$ that satisfy the *fundamental constraints* on references, arity (and equivalently) signatures. Figure 4 illustrates an endorelation. The large square represents the set of variables $\mathit{var}(\boldsymbol{L})$. The (grid) partition into the smaller squares represents the equivalence relation $\equiv_{\boldsymbol{V}}$ on variables. The set of six colors represents the set of entity types $\mathit{ent}(\boldsymbol{L})$. The color of each point in the larger square represents the reference function $\mathit{refer}(\boldsymbol{L}) : \mathit{var}(\boldsymbol{L}) \to \mathit{ent}(\boldsymbol{L})$. The fact that each square is a solid color represents the reference constraint. The two oval shapes outline two subsets of variables that are equivalent with respect to the relation $\equiv_{\#}$. They are equivalent because they intersect the same collection of $\equiv_{\boldsymbol{V}}$-equivalence classes.
  - **reference constraint:** if $(x_1, x_2) \in V$ then $\mathit{refer}(\boldsymbol{L})(x_1) \equiv_{\boldsymbol{E}} \mathit{refer}(\boldsymbol{L})(x_2)$.
  - **arity constraint:** if $(\rho_1, \rho_2) \in R$, then $\mathit{arity}(\boldsymbol{L})(\rho_1) \equiv_{\#} \mathit{arity}(\boldsymbol{L})(\rho_2)$, where $\equiv_{\#} = \wp \equiv_{\boldsymbol{V}}$ is the power equivalence relation $\equiv_{\#} \subseteq \wp\, \mathit{var}(\boldsymbol{L}) \times \wp\, \mathit{var}(\boldsymbol{L})$ defined on subsets as follows: $X_1 \equiv_{\#} X_2$ when
    for each variable $x_1 \in X_1$ there is a variable $x_2 \in X_2$ such that $x_1 \equiv_{\boldsymbol{E}} x_2$, and
    for each variable $x_2 \in X_2$ there is a variable $x_1 \in X_1$ such that $x_1 \equiv_{\boldsymbol{E}} x_2$.
    Note that there may be more than one associated variable – this is not a bijective situation.
- **signature constraint:** if $(\rho_1, \rho_2) \in R$, then $\mathit{sign}(\boldsymbol{L})(\rho_1) \equiv_{\partial} \mathit{sign}(\boldsymbol{L})(\rho_2)$, where $\equiv_{\partial}$ is the equivalence relation on tuple $\equiv_{\partial} \subseteq \mathit{sign}(\mathit{refer}(\boldsymbol{L})) \times \mathit{sign}(\mathit{refer}(\boldsymbol{L}))$ defined as follows: $(\tau_1, \tau_2) \in \equiv_{\partial}$ when

  $\mathit{arity}(\mathit{refer}(\boldsymbol{L}))(\tau_1) \equiv_{\#} \mathit{arity}(\mathit{refer}(\boldsymbol{L}))(\tau_2)$.

  Then $x_1 \in \mathit{arity}(\mathit{refer}(\boldsymbol{L}))(\tau_1)$, $x_2 \in \mathit{arity}(\mathit{refer}(\boldsymbol{L}))(\tau_2)$ and $x_1 \equiv_{\boldsymbol{E}} x_2$ imply that $\tau_1(x_1) \equiv_{\boldsymbol{E}} \tau_2(x_2)$.

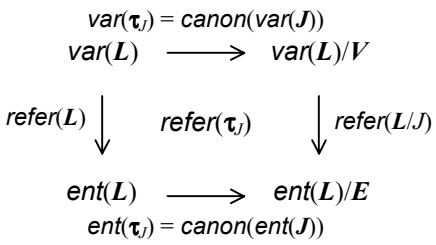
Here $\equiv_{\boldsymbol{V}}$ is the equivalence relation (reflexive, symmetric, transitive closure) generated by $\boldsymbol{V}$, $\equiv_{\boldsymbol{E}}$ is the equivalence relation generated by $\boldsymbol{E}$, and $\equiv_{\boldsymbol{R}}$ is the equivalence relation generated by $\boldsymbol{R}$.

The type language $\boldsymbol{L}$ is called the *base type language* of J – the type language on which J is based. An endorelation is determined by the quadruple consisting of its base and three endorelations.

```
(1) (SET$class endorelation)

(2) (SET.FTN$function language)
    (SET.FTN$function base)
    (= base language)
    (= (SET.FTN$source language) endorelation)
    (= (SET.FTN$target language) lang$language)

(3) (SET.FTN$function variable)
    (= (SET.FTN$source variable) endorelation)
    (= (SET.FTN$target variable) rel.endo$endorelation)
    (= (SET.FTN$composition [variable rel.endo$set])
       (SET.FTN$composition [language lang$variable]))

(4) (SET.FTN$function entity)
    (= (SET.FTN$source entity) endorelation)
    (= (SET.FTN$target entity) rel.endo$endorelation)
    (= (SET.FTN$composition [entity rel.endo$set])
       (SET.FTN$composition [language lang$entity]))

(5) (SET.FTN$function relation)
    (= (SET.FTN$source relation) endorelation)
    (= (SET.FTN$target relation) rel.endo$endorelation)
```

```
      (= (SET.FTN$composition [relation rel.endo$set])
         (SET.FTN$composition [language lang$relation]))

(6) (forall (?j (endorelation ?j))
        (and (forall (?x1 ((lang$variable (language ?j)) ?x1)
                      ?x2 ((lang$variable (language ?j)) ?x2))
                 (=> ((variable ?j) ?x1 ?x2)
                     ((rel.endo$equivalence-closure (entity ?j))
                         ((lang$reference (language ?j)) ?x1)
                         ((lang$reference (language ?j)) ?x2))))
             (forall (?r1 ((lang$relation (language ?j)) ?r1)
                      ?r2 ((lang$relation (language ?j)) ?r2))
                 (=> ((relation ?j) ?r1 ?r2)
                     ((rel.endo$power (rel.endo$equivalence-closure (variable ?j)))
                         ((lang$relation-arity (language ?j)) ?r1)
                         ((lang$relation-arity (language ?j)) ?r2))))))
```

- Often, the relations $\boldsymbol{V}$, $\boldsymbol{E}$ and $\boldsymbol{R}$ are equivalence relations on the variable, entity types and relation types, respectively. However, it is convenient and even important not to require this. In particular, the endorelations define by parallel pairs of type language morphisms (coequalizer diagrams) do not have component equivalence relations. For any relation type $\rho \in \mathit{rel}(\boldsymbol{L})$, write $[\rho]_{\boldsymbol{R}}$ for the $\boldsymbol{R}$-equivalence class of ρ. Same for variables and entity types. A simple inductive proof shows that the triple $\hat{J} = \langle \equiv_V, \equiv_E, \equiv_R \rangle$ is also an endorelation on $\boldsymbol{L}$.



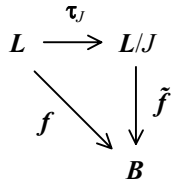


Diagram 1: Canonical Morphism and Universality of the Quotient

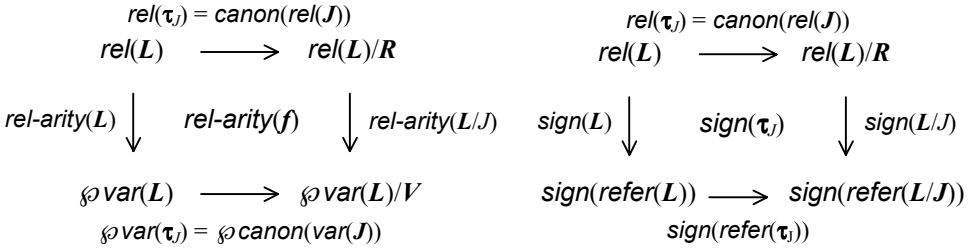


Figure 5: Type Language Quotient and Canonical Morphism - details

- The *quotient* $\boldsymbol{L}/J$ of an endorelation $\boldsymbol{J}$ on a type language $\boldsymbol{L}$ (see Figure 5, where arrows denote set functions, or Diagram 1, where arrows denote type language morphisms) is the type language defined as follows:
  - The set of *variables* is $\mathit{var}(\boldsymbol{L}/\boldsymbol{V}) = \mathit{var}(\boldsymbol{L})/\boldsymbol{V}$.
  - The set of *entity types* is $\mathit{ent}(\boldsymbol{L}/\boldsymbol{V}) = \mathit{ent}(\boldsymbol{L})/\boldsymbol{E}$.
  - The set of *relation types* is $\mathit{rel}(\boldsymbol{L}/\boldsymbol{V}) = \mathit{rel}(\boldsymbol{L})/\boldsymbol{R}$.

  The following definitions follow the diagrams above.
  - The *reference* function

    $$*_{\boldsymbol{L}/J} = \mathit{refer}(\boldsymbol{L}/J) : \mathit{var}(\boldsymbol{L})/\boldsymbol{V} \to \mathit{ent}(\boldsymbol{L})/\boldsymbol{E}$$

    is defined pointwise as follows

$*_{L/J}([x]_V) = [*_L(x)]_E$ for all variables $x \in \mathit{var}(\boldsymbol{L})$.

– The *relation arity* function

$\#_{L/J} = \mathit{rel\text{-}arity}(\boldsymbol{L}/J) : \mathit{rel}(\boldsymbol{L})/\boldsymbol{V} \to \wp\, \mathit{var}(\boldsymbol{L})/\boldsymbol{V}$

is defined pointwise as follows

$\#_{L/J}([\rho]_R) = \{[x]_V \mid x \in \#_L(\rho)\}$ for all relation types $\rho \in \mathit{rel}(\boldsymbol{L})$.

In Figure 4 this is the collection of equivalence classes that the arity $\#_L(\rho)$ intersects.

– The *signature* function

$\partial_{L/J} = \mathit{sign}(\boldsymbol{L}/J) : \mathit{rel}(\boldsymbol{L})/\boldsymbol{V} \to \mathit{sign}(\mathit{refer}(\boldsymbol{L}/\boldsymbol{J}))$

is defined pointwise as follows

$\partial_{L/J}([\rho]_R) = \mathit{sign}(\mathit{refer}(*_J))(\partial_L(\rho))$ for all relation types $\rho \in \mathit{rel}(\boldsymbol{L})$.

○ There is a *canonical quotient type language morphism*

$\boldsymbol{\tau}_J = \langle \mathit{refer}(\boldsymbol{\tau}_J), \mathit{rel\text{-}arity}(\boldsymbol{\tau}_J), \mathit{sign}(\boldsymbol{\tau}_J) \rangle : \boldsymbol{L} \to \boldsymbol{L}/J$

whose variable function is the canonical surjection

$\mathit{var}(\boldsymbol{\tau}_J) = \mathit{canon}(\mathit{var}(\boldsymbol{J})) = [\text{-}]_V : \mathit{var}(\boldsymbol{L}) \to \mathit{var}(\boldsymbol{L})/\boldsymbol{V}$,

whose entity type function is the canonical surjection

$\mathit{ent}(\boldsymbol{\tau}_J) = \mathit{canon}(\mathit{ent}(\boldsymbol{J})) = [\text{-}]_E : \mathit{ent}(\boldsymbol{L}) \to \mathit{ent}(\boldsymbol{L})/\boldsymbol{E}$,

and whose relation type function is the canonical surjection

$\mathit{rel}(\boldsymbol{\tau}_J) = \mathit{canon}(\mathit{rel}(\boldsymbol{J})) = [\text{-}]_R : \mathit{rel}(\boldsymbol{L}) \to \mathit{rel}(\boldsymbol{L})/\boldsymbol{V}$.

The fundamental property for this infomorphism is trivial, given the definition of the quotient type language above.

```
(7) (SET.FTN$function quotient)
    (= (SET.FTN$source coquotient) endorelation)
    (= (SET.FTN$target coquotient) lang$language)
    (forall (?j (endorelation ?j))
        (and (= (lang$variable (quotient ?j))
                (rel.endo$quotient (rel.endo$equivalence-closure (variable ?j))))
             (= (lang$entity (quotient ?j))
                (rel.endo$quotient (rel.endo$equivalence-closure (entity ?j))))
             (= (lang$relation (quotient ?j))
                (rel.endo$quotient (rel.endo$equivalence-closure (relation ?j))))
             (= (set.ftn$composition
                    [(rel.endo$canon (variable ?j)) (lang$reference (quotient ?j))])
                (set.ftn$composition
                    [(lang$reference (language ?j)) (rel.endo$canon (entity ?j))]))
             (= (set.ftn$composition
                    [(rel.endo$canon (relation ?j)) (lang$relation-arity (quotient ?j))])
                (set.ftn$composition
                    [(lang$relation-arity (language ?j))
                     (set.ftn$power (rel.endo$canon (variable ?j)))]))
             (= (set.ftn$composition
                    [(rel.endo$canon (relation ?j)) (lang$signature (quotient ?j))])
                (set.ftn$composition
                    [(lang$signature (language ?j)) (set.qtt$signature (reference ?j))])))))

(8) (SET.FTN$function canon)
    (= (SET.FTN$source canon) endorelation)
    (= (SET.FTN$target canon) lang.mor$language-morphism)
    (= (SET.FTN$composition [canon lang.mor$variable])
       (SET.FTN$composition [variable rel.endo$canon]))
    (= (SET.FTN$composition [canon lang.mor$entity])
       (SET.FTN$composition [entity rel.endo$canon]))
    (= (SET.FTN$composition [canon lang.mor$relation])
       (SET.FTN$composition [relation rel.endo$canon]))
```

- Let $\boldsymbol{J} = \langle \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{R} \rangle$ be an endorelation on a type language $\boldsymbol{L}$. A type language morphism $\boldsymbol{f} : \boldsymbol{L} \to \boldsymbol{K}$ *respects* $\boldsymbol{J}$ when:
  - for any two variables $x_0, x_1 \in \mathit{var}(\boldsymbol{A})$, if $(x_0, x_1) \in V$ then $\mathit{var}(\boldsymbol{f})(x_0) = \mathit{var}(\boldsymbol{f})(x_1)$,
  - for any two entity types $\alpha_0, \alpha_1 \in \mathit{ent}(\boldsymbol{A})$, if $(\alpha_0, \alpha_1) \in E$ then $\mathit{ent}(\boldsymbol{f})(\alpha_0) = \mathit{ent}(\boldsymbol{f})(\alpha_1)$, and
  - for any two relation types $\rho_0, \rho_1 \in \mathit{rel}(\boldsymbol{A})$, if $(\rho_0, \rho_1) \in R$ then $\mathit{rel}(\boldsymbol{f})(\rho_0) = \mathit{rel}(\boldsymbol{f})(\rho_1)$.

  If $\boldsymbol{f} : \boldsymbol{L} \to \boldsymbol{K}$ respects $\boldsymbol{J}$, then it factors uniquely through the quotient: there is a unique type language morphism $\tilde{\boldsymbol{f}} : \boldsymbol{L}/\boldsymbol{J} \to \boldsymbol{K}$ such that $\tau_J \circ \tilde{\boldsymbol{f}} = \boldsymbol{f}$. This means that:

  $[]_V \cdot \mathit{var}(\tilde{\boldsymbol{f}}) = \mathit{var}(\boldsymbol{f})$, or $\mathit{var}(\tilde{\boldsymbol{f}})([x]_V) = \mathit{var}(\boldsymbol{f})(x)$ for all variables $x \in \mathit{var}(\boldsymbol{L})$; and

  $[]_E \cdot \mathit{ent}(\tilde{\boldsymbol{f}}) = \mathit{ent}(\boldsymbol{f})$, or $\mathit{ent}(\tilde{\boldsymbol{f}})([\alpha]_E) = \mathit{ent}(\boldsymbol{f})(\alpha)$ for all entity types $\alpha \in \mathit{ent}(\boldsymbol{L})$; and

  $[]_R \cdot \mathit{rel}(\tilde{\boldsymbol{f}}) = \mathit{rel}(\boldsymbol{f})$, or $\mathit{rel}(\tilde{\boldsymbol{f}})([\rho]_E) = \mathit{rel}(\boldsymbol{f})(\rho)$ for all relation types $\rho \in \mathit{rel}(\boldsymbol{L})$.

  The respectful constraints on $\boldsymbol{f}$ imply that $\tilde{\boldsymbol{f}}$ is well-defined and is a type language morphism. The canonical quotient hypergraph morphism $\tau_J : \boldsymbol{A}/J \to \boldsymbol{A}$ respects J, and its unique morphism is the identity.

```
(9) (SET.LIM.PBK$opspan matches-opspan)
    (= (SET.LIM.PBK$class1 matches-opspan) lang.mor$language-morphism)
    (= (SET.LIM.PBK$class2 matches-opspan) endorelation)
    (= (SET.LIM.PBK$opvertex matches-opspan) lang$language)
    (= (SET.LIM.PBK$first matches-opspan) lang.mor$source)
    (= (SET.LIM.PBK$second matches-opspan) language)

(10) (REL$relation matches)
     (= (REL$class1 matches) lang.mor$language-morphism)
     (= (REL$class2 matches) endorelation)
     (= (REL$extent matches) (SET.LIM.PBK$pullback respects-opspan))

(11) (REL$relation respects)
     (= (REL$class1 respects) lang.mor$language-morphism)
     (= (REL$class2 respects) endorelation)
     (REL$subrelation respects matches)
     (forall (?f (lang.mor$language-morphism ?f) ?j (endorelation ?j) (matches ?f ?j))
         (<=> (respects ?f ?j)
              (and (forall (?x0 ((lang$variable (language ?j)) ?x0)
                           ?x1 ((lang$variable (language ?j)) ?x1)
                           ((variable ?j) ?r0 ?r1))
                       (= ((lang.mor$variable ?f) ?x0) ((lang.mor$variable ?f) ?x1)))
                   (forall (?e0 ((lang$entity (language ?j)) ?e0)
                            ?e1 ((lang$entity (language ?j)) ?e1)
                            ((entity ?j) ?e0 ?e1))
                       (= ((lang.mor$entity ?f) ?e0) ((lang.mor$entity ?f) ?e1)))
                   (forall (?r0 ((lang$relation (language ?j)) ?r0)
                            ?r1 ((lang$relation (language ?j)) ?r1)
                            ((relation ?j) ?r0 ?r1))
                       (= ((lang.mor$relation ?f) ?r0) ((lang.mor$relation ?f) ?r1))))))
```

Proposition. *For every endorelation* $\boldsymbol{J}$ *on a type language* $\boldsymbol{L}$ *and every type language morphism* $\boldsymbol{f} : \boldsymbol{L} \to \boldsymbol{K}$ *that respects* $\boldsymbol{J}$*, there is a unique comediating type language morphism* $\tilde{\boldsymbol{f}} : \boldsymbol{L}/\boldsymbol{J} \to \boldsymbol{K}$ *such that* $\boldsymbol{\tau}_J \circ \tilde{\boldsymbol{f}} = \boldsymbol{f}$.

Based on this proposition, a definite description is used to define a *comediator* function (Diagram 2) that maps a pair $(\boldsymbol{f}, \boldsymbol{J})$ consisting of an endorelation and a respectful type language morphism to their comediator $\tilde{\boldsymbol{f}}$.

```
(12) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) (REL$extent respects))
     (= (SET.FTN$target comediator) lang.mor$language-morphism)
     (forall (?f (lang.mor$language-morphism ?f) ?j (endorelation ?j) (respects ?f ?j))
         (= (comediator [?f ?j])
            (the (?ft (lang.mor$language-morphism ?ft))
                 (and (= (lang.mor$source ?ft) (quotient ?j))
                      (= (lang.mor$target ?ft) (lang.mor$target ?f))
                      (= (lang.mor$composition [(canon ?j) ?ft]) ?f))))))
```

Coequalizers

lang.col.coeq

A *coequalizer* is a finite colimit in the category Language for a diagram of shape *parallel-pair* = $\bullet \rightrightarrows \bullet$. Such a diagram is called a *parallel pair* of type language morphisms.

- A *parallel pair* (see Figure 6, where arrows denote type language morphisms) is the appropriate base diagram for a coequalizer. Each parallel pair consists of a pair of type language morphisms called *language-morphism1* and *language-morphism2* that share the same *source* and *target* type languages. We use either the generic term 'diagram' or the specific term 'parallel-pair' to denote the *parallel pair* class. Parallel pairs are determined by their two component type language morphisms.

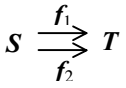


Figure 6: Parallel Pair

```
(1) (SET$class diagram)
    (SET$class parallel-pair)
    (= parallel-pair diagram)

(2) (SET.FTN$function source)
    (= (SET.FTN$source source) diagram)
    (= (SET.FTN$target source) lang$language)

(3) (SET.FTN$function target)
    (= (SET.FTN$source target) diagram)
    (= (SET.FTN$target target) lang$language)

(4) (SET.FTN$function language-morphism1)
    (= (SET.FTN$source language-morphism1) diagram)
    (= (SET.FTN$target language-morphism1) lang.mor$language-morphism)
    (= (SET.FTN$composition [language-morphism1 lang.mor$source]) source)
    (= (SET.FTN$composition [language-morphism1 lang.mor$target]) target)

(5) (SET.FTN$function language-morphism2)
    (= (SET.FTN$source language-morphism2) diagram)
    (= (SET.FTN$target language-morphism2) lang.mor$language-morphism)
    (= (SET.FTN$composition [language-morphism2 lang.mor$source]) source)
    (= (SET.FTN$composition [language-morphism2 lang.mor$target]) target)

    (forall (?p (diagram ?p) ?q (diagram ?q))
        (=> (and (= (language-morphism1 ?p) (language-morphism1 ?q))
                 (= (language-morphism2 ?p) (language-morphism2 ?q)))
            (= ?p ?q)))
```

- There is a *variable parallel pair* or *variable diagram* function, which maps a parallel pair of type language morphisms to the underlying (set.col.coeq) parallel pair of variable functions. Similarly, there is an *entity type parallel pair* or *entity type diagram* function, which maps a parallel pair of type language morphisms to the underlying parallel pair of entity type functions. And there is a *relation type parallel pair* or *relation type diagram* function, which maps a parallel pair of type language morphisms to the underlying parallel pair of relation type functions.

```
(6) (SET.FTN$function variable-diagram)
    (= (SET.FTN$source variable-diagram) diagram)
    (= (SET.FTN$target variable-diagram) set.col.coeq$diagram)
    (= (SET.FTN$composition [variable-diagram set.col.coeq$source])
       (SET.FTN$composition [source lang$variable]))
    (= (SET.FTN$composition [variable-diagram set.col.coeq$target])
       (SET.FTN$composition [target lang$variable]))
    (= (SET.FTN$composition [variable-diagram set.col.coeq$function1])
       (SET.FTN$composition [language-morphism1 lang.mor$variable]))
    (= (SET.FTN$composition [variable-diagram set.col.coeq$function2])
       (SET.FTN$composition [language-morphism2 lang.mor$variable]))

(7) (SET.FTN$function entity-diagram)
    (= (SET.FTN$source entity-diagram) diagram)
    (= (SET.FTN$target entity-diagram) set.col.coeq$diagram)
```

```
      (= (SET.FTN$composition [entity-diagram set.col.coeq$source])
         (SET.FTN$composition [source lang$entity]))
      (= (SET.FTN$composition [entity-diagram set.col.coeq$target])
         (SET.FTN$composition [target lang$entity]))
      (= (SET.FTN$composition [entity-diagram set.col.coeq$function1])
         (SET.FTN$composition [language-morphism1 lang.mor$entity]))
      (= (SET.FTN$composition [entity-diagram set.col.coeq$function2])
         (SET.FTN$composition [language-morphism2 lang.mor$entity]))

(8) (SET.FTN$function relation-diagram)
    (= (SET.FTN$source relation-diagram) diagram)
    (= (SET.FTN$target relation-diagram) set.col.coeq$diagram)
    (= (SET.FTN$composition [relation-diagram set.col.coeq$source])
       (SET.FTN$composition [source lang$relation]))
    (= (SET.FTN$composition [relation-diagram set.col.coeq$target])
       (SET.FTN$composition [target lang$relation]))
    (= (SET.FTN$composition [relation-diagram set.col.coeq$function1])
       (SET.FTN$composition [language-morphism1 lang.mor$relation]))
    (= (SET.FTN$composition [relation-diagram set.col.coeq$function2])
       (SET.FTN$composition [language-morphism2 lang.mor$relation]))
```

- The information in a coequalizer diagram (parallel pair of type language morphisms) is equivalently expressed as an *endorelation* based at the target type language of the parallel pair, whose variable endorelation is the coequalizer endorelation of the variable diagram, whose entity type endorelation is the coequalizer endorelation of the entity type diagram, and whose relation type endorelation is the coequalizer endorelation of the relation type diagram.

```
(9) (SET.FTN$function endorelation)
    (= (SET.FTN$source endorelation) diagram)
    (= (SET.FTN$target endorelation) lang.col.endo$endorelation)
    (= (SET.FTN$composition [endorelation lang.col.endo$language]) target)
    (= (SET.FTN$composition [endorelation lang.col.endo$variable])
       (SET.FTN$composition [variable-diagram set.col.coeq$endorelation]))
    (= (SET.FTN$composition [endorelation lang.col.endo$entity])
       (SET.FTN$composition [entity-diagram set.col.coeq$endorelation]))
    (= (SET.FTN$composition [endorelation lang.col.endo$relation])
       (SET.FTN$composition [relation-diagram set.col.coeq$endorelation]))
```

- The notion of a *coequalizer cocone* is used to specify and axiomatize coequalizers. Each coequalizer cocone is situated under a coequalizer *diagram* (parallel pair of type language morphisms). And each coequalizer cocone (see Figure 7, where arrows denote type language morphisms) has an *opvertex* type language, and a *type language morphism* whose source type language is the target type language of the type language morphisms in the underlying cocone diagram (parallel-pair) and whose target type language is the opvertex. Since Figure 7 is a commutative diagram, the composite type language morphism is not needed.

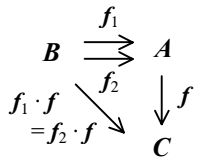


Figure 7: Coequalizer Cocone

```
(10) (SET$class cocone)

(11) (SET.FTN$function cocone-diagram)
     (= (SET.FTN$source cocone-diagram) cocone)
     (= (SET.FTN$target cocone-diagram) diagram)

(12) (SET.FTN$function opvertex)
     (= (SET.FTN$source opvertex) cocone)
     (= (SET.FTN$target opvertex) lang$language)

(13) (SET.FTN$function language-morphism)
     (= (SET.FTN$source language-morphism) cocone)
     (= (SET.FTN$target language-morphism) lang.mor$language-morphism)
     (= (SET.FTN$composition [language-morphism lang.mor$source])
        (SET.FTN$composition [cocone-diagram target]))
     (= (SET.FTN$composition [language-morphism lang.mor$target]) opvertex)

(14) (forall (?s (cocone ?s))
        (and (= (lang.mor$composition
                    [(language-morphism1 (cocone-diagram ?s)) (language-morphism ?s)])
                (lang.mor$composition
```

```
                    [(language-morphism2 (cocone-diagram ?s)) (language-morphism ?s)])))))
```

- The *variable* coequalizer cocone function maps a coequalizer cocone of type languages and type language morphisms to the underlying coequalizer cocone of variable sets and variable set functions. The *entity* type coequalizer cocone function maps a coequalizer cocone of type languages and type language morphisms to the underlying coequalizer cocone of entity type sets and entity type set functions. The *relation* type coequalizer cocone function maps a coequalizer cocone of type languages and type language morphisms to the underlying coequalizer cocone of relation type sets and relation type set functions.

```
(15) (SET.FTN$function variable-cocone)
     (= (SET.FTN$source variable-cocone) cocone)
     (= (SET.FTN$target variable-cocone) set.col.coeq$cocone)
     (= (SET.FTN$composition [variable-cocone set.col.coeq$cocone-diagram])
        (SET.FTN$composition [cocone-diagram variable-diagram]))
     (= (SET.FTN$composition [variable-cocone set.col.coeq$opvertex])
        (SET.FTN$composition [opvertex lang$variable]))
     (= (SET.FTN$composition [variable-cocone set.col.coeq$function])
        (SET.FTN$composition [language-morphism lang.mor$variable]))

(16) (SET.FTN$function entity-cocone)
     (= (SET.FTN$source entity-cocone) cocone)
     (= (SET.FTN$target entity-cocone) set.col.coeq$cocone)
     (= (SET.FTN$composition [entity-cocone set.col.coeq$cocone-diagram])
        (SET.FTN$composition [cocone-diagram entity-diagram]))
     (= (SET.FTN$composition [entity-cocone set.col.coeq$opvertex])
        (SET.FTN$composition [opvertex lang$entity]))
     (= (SET.FTN$composition [entity-cocone set.col.coeq$function])
        (SET.FTN$composition [language-morphism lang.mor$entity]))

(17) (SET.FTN$function relation-cocone)
     (= (SET.FTN$source relation-cocone) cocone)
     (= (SET.FTN$target relation-cocone) set.col.coeq$cocone)
     (= (SET.FTN$composition [relation-cocone set.col.coeq$cocone-diagram])
        (SET.FTN$composition [cocone-diagram relation-diagram]))
     (= (SET.FTN$composition [relation-cocone set.col.coeq$opvertex])
        (SET.FTN$composition [opvertex lang$relation]))
     (= (SET.FTN$composition [relation-cocone set.col.coeq$function])
        (SET.FTN$composition [language-morphism lang.mor$relation]))
```

- It is important to observe that the type language morphism in a cocone respects the endorelation of the underlying diagram of the cocone. This fact is needed to help define the comediator of a cocone.

```
(18) (forall (?s (cocone ?s))
         (lang.col.endo$respects (language-morphism ?s) (endorelation (cocone-diagram ?s))))
```

- There is a class function '`limiting-cone`' that maps a parallel pair (of type language morphisms) to its coequalizer (colimiting coequalizer cocone) (see Figure 8, where arrows denote type language morphisms). The totality of this function, along with the universality of the comediator type language morphism, implies that a coequalizer exists for any parallel pair of type language morphisms. The opvertex of the colimiting coequalizer cocone is a specific *coequalizer* type language. It comes equipped with a *canon* type language morphism. The coequalizer and canon are expressed both abstractly, and, in the last axiom, as the coquotient and canon of the endorelation of the diagram.

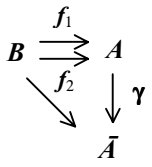


Figure 8: Colimiting Cocone

```
(19) (SET.FTN$function colimiting-cocone)
     (= (SET.FTN$source colimiting-cocone) diagram)
     (= (SET.FTN$target colimiting-cocone) cocone)
     (= (SET.FTN$composition [colimiting-cocone cocone-diagram])
        (SET.FTN$identity diagram))

(20) (SET.FTN$function colimit)
     (SET.FTN$function coequalizer)
     (= coequalizer colimit)
     (= (SET.FTN$source colimit) diagram)
     (= (SET.FTN$target colimit) lang$language)
```

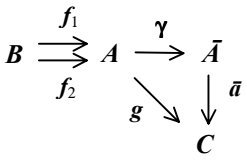
```
     (= colimit (SET.FTN$composition [colimiting-cocone opvertex]))

(21) (SET.FTN$function canon)
     (= (SET.FTN$source canon) diagram)
     (= (SET.FTN$target canon) lang.mor$language-morphism)
     (= (SET.FTN$composition [canon lang.mor$source]) target)
     (= (SET.FTN$composition [canon lang.mor$target]) colimit)
     (= canon (SET.FTN$composition [colimiting-cocone infomorphism]))

(22) (= colimit (SET.FTN$composition [endorelation lang.col.endo$coquotient]))
     (= canon (SET.FTN$composition [endorelation lang.col.endo$canon]))
```

- The following three axioms are the necessary conditions that the variable, entity type and relation type functors preserve concrete colimits. These ensure that both this coequalizer type language and its canon type language morphism are specific.

```
(23) (= (SET.FTN$composition [colimiting-cocone variable-cocone])
        (SET.FTN$composition [variable-diagram set.col.coeq$colimiting-cocone]))
     (= (SET.FTN$composition [colimit lang$variable])
        (SET.FTN$composition [variable-diagram set.col.coeq$colimit]))
     (= (SET.FTN$composition [canon lang.mor$variable])
        (SET.FTN$composition [variable-diagram set.col.coeq$canon]))

(24) (= (SET.FTN$composition [colimiting-cocone entity-cocone])
        (SET.FTN$composition [entity-diagram set.col.coeq$colimiting-cocone]))
     (= (SET.FTN$composition [colimit lang$entity])
        (SET.FTN$composition [entity-diagram set.col.coeq$colimit]))
     (= (SET.FTN$composition [canon lang.mor$entity])
        (SET.FTN$composition [entity-diagram set.col.coeq$canon]))

(25) (= (SET.FTN$composition [colimiting-cocone relation-cocone])
        (SET.FTN$composition [relation-diagram set.col.coeq$colimiting-cocone]))
     (= (SET.FTN$composition [colimit lang$relation])
        (SET.FTN$composition [relation-diagram set.col.coeq$colimit]))
     (= (SET.FTN$composition [canon lang.mor$relation])
        (SET.FTN$composition [relation-diagram set.col.coeq$canon]))
```

- The *comediator* type language morphism, from the coequalizer of a parallel pair of type language morphisms to the opvertex of a cocone under the parallel pair (see Figure 9, where arrows denote type language morphisms), is the unique type language morphism that commutes with cocone type language morphisms. This is defined abstractly by using a definite description, and is defined concretely as the comediator of the associated (language-morphism, endorelation) pair.

Figure 9: Comediator




```
(26) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) cocone)
     (= (SET.FTN$target comediator) lang.mor$language-morphism)
     (= (SET.FTN$composition [comediator lang.mor$source])
        (SET.FTN$composition [cocone-diagram colimit]))
     (= (SET.FTN$composition [comediator lang.mor$target]) opvertex)
     (forall (?s (cocone ?s))
         (= (comediator ?s)
            (the (?m (lang.mor$language-morphism ?m))
                 (= (composition [(canon (cocone-diagram ?s)) ?m])
                    (language-morphism ?s))))))

(27) (forall (?s (cocone ?s))
         (= (comediator ?s)
            (lang.col.endo$comediator
                [(language-morphism ?s) (endorelation (cocone-diagram ?s))]))))
```

Pushouts

lang.col.psh

A *pushout* is a finite colimit in the category Language for a diagram of shape *span* = • ← • → •. Such a diagram (of type languages and type language morphisms) is called an *span* (see Figure 10, where arrows denote type language morphisms)

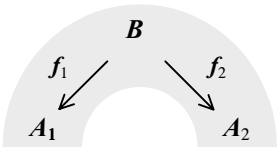


Figure 10: Pushout Diagram = Span

- A *span* is the appropriate base diagram for a pushout. Each span consists of a pair of type language morphisms called *first* and *second*. These are required to have a common source type language called the *vertex*. We use either the generic term '`diagram`' or the specific term '`span`' to denote the *span* class. A span is the special case of a general diagram whose shape is the graph that is also named *span*. Spans are determined by their pair of component type language morphisms.

```
(1) (SET$class diagram)
    (SET$class span)
    (= span diagram)

(2) (SET.FTN$function language1)
    (= (SET.FTN$source language1) diagram)
    (= (SET.FTN$target language1) lang$language)

(3) (SET.FTN$function language2)
    (= (SET.FTN$source language2) diagram)
    (= (SET.FTN$target language2) lang$language)

(4) (SET.FTN$function vertex)
    (= (SET.FTN$source vertex) diagram)
    (= (SET.FTN$target vertex) lang$language)

(5) (SET.FTN$function first)
    (= (SET.FTN$source first) diagram)
    (= (SET.FTN$target first) lang.mor$language-morphism)
    (= (SET.FTN$composition [first lang.mor$source]) vertex)
    (= (SET.FTN$composition [first lang.mor$target]) language1)

(6) (SET.FTN$function second)
    (= (SET.FTN$source second) diagram)
    (= (SET.FTN$target second) lang.mor$language-morphism)
    (= (SET.FTN$composition [second lang.mor$source]) vertex)
    (= (SET.FTN$composition [second lang.mor$target]) language2)

    (forall (?r1 (diagram ?r1) ?r2 (diagram ?r2))
        (=> (and (= (first ?r1) (first ?r2))
                 (= (second ?r1) (second ?r2)))
            (= ?r1 ?r2)))
```

- The *pair* of target type languages (suffixing discrete diagram) underlying any span is named. This construction is derived from the fact that the pair shape is a subshape of the span shape.

```
(7) (SET.FTN$function pair)
    (= (SET.FTN$source pair) diagram)
    (= (SET.FTN$target pair) lang.col.coprd2$diagram)
    (= (SET.FTN$composition [pair lang.col.coprd2$language1]) language1)
    (= (SET.FTN$composition [pair lang.col.coprd2$language2]) language2)
```

- Every span has an opposite.

```
(8) (SET.FTN$function opposite)
    (= (SET.FTN$source opposite) span)
    (= (SET.FTN$target opposite) span)
    (= (SET.FTN$composition [opposite language1]) language2)
    (= (SET.FTN$composition [opposite language2]) language1)
    (= (SET.FTN$composition [opposite vertex]) vertex)
    (= (SET.FTN$composition [opposite first]) second)
    (= (SET.FTN$composition [opposite second]) first)
```

- The opposite of the opposite is the original opspan – the following theorem can be proven.

```
(9) (= (SET.FTN$composition [opposite opposite])
       (SET.FTN$identity span))
```

- The *variable span* or *variable diagram* function maps a span of type language morphisms to the underlying (`set.col.psh`) span of variable functions. The *entity type span* or *entity type diagram* function maps a span of type language morphisms to the underlying (`set.col.psh`) span of entity type functions. The *relation type span* or *relation type diagram* function maps a span of type language morphisms to the underlying (`set.col.psh`) span of relation type functions.

```
(10) (SET.FTN$function variable-diagram)
     (= (SET.FTN$source variable-diagram) diagram)
     (= (SET.FTN$target variable-diagram) set.col.psh$diagram)
     (= (SET.FTN$composition [variable-diagram set.col.psh$set1])
        (SET.FTN$composition [language1 lang$variable]))
     (= (SET.FTN$composition [variable-diagram set.col.psh$set2])
        (SET.FTN$composition [language2 lang$variable]))
     (= (SET.FTN$composition [variable-diagram set.col.psh$vertex])
        (SET.FTN$composition [vertex lang$variable]))
     (= (SET.FTN$composition [variable-diagram set.col.psh$first])
        (SET.FTN$composition [first lang.mor$variable]))
     (= (SET.FTN$composition [variable-diagram set.col.psh$second])
        (SET.FTN$composition [second lang.mor$variable]))

(11) (SET.FTN$function entity-diagram)
     (= (SET.FTN$source entity-diagram) diagram)
     (= (SET.FTN$target entity-diagram) set.col.psh$diagram)
     (= (SET.FTN$composition [entity-diagram set.col.psh$set1])
        (SET.FTN$composition [language1 lang$entity]))
     (= (SET.FTN$composition [entity-diagram set.col.psh$set2])
        (SET.FTN$composition [language2 lang$entity]))
     (= (SET.FTN$composition [entity-diagram set.col.psh$vertex])
        (SET.FTN$composition [vertex lang$entity]))
     (= (SET.FTN$composition [entity-diagram set.col.psh$first])
        (SET.FTN$composition [first lang.mor$entity]))
     (= (SET.FTN$composition [entity-diagram set.col.psh$second])
        (SET.FTN$composition [second lang.mor$entity]))

(12) (SET.FTN$function relation-diagram)
     (= (SET.FTN$source relation-diagram) diagram)
     (= (SET.FTN$target relation-diagram) set.col.psh$diagram)
     (= (SET.FTN$composition [relation-diagram set.col.psh$set1])
        (SET.FTN$composition [language1 lang$relation]))
     (= (SET.FTN$composition [relation-diagram set.col.psh$set2])
        (SET.FTN$composition [language2 lang$relation]))
     (= (SET.FTN$composition [relation-diagram set.col.psh$vertex])
        (SET.FTN$composition [vertex lang$relation]))
     (= (SET.FTN$composition [relation-diagram set.col.psh$first])
        (SET.FTN$composition [first lang.mor$relation]))
     (= (SET.FTN$composition [relation-diagram set.col.psh$second])
        (SET.FTN$composition [second lang.mor$relation]))
```

- The *parallel pair* or *coequalizer diagram* function maps a span of type language morphisms to the associated (`lang.col.coeq`) parallel pair of type language morphisms (see Figure 11, where arrows denote type language morphisms), which are the composite of the first and second type language morphisms of the span with the coproduct injection type language morphisms for the binary coproduct of the component languages in the span. The coequalizer and canon of the associated parallel pair will be used to define the pushout.

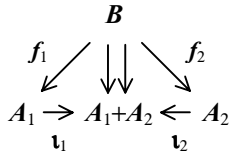


Figure 11: Coequalizer Diagram

```
(13) (SET.FTN$function coequalizer-diagram)
     (SET.FTN$function parallel-pair)
     (= parallel-pair coequalizer-diagram)
     (= (SET.FTN$source coequalizer-diagram) diagram)
     (= (SET.FTN$target coequalizer-diagram) lang.col.coeq$diagram)
     (= (SET.FTN$composition [coequalizer-diagram lang.col.coeq$source]) vertex)
```

```
(= (SET.FTN$composition [coequalizer-diagram lang.col.coeq$target])
   (SET.FTN$composition [pair lang.col.coprd2$binary-coproduct]))
(forall (?r (diagram ?r))
    (and (= (lang.col.coeq$infomorphism1 (coequalizer-diagram ?r))
            (lang.mor$composition
                [(first ?r) (lang.col.coprd2$injection1 (pair ?r))]))
         (= (lang.col.coeq$infomorphism2 (coequalizer-diagram ?r))
            (lang.mor$composition
                [(second ?r) (lang.col.coprd2$injection2 (pair ?r))])))))
```

- There are two important categorical identities (not just isomorphisms) that relate (1) the underlying instance limit and (2) the underlying type colimit of the coequalizer of the parallel pair of any span to (1′) the limit (equalizer) of the equalizer diagram of the underlying instance opspan and (2′) the colimit (coequalizer) of the coequalizer diagram of the underlying type span. These identities are assumed in the definition of the colimiting cocone below.

```
(14) (= (SET.FTN$composition [coequalizer-diagram lang.col.coeq$variable-diagram])
        (SET.FTN$composition [variable-diagram set.col.coeq$coequalizer-diagram]))

(15) (= (SET.FTN$composition [coequalizer-diagram lang.col.coeq$entity-diagram])
        (SET.FTN$composition [entity-diagram set.col.coeq$coequalizer-diagram]))

(16) (= (SET.FTN$composition [coequalizer-diagram lang.col.coeq$relation-diagram])
        (SET.FTN$composition [relation-diagram set.col.coeq$coequalizer-diagram]))

(17) (= (SET.FTN$composition [(SET.FTN$composition
                [coequalizer-diagram
                 lang.col.coeq$colimiting-cocone])
                 lang.col.coeq$variable-diagram])
        (SET.FTN$composition [(SET.FTN$composition
                [variable-diagram
                 lang.col.coeq$coequalizer-diagram])
                 set.col.coeq$colimiting-cocone]))

(18) (= (SET.FTN$composition [(SET.FTN$composition
                [coequalizer-diagram
                 lang.col.coeq$colimiting-cocone])
                 lang.col.coeq$entity-diagram])
        (SET.FTN$composition [(SET.FTN$composition
                [entity-diagram
                 lang.col.coeq$coequalizer-diagram])
                 set.col.coeq$colimiting-cocone]))

(19) (= (SET.FTN$composition [(SET.FTN$composition
                [coequalizer-diagram
                 lang.col.coeq$colimiting-cocone])
                 lang.col.coeq$relation-diagram])
        (SET.FTN$composition [(SET.FTN$composition
                [relation-diagram
                 lang.col.coeq$coequalizer-diagram])
                 set.col.coeq$colimiting-cocone]))
```

- *Pushout cocones* are used to specify and axiomatize pushouts. Each pushout cocone (Figure 12, where arrows denote type language morphisms) has an underlying *diagram* (the shaded part of Figure 12), an *opvertex* type language ***A***, and a pair of type language morphisms called *opfirst* and *opsecond*, whose common target type language is the opvertex and whose source type languages are the target type languages of the type language morphisms in the span. The opfirst and opsecond type language morphisms form a commutative diagram with the span. A pushout cocone is the very special case of a colimiting cocone under a span. The term '`cocone`' denotes the *pushout cocone* class. The term '`cocone-diagram`' represents the underlying diagram.

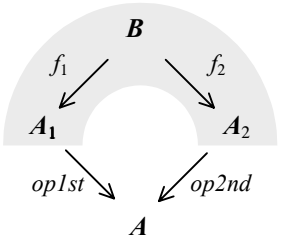


Figure 12: Pushout Cocone

```
(20) (SET$class cocone)

(21) (SET.FTN$function cocone-diagram)
```

```
     (= (SET.FTN$source cocone-diagram) cocone)
     (= (SET.FTN$target cocone-diagram) diagram)

(22) (SET.FTN$function opvertex)
     (= (SET.FTN$source opvertex) cocone)
     (= (SET.FTN$target opvertex) lang$language)

(23) (SET.FTN$function opfirst)
     (= (SET.FTN$source opfirst) cocone)
     (= (SET.FTN$target opfirst) lang.mor$language-morphism)
     (= (SET.FTN$composition [opfirst lang.mor$source])
        (SET.FTN$composition [cocone-diagram classification1]))
     (= (SET.FTN$composition [opfirst lang.mor$target]) opvertex)

(24) (SET.FTN$function opsecond)
     (= (SET.FTN$source opsecond) cocone)
     (= (SET.FTN$target opsecond) lang.mor$language-morphism)
     (= (SET.FTN$composition [opsecond lang.mor$source])
        (SET.FTN$composition [cocone-diagram classification2]))
     (= (SET.FTN$composition [opsecond lang.mor$target]) opvertex)

(25) (forall (?s (cocone ?s))
        (= (lang.mor$composition [(first (cocone-diagram ?s)) (opfirst ?s)])
           (lang.mor$composition [(second (cocone-diagram ?s)) (opsecond ?s)])))
```

- The *binary-coproduct cocone* underlying any cocone (pushout diagram) is named.

```
(26) (SET.FTN$function binary-coproduct-cocone)
     (= (SET.FTN$source binary-coproduct-cocone) cocone)
     (= (SET.FTN$target binary-coproduct-cocone) lang.col.coprd2$cocone)
     (= (SET.FTN$composition [binary-coproduct-cocone lang.col.coprd2$cocone-diagram])
        (SET.FTN$composition [cocone-diagram pair]))
     (= (SET.FTN$composition [binary-coproduct-cocone lang.col.coprd2$opvertex]) opvertex)
     (= (SET.FTN$composition [binary-coproduct-cocone lang.col.coprd2$opfirst]) opfirst)
     (= (SET.FTN$composition [binary-coproduct-cocone lang.col.coprd2$opsecond]) opsecond)
```

- The *coequalizer cocone* function maps a pushout cocone of type language morphisms to the associated (`lang.col.coeq`) coequalizer cocone of type language morphisms (see Figure 13, where arrows denote type language morphisms), whose type language morphism is the coequalizer comediator of the opfirst and opsecond type language morphisms with respect to the coequalizer diagram of a cocone. This is the first step in the definition of the pushout of a pushout cocone. The following string of equalities demonstrates that this cocone is well-defined.

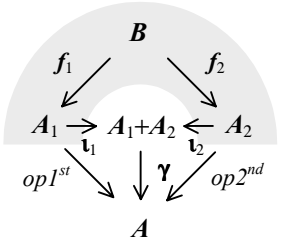


Figure 13: Coequalizer Cocone

$$f_1 \cdot \iota_1 \cdot \gamma = f_1 \cdot op1^{st} = f_2 \cdot op2^{nd} = f_2 \cdot \iota_2 \cdot \gamma$$

```
(27) (SET.FTN$function coequalizer-cocone)
     (= (SET.FTN$source coequalizer-cocone) cocone)
     (= (SET.FTN$target coequalizer-cocone) lang.col.coeq$cocone)
     (= (SET.FTN$composition [coequalizer-cocone lang.col.coeq$cocone-diagram])
        (SET.FTN$composition [cocone-diagram coequalizer-diagram]))
     (= (SET.FTN$composition [coequalizer-cocone lang.col.coeq$opvertex]) opvertex)
     (= (SET.FTN$composition [coequalizer-cocone lang.col.coeq$language-morphism])
        (SET.FTN$composition [binary-coproduct-cocone lang.col.coprd2$comediator]))
```

- The *variable* pushout cocone function maps a pushout cocone of type languages and type language morphisms to the underlying pushout cocone of variable sets and variable set functions. The *entity* type pushout cocone function maps a pushout cocone of type languages and type language morphisms to the underlying pushout cocone of entity type sets and entity type set functions. The *relation* type pushout cocone function maps a pushout cocone of type languages and type language morphisms to the underlying pushout cocone of relation type sets and relation type set functions.

```
(28) (SET.FTN$function variable-cocone)
     (= (SET.FTN$source variable-cocone) cocone)
     (= (SET.FTN$target variable-cocone) set.col.psh$cocone)
     (= (SET.FTN$composition [variable-cocone set.col.psh$cocone-diagram])
```

```
      (SET.FTN$composition [cocone-diagram variable-diagram]))
   (= (SET.FTN$composition [variable-cocone set.col.psh$opvertex])
      (SET.FTN$composition [opvertex lang$variable]))
   (= (SET.FTN$composition [variable-cocone set.col.psh$opfirst])
      (SET.FTN$composition [opfirst lang.mor$variable]))

(29) (SET.FTN$function entity-cocone)
   (= (SET.FTN$source entity-cocone) cocone)
   (= (SET.FTN$target entity-cocone) set.col.psh$cocone)
   (= (SET.FTN$composition [entity-cocone set.col.psh$cocone-diagram])
      (SET.FTN$composition [cocone-diagram entity-diagram]))
   (= (SET.FTN$composition [entity-cocone set.col.psh$opvertex])
      (SET.FTN$composition [opvertex lang$entity]))
   (= (SET.FTN$composition [entity-cocone set.col.psh$opfirst])
      (SET.FTN$composition [opfirst lang.mor$entity]))

(30) (SET.FTN$function relation-cocone)
   (= (SET.FTN$source relation-cocone) cocone)
   (= (SET.FTN$target relation-cocone) set.col.psh$cocone)
   (= (SET.FTN$composition [relation-cocone set.col.psh$cocone-diagram])
      (SET.FTN$composition [cocone-diagram relation-diagram]))
   (= (SET.FTN$composition [relation-cocone set.col.psh$opvertex])
      (SET.FTN$composition [opvertex lang$relation]))
   (= (SET.FTN$composition [relation-cocone set.col.psh$opfirst])
      (SET.FTN$composition [opfirst lang.mor$relation]))
```

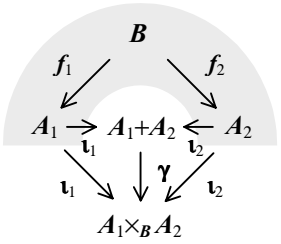


Figure 14: Colimiting Cocone

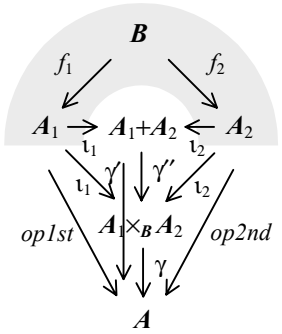


Figure 15: Comediator

- The class function ‘`colimiting-cocone`’ maps a span to its pushout (colimiting pushout cocone) (see Figure 14, where arrows denote type language morphisms). For convenience of reference, we define three terms that represent the components of this pushout cocone. The opvertex of the pushout cocone is a specific *pushout* type language, which comes equipped with two *injection* type language morphisms. The last axiom expresses concreteness of the colimit – it expresses pushouts in terms of coproducts and coequalizers: the colimit of the coequalizer diagram is (not just isomorphic but) equal to the pushout; likewise, the compositions of the coproduct injections of the pair diagram with the canon of the coequalizer diagram are equal to the pushout injections.

```
(31) (SET.FTN$function colimiting-cocone)
   (= (SET.FTN$source colimiting-cocone) diagram)
   (= (SET.FTN$target colimiting-cocone) cocone)
   (= (SET.FTN$composition [colimiting-cocone cocone-diagram])
      (SET.FTN$identity diagram))

(32) (SET.FTN$function colimit)
   (SET.FTN$function pushout)
   (= pushout colimit)
   (= (SET.FTN$source colimit) diagram)
   (= (SET.FTN$target colimit) lang$language)
   (= colimit (SET.FTN$composition [colimiting-cocone opvertex]))

(33) (SET.FTN$function injection1)
   (= (SET.FTN$source injection1) diagram)
```

```
     (= (SET.FTN$target injection1) lang.mor$language-morphism)
     (= (SET.FTN$composition [injection1 lang.mor$source]) language1)
     (= (SET.FTN$composition [injection1 lang.mor$target]) colimit)
     (= injection1 (SET.FTN$composition [colimiting-cocone opfirst])

(34) (SET.FTN$function injection2)
     (= (SET.FTN$source injection2) diagram)
     (= (SET.FTN$target injection2) lang.mor$language-morphism)
     (= (SET.FTN$composition [injection2 lang.mor$source]) language2)
     (= (SET.FTN$composition [injection2 lang.mor$target]) colimit)
     (= injection2 (SET.FTN$composition [colimiting-cocone opsecond])

(35) (forall (?r (diagram ?r))
         (and (= (colimit ?r)
                 (lang.col.coeq$coequalizer (coequalizer-diagram ?r)))
              (= (injection1 ?r)
                 (lang.mor$composition
                     [(lang.col.coprd2$injection1 (pair ?r))
                      (lang.col.coeq$canon (coequalizer-diagram ?r))]))
              (= (injection2 ?r)
                 (lang.mor$composition
                     [(lang.col.coprd2$injection2 (pair ?r))
                      (lang.col.coeq$canon (coequalizer-diagram ?r))])))))
```

- The following three axioms are the necessary conditions that the variable, entity and relation functors preserve concrete colimits. These ensure that both this pushout and its two pushout injection infomorphisms are specific.

```
(36) (= (SET.FTN$composition [colimiting-cocone variable-cocone])
        (SET.FTN$composition [variable-diagram set.col.psh$colimiting-cocone]))
     (= (SET.FTN$composition [colimit lang$variable])
        (SET.FTN$composition [variable-diagram set.col.psh$colimit]))
     (= (SET.FTN$composition [injection1 lang.mor$variable])
        (SET.FTN$composition [variable-diagram set.col.psh$injection1]))
     (= (SET.FTN$composition [injection2 lang.mor$variable])
        (SET.FTN$composition [variable-diagram set.col.psh$injection2]))

(37) (= (SET.FTN$composition [colimiting-cocone entity-cocone])
        (SET.FTN$composition [entity-diagram set.col.psh$colimiting-cocone]))
     (= (SET.FTN$composition [colimit lang$entity])
        (SET.FTN$composition [entity-diagram set.col.psh$colimit]))
     (= (SET.FTN$composition [injection1 lang.mor$entity])
        (SET.FTN$composition [entity-diagram set.col.psh$injection1]))
     (= (SET.FTN$composition [injection2 lang.mor$entity])
        (SET.FTN$composition [entity-diagram set.col.psh$injection2]))

(38) (= (SET.FTN$composition [colimiting-cocone relation-cocone])
        (SET.FTN$composition [relation-diagram set.col.psh$colimiting-cocone]))
     (= (SET.FTN$composition [colimit lang$relation])
        (SET.FTN$composition [relation-diagram set.col.psh$colimit]))
     (= (SET.FTN$composition [injection1 lang.mor$relation])
        (SET.FTN$composition [relation-diagram set.col.psh$injection1]))
     (= (SET.FTN$composition [injection2 lang.mor$relation])
        (SET.FTN$composition [relation-diagram set.col.psh$injection2]))
```

- The *comediator* type language morphism, from the pushout of a span to the opvertex of a cocone over the span (see Figure 15, where arrows denote type language morphisms), is the unique type language morphism that commutes with opfirst and opsecond. This is defined abstractly by using a definite description, and is defined concretely as the comediator of coequalizer cocone.

```
(39) (SET.FTN$function comediator)
     (= (SET.FTN$source comediator) cocone)
     (= (SET.FTN$target comediator) lang.mor$language-morphism)
     (= (SET.FTN$composition [comediator lang.mor$source])
        (SET.FTN$composition [cocone-diagram colimit]))
     (= (SET.FTN$composition [comediator lang.mor$target]) opvertex)
     (forall (?s (cocone ?s))
         (= (comediator ?s)
            (the (?m (lang.mor$language-morphism ?m))
```

```
                  (and (= (composition [(injection1 (cocone-diagram ?s)) ?m])
                          (opfirst ?s))
                       (= (composition [(injection2 (cocone-diagram ?s)) ?m])
                          (opsecond ?s))))))

(40) (= comediator
        (SET.FTN$composition [coequalizer-cocone lang.col.coeq$comediator]))
```