The Classification Ontology

The Namespace of Large Orders

This namespace will represent large orders and their morphisms: monotonic functions, adjoint pairs and Galois connections. Some of the terms introduced in this namespace are listed in Table 1. The *italicized terms* below remain to be defined.

Table 1: Terms introduced into the large order namespace

	Conglomerate	Function	Example
`ORD`	‘`preorder`’ ‘`partial-order`’ ‘`total-order`’	‘`class`’, ‘`underlying`’, ‘`extent`’ ‘**`classification`**’ ‘`identity`’, ‘`power`’ ‘`up-class`’, ‘`down-class`’ ‘`up`’, ‘`down`’ (uses $\exists$ quantifier) ‘`up-embedding`’, ‘`down-embedding`’ ‘`greatest`’, ‘`least`’	‘`successor`’ ‘`natural-numbers-order`’
`ORD .DM`		‘`upper-bound`’, ‘`lower-bound`’ (uses $\forall$ quantifier) ‘*`upper-lower-closure`*’, ‘*`lower-upper-closure`*’ ‘*`cut`*’, ‘*`cut-down`*’, ‘*`cut-up`*’ ‘*`cut-order`*’, ‘*`meet`*’, ‘*`join`*’ ‘`join-dense`’, ‘`meet-dense`’ ‘***`complete-lattice`***’ ‘*`element-embedding`*’, ‘*`element-concept`*’ ‘***`dedekind-macneille`***’	
`ORD .FTN`	‘`monotonic-function`’	‘`source`’, ‘`target`’, ‘`function`’ ‘`opposite`’, ‘`composition`’, ‘`identity`’ ‘`left`’, ‘`right`’	
`ORD .BIMOD`	‘`bimodule`’	‘`source`’, ‘`target`’, ‘`relation`’ ‘`opposite`’, ‘`composition`’, ‘`identity`’	
`ORD .ADJ`	‘`adjoint-pair`’	‘`source`’, ‘`target`’, ‘`left`’, ‘`right`’ ‘**`infomorphism`**’, ‘**`bond`**’ ‘`opposite`’, ‘`composition`’, ‘`identity`’	
`ORD .GC`	‘`galois-connection`’	‘`source`’, ‘`target`’, ‘`left`’, ‘`right`’ ‘`source-closure`’, ‘`target-closure`’	

Table 2 lists (needs to be completed) the correspondence between standard mathematical notation and the ontological terminology in the order namespace.

Table 2: Correspondence between Mathematical Notation and Ontological Terminology

Math	Ontological Terminology	Natural Language Description
$\leq$	‘`ORD$extent`’	the order relation class of a preorder

Orders

ORD

- A *preorder* $\boldsymbol{A} = \langle A, \leq_{\boldsymbol{A}} \rangle = \langle \mathit{class}(\boldsymbol{A}), \mathit{ext}(\boldsymbol{A}) \rangle$ is a reflexive and transitive endorelation. For convenience of reference the class terms $A = \mathit{class}(\boldsymbol{A})$ and $\leq_{\boldsymbol{A}} = \mathit{ext}(\boldsymbol{A})$ are reissued here.

```
(1) (CNG$conglomerate preorder)
    (CNG$subconglomerate preorder REL.ENDO$endorelation)

(2) (CNG$function class)
    (CNG$function underlying)
    (= underlying class)
    (CNG$signature class preorder SET$class)
    (forall (?o (preorder ?o))
        (= (class ?o) (REL.ENDO$class ?o)))

(3) (CNG$function extent)
    (CNG$signature extent preorder SET$class)
    (forall (?o (preorder ?o))
        (= (extent ?o) (REL.ENDO$extent ?o)))

    (forall (?o (preorder ?o))
        (and (reflexive ?o) (transitive ?o)))
```

- Any preorder $\boldsymbol{A} = \langle A, \leq_{\boldsymbol{A}} \rangle = \langle \mathit{class}(\boldsymbol{A}), \mathit{ext}(\boldsymbol{A}) \rangle$ has an associated classification $\mathit{cls}(\boldsymbol{A}) = \langle \mathit{class}(\boldsymbol{A}), \mathit{class}(\boldsymbol{A}), \mathit{ext}(\boldsymbol{A}) \rangle$. In axiom (4) the CNG function '`classification`' represents this.

```
(4) (CNG$function classification)
    (CNG$signature classification preorder CLS$classification)
    (forall (?o (preorder ?o))
        (and (= (instance (classification ?o)) (class ?o))
             (= (type (classification ?o)) (class ?o))
             (= (incidence (classification ?o)) (extent ?o))))
```

$$\begin{tikzcd}
\mathit{class}(\boldsymbol{A}) \\
\mathit{class}(\boldsymbol{A}) \arrow[u, "\boldsymbol{A}"']
\end{tikzcd}$$

Figure 1: Preorder as Classification

- A *partial order E* is a preorder that is antisymmetric.

```
(5) (CNG$conglomerate partial-order)
    (CNG$subconglomerate partial-order preorder)
    (forall (?o (REL.ENDO$endorelation ?o))
        (<=> (partial-order ?o)
             (and (preorder ?o) (antisymmetric ?o))))
```

- The identity endorelation on any class A is a partial order. The CNG function in axiom (6) realizes this.

```
(6) (CNG$function identity)
    (CNG$signature identity class partial-order)
    (forall (?c (class ?c))
        (= (identity ?c) (REL.ENDO$identity ?c)))
```

- For any class C the power class $\wp(C)$ using the subclass ordering is a partial order. There is a CNG function $\mathit{power} : \mathsf{class} \rightarrow \mathsf{order}$ that maps a class to its power order.

```
(7) (CNG$function power)
    (CNG$signature power class partial-order)
    (forall (?c (class ?c))
        (and (= (class (power ?c)) (SET$power ?c))
             (forall (?c1 (SET$subclass ?c1 ?c)
                      ?c2 (SET$subclass ?c2 ?c))
                 (<=> ((extent (power ?c)) [?c1 ?c2])
                      (SET$subclass ?c1 ?c2)))))
```

- A *total order E* is a partial order where all pairs are comparable.

```
(8) (CNG$conglomerate total-order)
    (CNG$subconglomerate total-order partial-order)
    (forall (?o (REL.ENDO$endorelation ?o))
        (<=> (total-order ?o)
             (and (partial-order ?o)
                  (forall (?x1 ((class ?o) ?x1)
                           ?x2 ((class ?o) ?x2))
                      (or ((extent ?o) [?x1 ?x2])
                          ((extent ?o) [?x2 ?x1]))))))
```

○ The natural numbers object in the core namespace has a “naturally defined” preorder. We can prove that this is a total order.

```
(9) (REL.ENDO$endorelation successor)
    (= (REL.ENDO$class successor) SET.TOP$natural-numbers)
    (<=> (REL.ENDO$extent successor) [?m ?n])
         (= (SET.TOP$successor ?m) ?n))

(10) (partial-order natural-numbers-order)
     (= natural-numbers-order (closure successor))

     (total-order natural-numbers-order)
```

○ Let $\boldsymbol{P} = \langle P, \leq \rangle$ be a partial order and let $Q \subseteq P$. Q is a *down-class* (*deceasing class* or *order ideal*) if, whenever $x \in Q$, $y \in P$ and $y \leq x$, we have $y \in Q$. An *up-class* (*increasing class* or *order filter*) has a dual definition.

```
(11) (KIF$function down-class)
     (KIF$signature down-class partial-order CNG$conglomerate)
     (forall (?o (partial-order ?o)
              ?c (SET$subclass ?c (class ?o)))
         (<=> ((down-class ?o) ?c)
              (forall (?x (?c ?x) ?y (?o ?y))
                  (=> ((extent ?o) [?y ?x]) (?c ?y)))))

(12) (KIF$function up-class)
     (KIF$signature up-class partial-order CNG$conglomerate)
     (forall (?o (partial-order ?o)
              ?c (SET$subclass ?c (class ?o)))
         (<=> ((up-class ?o) ?c)
              (forall (?x (?c ?x) ?y (?o ?y))
                  (=> ((extent ?o) [?x ?y]) (?c ?y)))))
```

○ Let $\boldsymbol{P} = \langle P, \leq \rangle$ be a partial order and let $Q \subseteq P$. The class $\uparrow_{\boldsymbol{P}}(Q) = up_{\boldsymbol{P}}(Q)$ is the class of all $\boldsymbol{P}$-elements that are above <u>some</u> (existential quantifier) $\boldsymbol{Q}$-element. Dually, the class $\downarrow_{\boldsymbol{P}}(Q) = down_{\boldsymbol{P}}(Q)$ is the class of all $\boldsymbol{P}$-elements that are below <u>some</u> (existential quantifier) $\boldsymbol{Q}$-element.

```
(13) (CNG$function up)
     (CNG$signature up partial-order SET.FTN$function)
     (forall (?o (partial-order ?o))
         (and (SET.FTN$source (up ?o)) (SET$power (class ?o)))
              (SET.FTN$target (up ?o)) (SET$power (class ?o)))
              (forall (?q (SET$subclass ?q (class ?o)) ?x ((class ?o) ?x))
                  (<=> (((up ?o) ?q) ?x)
                       (exists (?y (?q ?y))
                           ((extent ?o) [?y ?x]))))))

(14) (CNG$function down)
     (CNG$signature down partial-order SET.FTN$function)
     (forall (?o (partial-order ?o))
         (and (SET.FTN$source (down ?o)) (SET$power (class ?o)))
              (SET.FTN$target (down ?o)) (SET$power (class ?o)))
              (forall (?q (SET$subclass ?q (class ?o)) ?x ((class ?o) ?x))
                  (<=> (((down ?o) ?q) ?x)
                       (exists (?y (?q ?y))
                           ((extent ?o) [?x ?y]))))))
```

○ It is easy to check that $\downarrow_{\boldsymbol{P}}(Q)$ is the smallest down-class containing Q and that Q is a down-class iff $Q = \downarrow_{\boldsymbol{P}}(Q)$. Dually for $\uparrow_{\boldsymbol{P}}(Q)$.

```
     (forall (?o (partial-order ?o)
              ?q (SET$subclass ?q (class ?o)))
         (and ((up-class ?o) ((up ?o) ?q))
              (SET$subclass ?q ((up ?o) ?q))
```

```
                  (forall (?c (SET$subclass ?c (class ?o)))
                      (=> (and ((up-class ?o) ?c) (SET$subclass ?q ?c))
                          (SET$subclass ((up ?o) ?q) ?c)))))

         (forall (?o (partial-order ?o)
                  ?q (SET$subclass ?q (class ?o)))
             (<=> ((up-class ?o) ?q)
                  (= ?q ((up ?o) ?q))))

         (forall (?o (partial-order ?o)
                  ?q (SET$subclass ?q (class ?o)))
             (and ((down-class ?o) ((down ?o) ?q))
                  (SET$subclass ?q ((down ?o) ?q))
                  (forall (?c (SET$subclass ?c (class ?o)))
                      (=> (and ((down-class ?o) ?c) (SET$subclass ?q ?c))
                          (SET$subclass ((down ?o) ?q) ?c)))))

         (forall (?o (partial-order ?o)
                  ?q (SET$subclass ?q (class ?o)))
             (<=> ((down-class ?o) ?q)
                  (= ?q ((down ?o) ?q))))
```

- Let $\boldsymbol{P} = \langle P, \leq\rangle$ be a partial order and let $q \in P$. The class $\uparrow_{\boldsymbol{P}} q$ = *up-embed*$_{\boldsymbol{P}}(q)$ is the class of all $\boldsymbol{P}$-elements that are above q. Dually, the class $\downarrow_{\boldsymbol{P}} q$ = *down-embed*$_{\boldsymbol{P}}(q)$ is the class of all $\boldsymbol{P}$-elements that are below q.

```
    (15) (CNG$function up-embedding)
         (CNG$signature up-embedding partial-order SET.FTN$function)
         (forall (?o (partial-order ?o))
             (and (SET.FTN$source (up-embedding ?o)) (class ?o))
                  (SET.FTN$target (up-embedding ?o)) (SET$power (class ?o)))
                  (forall (?q ((class ?o) ?q) ?y ((class ?o) ?y))
                      (<=> (((up-embedding ?o) ?q) ?y)
                               ((extent ?o) [?q ?y])))))

    (16) (CNG$function down-embedding)
         (CNG$signature down-embedding partial-order SET.FTN$function)
         (forall (?o (partial-order ?o))
             (and (SET.FTN$source (down-embedding ?o)) (class ?o))
                  (SET.FTN$target (down-embedding ?o)) (SET$power (class ?o)))
                  (forall (?q ((class ?o) ?q) ?x ((class ?o) ?x))
                      (<=> (((down-embedding ?o) ?q) ?x)
                               ((extent ?o) [?x ?q])))))
```

- It is easy to check that $\downarrow_{\boldsymbol{P}}\{p\} = \downarrow_{\boldsymbol{P}} p$. Dually for $\uparrow_{\boldsymbol{P}} q$.

```
         (forall (?o (partial-order ?o)
             (and (= (SET.FTN$composition (SET.FTN$singleton (class ?o)) (down ?o))
                     (down-embedding ?o))
                  (= (SET.FTN$composition (SET.FTN$singleton (class ?o)) (up ?o))
                     (up-embedding ?o))))
```

- Let $\boldsymbol{P} = \langle P, \leq\rangle$ be a partial order and let $Q \subseteq P$. An element $a \in Q$ is a *greatest* (or *maximum*) element of Q if $x \leq a$ for any element $x \in Q$. Dually, an element $a \in Q$ is a *least* (or *minimum*) element of Q if $a \leq x$ for any element $x \in Q$.

```
    (17) (KIF$function greatest)
         (KIF$signature greatest partial-order CNG$function)
         (forall (?o (partial-order ?o))
             (and (CNG$signature (greatest ?o) SET$class SET$class)
                  (forall (?c (SET$subclass ?c (class ?o))
                           ?a (?c ?a))
                      (<=> (((greatest ?o) ?c) ?a)
                           (forall (?x (?c ?x))
                               ((extent ?o) [?x ?a]))))))

    (18) (KIF$function least)
         (KIF$signature least partial-order CNG$function)
         (forall (?o (partial-order ?o))
             (and (CNG$signature (least ?o) SET$class SET$class)
                  (forall (?c (SET$subclass ?c (class ?o))
                           ?a (?c ?a))
                      (<=> (((least ?o) ?c) ?a)
                           (forall (?x (?c ?x))
                               ((extent ?o) [?a ?x]))))))
```

○ By antisymmetry, there is at most one greatest or least element. Therefore, we can prove the following theorems.

```
         (forall (?o (partial-order ?o)
                  ?c (SET$subclass ?c (class ?o))
                  ?x (((greatest ?o) ?c) ?x)
                  ?y (((greatest ?o) ?c) ?y))
             (= ?x ?y))

         (forall (?o (partial-order ?o)
                  ?c (SET$subclass ?c (class ?o))
                  ?x (((least ?o) ?c) ?x)
                  ?y (((least ?o) ?c) ?y))
             (= ?x ?y))
```

Dedekind-MacNeille Completion

ORD.DM

○ Let $\boldsymbol{P} = \langle P, \leq \rangle$ be a partial order and let $Q \subseteq P$. An element $x \in P$ is an *upper bound* of Q when $q \leq x$ for <u>all</u> (universal quantifier) $q \in Q$. A *lower bound* is defined dually. The set of all upper bounds of Q is denoted Q^{u}. Dually, the set of all lower bounds of Q is denoted Q^{l}. These classes may be empty.

```
(1) (CNG$function upper-bound)
    (CNG$signature upper-bound ORD$partial-order SET.FTN$function)
    (forall (?o (ORD$partial-order ?o))
        (and (SET.FTN$source (upper-bound ?o)) (SET$power (class ?o)))
             (SET.FTN$target (upper-bound ?o)) (SET$power (class ?o)))
             (forall (?q (SET$subclass ?q (ORD$class ?o)) ?y ((ORD$class ?o) ?y))
                 (<=> (((upper-bound ?o) ?q) ?y)
                      (forall (?x (?q ?x))
                          ((ORD$extent ?o) [?x ?y]))))))

(2) (CNG$function lower-bound)
    (CNG$signature lower-bound ORD$partial-order SET.FTN$function)
    (forall (?o (ORD$partial-order ?o))
        (and (SET.FTN$source (lower-bound ?o)) (SET$power (class ?o)))
             (SET.FTN$target (lower-bound ?o)) (SET$power (class ?o)))
             (forall (?q (SET$subclass ?q (ORD$class ?o)) ?x ((ORD$class ?o) ?x))
                 (<=> (((lower-bound ?o) ?q) ?x)
                      (forall (?y (?q ?y))
                          ((ORD$extent ?o) [?x ?y]))))))
```

○ Since the order is transitive, the upper bound of Q is always an up-class and the lower bound of Q is always a down-class.

```
    (forall (?o (ORD$partial-order ?o)
             ?q (SET$subclass ?q (ORD$class ?o)))
        (and ((ORD$up-class ?o) ((upper-bound ?o) ?q))
             ((ORD$down-class ?o) ((lower-bound ?o) ?q))))
```

○ It is easy to check that $\{p\}^{\mathrm{l}} = \downarrow_{\boldsymbol{P}} p$. Dually, $\{p\}^{\mathrm{u}} = \uparrow_{\boldsymbol{P}} p$.

```
     (forall (?o (partial-order ?o))
         (and (= (SET.FTN$composition (SET.FTN$singleton (class ?o)) (upper-bound ?o))
                 (up-embedding ?o))
              (= (SET.FTN$composition (SET.FTN$singleton (class ?o)) (lower-bound ?o))
                 (down-embedding ?o))))
```

○ The composition of bounds gives two senses of closure operator.

```
(3) (CNG$function upper-lower-closure)
    (CNG$signature upper-lower-closure ORD$partial-order SET.FTN$function)
    (forall (?o (ORD$partial-order ?o))
        (and (= (SET.FTN$source (upper-lower-closure ?o))
                 (SET$power (ORD$class ?o)))
              (= (SET.FTN$target (upper-lower-closure ?o))
                 (SET$power (ORD$class ?o)))
              (= (upper-lower-closure ?o)
                 (SET.FTN$composition (upper-bound ?o) (lower-bound ?o)))))

(4) (CNG$function lower-upper-closure)
    (CNG$signature lower-upper-closure ORD$partial-order SET.FTN$function)
    (forall (?o (ORD$partial-order ?o))
        (and (= (SET.FTN$source (lower-upper-closure ?o))
                (SET$power (ORD$class ?o)))
             (= (SET.FTN$target (lower-upper-closure ?o))
                (SET$power (ORD$class ?o)))
             (= (lower-upper-closure ?o)
                (SET.FTN$composition (lower-bound ?o) (upper-bound ?o)))))
```

- The following (easily proven) results confirm that there is a Galois connection between bound operators and that the composites are closure operators.

  $X \subseteq X^{\mathsf{ul}}$. and $X^{\mathsf{u}} = X^{\mathsf{ulu}}$ for all $X \subseteq \mathit{inst}(\boldsymbol{A})$.

  If $X \subseteq Y$ then $X^{\mathsf{u}} \supseteq Y^{\mathsf{u}}$ and $X^{\mathsf{l}} \supseteq Y^{\mathsf{l}}$.

  $Y \subseteq Y^{\mathsf{lu}}$ and $Y^{\mathsf{l}} = Y^{\mathsf{lul}}$ for all $Y \subseteq \mathit{typ}(\boldsymbol{A})$.

```
(forall (?o (ORD$partial-order ?o))
    (and (= (upper-bound ?o)
            (SET.FTN$composition (upper-lower-closure ?o) (upper-bound ?o)))
         (forall (?x (SET$subclass ?x (ORD$class ?o)))
             (SET$subclass ?x ((upper-lower-closure ?o) ?x)))))

(forall (?o (ORD$partial-order ?o))
         ?x (SET$subclass ?x (ORD$class ?o))
         ?y (SET$subclass ?y (ORD$class ?o)))
    (=> (SET$subclass ?x ?y)
        (and (SET$subclass ((upper-bound ?o) ?y) ((upper-bound ?o) ?x))
             (SET$subclass ((lower-bound ?o) ?y) ((lower-bound ?o) ?x)))))

(forall (?o (ORD$partial-order ?o))
    (and (= (lower-bound ?o)
            (SET.FTN$composition (lower-upper-closure ?o) (lower-bound ?o)))
         (forall (?y (SET$subclass ?y (ORD$class ?o)))
             (SET$subclass ?y ((lower-upper-closure ?o) ?y)))))
```

- Further properties of Galois connection relate to continuity – the closure of a union of a family of classes is the intersection of the closures.

  $(\cup_{j \in J} X_{\mathrm{j}})^{\mathsf{u}} = \cap_{j \in J} X_j^{\mathsf{u}}$ for any family of subsets $X_j \subseteq \mathit{inst}(\boldsymbol{A})$ for $j \in J$.

  $(\cup_{k \in K} Y_k)^{\mathsf{l}} = \cap_{k \in K} Y_k^{\mathsf{l}}$ for any family of subsets $Y_k \subseteq \mathit{typ}(\boldsymbol{A})$ for $k \in K$.

- It is important to note that the notions of upper bound and lower bound for orders are related to the notions of intent and extent for classifications.
  - The up operator is the left-derivation of the classification of a preorder, and the down operator is the left-derivation of the classification of a preorder.
  - The cut class is the concept class of the classification of a preorder.
  - The Dedekind-MacNeille completion of a partial order is the concept lattice of the classification of a preorder, but with the instance/type classes identified with the preorder class, and the instance/type embeddings identified with the preorder embedding.

```
(forall (?o (partial-order ?o)
         ?q (SET$subclass ?q (class ?o)))
    (and ((up-class ?o) ((upper-bound ?o) ?q))
         ((down-class ?o) ((lower-bound ?o) ?q))))
```

- Let $\boldsymbol{P} = \langle P, \leq \rangle$ be a partial order and let $Q \subseteq P$. Then Q is *join-dense* in P when for every element $a \in P$ there is a subset $X \subseteq Q$ such that $a = \sqcup_P(X)$. The notion of *meet-dense* is dual.

```
(21) (KIF$function join-dense)
     (KIF$signature join-dense partial-order SET$class)
     (forall (?o (partial-order ?o))
         (and (SET$subclass (join-dense ?o) (SET$power (class ?o)))
              (forall (?q (SET$subclass ?q (class ?o)))
                  (<=> ((join-dense ?o) ?q)
                       (forall (?a ((class ?o) ?a))
                           (exists (?x (SET$subclass ?x ?q))
                               (((least ?o) ((upper-bound ?o) ?x)) ?a)))))))

(22) (KIF$function meet-dense)
     (KIF$signature meet-dense partial-order SET$class)
     (forall (?o (partial-order ?o))
         (and (SET$subclass (meet-dense ?o) (SET$power (class ?o)))
              (forall (?q (SET$subclass ?q (class ?o)))
                  (<=> ((meet-dense ?o) ?q)
                       (forall (?a ((class ?o) ?a))
                           (exists (?x (SET$subclass ?x ?q))
                               (((greatest ?o) ((lower-bound ?o) ?x)) ?a)))))))
```

○ Let $\boldsymbol{L} = \langle L, \leq_L \rangle$ be a partial order. For any class $S \subseteq L$, if it exists, the *meet* (*greatest lower bound* or *infimum*) $\sqcap_L(S)$ is the greatest element in the lower bound of S. For any class $S \subseteq P$, if it exists, the *join* (*least upper bound* or *infimum*) $\sqcup_L(S)$ is the least element in the upper bound of S.

Monotonic Functions

`ORD.FTN`

Preorders and partial orders are related through monotonic functions.

- A monotonic function $f : \boldsymbol{P} \rightarrow \boldsymbol{Q}$, from preorder $\boldsymbol{P} = \langle P, \leq_{\boldsymbol{P}} \rangle$ to preorder $\boldsymbol{Q} = \langle Q, \leq_{\boldsymbol{Q}} \rangle$, is a function $f : P \rightarrow Q$ between the underlying classes that preserves order:

  if $p_1 \leq_{\boldsymbol{P}} p_2$ then $f(p_1) \leq_{\boldsymbol{Q}} f(p_2)$ for all $p_1, p_2 \in P$.

```
(1) (CNG$conglomerate monotonic-function)

(2) (CNG$function source)
    (CNG$signature source monotonic-function ORD$preorder)

(3) (CNG$function target)
    (CNG$signature target monotonic-function ORD$preorder)

(4) (CNG$function function)
    (CNG$signature function monotonic-function SET.FTN$function)
    (forall (?f (monotonic-function ?f))
        (and (= (SET.FTN$source (function ?f)) (ORD$class (source ?f)))
             (= (SET.FTN$target (function ?f)) (ORD$class (target ?f)))))

(5) (forall (?f (monotonic-function ?f)
             ?p1 ((ORD$class (source ?f)) ?p1)
             ?p2((ORD$class (source ?f)) ?p2)
        (=> ((ORD$extent (source ?f)) [?p1 ?p2])
            ((ORD$extent (target ?f)) [(?f ?p1) (?f ?p2)])))
```

- Two monotonic functions are composable when the target preorder of the first is the source preorder of the second. The *composition* of two composable monotonic functions $f : \boldsymbol{P} \rightarrow \boldsymbol{Q}$ and $g : \boldsymbol{Q} \rightarrow \boldsymbol{N}$ is defined via the composition of the underlying functions.

```
(6) (CNG$function composition)
    (CNG$signature composition monotonic-function monotonic-function monotonic-function)
    (forall (?f (monotonic-function ?f) ?g (monotonic-function ?g))
        (<=> (exists (?h (monotonic-function ?h)) (= (composition ?f ?g) ?h))
             (= (target ?f) (source ?g))))
    (forall (?f (monotonic-function ?f) ?g (monotonic-function ?g))
        (=> (= (target ?f) (source ?g))
            (and (= (source (composition ?f ?g)) (source ?f))
                 (= (target (composition ?f ?g)) (target ?g))
                 (= (function (composition ?f ?g))
                    (SET.FTN$composition (function ?f) (function ?g))))))
```

- The identity monotonic function at a preorder $\boldsymbol{P}$ is the identity function of the underlying class.

```
(7) (CNG$function identity)
    (CNG$signature identity ORD$preorder monotonic-function)
    (forall (?p (ORD$preorder ?p))
        (and (= (source (identity ?p)) ?p)
             (= (target (identity ?p)) ?p)
             (= (function (identity ?p)) (SET.FTN$identity (ORD$underlying ?p)))))
```

- Duality can be extended from orders to monotonic functions. For any monotonic function $f : \boldsymbol{P} \rightarrow \boldsymbol{Q}$, the *opposite* or *dual* of f is the monotonic function $f^{\perp} : \boldsymbol{P}^{\perp} \rightarrow \boldsymbol{Q}^{\perp}$, whose source preorder is the opposite of the source of f, and whose target preorder is the opposite of the target of f. Note that the source/target polarity has not changed.

  Axiom (8) specifies the opposite operator on monotonic functions.

```
(8) (CNG$function opposite)
    (CNG$signature opposite monotonic-function monotonic-function)
    (forall (?f (monotonic-function ?f))
        (and (= (source (opposite ?f)) (ORD$opposite (source ?f)))
             (= (target (opposite ?f)) (ORD$opposite (target ?f)))
             (= (function (opposite ?f)) (function ?f))))
```

- In the presence of a preorder $\boldsymbol{A} = \langle A, \leq_A \rangle$, there are two ways that monotonic functions are transformed into order bimodules – both by composition. For any monotonic function $f : \boldsymbol{B} \rightarrow \boldsymbol{A}$, the *left* bimodule $f_@ : A \rightarrow B$ is defined as

  $$f_@(a, b) \text{ iff } a \leq_A f(b),$$

  and the *right* bimodule $f^@ : B \rightarrow A$ as follows

  $$f^@(b, a) \text{ iff } f(b) \leq_A a.$$

  The left and right operators for monotonic functions are defined in terms of the left and right operators for ordinary functions.

```
(9) (KIF$function left)
    (KIF$signature left preorder CNG$function)
    (forall (?o (ORD$preorder ?o))
        (CNG$signature (left ?o) monotonic-function BIMOD$bimodule))
    (forall (?o (ORD$preorder ?o)
             ?f (monotonic-function ?f))
        (<=> (exists (?r (BIMOD$bimodule ?r)) (= ((left ?o) ?f) ?r))
             (= (target ?f) ?o)))
    (forall (?o (ORD$preorder ?o) ?f (monotonic-function ?f))
        (=> (= (target ?f) ?o)
            (and (= (BIMOD$source ((left ?o) ?f)) ?o)
                 (= (BIMOD$target ((left ?o) ?f)) (source ?f))
                 (= (BIMOD$relation ((left ?o) ?f))
                    ((SET.FTN$left ?o) (function ?f))))))

(10) (KIF$function right)
     (KIF$signature right ORD$preorder CNG$function)
     (forall (?o (ORD$preorder ?o))
         (CNG$signature (right ?o) monotonic-function BIMOD$bimodule))
     (forall (?o (ORD$preorder ?o)
              ?f (monotonic-function ?f))
         (<=> (exists (?r (BIMOD$bimodule ?r)) (= ((right ?o) ?f) ?r))
              (= (target ?f) ?o)))
     (forall (?o (ORD$preorder ?o) ?f (monotonic-function ?f))
         (=> (= (target ?f) ?o)
             (and (= (BIMOD$source ((right ?o) ?f)) (source ?f))
                  (= (BIMOD$target ((right ?o) ?f)) ?o)
                  (= (BIMOD$relation ((right ?o) ?f))
                     ((SET.FTN$right ?o) (function ?f))))))
```

Order Bimodules

ORD.BIMOD

Preorders and partial orders are related through order bimodules. Order bimodules extend binary relations to the order realm. An order bimodule is a binary relation, whose source and target are preorders, and which is order-closed on the left (at the source) and on the right (at the target).

- A(n *order*) *bimodule* $\boldsymbol{R} : \boldsymbol{P} \rightarrow \boldsymbol{Q}$, from preorder $\boldsymbol{P} = \langle P, \leq_{\boldsymbol{P}} \rangle$ to preorder $\boldsymbol{Q} = \langle Q, \leq_{\boldsymbol{Q}} \rangle$, is a binary relation function $\boldsymbol{R} : P \rightarrow Q$ between the underlying classes that is order-closed on left and right:

  if $p_2 \leq_{\boldsymbol{P}} p_1$ and then $p_1 R q$ for all $p_2 R q$, and

  if $p R q_1$ and $q_1 \leq_{\boldsymbol{Q}} q_2$ then for all $p R q_2$.

```
(1) (CNG$conglomerate bimodule)

(2) (CNG$function source)
    (CNG$signature source bimodule ORD$preorder)

(3) (CNG$function target)
    (CNG$signature target bimodule ORD$preorder)

(4) (CNG$function relation)
    (CNG$signature relation bimodule REL$relation)
    (forall (?r (bimodule ?r))
        (and (= (REL$source (relation ?r)) (ORD$class (source ?r)))
             (= (REL$target (relation ?r)) (ORD$class (target ?r)))))

(5) (forall (?r (bimodule ?r)
             ?p2 ((ORD$class (source ?r)) ?p2)
             ?p1 ((ORD$class (source ?r)) ?p1)
             ?q ((ORD$class (target ?r)) ?q))
        (=> (and ((ORD$extent (source ?r)) [?p2 ?p1])
                 ((ORD$extent (relation ?r)) [?p1 ?q]))
            ((ORD$extent (relation ?r)) [?p2 ?q])))

    (forall (?r (bimodule ?r)
             ?p ((ORD$class (source ?r)) ?p)
             ?q1 ((ORD$class (target ?r)) ?q1)
             ?q2 ((ORD$class (target ?r)) ?q2))
        (=> (and ((ORD$extent (relation ?r)) [?p ?q1]))
                 ((ORD$extent (target ?r)) [?q1 ?q2])
            ((ORD$extent (relation ?r)) [?p ?q2])))
```

- Duality can be extended from monotonic functions to order bimodules. For any order bimodule $\boldsymbol{R} : \langle P, \leq_{\boldsymbol{P}} \rangle \rightarrow \langle Q, \leq_{\boldsymbol{Q}} \rangle$, from preorder $\boldsymbol{P} = \langle P, \leq_{\boldsymbol{P}} \rangle$ to preorder $\boldsymbol{Q} = \langle Q, \leq_{\boldsymbol{Q}} \rangle$, the *opposite* (*dual* or *transpose*) of $\boldsymbol{R}$ is the order bimodule $\boldsymbol{R}^{\perp} : \langle Q, \geq_{\boldsymbol{Q}} \rangle \rightarrow \langle P, \geq_{\boldsymbol{P}} \rangle$, from preorder $\boldsymbol{Q}^{\perp} = \langle Q, \geq_{\boldsymbol{Q}} \rangle$ to preorder $\boldsymbol{P}^{\perp} = \langle P, \geq_{\boldsymbol{P}} \rangle$, whose source preorder is the opposite of the target of $\boldsymbol{R}$, whose target preorder is the opposite of the source of $\boldsymbol{R}$, and whose underlying relation is the transpose (opposite) of the relation of $\boldsymbol{R}$. Note that the source/target polarity has changed – source to target and target to source.

  Axiom (6) specifies the opposite operator on order bimodules.

```
(6) (CNG$function opposite)
    (CNG$signature opposite bimodule bimodule)
    (forall (?r (bimodule ?r))
        (and (= (source (opposite ?r)) (ORD$opposite (target ?r)))
             (= (target (opposite ?r)) (ORD$opposite (source ?r)))
             (= (relation (opposite ?r)) (ORD$opposite (relation ?r)))))
```

- Two order bimodules $\boldsymbol{R} : \boldsymbol{O} \rightarrow \boldsymbol{P}$ and $\boldsymbol{S} : \boldsymbol{P} \rightarrow \boldsymbol{Q}$ are *composable* when the target order of $\boldsymbol{R}$ is the same as the source order of $\boldsymbol{S}$. There is a binary CNG function *composition*, which takes two composable order bimodules and returns their composition.

```
(7) (CNG$function composition)
```

```
        (CNG$signature composition bimodule bimodule bimodule)
         (forall (?r (bimodule ?r) ?s (bimodule ?s))
             (<=> (exists (?t (bimodule ?t)) (= (composition ?r ?s) ?t))
                  (= (target ?r) (source ?s)))))
         (forall (?r (bimodule ?r) ?s (bimodule ?s))
             (=> (= (target ?r) (source ?s))
                 (and (= (source (composition ?r ?s)) (source ?r))
                      (= (target (composition ?r ?s)) (target ?s))
                      (= (relation (composition ?r ?s))
                         (REL$composition (relation ?r) (relation ?s))))))
```

o For any preorder ***P*** there is an identity order bimodule *identity*$_{\boldsymbol{P}}$.

```
    (8) (CNG$function identity)
        (CNG$signature identity ORD$preorder bimodule)
        (forall (?o (ORD$preorder ?o))
            (and (= (source (identity ?o)) ?o)
                 (= (target (identity ?o)) ?o)
                 (= (relation (identity ?o))
                    (REL$identity (ORD$class ?o)))))
```

Adjoint Pairs

ORD.ADJ

$$\begin{tikzcd} \boldsymbol{P} \arrow[r, "{\langle \mathit{left}(\boldsymbol{f}), \mathit{right}(\boldsymbol{f}), \eta, \varepsilon \rangle}"] & \boldsymbol{Q} \end{tikzcd}$$

Figure 1: Abstract Notation

- An *adjoint pair* $\boldsymbol{f} : \boldsymbol{P} \rightleftarrows \boldsymbol{Q}$ from preorder $\boldsymbol{P}$ to preorder $\boldsymbol{Q}$ is a pair $\boldsymbol{f} = \langle \mathit{right}(\boldsymbol{f}), \mathit{left}(\boldsymbol{f}) \rangle$ of oppositely directed monotonic functions, $\mathit{right}(\boldsymbol{f}) : \boldsymbol{Q} \rightarrow \boldsymbol{P}$ and $\mathit{left}(\boldsymbol{f}) : \boldsymbol{P} \rightarrow \boldsymbol{Q}$, which satisfy the *fundamental property*:

  $$\mathit{left}(\boldsymbol{f})(p) \leq_{\boldsymbol{Q}} q \text{ iff } p \leq_{\boldsymbol{P}} \mathit{right}(\boldsymbol{f})(q)$$

  for all elements $q \in \boldsymbol{Q}$ and $p \in \boldsymbol{P}$.

  – An adjoint pair $\boldsymbol{f} : \boldsymbol{P} \rightleftarrows \boldsymbol{Q}$ is an adjunction $\langle \mathit{left}(\boldsymbol{f}), \mathit{right}(\boldsymbol{f}), \eta, \varepsilon \rangle : \boldsymbol{P} \rightarrow \boldsymbol{Q}$, where the preorders are consider categories, the unit $\eta : Id_P \Rightarrow \mathit{left}(\boldsymbol{f}) \cdot \mathit{right}(\boldsymbol{f})$ corresponds to the induced closure operator on $\boldsymbol{P}$, and the counit $\varepsilon : \mathit{right}(\boldsymbol{f}) \cdot \mathit{left}(\boldsymbol{f}) \Rightarrow Id_Q$ corresponds to the induced interior operator on $\boldsymbol{Q}$.

  – An adjoint pair $\boldsymbol{f} : \boldsymbol{P} \rightleftarrows \boldsymbol{Q}$ is an infomorphism $\boldsymbol{f} : \boldsymbol{Q} \rightleftarrows \boldsymbol{P}$ from classification $\boldsymbol{Q} = \langle Q, Q, \leq_{\boldsymbol{Q}} \rangle$ to classification $\boldsymbol{P} = \langle P, P, \leq_{\boldsymbol{P}} \rangle$, where the preorders are regarded as classifications.

```
(1) (CNG$conglomerate adjoint-pair)

(2) (CNG$function source)
    (CNG$signature source adjoint-pair ORD$preorder)

(3) (CNG$function target)
    (CNG$signature target adjoint-pair ORD$preorder)

(4) (CNG$function left)
    (CNG$signature left adjoint-pair ORD.FTN$monotonic-function)
    (forall (?a (adjoint-pair ?a))
        (and (= (ORD.FTN$source (left ?a)) (source ?a))
             (= (ORD.FTN$target (left ?a)) (target ?a))))

(5) (CNG$function right)
    (CNG$signature right adjoint-pair ORD.FTN$monotonic-function)
    (forall (?a (adjoint-pair ?a))
        (and (= (ORD.FTN$source (right ?a)) (target ?a))
             (= (ORD.FTN$target (right ?a)) (source ?a))))

    (forall (?a (adjoint-pair ?a)
             ?p ((ORD$class (source ?a)) ?p)
             ?q ((ORD$class (target ?a)) ?q))
        (<=> ((ORD$extent (target ?a)) [((left ?a) ?p) ?q])
             ((ORD$extent (source ?a)) [?p ((right ?a) ?q)])))
```

- Any adjoint pair is an infomorphism. The CNG function in axiom (6) realizes this assertion.

```
(6) (CNG$function infomorphism)
    (CNG$signature infomorphism
       adjoint-pair CLS.INFO$infomorphism)
    (forall (?a (adjoint-pair ?a))
       (and (= (CLS.INFO$source (infomorphism ?a))
               (ORD$classification (target ?a)))
            (= (CLS.INFO$target (infomorphism ?a))
               (ORD$classification (source ?a)))
            (= (instance (classification ?a))
               (left ?a))
            (= (type (classification ?a))
               (right ?a))))
```

$$\begin{tikzcd} \boldsymbol{Q} \arrow[r, "\mathit{right}(\boldsymbol{f})"] \arrow[d, dash, "\leq_{\boldsymbol{P}}"'] \arrow[dr, dash] & \boldsymbol{P} \arrow[d, dash, "\leq_{\boldsymbol{Q}}"] \\ \boldsymbol{Q} & \boldsymbol{P} \arrow[l, "\mathit{left}(\boldsymbol{f})"] \end{tikzcd}$$

Figure 2: Adjoint Pair as an Infomorphism

- The fundamental property of an adjoint pair $\boldsymbol{f} : \boldsymbol{P} \rightleftarrows \boldsymbol{Q}$ expresses the bonding classification of a bond $\mathit{bond}(\boldsymbol{f}) : \mathit{cls}(\boldsymbol{B}) \rightleftarrows \mathit{cls}(\boldsymbol{A})$ associated with the adjoint pair. Although this could be defined "from scratch," here it is defined in terms of the bond of the functional infomorphism associated with $\boldsymbol{f}$.

```
(7) (CNG$function bond)
    (CNG$signature bond adjoint-pair CLS.BND$bond)
    (forall (?a (adjoint-pair ?a))
        (and (= (CLS.BND$source (bond ?a)) (ORD$classification (target ?a)))
             (= (CLS.BND$target (bond ?a)) (ORD$classification (source ?a)))
             (= (CLS.BND$classification (bond ?a))
                (CLS.INFO$bond (infomorphism ?a)))))
```

- Duality can be extended to adjoint pairs. For any adjoint pair $\boldsymbol{f} : \boldsymbol{P} \rightleftarrows \boldsymbol{Q}$, the *opposite* or *dual* of $\boldsymbol{f}$ is the adjoint pair $\boldsymbol{f}^{\perp} : \boldsymbol{Q}^{\perp} \rightleftarrows \boldsymbol{P}^{\perp}$, whose source preorder is the opposite of the target of $\boldsymbol{f}$, whose target preorder is the opposite of the source of $\boldsymbol{f}$, whose left monotonic function is the right of $\boldsymbol{f}$, and whose right monotonic function is the left of $\boldsymbol{f}$.

  Axiom (8) specifies the opposite operator on adjoint pairs.

```
(8) (CNG$function opposite)
    (CNG$signature opposite adjoint-pair adjoint-pair)
    (forall (?f (adjoint-pair ?f))
        (and (= (source (opposite ?f)) (ORD$opposite (target ?f)))
             (= (target (opposite ?f)) (ORD$opposite (source ?f)))
             (= (left (opposite ?f)) (ORD.FTN$opposite (right ?f)))
             (= (right (opposite ?f)) (ORD.FTN$opposite (left ?f)))))
```

- Two adjoint pairs $\boldsymbol{f} : \boldsymbol{O} \rightleftarrows \boldsymbol{P}$ and $\boldsymbol{g} : \boldsymbol{P} \rightleftarrows \boldsymbol{Q}$ are *composable* when the target order of $\boldsymbol{f}$ is the same as the source order of $\boldsymbol{g}$. There is a binary CNG function *composition*, which takes two composable adjoint pairs and returns their composition.

```
(10) (CNG$function composition)
     (CNG$signature composition adjoint-pair adjoint-pair adjoint-pair)
      (forall (?f (adjoint-pair ?f) ?g (adjoint-pair ?g))
          (<=> (exists (?h (adjoint-pair ?h)) (= (composition ?f ?g) ?h))
               (= (target ?f) (source ?g)))))
      (forall (?f (adjoint-pair ?f) ?g (adjoint-pair ?g))
          (=> (= (target ?f) (source ?g))
              (and (= (source (composition ?f ?g)) (source ?f))
                   (= (target (composition ?f ?g)) (target ?g))
                   (= (left (composition ?f ?g))
                      (ORD.FTN$composition (left ?f) (left ?g)))
                   (= (right (composition ?f ?g))
                      (ORD.FTN$composition (right ?g) (right ?f))))))
```

- For any preorder $\boldsymbol{P}$ there is an identity adjoint pair $identity_{\boldsymbol{P}}$.

```
(11) (CNG$function identity)
     (CNG$signature identity ORD$preorder adjoint-pair)
     (forall (?o (ORD$preorder ?o))
         (and (= (source (identity ?o)) ?o)
              (= (target (identity ?o)) ?o)
              (= (left (identity ?o)) (ORD.FTN$identity ?o))
              (= (right (identity ?o)) (ORD.FTN$identity ?o))))
```

Galois Connections

ORD.GC

- A *Galois connection* is an adjoint pair $\boldsymbol{f} : \boldsymbol{P} \rightleftarrows \boldsymbol{Q}^{\mathrm{op}}$ from preorder $\boldsymbol{P}$ to the opposite of preorder $\boldsymbol{Q}$; that is, it is a pair $\boldsymbol{f} = \langle right(\boldsymbol{f}), left(\boldsymbol{f})\rangle$ of oppositely directed monotonic functions, $right(\boldsymbol{f}) : \boldsymbol{Q}^{\mathrm{op}} \rightarrow \boldsymbol{P}$ and $left(\boldsymbol{f}) : \boldsymbol{P} \rightarrow \boldsymbol{Q}^{\mathrm{op}}$, which satisfy the *fundamental property*:

  $$left(\boldsymbol{f})(p) \geq_{\boldsymbol{Q}} q \text{ iff } p \leq_{\boldsymbol{P}} right(\boldsymbol{f})(q)$$

  or

  $$right(\boldsymbol{f})(q) \geq_{\boldsymbol{P}} p \text{ iff } q \leq_{\boldsymbol{Q}} left(\boldsymbol{f})(p)$$

  for all elements $q \in \boldsymbol{Q}$ and $p \in \boldsymbol{P}$.

$$\begin{tikzcd}
\boldsymbol{P} \arrow[r, "left(\boldsymbol{f})"] \arrow[d, dash, "\geq_{\boldsymbol{P}}"'] \arrow[dr, dash] & \boldsymbol{Q} \arrow[d, dash, "\leq_{\boldsymbol{Q}}"] \\
\boldsymbol{P} & \boldsymbol{Q} \arrow[l, "right(\boldsymbol{f})"]
\end{tikzcd}$$

Figure 3: Galois Connection

```
(1) (CNG$conglomerate galois-connection)

(2) (CNG$function source)
```

```
        (CNG$signature source galois-connection preorder)

    (3) (CNG$function target)
        (CNG$signature target galois-connection preorder)

    (4) (CNG$function left)
        (CNG$signature left galois-connection monotonic-function)
        (forall (?a (galois-connection ?a))
            (and (= (ORD.FTN$source (left ?a)) (source ?a))
                 (= (ORD.FTN$target (left ?a)) (opposite (target ?a)))))

    (5) (CNG$function right)
        (CNG$signature right galois-connection monotonic-function)
        (forall (?a (galois-connection ?a))
            (and (= (ORD.FTN$source (right ?a)) (opposite (target ?a)))
                (= (ORD.FTN$target (right ?a)) (source ?a))))

        (forall (?a (galois-connection ?a)
                 ?p ((ORD$class (source ?a)) ?p)
                 ?q ((ORD$class (target ?a)) ?q))
            (<=> ((ORD$extent (target ?a)) [?q ((left ?a) ?p)])
                 ((ORD$extent (source ?a)) [?p ((right ?a) ?q)])))
```

- The following properties can be proven.

```
        (forall (?a (galois-connection ?a)
                 ?p ((ORD$class (source ?a)) ?p))
            (and ((ORD$extent (source ?a)) [?p ((right ?a) ((left ?a) ?p))])
                 (= ?p ((left ?a) ((right ?a) ((left ?a) ?p))))))

        (forall (?a (galois-connection ?a)
                 ?q ((ORD$class (target ?a)) ?q))
            (and ((ORD$extent (target ?a)) [?q ((left ?a) ((right ?a) ?q))])
                 (= ?q ((right ?a) ((left ?a) ((right ?a) ?q))))))
```

- Using these properties, we can show that there are two closure operators for any Galois connection.

```
    (6) (CNG$function source-closure)
        (CNG$signature source-closure galois-connection ORD.CLSR$closure-operator)
        (forall (?a (galois-connection ?a))
            (and (= (ORD.CLSR$class (source-closure ?a))
                    (ORD$class (source ?a)))
                 (= (ORD.CLSR$function (source-closure ?a))
                    (SET.FTN$composition
                        (ORD.FTN$function (left ?a))
                        (ORD.FTN$function (right ?a))))))

    (7) (CNG$function target-closure)
        (CNG$signature target-closure galois-connection ORD.CLSR$closure-operator)
        (forall (?a (galois-connection ?a))
            (and (= (ORD.CLSR$class (target-closure ?a))
                    (ORD$class (target ?a)))
                 (= (ORD.CLSR$function (target-closure ?a))
                    (SET.FTN$composition
                        (ORD.FTN$function (right ?a))
                        (ORD.FTN$function (left ?a))))))
```

The Namespace of Large Concept Lattices

This namespace will represent large concept lattices and their morphisms. Some of the terms introduced in this namespace are listed in Table 1.

Table 1: Terms introduced into the large concept lattice namespace

	Conglomerate	Function
`CL`	‘`concept-lattice`’	‘**`complete-lattice`**’, ‘`instance`’, ‘`type`’ ‘`instance-embedding`’, ‘`type-embedding`’ ‘**`classification`**’, ‘`opposite`’, ‘`instance-power`’
`CL .MOR`	‘`concept-morphism`’	‘`source`’, ‘`target`’, ‘**`adjoint-pair`**’, ‘`instance`’, ‘`type`’ ‘**`infomorphism`**’ ‘`instance-join`’, ‘`type-meet`’, ‘`right-representation`’, ‘`extent`’, ‘`intent`’, ‘`left-representation`’ ‘`representation`’ ‘`opposite`’, ‘`composition`’, ‘`identity`’ ‘`instance-power`’

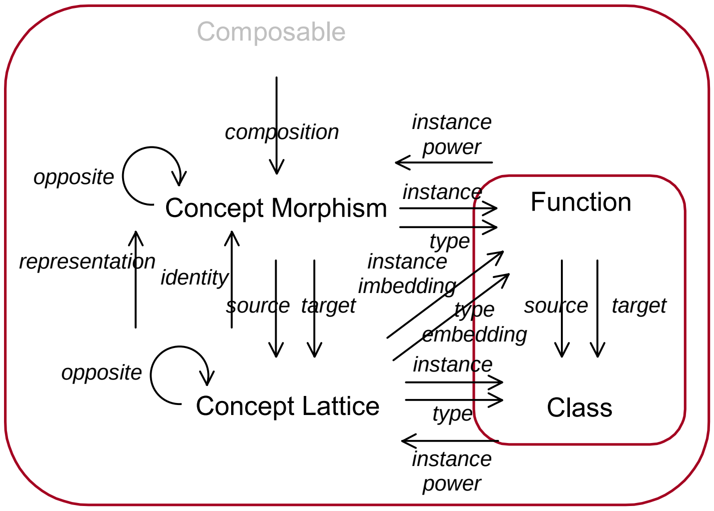


Diagram 3: Core Conglomerates and Functions for Concept Lattices

Concept Lattices

`CL`

$$\begin{tikzcd} \mathit{typ}(\boldsymbol{L}) \arrow[d, "\tau_{\boldsymbol{L}}"] \\ \mathit{latt}(\boldsymbol{L}) \arrow[d, dash, "\leq_{\boldsymbol{L}}"] \\ \mathit{latt}(\boldsymbol{L}) \\ \mathit{inst}(\boldsymbol{L}) \arrow[u, "\iota_{\boldsymbol{L}}"] \end{tikzcd}$$

Figure 4: Concept Lattice

- An (abstract) *concept lattice* $\boldsymbol{L} = \langle \mathit{latt}(\boldsymbol{L}), \mathit{inst}(\boldsymbol{L}), \mathit{typ}(\boldsymbol{L}), \iota_{\boldsymbol{L}}, \tau_{\boldsymbol{L}} \rangle$ consists of a complete lattice $\mathit{latt}(\boldsymbol{L})$, two classes $\mathit{inst}(\boldsymbol{L})$ and $\mathit{typ}(\boldsymbol{L})$ called the instance class and the type class of $\boldsymbol{L}$, respectively; along with two functions, an instance embedding function $\iota_{\boldsymbol{L}} : \mathit{inst}(\boldsymbol{L}) \rightarrow \mathit{latt}(\boldsymbol{L})$ and a type embedding function $\tau_{\boldsymbol{L}} : \mathit{typ}(\boldsymbol{L}) \rightarrow \mathit{latt}(\boldsymbol{L})$, which satisfying the following conditions.
  - The image $\iota_{\boldsymbol{L}}(\mathit{inst}(\boldsymbol{A}))$ is join-dense in $\mathit{latt}(\boldsymbol{L})$.
  - The image $\tau_{\boldsymbol{L}}(\mathit{typ}(\boldsymbol{A}))$ is meet-dense in $\mathit{latt}(\boldsymbol{L})$.

```
(1) (CNG$conglomerate concept-lattice)

(2) (CNG$function complete-lattice)
    (CNG$signature lattice concept-lattice LAT$complete-lattice)

(3) (CNG$function instance)
    (CNG$signature instance concept-lattice SET$class)

(4) (CNG$function type)
    (CNG$signature type concept-lattice SET$class)

(5) (CNG$function instance-embedding)
    (CNG$signature instance-embedding concept-lattice SET.FTN$function)
    (forall (?l (concept-lattice ?l))
        (and (= (SET.FTN$source (instance-embedding ?l))
                (instance ?l))
             (= (SET.FTN$target (instance-embedding ?l))
                (ORD$class (LAT$underlying (complete-lattice ?l))))))

(5) (CNG$function type-embedding)
    (CNG$signature type-embedding concept-lattice SET.FTN$function)
    (forall (?l (concept-lattice ?l))
        (and (= (SET.FTN$source (type-embedding ?l))
                (type ?l))
             (= (SET.FTN$target (type-embedding ?l))
                (ORD$class (LAT$underlying (complete-lattice ?l))))))

    (forall (?a (CLS$classification ?a))
        (and ((join-dense (complete-lattice ?l))
                 (SET.FTN$image (instance-embedding ?l)))
             ((meet-dense (complete-lattice ?l))
                 (SET.FTN$image (type-embedding ?l)))))
```

- Any concept lattice $\boldsymbol{L} = \langle \mathit{latt}(\boldsymbol{L}), \mathit{inst}(\boldsymbol{L}), \mathit{typ}(\boldsymbol{L}), \iota_{\boldsymbol{L}}, \tau_{\boldsymbol{L}} \rangle$ has an associated classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \vDash_{\boldsymbol{A}} \rangle$ whose incidence relation is defined by $i \vDash_{\boldsymbol{A}} t$ iff $\iota(i) \leq_{\boldsymbol{A}} \tau(t)$.

```
(6) (CNG$function classification)
    (CNG$signature classification concept-lattice CLS$classification)
    (forall (?l (concept-lattice ?l))
        (and (= (CLS$instance (classification ?l)) (instance ?l))
             (= (CLS$type (classification ?l)) (type ?l))
             (forall (?i ((instance ?l) ?i) ?t ((type ?l) ?t))
                 (<=> ((CLS$incidence (classification ?l)) [?i ?t])
                      ((ORD$extent (LAT$underlying (complete-lattice ?l)))
                          [(instance-embedding ?l) ?i) (type-embedding ?l) ?t)])))))
```

- From properties discussed above, it can be immediately proven that the composition of '`concept-lattice`' and '`classification`' is the identity on the '`classification`' conglomerate. We state this in an external namespace.

```
    (forall (?c (CLS$classification ?c))
        (= (CL$classification (CLS.CL$concept-lattice ?c)) ?c))
```

- Also, from properties discussed above, it can be immediately proven that for any complete lattice L, the complete lattice of the concept lattice of *L* is *L* itself; that is, that the composition of '`concept-lattice`' and '`complete-lattice`' is the identity on the '`complete-lattice`' conglomerate. We state this in an external namespace.

```
(forall (?l (LAT$complete-lattice ?l))
    (= (CL$complete-lattice (LAT$concept-lattice ?l)) ?l))
```

- For any concept lattice $\boldsymbol{L} = \langle \mathit{latt}(\boldsymbol{L}), \mathit{inst}(\boldsymbol{L}), \mathit{typ}(\boldsymbol{L}), \iota_{\boldsymbol{L}}, \tau_{\boldsymbol{L}} \rangle$, the *opposite* or *dual* of $\boldsymbol{L}$ is the concept lattice $\boldsymbol{L}^{\perp} = \langle \mathit{latt}(\boldsymbol{L})^{\perp}, \mathit{typ}(\boldsymbol{L}), \mathit{inst}(\boldsymbol{L}), \tau_{\boldsymbol{L}}, \iota_{\boldsymbol{L}} \rangle$, whose instances are the types of $\boldsymbol{L}$, whose types are the instances of $\boldsymbol{L}$, whose instance embedding function is the type embedding function of $\boldsymbol{L}$, whose type embedding function is the instance embedding function of $\boldsymbol{L}$, and whose complete lattice is the opposite of the complete lattice of $\boldsymbol{L}$ (turn it upside down). Axiom (7) specifies the opposite operator on concept lattices.

```
(7) (CNG$function opposite)
    (CNG$signature opposite concept-lattice concept-lattice)
    (forall (?l (concept-lattice ?l))
        (and (= (complete-lattice (opposite ?l))
                (LAT$opposite (complete-lattice ?l)))
             (= (instance (opposite ?l)) (type ?l))
             (= (type (opposite ?l)) (instance ?l))
             (= (instance-embedding (opposite ?l)) (type-embedding ?l))
             (= (type-embedding (opposite ?l)) (instance-embedding ?l))))
```

- For any class *A* the *instance power concept lattice* $\wp A = \langle\langle \wp A, \subseteq_A, \cap_A, \cup_A \rangle, A, \wp A, \{\text{-}\}_A, id_{\wp A} \rangle$ over *A* is defined as follows: the complete lattice is the power lattice generated by *A*, the instance class is *A*; the type class is the power class $\wp A$ (so that a type is a subclass of *A*), the instance embedding function is the singleton function for *A*, and the type-embedding function is the identity. Axiom (8) specifies the instance power operator from classes to concept lattices.

```
(8) (CNG$function instance-power)
    (CNG$signature instance-power SET$class concept-lattice)
    (forall (?c (SET$class ?c))
        (and (= (complete-lattice (instance-power ?c)) (LAT$power ?c))
             (= (instance (instance-power ?c)) ?c)
             (= (type (instance-power ?c)) (SET$power ?c))
             (= (instance-embedding (instance-power ?c)) (SET.FTN$singleton ?c))
             (= (type-embedding (instance-power ?c))
                (SET.FTN$identity (SET$power ?c)))))
```

Concept Morphisms

`CL.MOR`

- Concept lattices are related through concept morphisms. An (abstract) *concept morphism* $\boldsymbol{f} : \boldsymbol{L} \rightleftarrows \boldsymbol{K}$ from (abstract) concept lattice $\boldsymbol{L}$ to (abstract) concept lattice $\boldsymbol{K}$ consists of a pair of ordinary oppositely directed functions, $\mathit{inst}(\boldsymbol{f}) : \mathit{inst}(\boldsymbol{K}) \rightarrow \mathit{inst}(\boldsymbol{L})$ and $\mathit{typ}(\boldsymbol{f}) : \mathit{typ}(\boldsymbol{L}) \rightarrow \mathit{typ}(\boldsymbol{K})$, between instance classes and type classes, and an adjoint pair $\mathit{adj}(\boldsymbol{f}) : \boldsymbol{K} \rightleftarrows \boldsymbol{L}$ of monotonic functions, where the right adjoint $\mathit{right}(\boldsymbol{f}) : \boldsymbol{L} \rightarrow \boldsymbol{K}$ is a monotonic function in the forward direction (for the concept lattice morphism, not the adjoint pair) that preserves types (in the sense that the upper rectangle in Figure 5 is commutative)

$$\tau_{\boldsymbol{L}} \cdot \mathit{right}(\mathit{adj}(\boldsymbol{f})) = \mathit{typ}(\boldsymbol{f}) \cdot \tau_{\boldsymbol{K}}$$

and the left adjoint $\mathit{left}(\boldsymbol{f}) : \boldsymbol{K} \rightarrow \boldsymbol{L}$ is a monotonic function in the reverse direction that preserves instances (in the sense that the lower rectangle in Figure 5 is commutative)

$$\iota_{\boldsymbol{K}} \cdot \mathit{left}(\mathit{adj}(\boldsymbol{f})) = \mathit{inst}(\boldsymbol{f}) \cdot \iota_{\boldsymbol{L}} \, .$$

$$\begin{tikzcd}
\mathit{typ}(\boldsymbol{L}) \arrow[r, "\mathit{typ}(\boldsymbol{f})"] \arrow[d, "\tau_{\boldsymbol{L}}"] & \mathit{typ}(\boldsymbol{K}) \arrow[d, "\tau_{\boldsymbol{K}}"] \\
\boldsymbol{L} \arrow[r, "\mathit{right}(\mathit{adj}(\boldsymbol{f}))"] \arrow[d, dash, "\leq_{\boldsymbol{L}}"] & \boldsymbol{K} \arrow[d, dash, "\leq_{\boldsymbol{K}}"] \\
\boldsymbol{L} & \boldsymbol{K} \arrow[l, "\mathit{left}(\mathit{adj}(\boldsymbol{f}))"'] \\
\mathit{inst}(\boldsymbol{L}) \arrow[u, "\iota_{\boldsymbol{L}}"] & \mathit{inst}(\boldsymbol{K}) \arrow[l, "\mathit{inst}(\boldsymbol{f})"'] \arrow[u, "\iota_{\boldsymbol{K}}"]
\end{tikzcd}$$

Figure 5: Concept Lattice Morphism

Axiom (1) specifies the conglomerate of concept morphisms. Axiom (4) defines the overlying adjoint pair. Note the contravariance between the concept morphism and the adjoint pair – the concept morphism is oriented in the same direction as the type function, whereas the adjoint pair is oriented in the same direction as the left adjoint.

```
(1) (CNG$conglomerate concept-morphism)

(2) (CNG$function source)
    (CNG$signature source concept-morphism CL$concept-lattice)

(3) (CNG$function target)
    (CNG$signature target concept-morphism CL$concept-lattice)

(4) (CNG$function adjoint-pair)
    (CNG$signature adjoint-pair concept-morphism LAT.ADJ$adjoint-pair)
    (forall (?f (concept-morphism ?f))
        (and (= (LAT.ADJ$source (adjoint-pair ?f))
                (CL$complete-lattice (target ?f)))
             (= (LAT.ADJ$target (adjoint-pair ?f))
                (CL$complete-lattice (source ?f)))))

(5) (CNG$function instance)
    (CNG$signature instance concept-morphism SET.FTN$function)
    (forall (?f (concept-morphism ?f))
        (and (= (SET.FTN$source (instance ?f)) (CL$instance (target ?f)))
             (= (SET.FTN$target (instance ?f)) (CL$instance (source ?f)))
             (= (SET.FTN$composition
                    (CL$instance-embedding (target ?f))
                    (LAT.ADJ$left (adjoint-pair ?f)))
                (SET.FTN$composition
                    (instance ?f)
                    (CL$instance-embedding (source ?f))))))

(6) (CNG$function type)
    (CNG$signature type concept-morphism SET.FTN$function)
    (forall (?f (concept-morphism ?f))
        (and (= (SET.FTN$source (type ?f)) (CL$type (source ?f)))
             (= (SET.FTN$target (type ?f)) (CL$type (target ?f)))
             (= (SET.FTN$composition
                    (CL$type-embedding (source ?f))
                    (LAT.ADJ$right (adjoint-pair ?f)))
                (SET.FTN$composition
                    (type ?f)
                    (CL$type-embedding (target ?f))))))
```

- Any concept morphism $\boldsymbol{f} : \boldsymbol{L} \rightleftarrows \boldsymbol{K}$ has an associated infomorphism $\mathsf{info}(\boldsymbol{f}) : \mathsf{cls}(\boldsymbol{L}) \rightleftarrows \mathsf{cls}(\boldsymbol{K})$ whose fundamental property, expressed as

  $$\mathsf{inst}(\mathsf{info}(\boldsymbol{f}))(i) \vDash_{\mathsf{cls}(\boldsymbol{L})} t \text{ iff } i \vDash_{\mathsf{cls}(\boldsymbol{K})} \mathsf{typ}(\mathsf{info}(\boldsymbol{f}))(t)$$

  for all instances $i \in \mathsf{inst}(\boldsymbol{K})$ and all types $t \in \mathsf{typ}(\boldsymbol{L})$, is an easy translation of the adjointness condition for the adjoint pair $\mathsf{adj}(\boldsymbol{f})$ and the commutativity of the instance/type functions with the left/right monotonic functions.

```
(7) (CNG$function infomorphism)
    (CNG$signature infomorphism concept-morphism CLS.INFO$infomorphism)
    (forall (?f (concept-morphism ?l))
        (and (= (CLS.INFO$instance (infomorphism ?f)) (instance ?f))
             (= (CLS.INFO$type (infomorphism ?f)) (type ?f))))
```

- From properties discussed above, it can be immediately proven that the composition of '`concept-morphism`' and '`infomorphism`' is the identity on the '`infomorphism`' conglomerate. We state this in an external namespace.

```
    (forall (?f (CLS.INFO$infomorphism ?f))
        (= (CL.MOR$infomorphism (CLS.CL$concept-morphism ?f)) ?f))
```

- For any concept lattice $\boldsymbol{L}$, the concept lattice of the classification of $\boldsymbol{L}$ is related to $\boldsymbol{L}$ through the following two functions.

```
(8) (CNG$function instance-join)
    (CNG$signature instance-join CL$concept-lattice SET.FTN$function)
```

```
        (forall (?l (CL$concept-lattice ?l))
            (and (= (source (instance-join ?l))
                    (ORD$class (LAT$underlying (CL$complete-lattice
                        (CLS$concept-lattice (CL$classification ?l))))))
                 (= (target (instance-join ?l))
                    (ORD$class (LAT$underlying (CL$complete-lattice ?l))))
                 (= (instance-join ?l)
                    (SET.FTN$composition
                        (SET.FTN$composition
                            (CLS$extent (CL$classification ?l))
                            (SET.FTN$power (CL$instance-embedding ?l)))
                        (LAT$join (CL$complete-lattice ?l))))))

    (9) (CNG$function type-meet)
        (CNG$signature type-meet CL$concept-lattice SET.FTN$function)
        (forall (?l (CL$concept-lattice ?l))
            (and (= (source (type-meet ?l))
                    (ORD$class (LAT$underlying (CL$complete-lattice
                        (CLS$concept-lattice (CL$classification ?l))))))
                 (= (target (type-meet ?l))
                    (ORD$class (LAT$underlying (CL$complete-lattice ?l))))
                 (= (type-meet ?l)
                    (SET.FTN$composition
                        (SET.FTN$composition
                            (CLS$intent (CL$classification ?l))
                            (SET.FTN$power (CL$type-embedding ?l)))
                        (LAT$meet (CL$complete-lattice ?l))))))
```

- The previous two functions can be shown to be identical monotonic functions.

```
    (forall (?l (CL$concept-lattice ?l))
        (and (= (instance-join ?l) (type-meet ?l)))
             (exists (?f (ORD.FTN$monotonic-function ?))
                 (= (instance-join ?l) (ORD.FTN$function ?f)))))
```

- Let us call this common function the *right representation* of ***L***.

```
    (10) (CNG$function right-representation)
         (CNG$signature right-representation CL$concept-lattice SET.FTN$function)
         (= right-representation instance-join)
```

- Any concept lattice ***L*** is indirectly related to the concept lattice of the classification of ***L*** through the following two functions. The *extent* of a concept in ***L*** is the class of all instances whose generated concept is at or below the concept. The *intent* is the dual notion. As we shall observe and axiomatize, both the extent and intent represent concepts of ***L***.

```
    (11) (CNG$function extent)
         (CNG$signature extent CL$concept-lattice SET.FTN$function)
         (forall (?l (CL$concept-lattice ?l))
             (and (= (source (extent ?l))
                     (ORD$class (LAT$underlying (CL$complete-lattice ?l))))
                  (= (target (extent ?l))
                     (SET$power (CL$instance ?l)))
                  (= (extent ?l)
                  (= (SET.FTN$composition (extent ?l) (CLS$extent (CL$classification ?l)))
                     (SET.FTN$composition
                         (SET.FTN$singleton
                             (ORD$class (LAT$underlying (CL$complete-lattice ?l))))
                         (SET.FTN$composition
                             ((ORD$down (LAT$underlying (CL$complete-lattice ?l)))
                             (SET.FTN$inverse-image (CL$instance-embedding ?l)))))))

    (12) (CNG$function intent)
         (CNG$signature intent CL$concept-lattice SET.FTN$function)
         (forall (?l (CL$concept-lattice ?l))
             (and (= (source (intent ?l))
                     (ORD$class (LAT$underlying (CL$complete-lattice ?l))))
                  (= (target (intent ?l))
                     (SET$power (CL$type ?l)))
                  (= (intent ?l)
                     (SET.FTN$composition
                         (SET.FTN$singleton
                             (ORD$class (LAT$underlying (CL$complete-lattice ?l))))
                         (SET.FTN$composition
                             ((ORD$up (LAT$underlying (CL$complete-lattice ?l)))
                             (SET.FTN$inverse-image (CL$type-embedding ?l)))))))
```

- The following fact can be proven: there is a unique function, whose source class is the source of the extent and intent functions, whose target is the class underlying the concept lattice of the classification of ***L***, whose composition with the extent function of the classification of ***L*** is the above extent function, and whose composition with the intent function of the classification of ***L*** is the above intent function. Let us call this function the *left representation* of ***L***.

```
(13) (CNG$function left-representation)
     (CNG$signature left-representation CL$concept-lattice SET.FTN$function)
     (forall (?l (CL$concept-lattice ?l))
         (and (= (source (left-representation ?l))
                 (ORD$class (LAT$underlying (CL$complete-lattice ?l))))
              (= (target (left-representation ?l))
                 (ORD$class (LAT$underlying (CL$complete-lattice
                     (CLS$concept-lattice (CL$classification ?l))))))
              (= (SET.FTN$composition
                     (left-representation ?l)
                     (CLS$extent (CL$classification ?l)))
                 (extent ?l))
              (= (SET.FTN$composition
                     (left-representation ?l)
                     (CLS$intent (CL$classification ?l)))
                 (intent ?l))))
```

- For any concept lattice ***L***, it can be proven that the left and right representation functions are inverse to each other. This demonstrates that the concept lattice of the classification of ***L*** represents ***L*** via left representation (extent and intent functions).

```
     (forall (?l (CL$concept-lattice ?l))
         (and (= (SET.FTN$composition
                     (left-representation ?l)
                     (right-representation ?l))
                 (SET.FTN$identity
                     (ORD$class (LAT$underlying (CL$complete-lattice ?l)))))
              (= (SET.FTN$composition
                     (right-representation ?l)
                     (left-representation ?l))
                 (SET.FTN$identity
                     (ORD$class (LAT$underlying (CL$complete-lattice
                         (CLS$concept-lattice (CL$classification ?l)))))))))
```

- We rephrase this in terms of concept morphisms: for any concept lattice ***L***, there is a *representation* concept morphism from ***L*** to the concept lattice of the classification of ***L***. This is the ***L***th component of a natural isomorphism, demonstrating that the following quasi-categories are categorically equivalent:

  Classification ≡ Concept Lattice.

```
(14) (CNG$function representation)
     (CNG$signature representation CL$concept-lattice concept-morphism)
     (forall (?l (CL$concept-lattice ?l))
         (and (= (source (representation ?l)) ?l)
              (= (target (representation ?l))
                 (CLS$concept-lattice (CL$classification ?l)))
              (= (LAT.ADJ$left (adjoint-pair (representation ?l)))
                 (left-representation ?l))
              (= (LAT.ADJ$right (adjoint-pair (representation ?l)))
                 (right-representation ?l))
              (= (instance (representation ?l))
                 (SET.FTN$identity (CL$instance ?l)))
              (= (type (representation ?l))
                 (SET.FTN$identity (CL$instance ?l)))))
```

- In addition, from properties discussed above, it can be immediately proven that for any (complete lattice) adjoint pair *f*, the adjoint pair of the concept morphism of *f* is *f* itself; that is, that the composition of '`ORD.LAT$concept-morphism`' and '`adjoint-pair`' is the identity on the '`adjoint-pair`' conglomerate. We state this in an external namespace.

```
(forall (?f (LAT.ADJ$adjoint-pair ?f))
   (= (CL.MOR$adjoint-pair (LAT.ADJ$concept-morphism ?f)) ?f))
```

Duality can be extended to concept morphisms. For any concept morphism $\boldsymbol{f} : \boldsymbol{L} \rightleftarrows \boldsymbol{K}$, the *opposite* or *dual* of $\boldsymbol{f}$ is the concept morphism $\boldsymbol{f}^{\perp} : \boldsymbol{K}^{\perp} \rightleftarrows \boldsymbol{L}^{\perp}$, whose source concept lattice is the opposite of the target of $\boldsymbol{f}$, whose target concept lattice is the opposite of the source of $\boldsymbol{f}$, whose adjoint pair is the opposite of the adjoint pair of $\boldsymbol{f}$, whose instance function is the type function of $\boldsymbol{f}$, whose type function is the instance function of $\boldsymbol{f}$, and whose preservation conditions have been dualized.

Axiom (15) specifies the opposite operator on concept morphisms.

```
(15) (CNG$function opposite)
     (CNG$signature opposite concept-morphism concept-morphism)
     (forall (?f (infomorphism ?f))
         (and (= (source (opposite ?f)) (CL$opposite (target ?f)))
              (= (target (opposite ?f)) (CL$opposite (source ?f)))
              (= (adjoint-pair (opposite ?f)) (LAT.ADJ$opposite (adjoint-pair ?f)))
              (= (instance (opposite ?f)) (type ?f))
              (= (type (opposite ?f)) (instance ?f))))
```

- The function '`composition`' operates on any two concept morphisms that are composable in the sense that the target concept lattice of the first is equal to the source concept lattice of the second. Composition, defined in axiom (16), produces a concept morphism, whose components are constructed using composition.

```
(16) (CNG$function composition)
     (CNG$signature composition concept-morphism concept-morphism concept-morphism)
     (forall (?f (concept-morphism ?f) ?g (concept-morphism ?g))
         (<=> (exists (?h (concept-morphism ?h)) (= (composition ?f ?g) ?h))
              (= (target ?f) (source ?g))))
     (forall (?f (concept-morphism ?f) ?g (concept-morphism ?g))
         (=> (= (target ?f) (source ?g))
             (and (= (source (composition ?f ?g)) (source ?f))
                  (= (target (composition ?f ?g)) (target ?g))
                  (= (adjoint-pair (composition ?f ?g))
                     (CL.MOR$composition (adjoint-pair ?g) (adjoint-pair ?f)))
                  (= (instance (composition ?f ?g))
                     (SET.FTN$composition (instance ?g) (instance ?f)))
                  (= (type (composition ?f ?g))
                     (SET.FTN$composition (type ?f) (type ?g))))))
```

- The function '`identity`' defined in axiom (17) associates a well-defined (identity) concept morphism with any concept lattice, whose components are identities.

```
(17) (CNG$function identity)
     (CNG$signature identity CL$concept-lattice concept-morphism)
     (forall (?l (CL$concept-lattice ?l))
         (and (= (source (identity ?l)) ?l)
              (= (target (identity ?l)) ?l)
              (= (adjoint-pair (identity ?l))
                 (CL.MOR$identity (complete-lattice ?l)))
              (= (instance (identity ?l))
                 (SET.FTN$identity (CL$instance ?l)))
              (= (type (identity ?c))
                 (SET.FTN$identity (CL$type ?l)))))
```

- A very useful generic concept morphism represents the "instance power concept morphism construction." For any class function $f : B \rightarrow A$ the components of the *instance power concept morphism*

$$\wp f = \langle\langle \wp f, f^{-1}\rangle, f, f^{-1}\rangle : \wp A \rightleftarrows \wp B$$

over f are defined as follows: the source concept lattice is the instance power concept lattice $\wp A$ = $\langle\langle \wp A, \subseteq_A, \cap_A, \cup_A\rangle, A, \wp A, \{\text{-}\}_A, id_{\wp A}\rangle$ over A, the target concept lattice is the instance power concept lattice $\wp B = \langle\langle \wp B, \subseteq_B, \cap_B, \cup_B\rangle, B, \wp B, \{\text{-}\}_{\boldsymbol{B}}, id_{\wp \boldsymbol{B}}\rangle$ over B, the adjoint pair is the (complete lattice) power adjoint pair over f, the instance function is f, and the type function is the inverse image function $f^{-1} : \wp A \rightarrow \wp B$ from the power-class of A to the power-class of B. Note the contravariance.

```
(18) (CNG$function instance-power)
     (CNG$signature instance-power SET.FTN$function concept-morphism)
     (forall (?f (SET.FTN$function ?f))
         (and (= (source (instance-power ?f)) (CL$instance-power (SET.FTN$target ?f)))
              (= (target (instance-power ?f)) (CL$instance-power (SET.FTN$source ?f)))
              (= (adjoint-pair (instance-power ?f)) (LAT.ADJ$power ?f))
              (= (instance (instance-power ?f)) ?f)
              (= (type (instance-power ?f)) (SET.FTN$inverse-image ?f))))
```

The Namespace of Large Complete Lattices

This namespace will represent large complete lattices and their adjoint pairs. Some of the terms introduced in this namespace are listed in Table 1.

Table 1: Terms introduced into the large complete lattice namespace

	Conglomerate	Function	Example
`LAT`	‘`complete-lattice`’	‘`underlying`’, ‘`meet`’, ‘`join`’ ‘**`classification`**’ ‘`cut`’ ‘`opposite`’, ‘`power`’ ‘**`concept-lattice`**’	
`LAT .ADJ`	‘`adjoint-pair`’	‘`source`’, ‘`target`’, ‘`underlying`’, ‘`left`’, ‘`right`’ ‘**`infomorphism`**’ ‘**`bond`**’ ‘**`cut-forward`**’, ‘**`cut-reverse`**’ ‘`opposite`’, ‘`composition`’, ‘`identity`’ ‘`power`’ ‘**`concept-morphism`**’	
`LAT .MOR`	‘`homomorphism`’	‘`source`’, ‘`target`’, ‘`function`’, ‘`forward`’, ‘`reverse`’ ‘**`bonding-pair`**’, ‘**`cut`**’	

Complete Lattices

`LAT`

○ A partial order $\boldsymbol{L}$ is a *complete lattice* when the meet and join exist for all classes $S \subseteq L$. Then we use the notation $\boldsymbol{L} = \langle L, \leq_L, \sqcap_L, \sqcup_L \rangle$. The underlying partial order is represented by $|\boldsymbol{L}| = \langle L, \leq_L \rangle$.

```
(1) (CNG$conglomerate complete-lattice)

(2) (CNG$function underlying)
    (CNG$signature underlying complete-lattice ORD$partial-order)

(3) (CNG$function meet)
    (CNG$signature meet complete-lattice SET.FTN$function)
    (forall (?l (complete-lattice ?l))
        (and (= (SET.FTN$source (meet ?l)) (SET$power (ORD$class (underlying ?l))))
             (= (SET.FTN$target (meet ?l)) (ORD$class (underlying ?l)))
             (forall (?c ((SET$power (ORD$class (underlying ?l))) ?c))
                 (((greatest ?o) ((lower-bound ?o) ?c)) ((meet ?l) ?c)))))

(4) (CNG$function join)
    (CNG$signature join complete-lattice SET.FTN$function)
    (forall (?l (complete-lattice ?l))
        (and (= (SET.FTN$source (join ?l)) (SET$power (ORD$class (underlying ?l))))
             (= (SET.FTN$target (join ?l)) (ORD$class (underlying ?l)))
             (forall (?c ((SET$power (ORD$class (underlying ?l))) ?c))
                 (((least ?o) ((upper-bound ?o) ?c)) ((join ?l) ?c)))))
```

o Associated with any complete lattice $\boldsymbol{L} = \langle L, \leq_L, \sqcap_L, \sqcup_L \rangle$ is the classification $cls(\boldsymbol{L}) = cls(|\boldsymbol{L}|) = \langle L, L, \leq_L \rangle$, which has L-elements as its instances and types, and the lattice order as its incidence.

```
(5) (CNG$function classification)
    (CNG$signature classification complete-lattice CLS$classification)
    (forall (?l (complete-lattice ?l))
       (= (classification ?l)
          (ORD$classification (underlying ?l))))
```

- For any complete lattice $\boldsymbol{L} = \langle L, \leq_L, \sqcap_L, \sqcup_L\rangle$, there is a special *cut* function $\mathsf{cut}(\boldsymbol{L}) = \updownarrow_L : L \rightarrow \mathsf{lat}(\mathsf{cls}(\boldsymbol{L}))$ defined by $x \mapsto (\downarrow_L x, \uparrow_L x)$, for every element $x \in L$.

```
(6) (CNG$function cut)
    (CNG$signature cut complete-lattice SET.FTN$function)
    (forall (?l (complete-lattice ?l))
        (and (SET.FTN$source (cut ?l)) (ORD$class (underlying ?l)))
             (SET.FTN$target (cut ?l)) (CLS.CL$concept (classification ?l)))
             (= (SET.FTN$composition (cut ?l) (CLS.CL$extent (classification ?l)))
                (ORD$down-embedding (underlying ?l)))
             (= (SET.FTN$composition (cut ?l) (CLS.CL$intent (classification ?l)))
                (ORD$up-embedding (underlying ?l)))))
```

- Here are some preliminary observations that pertain to this cut function.

  On the underlying partial-order of a complete lattice, the composition of the down-embedding and join is the identity on the underlying class. Dually, the composition of the up-embedding and meet is the identity on the underlying class.

```
    (forall (?l (complete-lattice ?l))
        (and (= (SET.FTN$composition (ORD$down-embedding (underlying ?l)) (join ?l))
                (SET.FTN$identity (ORD$class (underlying ?l))))
             (= (SET.FTN$composition (ORD$up-embedding (underlying ?l)) (meet ?l))
                (SET.FTN$identity (ORD$class (underlying ?l))))))
```

- On the associated classification of a complete lattice, the instance generation function factors as the composition of join and the above cut function. Dually, the type generation function is the composition of meet followed by cut.

```
    (forall (?l (complete-lattice ?l))
        (and (= (instance-generation (classification ?l))
                (SET.FTN$composition (join ?l) (cut ?l)))
             (= (type-generation (classification ?l))
                (SET.FTN$composition (meet ?l) (cut ?l)))))
```

- It is important to observe that any concept in $\mathsf{lat}(\mathsf{cls}(\boldsymbol{L}))$ is of the form $(\downarrow_L x, \uparrow_L x)$ for some element $x \in L$. In fact, the cut function is a bijection, and its inverse function has two expressions – it is the composition of conceptual extent and join, and it is the composition of conceptual intent and meet.

```
(forall (?l (complete-lattice ?l))
    (and (= (SET.FTN$composition
                (cut ?l)
                (SET.FTN$composition (CLS.CL$extent (classification ?l)) (join ?l)))
            (SET.FTN$identity (ORD$class (underlying ?l))))
         (= (SET.FTN$composition
                (SET.FTN$composition (CLS.CL$extent (classification ?l)) (join ?l))
                (cut ?l))
            (SET.FTN$identity (CLS.CL$concept (classification ?l))))
         (= (SET.FTN$composition
                (cut ?l)
                (SET.FTN$composition (CLS.CL$intent (classification ?l)) (meet ?l)))
            (SET.FTN$identity (ORD$class (underlying ?l))))
         (= (SET.FTN$composition
                (SET.FTN$composition (CLS.CL$intent (classification ?l)) (meet ?l))
                (cut ?l))
            (SET.FTN$identity (CLS.CL$concept (classification ?l))))))
```

- For any complete lattice $\boldsymbol{L} = \langle L, \leq_L, \sqcap_L, \sqcup_L\rangle$, the *opposite* or *dual* of $\boldsymbol{L}$ is the complete lattice $\boldsymbol{L}^{\perp} = \langle L, \geq_L, \sqcup_L, \sqcap_L\rangle$, whose underlying partial order is the opposite of the underlying partial order of $\boldsymbol{L}$, whose meet is the join of $\boldsymbol{L}$, and whose join is the meet of $\boldsymbol{L}$. Axiom (6) specifies the opposite operator on complete lattices.

```
(6) (CNG$function opposite)
    (CNG$signature opposite complete-lattice complete-lattice)
    (forall (?l (complete-lattice ?l))
        (and (= (underlying (opposite ?l)) (ORD$opposite (underlying ?l)))
             (= (meet (opposite ?l)) (join ?l))
             (= (join (opposite ?l)) (meet ?l))))
```

- For any class C, the power complete lattice $\wp(C) = \langle \wp C, \subseteq_C, \cap_C, \cup_C \rangle$ is the power class with subclass ordering, intersection as meet and union as join. There is a CNG function *power* : class $\rightarrow$ complete-lattice that maps a class to its power lattice.

```
(7) (CNG$function power)
    (CNG$signature power SET$class complete-lattice)
    (forall (?c (SET$class ?c))
        (and (= (underlying (power ?c)) (ORD$power ?c))
             (= (meet (power ?c)) (SET$intersection ?c))
             (= (join (power ?c)) (SET$union ?c))))
```

- Any complete lattice $\boldsymbol{L} = \langle L, \leq_L, \sqcap_L, \sqcup_L \rangle$ is a concept lattice $\boldsymbol{L} = \langle \boldsymbol{L}, L, L, id_{\boldsymbol{L}}, id_{\boldsymbol{L}} \rangle$, where the instance and type classes are the underlying class of the lattice and the instance and type embeddings are the identity function. Axiom (8) represents this as a CNG function.

```
(8) (CNG$function concept-lattice)
    (CNG$signature concept-lattice complete-lattice CL$concept-lattice)
    (forall (?l (complete-lattice ?l))
        (and (= (CL$complete-lattice (concept-lattice ?l)) ?l)
             (= (CL$instance (concept-lattice ?l)) (ORD$class (underlying ?l)))
             (= (CL$type (concept-lattice ?l)) (ORD$class (underlying ?l)))
             (= (CL$instance-embedding (concept-lattice ?l))
                (SET.FTN$identity (ORD$class (underlying ?l))))
             (= (CL$type-embedding (concept-lattice ?l))
                (SET.FTN$identity (ORD$class (underlying ?l))))))
```

- An easy check shows that the classification of the concept lattice of a complete lattice $\boldsymbol{L}$ is the same as the classification associated with $\boldsymbol{L}$.

```
    (forall (?l (complete-lattice ?l))
        (= (CL$classification (concept-lattice ?l))
           (classification ?l)))
```

Complete Adjoint

`LAT.ADJ`

$$\begin{tikzcd}
L \arrow[r, "left(f)"] \arrow[d, dash, "\geq_L"'] \arrow[dr, dash] & K \arrow[d, dash, "\geq_K"] \\
L & K \arrow[l, "right(f)"]
\end{tikzcd}$$

Figure 4: Adjoint Pair

- Complete lattices are related through *adjoint pairs*. This is a restriction to complete lattices of the adjoint pair notion for preorders. For an adjoint pair $\langle \varphi, \psi \rangle : \boldsymbol{L} \rightleftarrows \boldsymbol{K}$ between complete lattices $\boldsymbol{L} = \langle \boldsymbol{L}, \leq_{\boldsymbol{L}}, \sqcap_{\boldsymbol{L}}, \sqcup_{\boldsymbol{L}} \rangle$, and $\boldsymbol{K} = \langle \boldsymbol{K}, \leq_{\boldsymbol{K}}, \sqcap_{\boldsymbol{K}}, \sqcup_{\boldsymbol{K}} \rangle$, the left adjoint $\varphi : \boldsymbol{L} \rightarrow \boldsymbol{K}$ is join-preserving and the right-adjoint $\psi : \boldsymbol{K} \rightarrow \boldsymbol{L}$ is meet-preserving. The two functions determine each other as follows.

  $\varphi(l) = \sqcap_{\boldsymbol{K}} \{k \in K \mid l \leq_{\boldsymbol{L}} \psi(k)\}$
  $\psi(k) = \vee_{\boldsymbol{L}} \{l \in L \mid k \leq_{\boldsymbol{K}} \varphi(l)\}$

  For example, suppose $\psi : \boldsymbol{K} \rightarrow \boldsymbol{L}$ is a meet-preserving monotonic function, and define the function $\varphi : L \rightarrow K$ as above.
  - If $l_1 \leq_{\boldsymbol{L}} l_2$ then $\{k \in K \mid l_1 \leq_{\boldsymbol{L}} \psi(k)\} \supseteq \{k \in K \mid l_2 \leq_{\boldsymbol{L}} \psi(k)\}$. Hence, $\varphi(l_1) \leq_{\boldsymbol{K}} \varphi(l_2)$.
  - If $l \leq_{\boldsymbol{L}} \psi(k)$ then $k \in \{\tilde{k} \in K \mid l \leq_{\boldsymbol{L}} \psi(\tilde{k})\}$. Hence, $\varphi(l) \leq_{\boldsymbol{K}} k$.
  - $\psi(\varphi(l)) = \psi(\sqcap_{\boldsymbol{K}} \{k \in K \mid l \leq_{\boldsymbol{L}} \psi(k)\}) = \sqcap_{\boldsymbol{L}} \{\psi(k) \in L \mid l \leq_{\boldsymbol{L}} \psi(k)\} \geq_{\boldsymbol{L}} l$.
  - If $\varphi(l) \leq_{\boldsymbol{K}} k$ then $\psi(\varphi(l)) \leq_{\boldsymbol{L}} \psi(k)$. Hence, $l \leq_{\boldsymbol{L}} \psi(k)$.

```
(1) (CNG$conglomerate adjoint-pair)

(2) (CNG$function source)
    (CNG$signature source adjoint-pair LAT$complete-lattice)

(3) (CNG$function target)
    (CNG$signature target adjoint-pair LAT$complete-lattice)

(4) (CNG$function underlying)
    (CNG$signature underlying adjoint-pair ORD.ADJ$adjoint-pair)
    (forall (?a (adjoint-pair ?a))
        (and (= (ORD.ADJ$source (underlying ?a)) (LAT$underlying (source ?a)))
             (= (ORD.ADJ$target (underlying ?a)) (LAT$underlying (target ?a)))))
```

- We define the following two terms for convenience of reference.

```
(5) (CNG$function left)
    (CNG$signature left adjoint-pair ORD.FTN$monotonic-function)
    (forall (?a (adjoint-pair ?a))
        (= (left ?a) (ORD.ADJ$left (underlying ?a))))

(6) (CNG$function right)
    (CNG$signature right adjoint-pair ORD.FTN$monotonic-function)
    (forall (?a (adjoint-pair ?a))
        (= (right ?a) (ORD.ADJ$right (underlying ?a))))
```

- Any adjoint pair is an infomorphism. The CNG function in axiom (7) realizes this assertion.

```
(7) (CNG$function infomorphism)
    (CNG$signature infomorphism adjoint-pair CLS.INFO$infomorphism)
    (forall (?a (adjoint-pair ?a))
        (= (infomorphism ?a)
           (ORD.ADJ$infomorphism (underlying ?a))))
```

- Associated with any adjoint pair $\boldsymbol{f} = \langle \mathsf{left}(\boldsymbol{f}), \mathsf{right}(\boldsymbol{f})\rangle = \langle \varphi, \psi\rangle : \boldsymbol{L} \rightleftarrows \boldsymbol{K}$, from complete lattice $\boldsymbol{L}$ to complete lattice $\boldsymbol{K}$, is the bond $\mathsf{bnd}(\boldsymbol{f}) : \mathsf{cls}(\boldsymbol{K}) \rightleftarrows \mathsf{cls}(\boldsymbol{L})$ (whose classification relation is) defined by the adjointness property: $l\,\mathsf{bnd}(\boldsymbol{f})k$ iff $\varphi(l) \leq_{\boldsymbol{K}} k$ iff $l \leq_{\boldsymbol{L}} \psi(k)$ for all elements $l \in L$ and $k \in K$. The closure property of bonds is obvious, since $l\,\mathsf{bnd}(\boldsymbol{f}) = {\uparrow_{\boldsymbol{K}}}\,\varphi(l)$ for all elements $l \in L$ and $\mathsf{bnd}(\boldsymbol{f})k = {\downarrow_{\boldsymbol{L}}}\,\psi(k)$ for all elements $k \in K$.

  This can equivalently be defined in terms of either the left or the right monotonic function. Here we use the left monotonic function. In particular, we define the classification of the bond via the right operator that maps the left function to a classification relation in the presence of the underlying partial order of the target complete lattice.

```
(8) (CNG$function bond)
    (CNG$signature bond adjoint-pair CLS.BND$bond)
    (forall (?a (adjoint-pair ?a))
        (and (= (CLS.BND$source (bond ?a)) (LAT$classification (target ?a)))
             (= (CLS.BND$target (bond ?a)) (LAT$classification (source ?a)))
             (= (CLS.BND$classification (bond ?a))
                ((SET.FTN$right (LAT$underlying (target ?a))) (left ?a)))))
```

- This bond is the bond of the underlying adjoint pair. This fact could be used as a definition.

```
    (forall (?a (adjoint-pair ?a))
        (= (bond ?a)
           (ORD.ADJ$bond (underlying ?a))))
```

- The functor composition $\mathsf{B} \circ \mathsf{A}$ is naturally isomorphic to the identity functor $Id_{\mathsf{Complete\ Adjoint}}$. To see this, let $\boldsymbol{L} = \langle L, \leq_{\boldsymbol{L}}, \wedge_{\boldsymbol{L}}, \vee_{\boldsymbol{L}}\rangle$ be a complete lattice with associated classification $\mathsf{B}(\boldsymbol{L}) = \langle L, L, \leq_L\rangle$. Part of the fundamental theorem of concept lattices asserts the isomorphism $\boldsymbol{L} \cong \mathsf{A}(\mathsf{B}(\boldsymbol{L}))$. In particular, formal concepts of $\mathsf{A}(\mathsf{B}(\boldsymbol{L}))$ are cuts of the form $({\downarrow_{\boldsymbol{L}}}x, {\uparrow_{\boldsymbol{L}}}x)$ for elements $x \in L$. The cut monotonic function $\boldsymbol{L} \to \mathsf{A}(\mathsf{B}(\boldsymbol{L}))$ was defined in the complete lattice namespace by $x \mapsto ({\downarrow_{\boldsymbol{L}}}x, {\uparrow_{\boldsymbol{L}}}x)$ for every element $x \in L$. This function is a bijection. Let $\langle \varphi, \psi\rangle : L \rightleftarrows K$ be a complete adjoint, an adjoint pair of monotonic functions, between complete lattices $L = \langle L, \leq_L, \wedge_L, \vee_L\rangle$, and $K = \langle K, \leq_K, \wedge_K, \vee_K\rangle$ with associated bond $\mathsf{B}(\langle \varphi, \psi\rangle) : \mathsf{B}(L) \to \mathsf{B}(K)$. Then, the right adjoint of $\mathsf{A}(\mathsf{B}(\langle \varphi, \psi\rangle)$ maps $({\downarrow_L}x, {\uparrow_L}x) \mapsto ({\downarrow_K}\psi(x), {\uparrow_K}\psi(x))$ and the left adjoint of $\mathsf{A}(\mathsf{B}(\langle \varphi, \psi\rangle)$ maps $({\downarrow_K}y, {\uparrow_K}y) \mapsto ({\downarrow_L}\varphi(y), {\uparrow_L}\varphi(y))$. So, up to isomorphism, $\mathsf{A}(\mathsf{B}(\langle \varphi, \psi\rangle)$ is the same as $\langle \varphi, \psi\rangle$. This defines the natural isomorphism: $\mathsf{B} \circ \mathsf{A} \equiv Id_{\mathsf{Complete\ Adjoint}}$.

  The *cut-forward* adjoint pair has the cut function as its right monotonic function and the composition of extent and join (or the composition of intent and meet) as its left monotonic function. This adjoint pair is a pair of inverse functions, and hence is an isomorphism from the complete lattice of its associated classification $\mathsf{A}(\mathsf{B}(\boldsymbol{L}))$ to a complete lattice $\boldsymbol{L}$. The *cut-reverse* adjoint pair flips the inverses – it has the cut function as its left monotonic function and the composition of extent and join as its right monotonic function. This adjoint pair is a pair of inverse functions, and hence is an isomorphism from a complete lattice $\boldsymbol{L}$ to the complete lattice of its associated classification $\mathsf{A}(\mathsf{B}(\boldsymbol{L}))$.

```
(9) (CNG$function cut-forward)
```

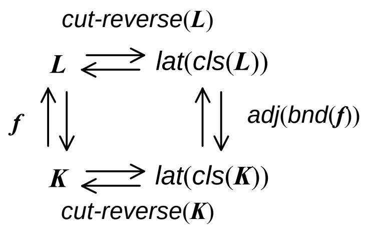
```
        (CNG$signature cut-forward LAT$complete-lattice adjoint-pair)
        (forall (?l (complete-lattice ?l))
            (and (= (source (cut-forward ?l))
                    (CLS.CL$complete-lattice (LAT$classification ?l)))
                 (= (target (cut-forward ?l)) ?l)
                 (= (ORD.FTN$function (left (cut-forward ?l)))
                    (SET.FTN$composition
                        (CLS.CL$extent (LAT$classification ?l))
                        (LAT$join ?l)))
                 (= (ORD.FTN$function (right (cut-forward ?l))) (LAT$cut ?l))))

    (10) (CNG$function cut-reverse)
         (CNG$signature cut-reverse LAT$complete-lattice adjoint-pair)
         (forall (?l (complete-lattice ?l))
             (and (= (source (cut-reverse ?l)) ?l)
                  (= (target (cut-reverse ?l))
                     (CLS.CL$complete-lattice (LAT$classification ?l)))
                  (= (ORD.FTN$function (left (cut-reverse ?l))) (LAT$cut ?l))
                  (= (ORD.FTN$function (right (cut-reverse ?l)))
                     (SET.FTN$composition
                         (CLS.CL$extent (LAT$classification ?l))
                         (LAT$join ?l)))))
```

- The *composition* of two composable adjoint pairs $\boldsymbol{F} : \boldsymbol{A} \rightarrow \boldsymbol{B}$ and $\boldsymbol{G} : \boldsymbol{B} \rightarrow \boldsymbol{C}$ is the composition of the underlying adjoint pairs.

```
    (11) (CNG$function composition)
         (CNG$signature composition adjoint-pair adjoint-pair adjoint-pair)
         (forall (?f (adjoint-pair ?f) ?g (adjoint-pair ?g))
             (<=> (exists (?h (adjoint-pair ?h)) (= (composition ?f ?g) ?h))
                  (= (target ?f) (source ?g))))
         (forall (?f (adjoint-pair ?f) ?g (adjoint-pair ?g))
             (=> (= (target ?f) (source ?g))
                 (and (= (source (composition ?f ?g)) (source ?f))
                      (= (target (composition ?f ?g)) (target ?g))
                      (= (underlying (composition ?f ?g))
                         (ORD.ADJ$composition (underlying ?f) (underlying ?g))))))
```

- Let $\boldsymbol{f} = \langle \varphi, \psi \rangle : \boldsymbol{L} \rightleftarrows \boldsymbol{K}$ be a complete adjoint, an adjoint pair of monotonic functions, between complete lattices $\boldsymbol{L} = \langle L, \leq_L, \wedge_L, \vee_L \rangle$, and $\boldsymbol{K} = \langle K, \leq_K, \wedge_K, \vee_K \rangle$ with associated bond $bnd(\boldsymbol{f}) : cls(\boldsymbol{L}) \rightarrow cls(\boldsymbol{K})$. Then, the right adjoint of $adj(bnd(\boldsymbol{f}))$ maps $(\downarrow_L x, \uparrow_L x) \mapsto (\downarrow_K \psi(x), \uparrow_K \psi(x))$ and the left adjoint of $adj(bnd(\boldsymbol{f}))$ maps $(\downarrow_K y, \uparrow_K y) \mapsto (\downarrow_L \varphi(y), \uparrow_L \varphi(y))$. This is equivalent to the natural isomorphism (commuting Diagram 3):

$$cut\text{-}reverse(\boldsymbol{L}) \cdot adj(bnd(\boldsymbol{f})) = \boldsymbol{f} \cdot cut\text{-}reverse(\boldsymbol{K}).$$



$$\begin{tikzcd}
\boldsymbol{L} \arrow[r, shift left, "cut\text{-}reverse(\boldsymbol{L})"] \arrow[d, shift left] & lat(cls(\boldsymbol{L})) \arrow[l, shift left] \arrow[d, shift left, "adj(bnd(\boldsymbol{f}))"] \\
\boldsymbol{K} \arrow[u, shift left, "\boldsymbol{f}"] \arrow[r, shift left] & lat(cls(\boldsymbol{K})) \arrow[l, shift left, "cut\text{-}reverse(\boldsymbol{K})"] \arrow[u, shift left]
\end{tikzcd}$$

Diagram 3: Natural isomorphism

So, up to isomorphism, $adj(bnd(\boldsymbol{f}))$ is the same as $\boldsymbol{f}$. This defines the natural isomorphism: $B \circ A \cong Id_{\text{Complete Adjoint}}$.

```
         (forall (?a (adjoint-pair ?a))
             (= (composition (cut-reverse (source ?a)) (adjoint-pair (bond ?a)))
                (composition ?a (cut-reverse (target ?a)))))
```

- The identity adjoint pair at a complete lattice $\boldsymbol{L}$ is the identity adjoint pair of the underlying order.

```
    (12) (CNG$function identity)
         (CNG$signature identity LAT$complete-lattice adjoint-pair)
         (forall (?l (LAT$complete-lattice ?l))
             (and (= (source (identity ?l)) ?l)
                  (= (target (identity ?l)) ?l)
                  (= (underlying (identity ?l)) (ORD.ADJ$identity (LAT$underlying ?l)))))
```

- Duality can be extended to adjoint pairs. For any adjoint pair $\boldsymbol{f} : \boldsymbol{L} \rightleftarrows \boldsymbol{K}$, the *opposite* or *dual* of $\boldsymbol{f}$ is the adjoint pair $\boldsymbol{f}^{\perp} : \boldsymbol{K}^{\perp} \rightleftarrows \boldsymbol{L}^{\perp}$, whose source complete lattice is the opposite of the target of $\boldsymbol{f}$, whose target complete lattice is the opposite of the source of $\boldsymbol{f}$, whose underlying adjoint pair is the opposite of the adjoint pair of $\boldsymbol{f}$.

Axiom (11) specifies the opposite operator on adjoint pairs.

```
(13) (CNG$function opposite)
     (CNG$signature opposite adjoint-pair adjoint-pair)
     (forall (?f (adjoint-pair ?f))
         (and (= (source (opposite ?f)) (CL$opposite (target ?f)))
              (= (target (opposite ?f)) (CL$opposite (source ?f)))
              (= (underlying (opposite ?f)) (ORD.ADJ$opposite (underlying ?f)))))
```

- For any class function $f : A \rightarrow B$ there is a *power adjoint pair* $\wp f : \wp A \rightleftarrows \wp B$ over f defined as follows: the source is the complete lattice $\wp(A) = \langle \wp A, \subseteq_A, \cap_A, \cup_A \rangle$, the target is the complete lattice $\wp(B) = \langle \wp B, \subseteq_B, \cap_B, \cup_B \rangle$, the left monotonic function is the direct image function $\wp f : \wp A \rightarrow \wp B$, whose right monotonic function is the inverse image function $f^{-1} : \wp B \rightarrow \wp A$, and whose fundamental property holds, since the following holds.

  $\wp f(X) \subseteq_B Y$ iff $X \subseteq_A f^{-1}(Y)$

  for all subclasses $X \subseteq A$ and $Y \subseteq B$.

```
(14) (CNG$function power)
     (CNG$signature power SET.FTN$function adjoint-pair)
      (forall (?f (SET.FTN$function ?f))
          (and (= (source (power ?f)) (LAT$power (SET.FTN$source ?f)))
               (= (target (power ?f)) (LAT$power (SET.FTN$target ?f)))
               (= (left (power ?f)) (SET.FTN$direct-image ?f))
               (= (right (power ?f)) (SET.FTN$inverse-image ?f))))
```

- Any adjoint pair $\boldsymbol{f} : \boldsymbol{L} = \langle L, \leq_L, \sqcap_L, \sqcup_L \rangle \rightleftarrows \boldsymbol{K} = \langle K, \leq_K, \sqcap_K, \sqcup_K \rangle$ between complete lattices is a concept morphism $\boldsymbol{f} : \langle \boldsymbol{K}, K, K, id_{\boldsymbol{K}}, id_{\boldsymbol{K}} \rangle \rightleftarrows \langle \boldsymbol{L}, L, L, id_{\boldsymbol{L}}, id_{\boldsymbol{L}} \rangle$ (in the reverse direction) between the concept lattices associated with the target and source complete lattices. Axiom (12) represents this as a CNG function. Note the contravariance.

```
(15) (CNG$function concept-morphism)
     (CNG$signature concept-morphism adjoint-pair CL.MOR$concept-morphism)
     (forall (?f (adjoint-pair ?f))
         (and (= (CL.MOR$source (concept-morphism ?f))
                 (LAT$concept-lattice (target ?f)))
              (= (CL.MOR$target (concept-morphism ?f))
                 (LAT$concept-lattice (source ?f)))
              (= (CL.MOR$adjoint-pair (concept-morphism ?f)) ?f)
              (= (CL.MOR$instance (concept-morphism ?f))
                 (ORD.ADJ$left (underlying ?f)))
              (= (CL.MOR$type (concept-morphism ?f))
                 (ORD.ADJ$right (underlying ?f)))))
```

- An easy check shows that the infomorphism of the concept morphism of an adjoint pair $\boldsymbol{f}$ is the same as the infomorphism associated with $\boldsymbol{f}$.

```
    (forall (?f (adjoint-pair ?f))
        (= (CL.MOR$infomorphism (concept-morphism ?f))
           (infomorphism ?f)))
```

$$\begin{tikzcd}
K \arrow[r, "\varphi"] \arrow[d, dash, "\geq_K"'] \arrow[dr, phantom, "forward"] & L \arrow[d, dash, "\geq_L"] \\
K & L \arrow[l, "\psi"]
\end{tikzcd}$$

$$\begin{tikzcd}
L \arrow[r, "\psi"] \arrow[d, dash, "\geq_L"'] \arrow[dr, phantom, "reverse"] & K \arrow[d, dash, "\geq_K"] \\
L & K \arrow[l, "\theta"]
\end{tikzcd}$$

Figure 5: Complete Homomorphism

Complete Lattice Homomorphism

`LAT.MOR`

Unfortunately, adjoint pairs are not the best morphisms for making structural comparisons between complete lattices. Another morphism between complete lattices called complete homomorphisms are best for this.

- A *complete* (*lattice*) *homomorphism* $\psi : \boldsymbol{L} \rightarrow \boldsymbol{K}$ between complete lattices L and K is a (monotonic) function that preserves both joins and meets. Being meet-preserving, ψ has a left adjoint $\varphi : K \rightarrow L$, and being join-preserving ψ has a right adjoint $\theta : K \rightarrow L$. Therefore, a complete homomorphism is the middle monotonic function in two adjunctions $\varphi \dashv \psi \dashv \theta$. Since it is more algebraic, we use the latter adjoint pair characterization in the definition of a complete lattice homomorphism.

```
(1) (CNG$conglomerate homomorphism)

(2) (CNG$function source)
    (CNG$signature source homomorphism LAT$complete-lattice)

(3) (CNG$function target)
    (CNG$signature target homomorphism LAT$complete-lattice)

(4) (CNG$function function)
    (CNG$signature function homomorphism ORD.FTN$monotonic-function)

(5) (CNG$function forward)
    (CNG$signature forward homomorphism LAT.ADJ$adjoint-pair)
    (forall (?h (homomorphism ?h))
        (and (= (LAT.ADJ$source (forward ?h)) (target ?h))
             (= (LAT.ADJ$target (forward ?h)) (source ?h))
             (= (LAT.ADJ$right (forward ?h)) (function ?h))))

(6) (CNG$function reverse)
    (CNG$signature reverse homomorphism LAT.ADJ$adjoint-pair)
    (forall (?h (homomorphism ?h))
        (and (= (LAT.ADJ$source (reverse ?h)) (source ?h))
             (= (LAT.ADJ$target (reverse ?h)) (target ?h))
             (= (LAT.ADJ$left (reverse ?h)) (function ?h))))
```

- The bond equivalent to a complete homomorphism would seem to be given by two bonds $\boldsymbol{F} : \boldsymbol{A} \multimap \boldsymbol{B}$ and $\boldsymbol{G} : \boldsymbol{B} \multimap \boldsymbol{A}$, where the right adjoint $\psi_{\boldsymbol{F}} : \boldsymbol{L}(\boldsymbol{A}) \rightarrow \boldsymbol{L}(\boldsymbol{B})$ of the complete adjoint $\boldsymbol{A}(\boldsymbol{F}) = \langle \varphi_{\boldsymbol{F}}, \psi_{\boldsymbol{F}} \rangle : \boldsymbol{L}(\boldsymbol{B}) \rightleftarrows \boldsymbol{L}(\boldsymbol{A})$ of the bond $\boldsymbol{F}$ is equal to the left adjoint $\varphi_{\boldsymbol{G}} : \boldsymbol{L}(\boldsymbol{A}) \rightarrow \boldsymbol{L}(\boldsymbol{B})$ of the complete adjoint $\boldsymbol{A}(\boldsymbol{G}) = \langle \varphi_{\boldsymbol{G}}, \psi_{\boldsymbol{G}} \rangle : \boldsymbol{L}(\boldsymbol{B}) \rightleftarrows \boldsymbol{L}(\boldsymbol{A})$ of the other bond $\boldsymbol{G}$ with the resultant adjunctions, $\varphi_{\boldsymbol{F}} \dashv \psi_{\boldsymbol{F}} = \varphi_{\boldsymbol{G}} \dashv \psi_{\boldsymbol{G}}$, where the middle adjoint is the complete homomorphism. This is indeed the case, but the question is what constraint to place on $\boldsymbol{F}$ and $\boldsymbol{G}$ in order for this to hold. The simple answer is to identify the actions of the two monotonic functions $\psi_{\boldsymbol{G}}$ and $\varphi_{\boldsymbol{F}}$. Let $(A, \Gamma) \in \boldsymbol{L}(\boldsymbol{A})$ be any formal concept in $\boldsymbol{L}(\boldsymbol{A})$. The action of the left adjoint $\varphi_{\boldsymbol{G}}$ on this concept is $(A, \Gamma) \mapsto (A^{\boldsymbol{GB}}, A^{\boldsymbol{G}})$, whereas the action of the right adjoint $\psi_{\boldsymbol{F}}$ on this concept is $(A, \Gamma) \mapsto (\Gamma^{\boldsymbol{F}}, \Gamma^{\boldsymbol{FB}})$. So the appropriate pointwise constraints are: $A^{\boldsymbol{GB}} = \Gamma^{\boldsymbol{F}}$ and $\Gamma^{\boldsymbol{FB}} = A^{\boldsymbol{G}}$, for every concept $(A, \Gamma) \in \boldsymbol{L}(\boldsymbol{A})$. The relational representation of these *pointwise constraints* is used in the definition of a bonding pair.

```
(7) (CNG$function bonding-pair)
    (CNG$signature bonding-pair homomorphism CLS.BNDPR$bonding-pair)
    (forall (?h (homomorphism ?h))
        (and (= (CLS.BND$source (bonding-pair ?h)) (LAT$classification (source ?a)))
             (= (CLS.BND$target (bonding-pair ?h)) (LAT$classification (target ?a)))
             (= (CLS.BND$forward (bonding-pair ?h))
                (LAT.ADJ$bond (forward ?h)))
             (= (CLS.BND$reverse (bonding-pair ?h))
                (LAT.ADJ$bond (reverse ?h)))))
```

- The functor composition $\boldsymbol{B}^2 \circ \boldsymbol{A}^2$ is naturally isomorphic to the identity functor $Id_{\text{Complete Lattice}}$. Consider any complete lattice $\boldsymbol{L}$. Any complete lattice $\boldsymbol{L}$ is isomorphic to the complete lattice $\boldsymbol{B}^2 \circ \boldsymbol{A}^2(\boldsymbol{L}) = \boldsymbol{L}(\langle \boldsymbol{L}, \boldsymbol{L}, \leq_{\boldsymbol{L}} \rangle)$ via the cut monotonic function $x \mapsto (\downarrow_{\boldsymbol{L}} x, \uparrow_{\boldsymbol{L}} x)$. This function is bijective, and it preserves meets and joins. The *cut* complete lattice homomorphism $\psi : \boldsymbol{L} \rightarrow A^2(B^2(\boldsymbol{L})) = B^2 \circ A^2(\boldsymbol{L})$ is a bijection from a complete lattice $\boldsymbol{L}$ to the complete lattice of its classification $A^2(B^2(\boldsymbol{L}))$. Its forward adjoint pair is the cut-forward adjoint pair and its reverse adjoint pair is the cut-reverse adjoint pair.

```
(8) (CNG$function cut)
    (CNG$signature cut LAT$complete-lattice homomorphism)
    (forall (?l (complete-lattice ?l))
        (and (= (source (cut-forward ?l)) ?l)
             (= (target (cut-forward ?l))
                (CLS.CL$complete-lattice (LAT$classification ?l)))
             (= (forward (cut ?l))
                (LAT.ADJ$cut-forward ?l))
             (= (reverse (cut ?l)) (LAT.ADJ$cut-reverse ?l))))
```

- Now consider any complete lattice homomorphism $\psi : \boldsymbol{L} \rightarrow \boldsymbol{K}$ between complete lattices $\boldsymbol{L}$ and $\boldsymbol{K}$ with associated adjunctions $\varphi \dashv \psi \dashv \theta$. The bonding pair functor maps this to the bonding pair $\boldsymbol{B}^2(\psi) = (\boldsymbol{B}(\langle\varphi, \psi\rangle), \boldsymbol{B}(\langle\psi, \theta\rangle))$, and the complete lattice functor maps this to the complete homomorphism $\boldsymbol{A}^2(\boldsymbol{B}^2(\psi)) = \tilde{\psi} : \boldsymbol{L}(\langle \boldsymbol{L}, \boldsymbol{L}, \leq_{\boldsymbol{L}}\rangle) \rightarrow \boldsymbol{L}(\langle \boldsymbol{K}, \boldsymbol{K}, \leq_{\boldsymbol{K}}\rangle)$ with associated adjunctions $\tilde{\varphi} \dashv \tilde{\psi} \dashv \tilde{\theta}$, where $\tilde{\varphi}((\downarrow_{\boldsymbol{K}} y, \uparrow_{\boldsymbol{K}} y)) = (\downarrow_{\boldsymbol{L}} \varphi(y), \uparrow_{\boldsymbol{L}} \varphi(y))$, $\tilde{\psi}((\downarrow_{\boldsymbol{L}} x, \uparrow_{\boldsymbol{L}} x)) = (\downarrow_{\boldsymbol{K}} \psi(x), \uparrow_{\boldsymbol{K}} \psi(x))$ and $\tilde{\theta}((\downarrow_{\boldsymbol{K}} y, \uparrow_{\boldsymbol{K}} y)) = (\downarrow_{\boldsymbol{L}} \theta(y), \uparrow_{\boldsymbol{L}} \theta(y))$. Clearly, the naturality condition holds between ψ and $\boldsymbol{B}^2 \circ \boldsymbol{A}^2(\psi)$.

```
(forall (?h (homomorphism ?h))
    (= (composition
           (cut (source ?h))
           (CLS.BNDPR$homomorphism (bonding-pair ?h)))
       (composition ?h (cut (target ?h)))))
```

$$\begin{tikzcd}
 & \text{Bonding Pair} \arrow[ddl, "\partial_0"'] \arrow[dr, "\partial_1"] \arrow[rrr, shift left, "A^2"] \arrow[rrr, phantom, "\equiv"] & & & \text{Complete Lattice} \arrow[lll, shift left, "B^2"] \arrow[ddl, "\partial_0"'] \arrow[dr, "\partial_1"] & \\
 & \textit{pairing constraints} & \text{Bond}^{\text{op}} \arrow[rrr, shift left, "A^{\text{op}}"] \arrow[rrr, phantom, "\equiv"] & & & \text{Complete Adjoint} \arrow[lll, shift left, "B^{\text{op}}"] \arrow[ddl, "\lambda"] \\
\text{Bond} \arrow[rrr, shift left, "A"] \arrow[rrr, phantom, "\equiv"] & & & \text{Complete Adjoint}^{\text{op}} \arrow[lll, shift left, "B"] \arrow[dr, "\rho"'] & \textit{pullback} & \\
 & & & & \text{Preorder} &
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Bond} \arrow[r, shift left, "A"] \arrow[r, phantom, "\equiv"] & \text{Complete Adjoint}^{\text{op}} \arrow[l, shift left, "B"] \\
\text{Classification}_{rel} \arrow[u, "bond"] & \\
\text{Classification} \arrow[u, "fn2rel"] \arrow[r, shift left, "L"] \arrow[r, phantom, "\equiv"] & \text{Concept Lattice} \arrow[l, shift left, "C"] \arrow[uu, "forget"']
\end{tikzcd}$$

Figure 1: Architectural Diagram of Distributed Conceptual Structures (functors and natural isomorphisms)

Table 1: Mathematical–Ontological Correspondences in the Architectural Diagram

Mathematical Notation		**Ontological Terminology**
A^2	object part:	‘`CLS.CL$complete-lattice`’
Complete Lattice Functor	morphism part:	‘`CLS.BNDPR$homomorphism`’
B^2	object part:	‘`LAT$classification`’
Bonding Pair Functor	morphism part:	‘`LAT.MOR$bonding-pair`’
$Id_{\text{Complete Lattice}} \equiv B^2 \circ A^2$ *Cut Natural Isomorphism*	component:	‘`LAT.MOR$cut`’
$A^2 \circ B^2 \equiv Id_{\text{Bonding Pair}}$ *Iota-Tau Natural Isomorphism*	component:	‘`CLS.BNDPR$iota-tau`’
A	object part:	‘`CLS.CL$complete-lattice`’
Complete Adjoint Functor	morphism part:	‘`CLS.BND$bond`’
B	object part:	‘`LAT$classification`’
Bond Functor	morphism part:	‘`LAT.ADJ$bond`’
$Id_{\text{Complete Adjoint}} \equiv B \circ A$ *Cut-Reverse Natural Isomorphism*	component:	‘`LAT.ADJ$cut-reverse`’
$A \circ B \equiv Id_{\text{Bond}}$ *Iota Natural Isomorphism*	component:	‘`CLS.BND$iota`’ (and ‘`CLS.BND$tau`’)
L	object part:	‘`CLS.CL$concept-lattice`’
Concept Lattice Functor	morphism part:	‘`CLS.INFO$concept-morphism`’
C	object part:	‘`CL$classification`’
Classification Functor	morphism part:	‘`CL.MOR$infomorphism`’
$C \circ L \equiv Id_{\text{Concept Lattice}}$ *Representation Natural Isomorphism*	component:	‘`CL.MOR$representation`’
$L \circ C = Id_{\text{Classification}}$		
equality		
∂_0	object part:	implicit identity function
0th Bond Projection Functor	morphism part:	‘`CLS.BNDPR$forward`’
∂_1	object part:	implicit identity function
1st Bond Projection Functor	morphism part:	‘`CLS.BNDPR$reverse`’
∂_0	object part:	implicit identity function
0th Adjoint Pair Projection Functor	morphism part:	‘`LAT.MOR$forward`’
∂_1	object part:	implicit identity function
1st Adjoint Pair Projection Functor	morphism part:	‘`LAT.MOR$reverse`’
λ	object part:	‘`LAT$underlying`’
Left Projection Functor	morphism part:	‘`LAT.ADJ$left`’
ρ	object part:	‘`LAT$underlying`’
Right Projection Functor	morphism part:	‘`LAT.ADJ$right`’

- The *complete lattice functor* A^2 : Bonding Pair $\rightarrow$ Complete Lattice is the operator that maps a classification $\boldsymbol{A}$ to its concept lattice $A^2(\boldsymbol{A}) \triangleq L(\boldsymbol{A})$ regarded as a complete lattice only, and maps a bonding pair $\langle \boldsymbol{F}, \boldsymbol{G} \rangle : \boldsymbol{A} \multimap \boldsymbol{B}$ to its complete lattice homomorphism $A^2(\langle \boldsymbol{F}, \boldsymbol{G} \rangle) \triangleq \psi_F = \varphi_G : L(\boldsymbol{A}) \rightarrow L(\boldsymbol{B})$.
- The *bonding pair functor* B^2 : Complete Lattice $\rightarrow$ Bonding Pair is the operator that maps a complete lattice $\boldsymbol{L}$ to its classification $\langle L, L, \leq_{\boldsymbol{L}} \rangle$ and maps a complete lattice homomorphism to its bonding pair as above. Since the bond functor $\boldsymbol{B}$ is functorial, so is B^2.

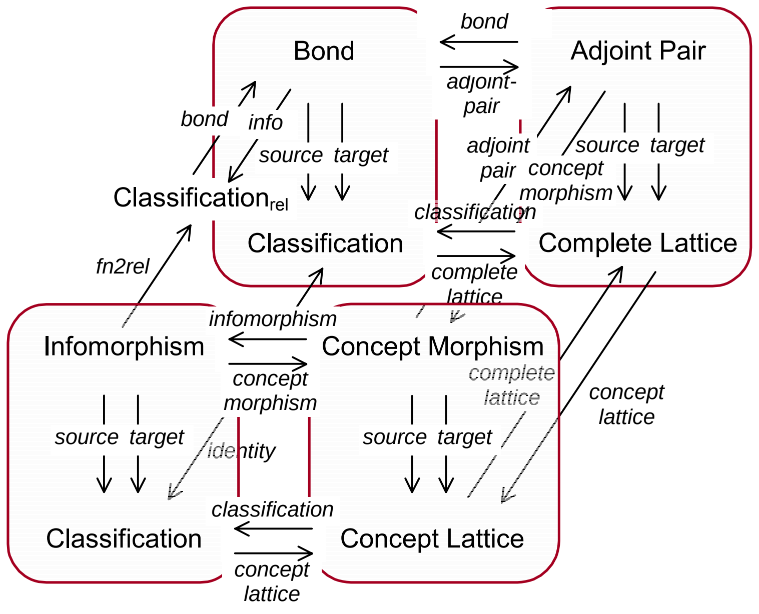


Diagram 2: Core Conglomerates and Functions in the Architectural Diagram

The Namespace of Large Classifications

This is the namespace for large classification and their morphisms: functional/relational infomorphisms, bonds and bonding pairs. In addition to strict classification terminology and axioms, this namespace will provide a bridge from classifications and their morphisms to complete/concept lattices and their morphisms. The terms introduced in this namespace are listed in Table 1.

Table 1: Terms introduced into the large classification namespace

	Conglomerate	Function	Example, Relation

`CLS`	`‘classification’`
	`‘separated’`
	`‘extensional’`
		`‘instance’`, `‘type’`, `‘incidence’`
		`‘extent’`, `‘intent’`
		`‘indistinguishable’`, `‘coextensive’`,
		`‘opposite’`, `‘instance-power’`	

`CLS .CL`		`‘left-derivation’`, `‘right-derivation’`
		`‘instance-closure’`, `‘type-closure’`
		`‘concept’`, `‘extent’`, `‘intent’`,
		`‘instance-generation’`, `‘type-generation’`
		`‘concept-order’`, `‘meet’`, `‘join’`,
		‘**`complete-lattice`**’
		`‘coreflection’`, `‘reflection’`
		`‘instance-embedding’`, `‘type-embedding’`,
		`‘instance-concept’`, `‘type-concept’`
		`‘instance-order’`, `‘type-order’`,
		‘**`concept-lattice`**’
			`‘truth-classification’`
			`‘truth-lattice’`

`CLS .FIB`		`‘instance’`, `‘instance-index’`
		`‘type’`, `‘type-index’`
		`‘left-derivation’`, `‘right-derivation’`
		`‘instance-closure’`, `‘type-closure’`
		`‘concept’`, `‘extent’`, `‘intent’`, `‘index’`
		`‘instance-generation’`, `‘type-generation’`	

`CLS .COLL`	`‘concept’`	`‘classification’`, `‘index’`
		`‘extent’`, `‘intent’`
		`‘opposite’`
		`‘2-cell’`, `‘source’`, `‘target’`
		`‘function’`, `‘terminal’`
		`‘mediator-function’`
		`‘instance-distribution’`
		`‘type-distribution’`	

`CLS .INFO`	`‘infomorphism’`	`‘source’`, `‘target’`, `‘instance’`, `‘type’`
		‘**`bond`**’
		`‘monotonic-instance’`, `‘monotonic-type’`
		`‘relational-instance’`, `‘relational-type’`
		‘**`relational-infomorphism`**’, ‘**`fn2rel`**’
		`‘opposite’`, `‘composition’`, `‘identity’`
		`‘instance-power’`, `‘eta’`
		`‘adjoint-pair’`, ‘**`concept-morphism`**’	

`CLS .REL`	`‘infomorphism’`	`‘source’`, `‘target’`, `‘instance’`, `‘type’`
		‘**`bond`**’
		`‘opposite’`, `‘composition’`, `‘identity’`	

CLS .BND	'bond'	'source', 'target', 'classification' **'bimodule'**, **'infomorphism'**, **'adjoint-pair'** 'opposite', 'composition', 'identity' **'iota'**, **'tau'**	
CLS .BNDPR	'bonding-pair'	'source', 'target', 'forward', 'reverse' 'conceptual-image' 'opposite', 'composition', 'identity' **'tau-iota'**, **'iota-tau'** **'homomorphism'**	
CLS .COL		'counique'	'initial'
CLS .COL .COPRD	'diagram' 'pair' 'cocone'	'classification1', 'classification2' 'instance-diagram', 'instance-pair' 'type-diagram', 'type-pair' 'opposite' 'cocone-diagram', 'opvertex', 'opfirst', 'opsecond' 'instance-cone', 'type-cocone' 'colimiting-cocone', 'colimit', 'binary-coproduct', 'injection1', 'injection2' 'comediator'	
CLS .COL .COINV	'coinvariant'	'classification', 'base', 'class', 'endorelation' 'coquotient', 'canon', 'comediator'	'respects'
CLS .COL .COEQ	'diagram' 'parallel-pair' 'cocone'	'source', 'target' 'infomorphism1', 'infomorphism2' 'instance-diagram', 'instance-parallel-pair' 'type-diagram', 'type-parallel-pair' 'coinvariant' 'cocone-diagram', 'opvertex', 'infomorphism' 'instance-cone', 'type-cocone' 'colimiting-cocone', 'colimit', 'coequalizer', 'canon'	
CLS .COL .PSH	'diagram' 'span' 'cocone'	'classification1', 'classification2', 'vertex', 'first', 'second' 'pair' 'opposite' 'instance-diagram', 'instance-opspan' 'type-diagram', 'type-span' 'coequalizer-diagram', 'parallel-pair' 'cocone-diagram', 'opvertex', 'opfirst', 'opsecond' 'binary-coproduct-cocone' 'coequalizer-cocone' 'colimiting-cocone', 'colimit', 'pushout', 'injection1', 'injection2', 'comediator'	

Table 2 lists the correspondence between standard mathematical notation and the ontological terminology in the namespace for classifications and functional/relation infomorphisms and bonds.

Table 2: Correspondence between Mathematical Notation and Ontological Terminology

Mathematical Notation	Ontological Terminology	Natural Language Description
CLS		
$\boldsymbol{A} = \langle \text{tok}(\boldsymbol{A}), \text{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$	`'classification'`	a *classification* – identified with a binary relation; it is determined by its three components
$\text{tok}(\boldsymbol{A})$	`'instance'`	the *instance* (token) class of a classification – identified with the source class of a binary relation
$\text{typ}(\boldsymbol{A})$	`'type'`	the *type* class of a classification – identified with the target class of a binary relation
$\models_{\boldsymbol{A}}$	`'incidence'`	the *incidence* (or *classification*) class of a classification – identified with the extent class of a binary relation
$\wp$	`'instance-power'`	the *instance power* operator on classes – this maps classes to classifications
$i_1 \sim_{\boldsymbol{A}} i_2$	`'indistinguishable'`	the information flow *indistinguishable* relation on instances
$t_1 \sim_{\boldsymbol{A}} t_2$	`'coextensive'`	the information flow *coextensive* relation on types
"separated"	`'separated'`	the *separated* subcollection of classifications
"extensional"	`'extensional'`	the *extensional* subcollection of classifications
$(\text{-})^{\propto}$ or $(\text{-})^{\perp}$ or $(\text{-})^{\text{op}}$	`'opposite'`	the *involution* (or *transpose* or *opposite* or *dual*) operator on classifications
CLS.CL		
$(\text{-})'$ or $(\text{-})^{A}$	`'left-derivation'`, `'right-derivation'`	*left/right derivation* operations for a classification
$(\text{-})''$ or $(\text{-})^{AA}$	`'instance-closure'` `'type-closure'`	*instance/type closure* operations for a classification
$\boldsymbol{c} = (X,Y)$	`'concept'`	a *formal concept* for a classification
$\boldsymbol{c}_1 \leq_{\boldsymbol{A}} \boldsymbol{c}_2$	`'concept-order'`	the concept order of a classification – this is the partial order underlying the concept lattice of a classification
$\underline{\mathfrak{B}}\boldsymbol{A}$	`'concept-lattice'`	the *concept lattice* of a classification – the German word for "formal concepts" is *die begriffe*
$\gamma_{\boldsymbol{A}} : \text{tok}(\boldsymbol{B}) \rightarrow \underline{\mathfrak{B}}\boldsymbol{A}$	`'instance-embedding'`	the *instance embedding function* – maps an instance to the concept that it generates
$\mu_{\boldsymbol{A}} : \text{typ}(\boldsymbol{B}) \rightarrow \underline{\mathfrak{B}}\boldsymbol{A}$	`'type-embedding'`	the *type embedding function* – maps a type to the concept that it generates
CLS.INFO		
$f = \langle f\hat{}, f\check{} \rangle : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$	`'infomorphism'`	an *infomorphism* from classification $\boldsymbol{A}$ to classification $\boldsymbol{B}$ – determined by its instance and type functions
$f\hat{} : \text{typ}(\boldsymbol{A}) \rightarrow \text{typ}(\boldsymbol{B})$	`'type'`	the *type* function of an infomorphism
$f\check{} : \text{tok}(\boldsymbol{B}) \rightarrow \text{tok}(\boldsymbol{A})$	`'instance'`	the *instance* (or token) function of an infomorphism
$gf : \boldsymbol{A} \rightleftarrows \boldsymbol{C}$	`'composition'`	the *composition* of two composable infomorphisms $f : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ and $g : \boldsymbol{B} \rightleftarrows \boldsymbol{C}$
$1_{\boldsymbol{A}} : \boldsymbol{A} \rightleftarrows \boldsymbol{A}$	`'identity'`	the *identity* infomorphism on classification $\boldsymbol{A}$
$(\text{-})^{\propto}$ or $(\text{-})^{\perp}$ or $(\text{-})^{\text{op}}$	`'opposite'`	the *involution* (or *transpose* or *opposite* or *dual*) operator on infomorphisms
$\wp$	`'instance-power'`	the *instance power* operator on functions – this maps functions to infomorphisms

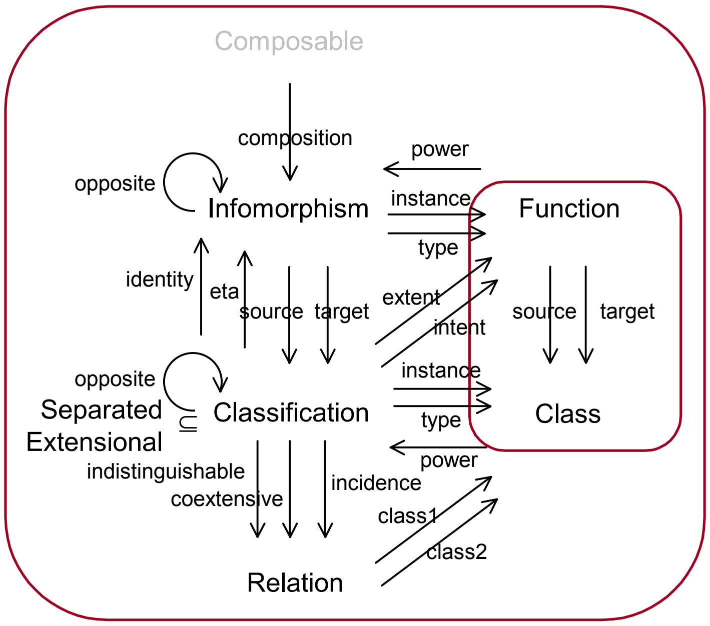


Diagram 3: Core Conglomerates and Functions for Classifications and Infomorphisms

Classifications

CLS

o A (large) *classification* $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$ is identical to a (large) binary relation. It consists of a class of instances $\mathit{inst}(\boldsymbol{A})$ identified with the first or source class of a binary relation, a class of types $\mathit{typ}(\boldsymbol{A})$ identified with the second or target class of a binary relation, and a class of incidence or classification $\models_{\boldsymbol{A}}$ identified with the extent class of a binary relation. We visualize a binary relation as in Figure 1.

$$\begin{tikzcd}
\mathit{typ}(\boldsymbol{A}) \\
\mathit{inst}(\boldsymbol{A}) \arrow[u, "\boldsymbol{A}"']
\end{tikzcd}$$

Figure 1: Classification

The following is a KIF representation for the elements of a classification. Classifications are specified by declaration and population. The elements in the KIF representation are useful for the declaration and population of a classification. The term ‘`classification`’ specified in axiom (1) allows one to *declare* classifications. The terms ‘`instance`’ and ‘`type`’ specified in axioms (2–3) and the term ‘`incidence`’ specified in axiom (4) resolve classifications into their parts, thus allowing one to *populate* classifications.

```
(1) (CNG$conglomerate classification)
    (= classification REL$relation)

(2) (CNG$function instance)
    (CNG$signature instance classification CLS$class)
    (= instance REL$class1)

(3) (CNG$function type)
    (CNG$signature type classification CLS$class)
    (= type REL$class2)

(4) (CNG$function incidence)
    (CNG$signature incidence classification CLS$class)
    (= incidence REL$extent)
```

o Associated with any classification is a function that produces the *extent* of a type and a function that produces the *intent* of an instance, both within the context of the classification. The extent of a type $t \in \mathit{typ}(\boldsymbol{A})$ in a classification $\boldsymbol{A}$ defined by

$extent_A(t) = \{i \in inst(A) \mid i \vDash_A t\}$.

- Dually, the intent of an instance $i \in inst(A)$ in a classification $A = \langle inst(A), typ(A), \vDash_A\rangle$ is defined by

  $intent_A(i) = \{t \in typ(A) \mid i \vDash_A t\}$.

  Axiom (5) specifies the extent function and axiom (6) specifies the intent function. The last axiom in (5) demonstrates that the relative instantiation-predication represented by the incidence relation is compatible with, generalizes and relativizes the absolute KIF instantiation-predication – an instance is a member of the extent of a type iff the instance is classified by the type.

```
(5) (CNG$function extent)
    (CNG$signature extent classification SET.FTN$function)
    (forall (?a (classification ?a))
        (= (SET.FTN$source (extent ?a)) (type ?a))
        (= (SET.FTN$target (extent ?a)) (SET$power (instance ?a)))
        (forall (?i ?t ((instance ?a) ?i) ((type ?a) ?t))
            (<=> (((extent ?a) ?t) ?i)
                 ((incidence ?a) [?i ?t]))))

(6) (CNG$function intent)
     (CNG$signature intent classification SET.FTN$function)
     (forall (?a (classification ?a))
         (= (SET.FTN$source (intent ?a)) (instance ?a))
         (= (SET.FTN$target (intent ?a)) (SET$power (type ?a)))
         (forall (?i ?t ((instance ?a) ?i) ((type ?a) ?t))
             (<=> (((intent ?a) ?i) ?t)
                  ((REL$extent (incidence ?a)) [?i ?t]))))
```

- For any classification $A = \langle inst(A), typ(A), \vDash_A\rangle$, two instances $i_1, i_2 \in inst(A)$ are *indistinguishable* in A (Barwise and Seligman, 1997), written $i_1 \sim_A i_2$, when $intent_A(i_1) = intent_A(i_2)$. Two types $t_1, t_2 \in typ(A)$ are *coextensive* in A, written $t_1 \sim_A t_2$, when $extent_A(t_1) = extent_A(t_2)$. A classification A is *separated* when there are no distinct indistinguishable instances, and *extensional* when there are no distinct coextensive types.

  The terms '`(indistinguishable ?a)`' and '`(coextensive ?a)`' that are specified in axioms (7–8) represent the Information Flow notions of instance *indistinguishability* and type *coextension*, respectively. The terms '`separated`' and '`extensional`' that are specified in axioms (9–10) represent the Information Flow notions of classification *separateness* and *extensionality*, respectively.

```
(7) (CNG$function indistinguishable)
    (CNG$signature indistinguishable classification REL.ENDO$relation)
    (forall (?a (classification ?a))
        (and (= (REL$class (indistinguishable ?a)) (instance ?a))
             (forall (?i1 ((instance ?a) ?i1)
                      ?i2 ((instance ?a) ?i2))
                 (<=> ((REL$extent (indistinguishable ?a)) [?i1 ?i2])
                      (= ((intent ?a) ?i1) ((intent ?a) ?i2))))))

(8) (CNG$function coextensive)
    (CNG$signature coextensive classification REL.ENDO$relation)
    (forall (?a (classification ?a))
        (and (= (REL$class (coextensive ?a)) (type ?a))
             (forall (?t1 ((type ?a) ?t1)
                      ?t2 ((type ?a) ?t2))
                 (<=> ((REL$extent (coextensive ?a)) [?t1 ?t2])
                      (= ((extent ?a) ?t1) ((extent ?a) ?t2))))))

(9) (CNG$conglomeration separated)
    (CNG$subconglomeration separated classification)
    (forall (?a (classification ?a))
        (<=> (separated ?a)
             (REL.ENDO$subendorelation
                 (indistinguishable ?a)
                 (REL.ENDO$identity (instance ?a)))))

(10) (CNG$conglomeration extensional)
     (CNG$subconglomeration extensional classification)
```

```
        (forall (?a (classification ?a))
            (<=> (extensional ?a)
                (REL.ENDO$subendorelation
                    (coextensive ?a)
                    (REL.ENDO$identity (type ?a)))))
```

- To quote (Barwise and Seligman, 1997), "in any classification, we think of the types as classifying the instances, but it is often useful to think of the instances as classifying the types." For any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$, the *opposite* or *dual* of $\boldsymbol{A}$ is the opposite binary relation; that is, the classification $\boldsymbol{A}^{\perp} = \langle \mathit{typ}(\boldsymbol{A}), \mathit{inst}(\boldsymbol{A}), {\models_{\boldsymbol{A}}}^{\perp} \rangle$, whose instances are types of $\boldsymbol{A}$, and whose types are instances of $\boldsymbol{A}$, and whose incidence is: $t \models^{\perp} i$ when $i \models t$. Axiom (11) specifies the opposite operator on classifications.

```
(11) (CNG$function opposite)
     (CNG$signature opposite classification classification)
     (= opposite REL$opposite)
```

- For any class A the *instance power classification* $\wp A = \langle A, \wp A, \in_A \rangle$ over A are defined as follows: the instance class is A; the type class is the power class $\wp A$ (so that a type is a subclass of A), and the incidence relation is the membership relation $\in_A$. Axiom (12) specifies the power operator from classes to classifications.

```
(12) (CNG$function instance-power)
     (CNG$signature instance-power SET$class classification)
     (forall (?a (SET$class ?a))
         (and (= (instance (instance-power ?a)) ?a)
              (= (type (instance-power ?a)) (SET$power ?a))
              (forall (?x ?y (?a ?x) ((SET$power ?a) ?y))
                  (<=> ((incidence (instance-power ?a)) [?x ?y])
                       (?y ?x)))))
```

Concept Lattices

CLS.CL

$$\begin{tikzcd}
 & \mathit{concept}(\boldsymbol{A}) \arrow[dl, shift left, "\mathit{extent}_{A}"] \arrow[dr, shift left, "\mathit{intent}_{A}"] & \\
\wp(\mathit{inst}(\boldsymbol{A})) \arrow[ur, shift left, "\mathit{inst\text{-}gen}_{A}"] \arrow[rr, shift left, "\mathit{left\text{-}deriv}_{A}"] \arrow[loop left, "\mathit{inst\text{-}clsr}_{A}"] & & \wp(\mathit{typ}(\boldsymbol{A})) \arrow[ul, shift left, "\mathit{typ\text{-}gen}_{A}"] \arrow[ll, shift left, "\mathit{right\text{-}deriv}_{A}"] \arrow[loop right, "\mathit{typ\text{-}clsr}_{A}"]
\end{tikzcd}$$

Diagram 4: Core classes/functions for concept lattices

○ For any large classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$ there are two senses of *derivation* corresponding to the two senses of relational residuation. Left derivation maps a subset of instances to the types on which they are all incident, and dually right derivation maps a subset of types to the instances, which are incident on all of them. For all $X \subseteq \mathit{inst}(\boldsymbol{A})$ and $Y \subseteq \mathit{typ}(\boldsymbol{A})$

$$X \mapsto X' = X\backslash\boldsymbol{A} = \{t \in \mathit{typ}(\boldsymbol{A}) \mid i \models t \text{ for all } i \in X\}$$

$$Y \mapsto Y' = \boldsymbol{A}/Y = \{i \in \mathit{inst}(\boldsymbol{A}) \mid i \models t \text{ for all } t \in Y\}.$$

```
(1) (CNG$function left-derivation)
    (CNG$signature left-derivation CLS$classification SET.FTN$function)
    (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (left-derivation ?a))
                (SET$power (CLS$instance ?a)))
             (= (SET.FTN$target (left-derivation ?a))
                (SET$power (CLS$type ?a)))
             (forall (?x (SET$subclass ?x (CLS$instance ?a)))
                      ?t ((CLS$type ?a) ?t))
                 (<=> (((left-derivation ?a) ?x) ?t)
                      (forall (?i ((CLS$instance ?a) ?i))
                          (=> (?x ?i) ((CLS$incidence ?a) [?i ?t])))))))

(2) (CNG$function right-derivation)
    (CNG$signature right-derivation CLS$classification SET.FTN$function)
    (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (right-derivation ?a))
                (SET$power (CLS$type ?a)))
             (= (SET.FTN$target (right-derivation ?a))
                (SET$power (CLS$instance ?a)))
             (forall (?y (SET$subclass ?y (CLS$type ?a)))
                      ?i ((CLS$instance ?a) ?i))
                 (<=> (((right-derivation ?a) ?y) ?i)
                      (forall (?t ((CLS$type ?a) ?t))
                          (=> (?y ?t) ((CLS$incidence ?a) [?i ?t])))))))
```

○ A simple fundamental result is the following equivalence. For all $X \subseteq \mathit{inst}(\boldsymbol{A})$ and $Y \subseteq \mathit{typ}(\boldsymbol{A})$

$$Y \subseteq X\backslash\boldsymbol{A} \text{ in } \wp(\mathit{typ}(\boldsymbol{A}))^{\mathrm{op}} \ \underline{\text{iff}}\ X{\times}Y \subseteq \models_{\boldsymbol{A}} \ \underline{\text{iff}}\ X \subseteq \boldsymbol{A}/Y \text{ in } \wp(\mathit{inst}(\boldsymbol{A})).$$

This describes a Galois connection (preorder adjunction) between the left derivation

$$(\text{-})\backslash\boldsymbol{A} : \wp(\mathit{inst}(\boldsymbol{A})) \rightarrow \wp(\mathit{typ}(\boldsymbol{A}))^{\mathrm{op}}$$

and the right derivation

$$\boldsymbol{A}/(\text{-}) : \wp(\mathit{typ}(\boldsymbol{A}))^{\mathrm{op}} \rightarrow \wp(\mathit{inst}(\boldsymbol{A})).$$

The first two facts assert (contravariant) monotonicity of derivation. The last fact asserts the adjointness condition.

```
    (forall (?x1 (SET$subclass ?x1 (CLS$instance ?a)
             ?x2 (SET$subclass ?x2 (CLS$instance ?a))
        (=> (SET$subclass ?x1 ?x2)
            (SET$subclass ((left-derivation ?a) ?x2) ((left-derivation ?a) ?x1))))

    (forall (?y1 (SET$subclass ?y1 (CLS$type ?a)
             ?y2 (SET$subclass ?y2 (CLS$type ?a))
        (=> (SET$subclass ?y2 ?y1)
            (SET$subclass ((right-derivation ?a) ?y1) ((right-derivation ?a) ?y2))))
    (forall (?x (SET$subclass ?x (CLS$instance ?a))
             ?y (SET$subclass ?y (CLS$type ?a)))
        (<=> (SET$subclass ?x ((right-derivation ?a) ?y))
             (SET$subclass ?y ((left-derivation ?a) ?x))))
```

- The composition of derivations gives two senses of closure operator.

```
(3) (CNG$function instance-closure)
    (CNG$signature instance-closure CLS$classification SET.FTN$function)
    (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (instance-closure ?a))
                (SET$power (CLS$instance ?a)))
             (= (SET.FTN$target (instance-closure ?a))
                (SET$power (CLS$instance ?a)))
             (= (instance-closure ?a)
                (SET.FTN$composition (left-derivation ?a) (right-derivation ?a)))))

(4) (CNG$function type-closure)
    (CNG$signature type-closure CLS$classification SET.FTN$function)
    (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (type-closure ?a))
                (SET$power (CLS$type ?a)))
             (= (SET.FTN$target (type-closure ?a))
                (SET$power (CLS$type ?a)))
             (= (type-closure ?a)
                (SET.FTN$composition (right-derivation ?a) (left-derivation ?a)))))
```

- The following (easily proven) results confirm that these are closure operators.

  $X \subseteq X''$ and $X' = X'''$ for all $X \subseteq \mathit{inst}(\boldsymbol{A})$.

  $Y \subseteq Y''$ and $Y' = Y'''$ for all $Y \subseteq \mathit{typ}(\boldsymbol{A})$.

```
    (forall (?a (CLS$classification ?a))
        (and (= (left-derivation ?a)
                (SET.FTN$composition (instance-closure ?a) (left-derivation ?a)))
             (forall (?x (SET$subclass ?x (CLS$instance ?a))
                 (SET$subclass ?x ((instance-closure ?a) ?x)))))

    (forall (?a (CLS$classification ?a))
        (and (= (right-derivation ?a)
                (SET.FTN$composition (type-closure ?a) (right-derivation ?a)))
             (forall (?y (SET$subclass ?y (CLS$type ?a))
                 (SET$subclass ?y ((type-closure ?a) ?y)))))
```

- Further properties of Galois connection relate to continuity – the closure of a union of a family of classes is the intersection of the closures.

  $(\cup_{j\in J} X_j)' = \cap_{j\in J} X_j'$ for any family of subsets $X_j \subseteq \mathit{inst}(\boldsymbol{A})$ for $j\in J$.

  $(\cup_{k\in K} Y_k)' = \cap_{k\in K} Y_k'$ for any family of subsets $Y_k \subseteq \mathit{typ}(\boldsymbol{A})$ for $k\in K$.

- By embedding the subsets of instances and types above as relations, we can connect derivation to residuation. We can prove the following simple identities: the embedding of the left-derivation of a subset of instances is the left-residuation of the classification along the opposite of the embedding of the subset, and dually for the right notions.

```
    (forall (?a (CLS$classification ?a))
        (and (forall (?x (SET$subclass ?x (CLS$instance ?a)))
                 (= ((REL$embed (CLS$type ?a)) ((left-derivation ?a) ?x))
                    (REL$left-residuation
                        (REL$opposite ((REL$embed (CLS$instance ?a)) ?x))
                        ?a))
             (forall (?y (SET$subclass ?y (CLS$type ?a)))
                 (= (REL$opposite
                        ((REL$embed (CLS$instance ?a)) ((right-derivation ?a) ?y)))
                    (REL$right-residuation
                        ((REL$embed (CLS$type ?a)) ?y)
                        ?a)))))
```

- For a classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \vDash_{\boldsymbol{A}} \rangle$, the closed elements of the derivation Galois connection are called formal concepts. There are several ways to define this notion, but traditionally it has been given the following (slightly redundant) definition. A (large) *formal concept* $c = \langle \mathit{extent}_{\boldsymbol{A}}(c), \mathit{intent}_{\boldsymbol{A}}(c) \rangle$ is a pair of classes, $\mathit{extent}_{\boldsymbol{A}}(c) \subseteq \mathit{inst}(\boldsymbol{A})$ and $\mathit{intent}_{\boldsymbol{A}}(c) \subseteq \mathit{typ}(\boldsymbol{A})$, that satisfy the equivalent conditions that $\mathit{extent}_{\boldsymbol{A}}(c) = \mathit{intent}_{\boldsymbol{A}}(c)'$ and $\mathit{intent}_{\boldsymbol{A}}(c) = \mathit{extent}_{\boldsymbol{A}}(c)'$. Let $\mathit{concept}(\boldsymbol{A})$ denote the class of all formal concepts of a classification $\boldsymbol{A}$. There are functions, $\mathit{extent}_{\boldsymbol{A}} : \mathit{concept}(\boldsymbol{A}) \rightarrow \wp(\mathit{inst}(\boldsymbol{A}))$ and $\mathit{intent}_{\boldsymbol{A}} : \mathit{concept}(\boldsymbol{A}) \rightarrow \wp(\mathit{typ}(\boldsymbol{A}))$, that map concepts to their extent and intent.

```
(5) (CNG$function concept)
    (CNG$signature concept CLS$classification SET$class)

(6) (CNG$function extent)
    (CNG$signature extent CLS$classification SET.FTN$function)
    (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (extent ?a)) (concept ?a))
             (= (SET.FTN$target (extent ?a)) (SET$power (CLS$instance ?a)))))

(7) (CNG$function intent)
    (CNG$signature intent CLS$classification SET.FTN$function)
    (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (intent ?a)) (concept ?a))
             (= (SET.FTN$target (intent ?a)) (SET$power (CLS$type ?a)))))

    (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$composition (intent ?a) (right-derivation ?a))
                (extent ?a))
             (= (SET.FTN$composition (extent ?a) (left-derivation ?a))
                (intent ?a))))
```

- There are surjective generator functions, called instance-generation and type-generation, that map subsets of instances and types to their generated concepts. For all $X \subseteq \mathit{inst}(\boldsymbol{A})$ and $Y \subseteq \mathit{typ}(\boldsymbol{A})$

  $X \mapsto \langle X'', X' \rangle$ “instance-generation”

  $Y \mapsto \langle Y', Y'' \rangle$ “type-generation”.

  See diagram 4 for a visualization of the following conditions.
  - The composition of instance-generation and intent equals left-derivation, and the composition of type-generation and extent equals right-derivation.
  - The composition of instance-generation and extent equals instance-closure, and the composition of type-generation and intent equals type-closure.
  - The composition of extent and instance-generation is identity, and the composition of intent and type-generation is identity.

  These conditions define generation, and also insure that all concepts are captured in the class ‘`(concept ?a)`’. Concepts are determined by either their extents or their intents – this is represented by the fact that the extent and intent functions are injective.

```
(8) (CNG$function instance-generation)
    (CNG$signature instance-generation CLS$classification SET.FTN$function)
    (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (instance-generation ?a))
                (SET$power (CLS$instance ?a)))
             (= (SET.FTN$target (instance-generation ?a))
                (concept ?a))
             (= (SET.FTN$composition (instance-generation ?a) (intent ?a))
                (left-derivation ?a))
             (= (SET.FTN$composition (instance-generation ?a) (extent ?a))
                (instance-closure ?a))
             (= (SET.FTN$composition (extent ?a) (instance-generation ?a))
                (SET.FTN$identity (concept ?a)))))

(9) (CNG$function type-generation)
    (CNG$signature type-generation CLS$classification SET.FTN$function)
    (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (type-generation ?a))
                (SET$power (CLS$type ?a)))
             (= (SET.FTN$target (type-generation ?a))
                (concept ?a))
             (= (SET.FTN$composition (type-generation ?a) (extent ?a))
                (right-derivation ?a))
             (= (SET.FTN$composition (type-generation ?a) (intent ?a))
                (type-closure ?a))
             (= (SET.FTN$composition (intent ?a) (type-generation ?a))
                (SET.FTN$identity (concept ?a)))))
```

○ A concept $c_1 = \langle \mathit{extent}_A(c_1), \mathit{intent}_A(c_1)\rangle$ is a *subconcept* of a concept $c_2 = \langle \mathit{extent}_A(c_2), \mathit{intent}_A(c_2)\rangle$, denoted $c_1 \leq_A c_2$, when $\mathit{extent}_A(c_1) \subseteq \mathit{extent}_A(c_2)$ or equivalently when $\mathit{intent}_A(c_1) \supseteq \mathit{intent}_A(c_2)$. Other language is that c_1 is "more specific" than c_2, and c_2 is "more generic" than c_1. For any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \vDash_{\boldsymbol{A}}\rangle$, the class of concepts together with the subconcept relation form a partial order $\mathit{order}(\boldsymbol{A}) = \langle \mathit{concept}(\boldsymbol{A}), \leq_{\boldsymbol{A}}\rangle$.

```
(10) (CNG$function concept-order)
     (CNG$signature concept-order CLS$classification ORD$partial-order)
     (forall (?a (CLS$classification ?a))
         (and (= (ORD$class (concept-order ?a)) (concept ?a))
              (forall (?c1 ((concept ?a) ?c1)
                       ?c2 ((concept ?a) ?c2))
                  (<=> ((ORD$extent (concept-order ?a)) [?c1 ?c2])
                       (SET$subclass ((extent ?a) ?c1) ((extent ?a) ?c2))))))
```

○ There are both a meet and a join operations defined on subclasses of concepts

$$\mathit{meet}_A : \wp(\mathit{concept}(\boldsymbol{A})) \rightarrow \mathit{concept}(\boldsymbol{A})$$

$$\mathit{join}_A : \wp(\mathit{concept}(\boldsymbol{A})) \rightarrow \mathit{concept}(\boldsymbol{A})$$

defined as follows:

$$\sqcap_L(C) = \sqcap_{c\in C} \langle \mathit{extent}_A(c), \mathit{intent}_A(c)\rangle = \langle \cap_{c\in C}\, \mathit{extent}_A(c), (\cup_{c\in C}\, \mathit{intent}_A(c))''\rangle$$

$$\sqcup_L(C) = \sqcup_{c\in C} \langle \mathit{extent}_A(c), \mathit{intent}_A(c)\rangle = \langle (\cup_{c\in C}\, \mathit{extent}_A(c))'', \cap_{c\in C}\, \mathit{intent}_A(c)\rangle$$

for any subclass $C \subseteq \mathit{concept}(\boldsymbol{A})$.

```
(11) (CNG$function meet)
     (CNG$signature meet CLS$classification SET.FTN$function)
     (forall (?a (CLS$classification ?a))
         (and (= (SET.FTN$source (meet ?a)) (SET$power (concept ?a)))
              (= (SET.FTN$target (meet ?a)) (concept ?a))
              (forall (?c (SET$subclass ?c (concept ?a)))
                  (and (= (SET.FTN$composition (meet ?a) (extent ?a))
                          (SET.FTN$composition
                              (SET.FTN$power (extent ?a))
                              (SET$intersection (instance ?a))))
                       (= (SET.FTN$composition (meet ?a) (intent ?a))
                          (SET.FTN$composition
                              (SET.FTN$composition
                                  (SET.FTN$power (intent ?a))
                                  (SET$union (type ?a)))
                              (type-closure ?a)))))))

(12) (CNG$function join)
     (CNG$signature join CLS$classification SET.FTN$function)
     (forall (?a (CLS$classification ?a))
         (and (= (SET.FTN$source (join ?a)) (SET$power (concept ?a)))
              (= (SET.FTN$target (join ?a)) (concept ?a))
              (forall (?c (SET$subclass ?c (concept ?a)))
                  (and (= (SET.FTN$composition (join ?a) (extent ?a))
                          (SET.FTN$composition
                              (SET.FTN$composition
                                  (SET.FTN$power (extent ?a))
                                  (SET$union (instance ?a)))
                              (instance-closure ?a)))
                       (= (SET.FTN$composition (join ?a) (intent ?a))
                          (SET.FTN$composition
                              (SET.FTN$power (intent ?a))
                              (SET$intersection (type ?a))))))))
```

- For any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$, the partial order $\mathit{order}(\boldsymbol{A}) = \langle \mathit{concept}(\boldsymbol{A}), \leq_{\boldsymbol{A}} \rangle$ of concepts together with the conceptual join and meet operators form a complete lattice $\mathit{latt}(\boldsymbol{A}) = \langle \mathit{order}(\boldsymbol{A}), \mathit{join}(\boldsymbol{A}), \mathit{meet}(\boldsymbol{A}) \rangle$.

```
(13) (CNG$function complete-lattice)
     (CNG$signature complete-lattice CLS$classification LAT$complete-lattice)
     (forall (?a (CLS$classification ?a))
         (and (= (LAT$underlying (complete-lattice ?a)) (concept-order ?a))
              (= (LAT$join (complete-lattice ?a)) (join ?a))
              (= (LAT$meet (complete-lattice ?a)) (meet ?a))))
```

- For any two classifications $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}, \models_{\boldsymbol{A}} \rangle$ and $\boldsymbol{C} = \langle \mathit{inst}(\boldsymbol{C}), \mathit{typ}, \models_{\boldsymbol{C}} \rangle$ that have the same class of types and where the classification (binary relation) $\boldsymbol{C}$ is *type-closed* $(\boldsymbol{A}/\boldsymbol{C})\backslash\boldsymbol{A} = \boldsymbol{C}$ with respect to $\boldsymbol{A}$, there is an associated *coreflection* (adjoint pair) $\mathit{int}_{\boldsymbol{C},\boldsymbol{A}} = \langle \partial_0, \tilde{\partial}_0 \rangle : \mathit{latt}(\boldsymbol{C}) \rightleftarrows \mathit{latt}(\boldsymbol{A})$ between their complete lattices, where $\tilde{\partial}_0 : \mathit{latt}(\boldsymbol{A}) \rightarrow \mathit{latt}(\boldsymbol{C})$ is right adjoint right inverse (rari) to $\partial_0 : \mathit{latt}(\boldsymbol{C}) \rightarrow \mathit{latt}(\boldsymbol{A})$. Hence, ∂_0 embeds $\mathit{latt}(\boldsymbol{C})$ as an internal part of $\mathit{latt}(\boldsymbol{A})$. These functions are defined as follows.

  $\partial_0(\langle X, Y \rangle) = (\langle Y', Y \rangle)$ for any concept $\langle X, Y \rangle \in \mathit{latt}(\boldsymbol{C})$
  $\tilde{\partial}_0(\langle X, Y \rangle) = (\langle Y', Y'' \rangle)$ for any concept $\langle X, Y \rangle \in \mathit{latt}(\boldsymbol{A})$

$$\begin{tikzcd}
\mathit{latt}(\boldsymbol{C}) \arrow[r, hook, "\partial_0"] \arrow[d, dash, "\geq_{\boldsymbol{C}}"'] \arrow[dr, dash] & \mathit{latt}(\boldsymbol{A}) \arrow[d, dash, "\geq_{\boldsymbol{B}}"] \\
\mathit{latt}(\boldsymbol{C}) & \mathit{latt}(\boldsymbol{A}) \arrow[l, tail, "\tilde{\partial}_0"]
\end{tikzcd}$$

Figure 2: Type-closed adjoint pair

- Dually, for any two classifications $\boldsymbol{C} = \langle \mathit{inst}, \mathit{typ}(\boldsymbol{C}), \models_{\boldsymbol{C}} \rangle$ and $\boldsymbol{B} = \langle \mathit{inst}, \mathit{typ}(\boldsymbol{B}), \models_{\boldsymbol{B}} \rangle$ that have the same class of instances and where the classification (binary relation) $\boldsymbol{C}$ is *instance-closed* $\boldsymbol{B}/(\boldsymbol{C}\backslash\boldsymbol{B}) = \boldsymbol{C}$ with respect to $\boldsymbol{B}$, there is an associated *reflection* (adjoint pair) $\mathit{ext}_{\boldsymbol{B},\boldsymbol{C}} = \langle \tilde{\partial}_1, \partial_1 \rangle : \mathit{latt}(\boldsymbol{B}) \rightleftarrows \mathit{latt}(\boldsymbol{C})$ between their complete lattices, where $\tilde{\partial}_1 : \mathit{latt}(\boldsymbol{B}) \rightarrow \mathit{latt}(\boldsymbol{C})$ is left adjoint right inverse (lari) to $\partial_1 : \mathit{latt}(\boldsymbol{C}) \rightarrow \mathit{latt}(\boldsymbol{B})$. Hence, ∂_1 embeds $\mathit{latt}(\boldsymbol{C})$ as an external part of $\mathit{latt}(\boldsymbol{A})$. These functions are defined as follows.

  $\partial_1(\langle X, Y \rangle) = (\langle X, X' \rangle)$ for any concept $\langle X, Y \rangle \in \mathit{latt}(\boldsymbol{C})$
  $\tilde{\partial}_1(\langle X, Y \rangle) = (\langle X'', X' \rangle)$ for any concept $\langle X, Y \rangle \in \mathit{latt}(\boldsymbol{B})$.

$$\begin{tikzcd}
\mathit{latt}(\boldsymbol{B}) \arrow[r, tail, "\tilde{\partial}_1"] \arrow[d, dash, "\geq_{\boldsymbol{A}}"'] \arrow[dr, dash] & \mathit{latt}(\boldsymbol{C}) \arrow[d, dash, "\geq_{\boldsymbol{B}}"] \\
\mathit{latt}(\boldsymbol{B}) & \mathit{latt}(\boldsymbol{C}) \arrow[l, hook', "\partial_1"]
\end{tikzcd}$$

Figure 3: Instance-closed adjoint pair

```
(14) (CNG$function coreflection)
     (CNG$signature coreflection
         CLS$classification CLS$classification LAT.ADJ$adjoint-pair)
     (forall (?c (CLS$classification ?c)
              ?a (CLS$classification ?a))
         (<=> (exists (?f (LAT.ADJ$adjoint-pair ?f))
                  (= (coreflection ?c ?a) ?f))
              (and (= (CLS$type ?c) (CLS$type ?a))
                   (= (REL$left-residuation (REL$right-residuation ?c ?a) ?a) ?c))))
     (forall (?c (CLS$classification ?c)
              ?a (CLS$classification ?a))
         (=> (and (= (CLS$type ?c) (CLS$type ?a))
                  (= (REL$left-residuation (REL$right-residuation ?c ?a) ?a) ?c))
             (and (= (LAT.ADJ$source (coreflection ?c ?a))
                     (complete-lattice c?))
                  (= (LAT.ADJ$target (coreflection ?c ?a))
                     (complete-lattice ?a))
                  (= (REL.FTN$composition (LAT.ADJ$left (coreflection ?c ?a)) (extent ?a))
                     (REL.FTN$composition (intent ?c) (right-derivation ?c)))
                  (= (REL.FTN$composition (LAT.ADJ$left (coreflection ?c ?a)) (intent ?a))
                     (intent ?c))
                  (= (REL.FTN$composition (LAT.ADJ$right (coreflection ?c ?a)) (extent ?c))
                     (REL.FTN$composition (intent ?a) (right-derivation ?a)))
                  (= (REL.FTN$composition (LAT.ADJ$right (coreflection ?c ?a)) (intent ?c))
                     (REL.FTN$composition (intent ?a) (type-closure ?a))))))
```

```
(15) (CNG$function reflection)
     (CNG$signature reflection
        CLS$classification CLS$classification LAT.ADJ$adjoint-pair)
     (forall (?b (CLS$classification ?b)
              ?c (CLS$classification ?c))
        (<=> (exists (?f (LAT.ADJ$adjoint-pair ?f))
                 (= (reflection ?b ?c) ?f))
             (and (= (CLS$instance ?b) (CLS$instance ?c))
                  (= (REL$right-residuation (REL$left-residuation ?c ?b) ?b) ?c))))
     (forall (?b (CLS$classification ?b)
              ?c (CLS$classification ?c))
        (=> (and (CLS$instance ?b) (= (CLS$instance ?c))
                 (= (REL$right-residuation (REL$left-residuation ?c ?b) ?b) ?c))
            (and (= (LAT.ADJ$source (reflection ?b ?c))
                    (complete-lattice b?))
                 (= (LAT.ADJ$target (reflection ?b ?c))
                    (complete-lattice c?))
                 (= (REL.FTN$composition (LAT.ADJ$right (reflection ?b ?c)) (extent ?b))
                    (extent ?c))
                 (= (REL.FTN$composition (LAT.ADJ$right (reflection ?b ?c)) (intent ?b))
                    (REL.FTN$composition (extent ?c) (left-derivation c?)))
                 (= (REL.FTN$composition (LAT.ADJ$left (reflection ?b ?c)) (extent ?c))
                    (REL.FTN$composition (extent ?b) (instance-closure ?b)))
                 (= (REL.FTN$composition (LAT.ADJ$left (reflection ?b ?c)) (intent ?c))
                    (REL.FTN$composition (extent ?b) (left-derivation ?b))))))
```

- For any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \vDash_{\boldsymbol{A}} \rangle$, we can restrict the instance-generation and type-generation to elements, thus defining an instance embedding function $\iota_{\boldsymbol{A}} : \mathit{inst}(\boldsymbol{A}) \rightarrow \mathit{latt}(\boldsymbol{A})$ and a type embedding function $\tau_{\boldsymbol{A}} : \mathit{typ}(\boldsymbol{A}) \rightarrow \mathit{latt}(\boldsymbol{A})$.

```
(16) (CNG$function instance-embedding)
     (CNG$signature instance-embedding CLS$classification SET.FTN$function)
     (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (instance-embedding ?a)) (CLS$instance ?a))
             (= (SET.FTN$target (instance-embedding ?a)) (concept ?a))
             (= (instance-embedding ?a)
                (SET.FTN$composition
                    (SET.FTN$singleton (CLS$instance ?a))
                    (instance-generation ?a)))))

(17) (CNG$function type-embedding)
     (CNG$signature type-embedding CLS$classification SET.FTN$function)
     (forall (?a (CLS$classification ?a))
        (and (= (SET.FTN$source (type-embedding ?a)) (CLS$type ?a))
             (= (SET.FTN$target (type-embedding ?a)) (concept ?a))
             (= (type-embedding ?a)
                (SET.FTN$composition
                    (SET.FTN$singleton (CLS$type ?a))
                    (type-generation ?a)))))
```

- By means of these two embedding function, we can prove that the notions of extent and intent for classifications extends to the notions of extent and intent for concept lattices.

  $\mathit{extent}_{\boldsymbol{A}}$ (for classifications) = $\tau_{\boldsymbol{A}} \cdot \mathit{extent}_{\boldsymbol{A}}$ (for concept lattices)

  $\mathit{intent}_{\boldsymbol{A}}$ (for classifications) = $\iota_{\boldsymbol{A}} \cdot \mathit{intent}_{\boldsymbol{A}}$ (for concept lattices)

  For clarity, we state these identities in an external namespace.

```
     (forall (?a (CLS$classification ?a))
        (and (= (CLS$extent ?a)
                (SET.FTN$composition (CLS.CL$type-embedding ?a) (CLS.CL$extent ?a)))
             (= (CLS$intent ?a)
                (SET.FTN$composition (CLS.CL$instance-embedding ?a) (CLS.CL$intent ?a)))
```

- Concepts in $\iota_{\boldsymbol{A}}(\mathit{inst}(\boldsymbol{A}))$ are called *instance concepts*, whereas concepts in $\tau_{\boldsymbol{A}}(\mathit{typ}(\boldsymbol{A}))$ are called *type concepts*.

```
(18) (CNG$function instance-concept)
     (CNG$signature instance-concept CLS$classification CLS$class)
```

```
      (forall (?a (CLS$classification ?a))
          (and (SET$subclass (instance-concept ?a) (concept ?a))
               (= (instance-concept ?a)
                  (SET.FTN$image (instance-embedding ?a)))))

(19) (CNG$function type-concept)
     (CNG$signature type-concept CLS$classification CLS$class)
     (forall (?a (CLS$classification ?a))
         (and (SET$subclass (type-concept ?a) (concept ?a))
              (= (type-concept ?a)
                 (SET.FTN$image (type-embedding ?a)))))
```

○ Because of the resolutions

$$c = \sqcap_{i \in extent_A(c)} \iota_A(i) = \sqcap_{t \in intent_A(c)} \tau_A(t)$$

for any concept $c \in \mathit{concept}(\boldsymbol{A})$, these functions satisfy the following conditions.

– The image $\iota_A(\mathit{inst}(\boldsymbol{A}))$ is join-dense in $\mathit{latt}(\boldsymbol{A})$.
– The image $\tau_A(\mathit{typ}(\boldsymbol{A}))$ is meet-dense in $\mathit{latt}(\boldsymbol{A})$.
– The classification incidence can be expressed in terms of embeddings and lattice order:

$$i \vDash_A t \text{ iff } \iota_A(i) \leq_A \tau_A(t).$$

```
      (forall (?a (CLS$classification ?a))
          (and ((ORD$join-dense (complete-lattice ?a)) (instance-concept ?a))
               ((ORD$meet-dense (complete-lattice ?a)) (type-concept ?a))
               (forall (?i ((instance ?a) ?i) ?t ((type ?a) ?t))
                   (<=> ((CLS$incidence ?a) [?i ?t])
                        ((ORD$extent (LAT$underlying (complete-lattice ?a)))
                            [((instance-embedding ?a) ?i) ((type-embedding ?a) ?t)])))))
```

o By the join-density property stated above, any concept in the lattice can be expressed as the join of a subset of join concepts.

```
      (forall (?a (CLS$classification ?a))
               ?c ((concept ?a) ?c))
          (and (exists (?X (SET$subclass ?X (instance-concept ?a)))
                   (= ?c ((join ?a) ?X)))
               (exists (?Y (SET$subclass ?Y (type-concept ?a)))
                   (= ?c ((meet ?a) ?Y)))))
```

o For any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \vDash_A \rangle$, there are two binary relations that correspond to the instance and type embedding functions.

Define the instance embedding classification $\iota_A : \mathit{inst}(\boldsymbol{A}) \rightarrow \mathit{concept}(\boldsymbol{A})$ called *iota* as follows: for every instance $a \in \mathit{inst}(\boldsymbol{A})$ and every formal concept $\boldsymbol{a} \in \mathit{concept}(\boldsymbol{A})$, the incidence relationship $a\iota_A\boldsymbol{a}$ holds when a is in the extent of $\boldsymbol{a}$; that is, $a \in \mathit{ext}_A(\boldsymbol{a})$. As a relation, this classification is closed on the right with respect to lattice order. The instance embedding relation can be defined in terms of the instance embedding function as follows:

$$\begin{tikzcd}
 & \mathit{typ}(\boldsymbol{A}) \\
\mathit{concept}(\boldsymbol{A}) \arrow[ur, "\tau_A"] & \\
 & \mathit{inst}(\boldsymbol{B}) \arrow[ul, "\iota_A"] \arrow[uu, "\boldsymbol{A}"']
\end{tikzcd}$$

Figure 4: *iota* & *tau*

$a\iota_A\boldsymbol{a}$ when $\iota_A(a) \leq_A \boldsymbol{a}$, for $a \in \mathit{inst}(\boldsymbol{A})$ and $\boldsymbol{a} \in \mathit{concept}(\boldsymbol{A})$.

This is an application of the right operator for a preorder that maps functions to binary relations.

Dually, define the type embedding classification $\tau_A : \mathit{concept}(\boldsymbol{A}) \rightarrow \mathit{typ}(\boldsymbol{A})$ called *tau* as follows: for every formal concept $\boldsymbol{a} \in \mathit{concept}(\boldsymbol{A})$ and every type $\alpha \in \mathit{typ}(\boldsymbol{A})$, the incidence relationship $\boldsymbol{a}\tau_A\alpha$ holds when α is in the intent of $\boldsymbol{a}$; that is, $\alpha \in \mathit{int}_A(\boldsymbol{a})$. As a relation, this classification is closed on the left with respect to lattice order. The type embedding relation can be defined in terms of the type embedding function as follows:

$\boldsymbol{a}\tau_A\alpha$ when $\boldsymbol{a} \leq_A \tau_A(\alpha)$, for $\boldsymbol{a} \in \mathit{concept}(\boldsymbol{A})$ and $\alpha \in \mathit{typ}(\boldsymbol{A})$.

This is an application of the left operator for a preorder that maps functions to binary relations.

Note the direction of the type embedding relation.

```
(20) (CNG$function iota)
     (CNG$signature iota CLS$classification CLS$classification)
     (forall (?a (CLS$classification ?a))
         (and (= (REL$instance (iota ?a)) (CLS$instance ?a))
              (= (REL$type (iota ?a)) (concept ?a))
              (= (iota ?a)
                 ((SET.FTN$right (concept-order ?a)) (instance-embedding ?a)))))

(21) (CNG$function tau)
     (CNG$signature tau CLS$classification CLS$classification)
     (forall (?a (CLS$classification ?a))
         (and (= (REL$instance (tau ?a)) (concept ?a))
              (= (REL$type (tau ?a)) (CLS$type ?a))
              (= (tau ?a)
                 ((SET.FTN$left (concept-order ?a)) (type-embedding ?a)))))
```

- The instance embedding function can be defined in terms of the instant embedding relation as follows:

  $\iota_A(a) = \sqcap_A(a\iota_A)$, for $a \in$ *inst*($\boldsymbol{A}$).

  The type embedding function can be defined in terms of the type embedding relation as follows:

  $\tau_A(\alpha) = \sqcup_A(\tau_A\alpha)$, for $\alpha \in$ *typ*($\boldsymbol{A}$).

  However, since we have already defined these functions by other means, these facts are expressed as theorems.

```
     (forall (?a (CLS$classification ?a))
              ?x ((CLS$instance ?a) ?x))
         (= ((instance-embedding ?a) ?x)
            ((meet ?a) ((REL$fiber12 (iota ?a)) ?x))))

     (forall (?a (CLS$classification ?a))
              ?y ((CLS$type ?a) ?y))
         (= ((type-embedding ?a) ?y)
            ((join ?a) ((REL$fiber21 (tau ?a)) ?y))))
```

- Intent induces a preorder on the instance class *inst*($\boldsymbol{A}$) defined by $i_1 \leq_A i_2$ when $\iota_A(i_1) \leq_A \iota_A(i_2)$ or $i_1' \supseteq i_2'$. Dually, extent induces a preorder on the type class *typ*($\boldsymbol{A}$) defined by $t_1 \leq_A t_2$ when $\tau_A(t_1) \leq_A \tau_A(t_2)$ or $t_1' \subseteq t_2'$.

```
(22) (CNG$function instance-order)
     (CNG$signature instance-order CLS$classification ORD$preorder)
     (forall (?a (CLS$classification ?a))
         (and (= (ORD$class (instance-order ?a)) (instance ?a))
              (forall (?i1 ((instance ?a) ?i1) (?i2 ((instance ?a) ?i2))
              (<=> ((ORD$extent (instance-order ?a)) [?i1 ?i2])
                   ((ORD$extent (concept-order ?a))
                       [((instance-embedding ?a) ?i1)
                        ((instance-embedding ?a) ?i2)]))))

(23) (CNG$function type-order)
     (CNG$signature type-order CLS$classification ORD$preorder)
     (forall (?a (CLS$classification ?a))
         (and (= (ORD$class (type-order ?a)) (type ?a))
              (forall (?t1 ((type ?a) ?t1) (?t2 ((type ?a) ?t2))
              (<=> ((ORD$extent (type-order ?a)) [?t1 ?t2])
                   ((ORD$extent (concept-order ?a))
                       [((type-embedding ?a) ?t1)
                        ((type-embedding ?a) ?t2)]))))
```

- Part of the "fundamental theorem" of Formal Concept Analysis states that every classification $\boldsymbol{A}$ = ⟨*inst*($\boldsymbol{A}$), *typ*($\boldsymbol{A}$), $\vDash_A$⟩ has an associated concept lattice *cl*($\boldsymbol{A}$) = ⟨*latt*($\boldsymbol{A}$), *inst*($\boldsymbol{A}$), *typ*($\boldsymbol{A}$), ι_A, τ_A⟩.

```
(24) (CNG$function concept-lattice)
     (CNG$signature concept-lattice CLS$classification CL$concept-lattice)
     (forall (?a (CLS$classification ?a))
         (and (= (CL$complete-lattice (concept-lattice ?a)) (complete-lattice ?a))
              (= (CL$instance (concept-lattice ?a)) (CLS$instance ?a))
```

```
                  (= (CL$type (concept-lattice ?a)) (CLS$type ?a))
                  (= (CL$instance-embedding (concept-lattice ?a)) (instance-embedding ?a))
                  (= (CL$type-embedding (concept-lattice ?a)) (type-embedding ?a))))
```

- The following fact of abstract Formal Concept Analysis, stated in terms of the embedding functions above, is straightforward to prove:

  $$\boldsymbol{A} = \iota_{\boldsymbol{A}} \circ \leq_{\boldsymbol{A}} \circ \tau_{\boldsymbol{A}}^{\text{op}}.$$

  This says that any classification (viewed as a binary relation) is identical to the composition of the instance-embedding relation followed by the lattice order of the classification followed by the type embedding relation.

```
(forall (?a (CLS$classification ?a))
    (= ?a
       (REL$composition (REL$composition (iota ?a) (concept-order ?a)) (tau ?a))))
```

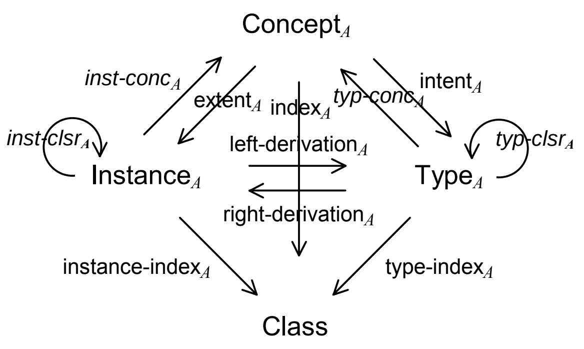


Diagram 5: Core Conglomerates and Functions for Fibers

Conceptual Fibers

`CLS.FIB`

This section defines fibers along the underlying classification function or compositions with this function (see Diagram 6).

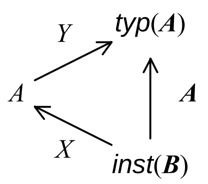

$$\begin{tikzcd} & typ(\mathbf{A}) \\ A \arrow[ur, "Y"] & \\ & inst(\mathbf{B}) \arrow[ul, "X"] \arrow[uu, "\mathbf{A}"'] \end{tikzcd}$$

Figure 4: *A*-instances & *A*-types

- For any classification $\mathbf{A} = \langle inst(\mathbf{A}), typ(\mathbf{A}), \models_{\mathbf{A}} \rangle$, a (*collective*) ***A**-instance* indexed by a class A is a binary relation $X \subseteq inst(\mathbf{A}) \times A$. For a fixed classification $\mathbf{A}$, the collection of all $\mathbf{A}$-instances is denoted '`(instance ?a)`'. Dually, a (*collective*) ***A**-type* indexed by a class A is a binary relation $Y \subseteq A \times typ(\mathbf{A})$. For a fixed classification $\mathbf{A}$, the collection of all $\mathbf{A}$-types is denoted '`(type ?a)`'.

```
(1) (KIF$function instance)
    (KIF$signature instance CLS$classification CNG$conglomerate)
    (forall (?a (CLS$classification ?a)
        (and (CNG$subconglomerate (instance ?a) REL$relation)
             (forall (?x (REL$relation ?x))
                 (<=> ((instance ?a) ?x)
                      (= (REL$source ?x) (CLS$instance ?a))))))

(2) (KIF$function instance-index)
    (KIF$signature instance-index CLS$classification CNG$function)
    (forall (?a (CLS$classification ?a))
        (and (CNG$signature (instance-index ?a) (instance ?a) SET$class)
             (forall (?x ((instance ?a) ?x))
                 (= ((instance-index ?a) ?x) (REL$target ?x)))))

(3) (KIF$function type)
    (KIF$signature type CLS$classification CNG$conglomerate)
    (forall (?a (CLS$classification ?a)
        (and (CNG$subconglomerate (type ?a) REL$relation)
             (forall (?y (REL$relation ?y))
                 (<=> ((type ?a) ?y)
                      (= (REL$target ?y) (CLS$type ?a))))))

(4) (KIF$function type-index)
    (KIF$signature type-index CLS$classification CNG$function)
    (forall (?a (CLS$classification ?a))
        (and (CNG$signature (type-index ?a) (type ?a) SET$class)
             (forall (?y ((type ?a) ?y))
                 (= ((type-index ?a) ?y) (REL$source ?y)))))
```

- Two derivation operators in this fibered setting correspond to the two relational residuation operators. For any large classification $\mathbf{A}$ there are two senses of *derivation* corresponding to the two senses of relational residuation. Derivation preserves indices. Left derivation maps an instance to the type on which it is "universally incident," and dually, right derivation maps a type to the instance which is "universally incident" on it: for all instances $X \subseteq inst(\mathbf{A}) \times A$ and all types $Y \subseteq A \times typ(\mathbf{A})$,

$X \mapsto X' = X\backslash\boldsymbol{A}$ and $Y \mapsto Y' = \boldsymbol{A}/Y$.

```
(5) (KIF$function left-derivation)
    (KIF$signature left-derivation CLS$classification CNG$function)
    (forall (?a (CLS$classification ?a))
        (and (CNG$signature (left-derivation ?a) (instance ?a) (type ?a))
             (forall (?x ((instance ?a) ?x))
                 (= ((left-derivation ?a) ?x)
                    (REL$left-residuation ?x ?a)))))

(6) (CNG$function right-derivation)
    (CNG$signature right-derivation CLS$classification SET.FTN$function)
    (forall (?a (CLS$classification ?a))
        (and (CNG$signature (right-derivation ?a) (type ?a) (instance ?a))
             (forall (?y ((type ?a) ?y))
                 (= ((right-derivation ?a) ?y)
                    (REL$right-residuation ?y ?a)))))
```

○ A simple fundamental result is the following equivalence. For all instances $X \subseteq \mathit{inst}(\boldsymbol{A}) \times A$ and all types $Y \subseteq A \times \mathit{typ}(\boldsymbol{A})$

$$Y \subseteq X\backslash\boldsymbol{A} \text{ in } \mathit{REL}[A, \mathit{typ}(\boldsymbol{A})] \ \underline{\text{iff}}\ X \circ Y \subseteq \boldsymbol{A} \ \underline{\text{iff}}\ X \subseteq \boldsymbol{A}/Y \text{ in } \mathit{REL}[\mathit{inst}(\boldsymbol{A}), A].$$

This describes a Galois connection between left derivation

$$(\text{-})\backslash\boldsymbol{A} : \mathit{REL}[\mathit{inst}(\boldsymbol{A}), A] \rightarrow (\mathit{REL}[A, \mathit{typ}(\boldsymbol{A})])^{\text{op}}$$

and right derivation

$$\boldsymbol{A}/(\text{-}) : (\mathit{REL}[A, \mathit{typ}(\boldsymbol{A})])^{\text{op}} \rightarrow \mathit{REL}[\mathit{inst}(\boldsymbol{A}), A].$$

The first two facts assert (contravariant) monotonicity of derivation. The last fact asserts the adjointness condition.

```
    (forall (?x1 ((instance ?a) ?x1)
             ?x2 ((instance ?a) ?x2)
             (= ((instance-index ?a) ?x1)
                ((instance-index ?a) ?x2)))
        (=> (REL$subrelation ?x1 ?x2)
            (SET$subrelation ((left-derivation ?a) ?x2) ((left-derivation ?a) ?x1))))

    (forall (?y1 ((type ?a) ?y1)
             ?y2 ((type ?a) ?y2)
             (= ((type-index ?a) ?y1)
                ((type-index ?a) ?y2)))
        (=> (REL$subrelation ?y2 ?y1)
            (REL$subrelation ((right-derivation ?a) ?y1) ((right-derivation ?a) ?y2))))

    (forall (?x ((instance ?a) ?x)
             ?y ((type ?a) ?y)
             (= ((instance-index ?a) ?x)
                ((type-index ?a) ?y)))
        (<=> (REL$subrelation ?y ((left-derivation ?a) ?x))
             (REL$subrelation ?x ((right-derivation ?a) ?y))))
```

○ The composition of derivations gives two senses of closure operator.

```
(7) (KIF$function instance-closure)
    (KIF$signature instance-closure CLS$classification CNG$function)
    (forall (?a (CLS$classification ?a))
        (and (CNG$signature (instance-closure ?a) (instance ?a) (instance ?a))
             (forall (?x ((instance ?a) ?x))
                 (and (= (instance-index ((instance-closure ?a) ?x))
                         (instance-index ?x))
                      (= ((instance-closure ?a) ?x)
                         ((right-derivation ?a) ((left-derivation ?a) ?x)))))))

(8) (CNG$function type-closure)
     (CNG$signature type-closure CLS$classification CNG$function)
        (and (CNG$signature (type-closure ?a) (type ?a) (type ?a))
             (forall (?y ((type ?a) ?y))
                 (and (= (type-index ((type-closure ?a) ?y))
                         (type-index ?y))
                      (= ((type-closure ?a) ?y)
                         ((left-derivation ?a) ((right-derivation ?a) ?y)))))))
```

○ The following (easily proven) results confirm that these are closure operators.

$X \subseteq X''$ and $X' = X'''$ for all instances $X \subseteq \mathit{inst}(\boldsymbol{A}) \times A$.

$Y \subseteq Y''$ and $Y' = Y'''$ for all types $Y \subseteq A \times \mathit{typ}(\boldsymbol{A})$.

```
(forall (?x ((instance ?a) ?x))
    (and (REL$subrelation ?x ((instance-closure ?a) ?x))
         (= ((left-derivation ?a) ?x)
            ((left-derivation ?a) ((instance-closure ?a) ?x)))))

(forall (?y ((type ?a) ?y))
    (and (REL$subrelation ?y ((type-closure ?a) ?y))
         (= ((right-derivation ?a) ?y)
            ((right-derivation ?a) ((type-closure ?a) ?y)))))
```

o For any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$, a (*collective*) ***A**-concept* $\boldsymbol{C} = \langle \mathit{ind}(\boldsymbol{C}), \mathit{ext}(\boldsymbol{C}), \mathit{int}(\boldsymbol{C}) \rangle$, consists of an *extent* **A**-instance $\mathit{ext}(\boldsymbol{C}) : \mathit{inst}(\boldsymbol{A}) \rightarrow \mathit{ind}(\boldsymbol{A})$ indexed by $\mathit{ind}(\boldsymbol{A})$, and an *intent* **A**-type $\mathit{int}(\boldsymbol{C}) : \mathit{ind}(\boldsymbol{A}) \rightarrow \mathit{typ}(\boldsymbol{A})$ indexed by $\mathit{ind}(\boldsymbol{A})$, which satisfy the following closure conditions:

$\mathit{ext}(\boldsymbol{C}) = \boldsymbol{A}/\mathit{int}(\boldsymbol{C})$ and $\mathit{int}(\boldsymbol{C}) = \mathit{ext}(\boldsymbol{C})\backslash\boldsymbol{A}$.

The collection of all **A**-concepts is denoted by '`(concept ?a)`'.

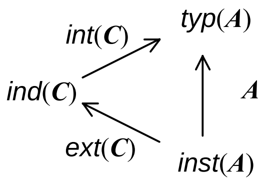

$$\begin{tikzcd} & & \mathit{typ}(\boldsymbol{A}) \\ \mathit{ind}(\boldsymbol{C}) \arrow[urr, "\mathit{int}(\boldsymbol{C})"] & & \\ & & \mathit{inst}(\boldsymbol{A}) \arrow[ull, "\mathit{ext}(\boldsymbol{C})"] \arrow[uu, "\boldsymbol{A}"'] \end{tikzcd}$$

Figure 5: *A*-Concept

```
(9) (KIF$function concept)
    (KIF$signature concept CLS$classification CNG$conglomerate)

(10) (KIF$function extent)
     (KIF$signature extent CLS$classification CNG$function)
     (forall (?a (CLS$classification ?a))
         (CNG$signature (extent ?a) (concept ?a) (instance ?a)))

(11) (KIF$function intent)
     (KIF$signature intent CLS$classification CNG$function)
     (forall (?a (CLS$classification ?a))
         (CNG$signature (intent ?a) (concept ?a) (type ?a)))

(14) (KIF$function index)
     (KIF$signature index CLS$classification CNG$function)
     (forall (?a (CLS$classification ?a))
         (and (CNG$signature (index ?a) (concept ?a) SET$class)
              (forall (?c ((concept ?a) ?c))
                  (and (= ((index ?a) ?c)
                          ((instance-index ?a) ((extent ?a) ?c)))
                       (= ((index ?a) ?c)
                          ((type-index ?a) ((intent ?a) ?c)))
                       (= ((left-derivation ?a) ((extent ?a) ?c))
                          ((intent ?a) ?c))
                       (= ((right-derivation ?a) ((intent ?a) ?c))
                          ((extent ?a) ?c))))))
```

○ There are surjective generator functions, called instance-generation and type-generation, that map instances and types to their generated concepts. For all instances $X \subseteq \mathit{inst}(\boldsymbol{A}) \times A$ and types $Y \subseteq A \times \mathit{typ}(\boldsymbol{A})$

$X \mapsto \langle X'', X' \rangle$ "instance-generation"

$Y \mapsto \langle Y', Y'' \rangle$ "type-generation".

– The composition of instance-generation and intent equals left-derivation, and the composition of type-generation and extent equals right-derivation.

- The composition of instance-generation and extent equals instance-closure, and the composition of type-generation and intent equals type-closure.
- The composition of extent and instance-generation is identity, and the composition of intent and type-generation is identity.

These conditions define generation, and also insure that all concepts are captured in the conglomerate ‘`(concept ?a)`’. Concepts are determined by either their extents or their intents – this is represented by the fact that the extent and intent functions are injective.

```
(15) (KIF$function instance-generation)
     (KIF$signature instance-generation CLS$classification CNG$function)
     (forall (?a (CLS$classification ?a))
         (and (CNG$signature (instance-generation ?a) (instance ?a) (concept ?a))
              (forall (?x ((instance ?a) ?x))
                  (and (= ((index ?a) ((instance-generation ?a) ?x))
                          ((instance-index ?a) ?x))
                       (= ((intent ?a) ((instance-generation ?a) ?x))
                          ((left-derivation ?a) ?x))
                       (= ((extent ?a) ((instance-generation ?a) ?x))
                          ((instance-closure ?a) ?x))))
              (forall (?c ((concept ?a) ?c))
                  (= ((instance-generation ?a) ((extent ?a) ?c)) ?c))))

(16) (KIF$function type-generation)
     (KIF$signature type-generation CLS$classification CNG$function)
     (forall (?a (CLS$classification ?a))
         (and (CNG$signature (type-generation ?a) (type ?a) (concept ?a))
              (forall (?y ((type ?a) ?y))
                  (and (= ((index ?a) ((type-generation ?a) ?y))
                          ((type-index ?a) ?y))
                       (= ((extent ?a) ((type-generation ?a) ?y))
                          ((right-derivation ?a) ?y))
                       (= ((intent ?a) ((type-generation ?a) ?y))
                          ((type-closure ?a) ?y))))
              (forall (?c ((concept ?a) ?c))
                  (= ((type-generation ?a) ((intent ?a) ?c)) ?c))))
```

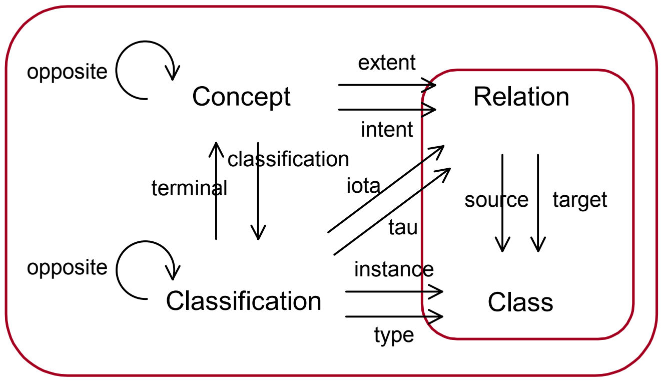


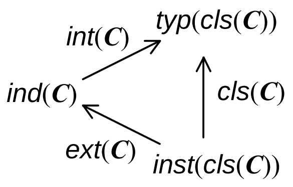
Diagram 6: Core Conglomerates and Functions for Concepts

Collective Concepts

`CLS.CONC`

o Here we define collective concepts that internally include their underlying classifications. A formal (collective) concept ***C*** = ⟨*cls*(***C***), *ind*(***C***), *ext*(***C***), *int*(***C***)⟩ consists of an underlying *classification* *cls*(***C***), an *index*ing class *ind*(***C***), an *extent* relation *ext*(***C***) and an *intent* relation *int*(***C***). These components satisfy the equivalent closure conditions:

ext(***C***) = *cls*(***C***)/*int*(***C***) and *int*(***C***) = *ext*(***C***)\cls(***C***).



$$\begin{tikzcd} & \mathit{typ}(\mathit{cls}(C)) \\ \mathit{ind}(C) \arrow[ur, "\mathit{int}(C)"] & \\ & \mathit{inst}(\mathit{cls}(C)) \arrow[ul, "\mathit{ext}(C)"] \arrow[uu, "\mathit{cls}(C)"'] \end{tikzcd}$$

Figure 6: Collective Concept

The conglomeration of all concepts is the disjoint union of all the conceptual fibers ‘`(CLS.FIB$concept ?a)`’ as ***A*** ranges over the conglomerate of all large classifications.

Axiom (1) specifies the (collective) concept conglomerate. Axiom (2) specifies the underlying classification function. This function is part of a fibration. The index function is specified in axiom (3). When this maps to unity (the unit class), the concept is a point. Axiom (6) expresses the closure conditions. Axiom (7) expresses the disjoint union property.

```
(1) (CNG$conglomerate concept)

(2) (CNG$function classification)
    (CNG$signature classification concept CLS$classification)

(3) (CNG$function index)
    (CNG$signature index concept SET$class)

(4) (CNG$function extent)
    (CNG$signature extent concept REL$relation)
    (forall (?c (concept ?c))
        (and (= (REL$source (extent ?c))
                (CLS$instance (classification ?c)))
             (= (REL$target (extent ?c))
                (index ?c))))

(5) (CNG$function intent)
    (CNG$signature intent concept REL$relation)
    (forall (?c (concept ?c))
        (and (= (REL$source (intent ?c))
                (index ?c))
             (= (REL$target (intent ?c))
                (CLS$type (classification ?c)))))

(6) (forall (?c (concept ?c))
        (and (= (extent ?c)
                (REL$right-residuation (intent ?c) (classification ?c)))
             (= (intent ?c)
                (REL$left-residuation (extent ?c) (classification ?c)))))

(7) (forall (?c)
        (<=>  (concept ?c)
              (exists (?cf ((CLS.FIB$concept (classfication ?c)) ?cf))
                  (and (= (index ?c)
                          ((CLS.FIB$index (classfication ?c)) ?cf))
                       (= (extent ?c)
                          ((CLS.FIB$extent (classfication ?c)) ?cf))
                       (= (intent ?c)
                          ((CLS.FIB$intent (classfication ?c)) ?cf))))))
```

- For any concept $\boldsymbol{C} = \langle \mathit{cls}(\boldsymbol{C}), \mathit{ind}(\boldsymbol{C}), \mathit{ext}(\boldsymbol{C}), \mathit{int}(\boldsymbol{C})\rangle$, the *opposite* or *dual* of $\boldsymbol{C}$ is the concept $\boldsymbol{C}^{\perp} = \langle \mathit{cls}(\boldsymbol{C})^{\perp}, \mathit{ind}(\boldsymbol{C}), \mathit{int}(\boldsymbol{C}), \mathit{ext}(\boldsymbol{C})\rangle$, whose classification is the opposite of the classification of $\boldsymbol{C}$, whose index is the same as the index of $\boldsymbol{C}$, whose extent is the intent of $\boldsymbol{C}$, and whose intent is the extent of $\boldsymbol{C}$. Axiom (8) specifies the opposite operator on classifications.

```
(8) (CNG$function opposite)
    (CNG$signature opposite concept concept)
    (forall (?c (concept ?c))
        (and (= (classification (opposite ?c)) (CLS$opposite (classification ?c)))
             (= (index (opposite ?c)) (index ?c))
             (= (extent (opposite ?c)) (intent ?c))
             (= (intent (opposite ?c)) (extent ?c))))
```

- For any two collective concepts $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ over the same classification $\mathit{cls}(\boldsymbol{C}_1) = \boldsymbol{A} = \mathit{cls}(\boldsymbol{C}_2)$, a *two cell* $f : \boldsymbol{C}_1 \Rightarrow \boldsymbol{C}_2$ from $\boldsymbol{C}_1$ to $\boldsymbol{C}_2$ is a function $f : \mathit{ind}(\boldsymbol{C}_1) \rightarrow \mathit{ind}(\boldsymbol{C}_2)$ between index classes, which satisfies the following conditions:

$$\mathit{ext}(\boldsymbol{C}_1) = \mathit{ext}(\boldsymbol{C}_2) \circ f^{\mathrm{op}} = \mathit{ext}(\boldsymbol{C}_2) \,/\, f$$

$$\mathit{int}(\boldsymbol{C}_1) = f \circ \mathit{int}(\boldsymbol{C}_2) = f^{\mathrm{op}} \setminus \mathit{int}(\boldsymbol{C}_2)$$

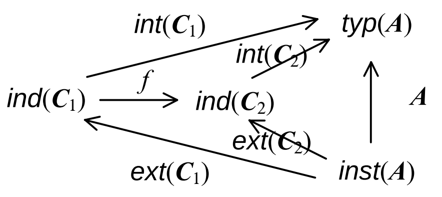


Figure 6: 2-cell

```
(9) (CNG$conglomerate two-cell)

(10) (CNG$function source)
     (CNG$signature source two-cell concept)

(11) (CNG$function target)
     (CNG$signature target two-cell concept)

(12) (CNG$function function)
     (CNG$signature function two-cell SET.FTN$function)

     (forall (?f (two-cell ?f))
         (and (= (classification (source ?f))
                 (classification (target ?f)))
              (= (index (source ?f))
                 (SET.FTN$source (function ?f)))
              (= (index (target ?f))
                 (SET.FTN$target (function ?f)))
              (= (SET.FTN$composition
                     (extent (target ?f))
                     (REL$opposite (SET.FTN$fn2rel (function ?f))))
                 (extent (source ?f)))
              (= (SET.FTN$composition
                     (SET.FTN$fn2rel (function ?f))
                     (intent (target ?f))
                 (intent (source ?f)))))
```

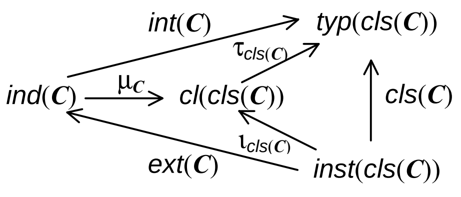
- The closure conditions $\iota_{\boldsymbol{A}} = \models_{\boldsymbol{A}}/\tau_{\boldsymbol{A}}$ and $\tau_{\boldsymbol{A}} = \iota_{\boldsymbol{A}}\backslash\models_{\boldsymbol{A}}$ hold between the instance embedding relation $\iota_{\boldsymbol{A}} : \mathit{inst}(\boldsymbol{A}) \rightarrow \mathit{concept}(\boldsymbol{A})$ and the type embedding relation $\tau_{\boldsymbol{A}} : \mathit{concept}(\boldsymbol{A}) \rightarrow \mathit{typ}(\boldsymbol{A})$. So, for any classification $\boldsymbol{A}$, a basic example of a collective concept is $\mathit{terminal}(\boldsymbol{A}) = \langle \boldsymbol{A}, \mathit{concept}(\boldsymbol{A}), \iota_{\boldsymbol{A}}, \tau_{\boldsymbol{A}}\rangle$ with $\iota_{\boldsymbol{A}}$ as collective extent, $\tau_{\boldsymbol{A}}$ as collective intent, and the concept class $\mathit{concept}(\boldsymbol{A})$ as index.

```
(13) (CNG$function terminal)
     (CNG$signature terminal CLS$classification concept)
     (forall (?a (CLS$classification ?a))
         (and (= (classification (terminal ?a)) ?a)
              (= (index (terminal ?a)) (CLS.CL$concept ?a))
              (= (extent (terminal ?a)) (iota ?a))
              (= (intent (terminal ?a)) (tau ?a))))
```

- Any collective concept $\boldsymbol{C} = \langle \boldsymbol{A}, A, X, Y\rangle = \langle \mathit{cls}(\boldsymbol{C}), \mathit{ind}(\boldsymbol{C}), \mathit{ext}(\boldsymbol{C}), \mathit{int}(\boldsymbol{C})\rangle$ induces a unique mediating function $\mu_{\boldsymbol{C}} : \mathit{ind}(\boldsymbol{C}) \rightarrow \mathit{cl}(\mathit{cls}(\boldsymbol{C}))$ that satisfies the following constraints.

  $$\mathit{ext}(\boldsymbol{C}) = \iota_{\mathit{cls}(\boldsymbol{C})} \circ {\mu_{\boldsymbol{C}}}^{\mathrm{op}} = \iota_{\mathit{cls}(\boldsymbol{C})} / \mu_{\boldsymbol{C}}$$

  $$\mathit{int}(\boldsymbol{C}) = \mu_{\boldsymbol{C}} \circ \tau_{\mathit{cls}(\boldsymbol{C})} = {\mu_{\boldsymbol{C}}}^{\mathrm{op}} \setminus \tau_{\mathit{cls}(\boldsymbol{C})}$$



$$\begin{tikzcd}
 & & \mathit{typ}(\mathit{cls}(\boldsymbol{C})) \\
\mathit{ind}(\boldsymbol{C}) \arrow[urr, "\mathit{int}(\boldsymbol{C})"] \arrow[r, "\mu_{\boldsymbol{C}}"] & \mathit{cl}(\mathit{cls}(\boldsymbol{C})) \arrow[ur, "\tau_{\mathit{cls}(\boldsymbol{C})}"] & \\
 & & \mathit{inst}(\mathit{cls}(\boldsymbol{C})) \arrow[ul, "\iota_{\mathit{cls}(\boldsymbol{C})}"'] \arrow[ull, "\mathit{ext}(\boldsymbol{C})"] \arrow[uu, "\mathit{cls}(\boldsymbol{C})"']
\end{tikzcd}$$

Figure 6: Mediating Function

  The definition $\mu_{\boldsymbol{C}}(a) = \langle Xa, aY\rangle$ is well-defined, since the closure conditions are equivalent to the (pointwise) fact that $Xa = (aY)'$ and $aY = (Xa)'$; that is, that $\langle Xa, aY\rangle \in \mathit{latt}(\boldsymbol{A})$. Conversely, for any classification $\boldsymbol{A}$ and for any function $f : \mathit{ind}(\boldsymbol{C}) \rightarrow \mathit{cl}(\boldsymbol{A})$, the quadruple $\langle \boldsymbol{A}, \mathit{concept}(\boldsymbol{A}), \iota_{\mathit{cls}(\boldsymbol{C})} \circ f^{\mathrm{op}}, f \circ \tau_{\mathit{cls}(\boldsymbol{C})}\rangle$ is an collective concept. So the CNG function '`(mediator-function ?c)`' defined below is a bijection.

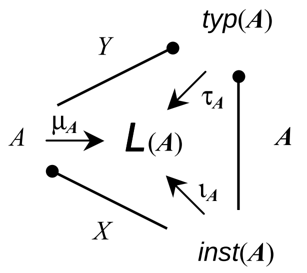
```
(14) (CNG$function mediator-function)
     (CNG$signature mediator-function concept SET.FTN$function)
     (forall (?c (concept ?c))
         (and (= (SET.FTN$source (mediator-function ?c))
                 (index ?c))
              (= (SET.FTN$target (mediator-function ?c))
                 (CLS.CL$concept (classification ?c)))
              (= (SET.FTN$composition
                     (mediator-function ?c)
                     (CLS.CL$extent (classification ?c)))
                 (SET.FTN$fiber21 (extent ?c)))
              (= (SET.FTN$composition
                     (mediator-function ?c)
                     (CLS.CL$intent (classification ?c)))
                 (SET.FTN$fiber12 (intent ?c)))))
```

- For any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}}\rangle$, there is a special (collective) concept that is in a sense is the largest concept over that classification. Its extent relation $\iota_{\boldsymbol{A}} \subseteq \mathit{inst}(\boldsymbol{A}) \times \mathit{concept}(\boldsymbol{A})$ is the instance embedding relation *iota*, and its intent relation $\tau_{\boldsymbol{A}} \subseteq \mathit{concept}(\boldsymbol{A}) \times \mathit{typ}(\boldsymbol{A})$ is the type embedding relation *tau*.

- There are bijections between subsets of instances/types and collective instances/types. Any subclass of instances $X \subseteq \mathit{inst}(\boldsymbol{A})$ is a (has an associated) collective $\boldsymbol{A}$-instance $\mathit{inst\text{-}dist}(X) \subseteq \mathit{inst}(\boldsymbol{A}) \times \mathit{1}$ indexed by the unit class $\mathit{1}$. A similar statement holds for types.

```
    (KIF$function instance-distribution)
    (KIF$signature instance-distribution CLS$classification CNG$function)
    (forall (?a (CLS$classification ?a))
        (and (CNG$signature (instance-distribution ?a)
                 (SET$power (CLS$instance ?a)) (instance ?a))
             (forall (?X (SET$subclass ?X (CLS$instance ?a)))
                 (and (= ((instance-index ?a) ((instance-distribution ?a) ?X))
                         SET.LIM$unit)
                      (forall (?x ((CLS$instance ?a) ?x))
                          (<=> ((REL$extent ((instance-distribution ?a) ?X)) [?x 0])
                               (?X ?x)))))))

     (KIF$function type-distribution)
     (KIF$signature type-distribution CLS$classification CNG$function)
     (forall (?a (CLS$classification ?a))
         (and (CNG$signature (type-distribution ?a)
                  (SET$power (CLS$type ?a)) (type ?a))
              (forall (?Y (SET$subclass ?Y (CLS$type ?a)))
                  (and (= ((type-index ?a) ((type-distribution ?a) ?Y))
                           SET.LIM$unit)
                       (forall (?y ((CLS$type ?a) ?y))
                           (<=> ((REL$extent ((type-distribution ?a) ?X)) [0 ?y])
                                (?Y ?y)))))))
```

○ A collective concept is also called a *concept space*. The indexed or named formal concepts in a concept space are also called *conceptual views*. Conceptual knowledge is represented the following components of a concept space.
  - The three preorders on instances, types and conceptual views.
  - The membership or instantiation relation between instances and conceptual views, whose columns, when regarded as a Boolean matrix, record the *extent* of all the conceptual views.



$\boldsymbol{A}$	*classification*
inst($\boldsymbol{A}$)	*instance preorder*
typ($\boldsymbol{A}$)	*type preorder*
A	*index preorder*
X	*extent*
Y	*intent*
L($\boldsymbol{A}$)	*concept lattice*

  - The abstraction relation between conceptual views and types, whose rows, when regarded as a Boolean matrix, record the *intent* of all the conceptual views.
  - The *classification* relation between instances and types.

The constraining relation ships between the extent and intent of a concept space can be expressed in three different forms: residuation, inclusion and derivation.

incidence constraints		
residuation	*inclusion*	*derivation*
$Y = X\backslash\boldsymbol{A}$ $X = \boldsymbol{A}/Y$	$\forall_{a \in A,\, t \in \mathit{typ}(\boldsymbol{A})}\, aYt$ *iff* $Xa \subseteq \boldsymbol{A}t$ $\forall_{i \in \mathit{inst}(\boldsymbol{A}),\, a \in A}\, iXa$ *iff* $i\boldsymbol{A} \supseteq aY$	$\forall_{a \in A}\;\; aY = (Xa)'$ $\forall_{a \in A}\;\; Xa = (aY)'$

In order to construct a concept space, we start out by specifying the class A to be a collection of names, with each $a \in A$ representing a subset of types $aY_0 \subseteq \mathit{typ}(\boldsymbol{A})$. We use the two residuation operators, which are a generalized form of the derivation operators of Formal Concept Analysis, in order to define the notion of a *collective formal concept*. We define the extent X to be the right residuation of $\boldsymbol{A}$ along Y_0:

$$X = \boldsymbol{A}/Y_0 = \{(i, a) \mid \forall_{t \in \mathit{typ}(\boldsymbol{A})} (aY_0t \Rightarrow i\boldsymbol{A}t)\}.$$

so that $X \circ Y_0 \subseteq \boldsymbol{A}$. We define Y to be the left residuation of $\boldsymbol{A}$ along X:

$$Y = X\backslash\boldsymbol{A} = \{(a, t) \mid \forall_{i \in \mathit{inst}(\boldsymbol{A})} (iXa \Rightarrow i\boldsymbol{A}t)\}.$$

From these definitions it is straightforward to show that X is the right residuation, $X = \boldsymbol{A}/Y$, of $\boldsymbol{A}$ along Y: First of all, since $Y_0 \subseteq X\backslash\boldsymbol{A} = Y$, by contravariance of residuation $X = \boldsymbol{A}/Y_0 \supseteq \boldsymbol{A}/Y$; and secondly, since $X \circ Y = X \circ (X\backslash\boldsymbol{A}) \subseteq \boldsymbol{A}$, we have $X \subseteq \boldsymbol{A}/Y$.

Table 2: Conglomerate Functions used to define the Concept Lattice Functor

left derivation : Classification → Function
right derivation : Classification → Function
instance closure : Classification → Function
type closure : Classification → Function
concept : Classification → Class
extent : Classification → Function
intent : Classification → Function
instance concept : Classification → Function
type concept : Classification → Function
concept order : Classification → Partial Order
meet : Classification → Function
join : Classification → Function
complete lattice : Classification → Complete Lattice
adjoint pair : Infomorphism → Adjoint Pair
instance embedding : Classification → Function
type embedding : Classification → Function
concept lattice : Classification → Concept Lattice
concept morphism : Infomorphism → Concept Morphism
instance embedding relation : Classification → Relation
type embedding relation : Classification → Relation

Table 3: Functors and Natural Isomorphisms

$L \circ C = Id_{\text{Classification}}$	`CLS.CL$concept-lattice . CL$classification = Id`
	`CLS.INFO$concept-morphism . CL.MOR$infomorphism = Id`
$C \circ L \cong Id_{\text{Classification}}$	`CL$classification . CLS.CL$concept-lattice ≅ Id`
	`CL.MOR$infomorphism . CLS.INFO$concept-morphism ≅ Id`

Functional Infomorphisms

CLS.INFO

o Classifications are related through (functional) infomorphisms. A (*functional*) *infomorphism* $\boldsymbol{f} : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ from classification $\boldsymbol{A}$ to classification $\boldsymbol{B}$ (see Figure 2) is a pair $\boldsymbol{f} = \langle \mathit{inst}(\boldsymbol{f}), \mathit{typ}(\boldsymbol{f})\rangle$ of oppositely directed functions, $\mathit{inst}(\boldsymbol{f}) : \mathit{inst}(\boldsymbol{B}) \rightarrow \mathit{inst}(\boldsymbol{A})$ and $\mathit{typ}(\boldsymbol{f}) : \mathit{typ}(\boldsymbol{A}) \rightarrow \mathit{typ}(\boldsymbol{B})$, which satisfy the fundamental property:

$$\mathit{inst}(\boldsymbol{f})(i) \vDash_{\boldsymbol{A}} t \text{ iff } i \vDash_{\boldsymbol{B}} \mathit{typ}(\boldsymbol{f})(t)$$

$$\begin{tikzcd}
\mathit{typ}(\boldsymbol{A}) \arrow[r, "\mathit{typ}(\boldsymbol{f})"] & \mathit{typ}(\boldsymbol{B}) \\
\mathit{inst}(\boldsymbol{A}) \arrow[u, "\boldsymbol{A}"'] & \mathit{inst}(\boldsymbol{B}) \arrow[l, "\mathit{inst}(\boldsymbol{f})"] \arrow[u, "\boldsymbol{B}"']
\end{tikzcd}$$

Figure 2: Functional Infomorphism

for all instances $i \in \mathit{inst}(\boldsymbol{B})$ and all types $t \in \mathit{typ}(\boldsymbol{A})$. Note that the relation expressed on the either side of this equivalence is the bond of the relational infomorphism generated by this functional infomorphism. This *relational infomorphism* is defined below in terms of the left relation of the *instance monotonic function* and the right relation of the *type monotonic function*.

The following is a KIF representation for the elements of an infomorphism. Such elements are useful for the definition of an infomorphism. In the same fashion as classifications, infomorphisms are specified by declaration and population. The term '`infomorphism`' specified in axiom (1) allows one to *declare* infomorphisms themselves, and the two terms '`source`' and '`target`' specified in axioms (2–3) allow one to *declare* their associated source (domain) and target (codomain) classifications, respectively. The terms '`instance`' and '`type`' specified in axioms (4–5) resolve infomorphisms into their parts, thus allowing one to *populate* infomorphisms. The Infomorphism namespace is a sub-namespace of the Classification namespace.

```
(1) (CNG$conglomeration infomorphism)

(2) (CNG$function source)
    (CNG$signature source infomorphism CLS$classification)

(3) (CNG$function target)
    (CNG$signature target infomorphism CLS$classification)

(4) (CNG$function instance)
    (CNG$signature instance infomorphism SET.FTN$function)
    (forall (?f (infomorphism ?f))
        (and (= (SET.FTN$source (instance ?f)) (CLS$instance (target ?f)))
             (= (SET.FTN$target (instance ?f)) (CLS$instance (source ?f)))))

(5) (CNG$function type)
    (CNG$signature type infomorphism SET.FTN$function)
    (forall (?f (infomorphism ?f))
        (and (= (SET.FTN$source (type ?f)) (CLS$type (source ?f)))
             (= (SET.FTN$target (type ?f)) (CLS$type (target ?f)))))
```

o Axiom (6) gives the KIF for the fundamental property of an infomorphism.

```
(6) (forall (?f (infomorphism ?f)
             ?i ((CLS$instance (target ?f)) ?i)
             ?t ((CLS$type (source ?f)) ?t))
        (<=> ((CLS$incidence (source ?f)) [((instance ?f) ?i) ?t])
             ((CLS$incidence (target ?f)) [?i ((type ?f) ?t)])))
```

o The fundamental property of an infomorphism $\boldsymbol{f} : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ expresses the bonding classification of a bond $\mathit{bond}(\boldsymbol{f}) : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ associated with the infomorphism. This can be equivalently define in terms of either the instance or the type function. Here we use the instance function. For comparison, see the right operator that maps a function to a relation in the presence of a preorder.

```
(7) (CNG$function bond)
    (CNG$signature bond infomorphism CLS.BND$bond)
    (forall (?f (infomorphism ?f))
        (and (= (CLS.BND$source (bond ?f)) (source ?f))
             (= (CLS.BND$target (bond ?f)) (target ?f))
```

```
                (forall (?i ((CLS$instance (target ?f)) ?i)
                         ?t ((CLS$type (source ?f)) ?t))
                (<=> ((CLS$incidence (CLS.BND$classification (bond ?f))) [?i ?t])
                     ((CLS$incidence (source ?f)) [((instance ?f) ?i) ?t]))))
```

- The instance function is a monotonic function between instance orders. Dually, the type function is a monotonic function between type orders. Axiom (8) defines the monotonic instance function in KIF, and axiom (9) does the same for the monotonic type function.

```
(8) (CNG$function monotonic-instance)
    (CNG$signature monotonic-instance infomorphism ORD.FTN$monotonic-function)
    (forall (?f (infomorphism ?f))
        (and (= (ORD.FTN$source (monotonic-instance ?f))
                (instance-order (target ?f)))
             (= (ORD.FTN$target (monotonic-instance ?f))
                (instance-order (source ?f)))
             (= (ORD.FTN$function (monotonic-instance ?f))
                (instance ?f))))

(9) (CNG$function monotonic-type)
    (CNG$signature monotonic-type infomorphism ORD.FTN$monotonic-function)
    (forall (?f (infomorphism ?f))
        (and (= (ORD.FTN$source (monotonic-type ?f))
                (type-order (source ?f)))
             (= (ORD.FTN$target (monotonic-type ?f))
                (type-order (target ?f)))
             (= (ORD.FTN$function (monotonic-type ?f))
                (type ?f))))
```

- The instance relation is the left relation of the monotonic instance function. The type relation is the right relation of the monotonic type function. Axiom (10) defines the instance relation of an infomorphism, and axiom (11) defines the type relation of an infomorphism.

```
(10) (CNG$function relational-instance)
     (CNG$signature relational-instance infomorphism REL$relation)
     (forall (?f (infomorphism ?f))
         (and (= (REL$source (relational-instance ?f))
                 (instance (source ?f)))
              (= (REL$target (relational-instance ?f))
                 (instance (target ?f)))
              (= (relational-instance ?f)
                 ((ORD.FTN$left (instance-order (source ?f)))
                     (monotonic-instance ?f)))))

(11) (CNG$function relational-type)
     (CNG$signature relational-type infomorphism REL$relation)
     (forall (?f (infomorphism ?f))
         (and (= (REL$source (relational-type ?f))
                 (type (source ?f)))
              (= (REL$target (relational-type ?f))
                 (type (target ?f)))
              (= (relational-type ?f)
                 ((ORD.FTN$right (type-order (target ?f)))
                     (monotonic-type ?f)))))
```

- Any functional infomorphism can be transformed into a relational infomorphism. In axiom (12) this *relational infomorphism* is defined below in terms of the instance and type relations.

```
(12) (CNG$function relational-infomorphism)
     (CNG$function fn2rel)
     (= fn2rel relational-infomorphism)
     (CNG$signature relational-infomorphism infomorphism CLS.REL$infomorphism)
     (forall (?f (infomorphism ?f))
         (and (= (CLS.REL$source (relational-infomorphism ?f))
                 (source ?f))
              (= (CLS.REL$target (relational-infomorphism ?f))
                 (target ?f))
              (= (CLS.REL$instance (relational-infomorphism ?f))
                 (relational-instance ?f))
              (= (CLS.REL$type (relational-infomorphism ?f))
                    (relational-type ?f))))
```

- It can be shown that the bond of the relational infomorphism of a functional infomorphism ***f*** is the bond associated with ***f***.

```
(forall (?f (infomorphism ?f))
    (= (CLS.REL$bond (relational-infomorphism ?f))
       (bond ?f)))
```

- The function '`composition`' operates on any two infomorphisms that are composable in the sense that the target classification of the first is equal to the source classification of the second. Composition, defined in axiom (13), produces an infomorphism, whose instance function is the composition of the instance functions of the components and whose type function is the composition of the type functions of the components.

```
(13) (CNG$function composition)
     (CNG$signature composition infomorphism infomorphism infomorphism)
     (forall (?f (infomorphism ?f) ?g (infomorphism ?g))
         (<=> (exists (?h (infomorphism ?h)) (= (composition ?f ?g) ?h))
              (= (target ?f) (source ?g))))
     (forall (?f (infomorphism ?f) ?g (infomorphism ?g))
         (=> (= (target ?f) (source ?g))
             (and (= (source (composition ?f ?g)) (source ?f))
                  (= (target (composition ?f ?g)) (target ?g))
                  (= (instance (composition ?f ?g))
                     (SET.FTN$composition (instance ?g) (instance ?f)))
                  (= (type (composition ?f ?g))
                     (SET.FTN$composition (type ?f) (type ?g))))))
```

- The function '`identity`' defined in axiom (14) associates a well-defined (identity) infomorphism with any classification, whose instance function is the identity class function on instances and whose type function is the identity class function on types.

```
(14) (CNG$function identity)
     (CNG$signature identity CLS$classification infomorphism)
     (forall (?c (CLS$classification ?c))
         (and (= (source (identity ?c)) ?c)
              (= (target (identity ?c)) ?c)
              (= (instance (identity ?c))
                 (SET.FTN$identity (CLS$instance ?c)))
              (= (type (identity ?c))
                 (SET.FTN$identity (CLS$type ?c)))))
```

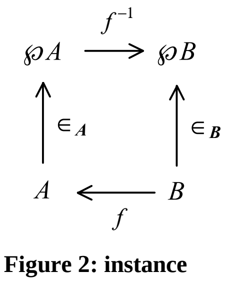
- Duality can be extended to infomorphisms. For any infomorphism $\boldsymbol{f} : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$, the *opposite* or *dual* of ***f*** is the infomorphism $\boldsymbol{f}^{\perp} : \boldsymbol{B}^{\perp} \rightleftarrows \boldsymbol{A}^{\perp}$ whose source classification is the opposite of the target classification of ***f***, whose target classification is the opposite of the source classification of ***f***, whose instance function is the type function of ***f***, whose type function is the instance function of ***f***, and whose fundamental condition is equivalent to that of ***f***:

  $typ(f)(t) \models^{\perp} i$ iff $t \models^{\perp} inst(f)(i)$.

  Axiom (15) specifies the opposite operator on infomorphisms.

```
(15) (CNG$function opposite)
     (CNG$signature opposite infomorphism infomorphism)
     (forall (?f (infomorphism ?f))
         (and (= (source (opposite ?f)) (CLS$opposite (target ?f)))
              (= (target (opposite ?f)) (CLS$opposite (source ?f)))
              (= (instance (opposite ?f)) (type ?f))
              (= (type (opposite ?f)) (instance ?f))))
```

- For any class function $f : B \rightarrow A$ the components of the *instance power infomorphism* $\wp f : \wp A \rightleftarrows \wp B$ over f are defined as follows: the source classification $\wp A = \langle A, \wp A, \in_A\rangle$ is the instance power classification over the target class A, the target classification $\wp B = \langle B, \wp B, \in_B\rangle$ is the instance power classification over the source class B, the instance function is f, and the type



$$\begin{tikzcd}
\wp A \arrow[r, "f^{-1}"] & \wp B \\
A \arrow[u, "\in_A"'] & B \arrow[l, "f"] \arrow[u, "\in_B"']
\end{tikzcd}$$

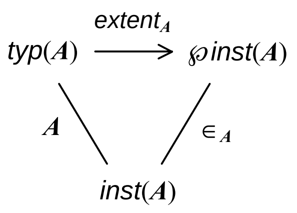
Figure 2: instance power infomorphism

function is the inverse image function $f^{-1} : \wp A \rightarrow \wp B$ from the power-class of A to the power-class of B. Note the contravariance.

```
(16) (CNG$function instance-power)
     (CNG$signature instance-power SET.FTN$function infomorphism)
     (forall (?f (SET.FTN$function ?f))
         (and (= (source (instance-power ?f)) (SET.FTN$power (SET.FTN$target ?f)))
              (= (target (instance-power ?f)) (SET.FTN$power (SET.FTN$source ?f)))
              (= (instance (instance-power ?f)) ?f)
              (= (type (instance-power ?f))
                 (SET.FTN$inverse-image ?f))))
```

o It is a standard fact in Information Flow that from any classification $\boldsymbol{A}$ there is a *canonical extent infomorphism* $\eta_{\boldsymbol{A}} : \boldsymbol{A} \rightleftarrows \wp\mathit{inst}(\boldsymbol{A})$ from $\boldsymbol{A}$ to the instance power classification $\wp\mathit{inst}(\boldsymbol{A}) = \langle \mathit{inst}(\boldsymbol{A}), \wp\mathit{inst}(\boldsymbol{A}), \in \rangle$ over the instance class. This infomorphism is the $\boldsymbol{A}$th component of a natural quasi-transformation called *eta*. The instance function is the identity function on the instance class $\mathit{inst}(\boldsymbol{A})$, and the type function is the extent function $\mathit{extent}_{\boldsymbol{A}} : \mathit{typ}(\boldsymbol{A}) \rightarrow \wp\mathit{inst}(\boldsymbol{A})$.

Axiom (17) gives the KIF definition of the extent infomorphism.



$$\begin{tikzcd}
\mathit{typ}(\boldsymbol{A}) \arrow[r, "\mathit{extent}_{\boldsymbol{A}}"] \arrow[dr, dash, "\boldsymbol{A}"'] & \wp\mathit{inst}(\boldsymbol{A}) \arrow[d, dash, "\in_{\boldsymbol{A}}"] \\
& \mathit{inst}(\boldsymbol{A})
\end{tikzcd}$$

Figure 2: η_A, the extent infomorphism

```
(17) (CNG$function eta)
     (CNG$signature eta CLS$classification infomorphism)
     (forall (?a (classification ?a))
         (and (= (source (eta ?a)) ?a)
              (= (target (eta ?a))
                 (CLS$instance-power (CLS$instance ?a)))
              (= (instance (eta ?a))
                 (SET.FTN$identity (CLS$instance ?a)))
              (= (type (eta ?a)) (CLS$extent ?a))))
```

o Let $\boldsymbol{f} = \langle \mathit{inst}(\boldsymbol{f}), \mathit{typ}(\boldsymbol{f}) \rangle : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ be any infomorphism from classification $\boldsymbol{A}$ to classification $\boldsymbol{B}$ with instance function $\mathit{inst}(\boldsymbol{f}) : \mathit{inst}(\boldsymbol{B}) \rightarrow \mathit{inst}(\boldsymbol{A})$ and type function $\mathit{typ}(\boldsymbol{f}) : \mathit{typ}(\boldsymbol{A}) \rightarrow \mathit{typ}(\boldsymbol{B})$. There is an adjoint pair

$$\mathit{adj}(\boldsymbol{f}) = \langle \mathit{left}(\mathit{adj}(\boldsymbol{f})), \mathit{right}(\mathit{adj}(\boldsymbol{f})) \rangle : \mathit{complete\text{-}lattice}(\boldsymbol{A}) \rightleftarrows \mathit{complete\text{-}lattice}(\boldsymbol{B}),$$

defined as follows.

$$\begin{aligned}
&\mathit{left}(\mathit{adj}(\boldsymbol{f}))(d) \\
&\quad = \mathit{left}(\mathit{adj}(\boldsymbol{f}))(\langle \mathit{extent}_{\boldsymbol{B}}(d), \mathit{intent}_{\boldsymbol{B}}(d) \rangle) \\
&\quad = \langle (\mathit{typ}(\boldsymbol{f})^{-1}(\mathit{intent}_{\boldsymbol{B}}(d)))', \ (\mathit{typ}(\boldsymbol{f})^{-1}(\mathit{intent}_{\boldsymbol{B}}(d)) \rangle
\end{aligned}$$

for all concepts $d \in \mathit{complete\text{-}lattice}(\boldsymbol{B})$, and

$$\begin{aligned}
&\mathit{right}(\mathit{adj}(\boldsymbol{f}))(c) \\
&\quad = \mathit{right}(\mathit{adj}(\boldsymbol{f}))(\langle \mathit{extent}_{\boldsymbol{A}}(c), \mathit{intent}_{\boldsymbol{A}}(c) \rangle) \\
&\quad = \langle (\mathit{inst}(\boldsymbol{f})^{-1}(\mathit{extent}_{\boldsymbol{A}}(c)), (\mathit{inst}(\boldsymbol{f})^{-1}(\mathit{extent}_{\boldsymbol{A}}(c)))' \rangle
\end{aligned}$$

for all concepts $c \in \mathit{complete\text{-}lattice}(\boldsymbol{A})$.

```
(18) (CNG$function adjoint-pair)
     (CNG$signature adjoint-pair infomorphism LAT.ADJ$adjoint-pair)
     (forall (?f (infomorphism ?f))
         (and (= (LAT.ADJ$source (adjoint-pair ?f))
                 (complete-lattice (target ?f)))
              (= (LAT.ADJ$target (adjoint-pair ?f))
                 (complete-lattice (source ?f)))
              (= (SET.FTN$composition
                     (ORD.FTN$function (LAT.ADJ$left (adjoint-pair ?f)))
                     (CLS.CL$intent (source ?f)))
                  (SET.FTN$composition
                     (CLS.CL$intent (target ?f))
                     (SET.FTN$inverse-image (type ?f))))
              (= (SET.FTN$composition
                     (ORD.FTN$function (LAT.ADJ$right (adjoint-pair ?f)))
                     (CLS.CL$extent (target ?f)))
                 (SET.FTN$composition
                     (CLS.CL$extent (source ?f))
                     (SET.FTN$inverse-image (instance ?f))))))
```

- Let $\boldsymbol{f} : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ be any infomorphism from classification $\boldsymbol{A}$ to classification $\boldsymbol{B}$ with instance function $\mathit{inst}(\boldsymbol{f}) : \mathit{inst}(\boldsymbol{B}) \rightarrow \mathit{inst}(\boldsymbol{A})$ and type function $\mathit{typ}(\boldsymbol{f}) : \mathit{typ}(\boldsymbol{A}) \rightarrow \mathit{typ}(\boldsymbol{B})$. There is a concept morphism

  $$\mathit{concept\text{-}morphism}(\boldsymbol{f}) = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \mathit{adj}(\boldsymbol{A}) \rangle : \mathit{concept\text{-}lattice}(\boldsymbol{A}) \rightleftarrows \mathit{concept\text{-}lattice}(\boldsymbol{B}),$$

  whose instance/type functions are the same as $\boldsymbol{f}$, and whose adjoint pair the adjoint pair of $\boldsymbol{f}$.

```
(19) (CNG$function concept-morphism)
     (CNG$signature concept-morphism CLS.INFO$infomorphism CL.MOR$concept-morphism)
     (forall (?f (CLS.INFO$infomorphism ?f))
         (and (= (CL.MOR$source (concept-morphism ?f))
                 (concept-lattice (CLS.INFO$source ?f)))
              (= (CL.MOR$target (concept-morphism ?f))
                 (concept-lattice (CLS.INFO$target ?f)))
              (= (CL.MOR$adjoint-pair (concept-morphism ?f)) (adjoint-pair ?f))
              (= (CL.MOR$instance (concept-morphism ?f)) (instance ?f))
              (= (CL.MOR$type (concept-morphism ?f)) (type ?f))))
```

Relational Infomorphisms

CLS.REL

- Classifications are also related through (relational) infomorphisms. A (*relational*) *infomorphism* $\boldsymbol{r} : \boldsymbol{A} \rightrightarrows \boldsymbol{B}$ from classification $\boldsymbol{A}$ to classification $\boldsymbol{B}$ (see Figure 5) is a pair $\boldsymbol{r} = \langle \mathit{inst}(\boldsymbol{r}), \mathit{typ}(\boldsymbol{r})\rangle$ of binary relations, a relation between instances $\mathit{inst}(\boldsymbol{r}) : \mathit{inst}(\boldsymbol{A}) \rightarrow \mathit{inst}(\boldsymbol{B})$ and a relation between types $\mathit{typ}(\boldsymbol{r}) : \mathit{typ}(\boldsymbol{A}) \rightarrow \mathit{typ}(\boldsymbol{B})$, which satisfy the fundamental property:

  $$\mathit{inst}(\boldsymbol{r}) \backslash \boldsymbol{A} \;=\; \boldsymbol{B} \,/\, \mathit{typ}(\boldsymbol{r}).$$

$$\begin{tikzcd}
\mathit{typ}(\boldsymbol{A}) \arrow[r, "\mathit{typ}(\boldsymbol{r})"] & \mathit{typ}(\boldsymbol{B}) \\
\mathit{inst}(\boldsymbol{A}) \arrow[u, "\boldsymbol{A}"'] \arrow[r, "\mathit{inst}(\boldsymbol{r})"'] & \mathit{inst}(\boldsymbol{B}) \arrow[u, "\boldsymbol{B}"'] \arrow[ul, "\mathit{bond}(\boldsymbol{f})"']
\end{tikzcd}$$

Figure 5: Relational Infomorphism

The following is a KIF representation for the elements of a relation infomorphism.

```
(1) (CNG$conglomeration infomorphism)

(2) (CNG$function source)
    (CNG$signature source infomorphism CLS$classification)

(3) (CNG$function target)
    (CNG$signature target infomorphism CLS$classification)

(4) (CNG$function instance)
    (CNG$signature instance infomorphism REL$relation)
    (forall (?r (infomorphism ?r))
        (and (= (REL$source (instance ?r)) (CLS$instance (source ?r)))
             (= (REL$target (instance ?r)) (CLS$instance (target ?r)))))

(5) (CNG$function type)
    (CNG$signature type infomorphism REL$relation)
    (forall (?r (infomorphism ?r))
        (and (= (REL$source (type ?r)) (CLS$type (source ?r)))
             (= (REL$target (type ?r)) (CLS$type (target ?r)))))
```

- Axiom (6) gives the KIF for the fundamental property of a relational infomorphism.

```
(6) (forall (?r (infomorphism ?r))
        (= (REL$left-residuation (instance ?r) (CLS$incidence (source ?r)))
           (REL$right-residuation (type ?r) (CLS$incidence (target ?r)))))
```

- For any relational infomorphism $\boldsymbol{r} : \boldsymbol{A} \rightrightarrows \boldsymbol{B}$, the common relation $\mathit{bond}(\boldsymbol{r}) = \mathit{inst}(\boldsymbol{r})\backslash\boldsymbol{A} = \boldsymbol{B}/\mathit{typ}(\boldsymbol{r}) : \mathit{inst}(\boldsymbol{A}) \rightarrow \mathit{inst}(\boldsymbol{B})$ in the fundamental property is called the *bond* of $\boldsymbol{r}$ – it bonds the instance and type relations into a unity.

```
(7) (CNG$function bond)
    (CNG$signature bond infomorphism CLS.BND$bond)
    (forall (?r (infomorphism ?r))
        (and (= (CLS.BND$source (bond ?r) (source ?r))
             (= (CLS.BND$target (bond ?r) (target ?r))
             (= (CLS.BND$classification (bond ?r)
                (REL$left-residuation (instance ?r) (CLS$incidence (source ?r))))))
```

- Given any two relational infomorphisms $\langle r_1, s_1\rangle : \boldsymbol{A} \rightrightarrows \boldsymbol{B}$ and $\langle r_2, s_2\rangle : \boldsymbol{B} \rightrightarrows \boldsymbol{C}$, which are composable in the sense the target classification of the first is the source classification of the second, there is a *composite infomorphism* $\langle r_1, s_1\rangle \circ \langle r_2, s_2\rangle = \langle r_1{\circ}r_2, s_1{\circ}s_2\rangle : \boldsymbol{A} \rightrightarrows \boldsymbol{C}$ defined by composing the type and instance relations, and whose fundamental property follows from composition and associative laws.

```
(8) (CNG$function composition)
    (CNG$signature composition infomorphism infomorphism infomorphism)
    (forall (?r (infomorphism ?r) ?s (infomorphism ?s))
        (<=> (exists (?t (infomorphism ?t)) (= (composition ?r ?s) ?t))
             (= (target ?r) (source ?s))))
    (forall (?r (infomorphism ?r) ?s (infomorphism ?s))
        (=> (= (target ?r) (source ?s))
            (and (= (source (composition ?r ?s)) (source ?r))
                 (= (target (composition ?r ?s)) (target ?s))
                 (= (instance (composition ?r ?s))
                    (REL$composition (instance ?r) (instance ?s)))
                 (= (type (composition ?r ?s))
                    (REL$composition (type ?r) (type ?s))))))
```

o Given any classification $\boldsymbol{A} = \langle inst(\boldsymbol{A}), typ(\boldsymbol{A}), \models_{\boldsymbol{A}}\rangle$, the pair of identity relations on types and instances, with the bond being $\boldsymbol{A}$, forms an *identity infomorphism* $Id_{\boldsymbol{A}} : \boldsymbol{A} \rightrightarrows \boldsymbol{A}$ (with respect to composition).

```
(9) (CNG$function identity)
    (CNG$signature identity CLS$classification infomorphism)
    (forall (?a (CLS$classification ?a))
        (and (= (source (identity ?a)) ?a)
             (= (target (identity ?a)) ?a)
             (= (instance (identity ?a)) (REL$identity (CLS$instance ?a)))
             (= (type (identity ?a)) (REL$identity (CLS$type ?a)))))
```

o For any given infomorphism $\langle r, s\rangle : \boldsymbol{A} \rightrightarrows \boldsymbol{B}$ the *dual infomorphism* $\langle r, s\rangle^{\mathrm{op}} = \langle s^{\mathrm{op}}, r^{\mathrm{op}}\rangle : \boldsymbol{B}^{\mathrm{op}} \rightrightarrows \boldsymbol{A}^{\mathrm{op}}$ is the relational infomorphism with type and instance relations switched and transposed.

```
(10) (CNG$function opposite)
     (CNG$signature opposite infomorphism infomorphism)
     (forall (?r (infomorphism ?r))
         (and (= (source (opposite ?r)) (CLS$opposite (target ?r)))
              (= (target (opposite ?r)) (CLS$opposite (source ?r)))
              (= (instance (opposite ?r)) (REL$opposite (type ?r)))
              (= (type (opposite ?r)) (REL$opposite (instance ?r)))))
```

o The fundamental property of relational infomorphisms for composition, identity and involution follow from basic properties of residuation.

Bonds

`CLS.BND`

- Classifications are also related through bonds. A *bond* $\boldsymbol{F} : \boldsymbol{A} -\!\bullet\, \boldsymbol{B}$ from classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \vDash_{\boldsymbol{A}} \rangle$ to classification $\boldsymbol{B} = \langle \mathit{inst}(\boldsymbol{B}), \mathit{typ}(\boldsymbol{B}), \vDash_{\boldsymbol{B}} \rangle$ (see Figure 6) is a classification $\mathit{cls}(\boldsymbol{F}) = \langle \mathit{inst}(\boldsymbol{B}), \mathit{typ}(\boldsymbol{A}), \vDash_{\boldsymbol{F}} \rangle$ sharing types with $\boldsymbol{A}$ and instances with $\boldsymbol{B}$, that is compatible with $\boldsymbol{A}$ and $\boldsymbol{B}$ in the sense of closure: type sets $\{j\boldsymbol{F} \mid j \in \mathit{inst}(\boldsymbol{B})\}$ are intents of $\boldsymbol{A}$ and instance sets $\{\boldsymbol{F}t \mid t \in \mathit{typ}(\boldsymbol{B})\}$ are extents of $\boldsymbol{B}$. Closure can be expressed relationally (in terms of residuation) as the following fundamental properties.

$$\begin{tikzcd}
\mathit{typ}(\boldsymbol{A}) \arrow[r, "\boldsymbol{F}\backslash\boldsymbol{B}"] & \mathit{typ}(\boldsymbol{B}) \\
\mathit{inst}(\boldsymbol{A}) \arrow[u, "\boldsymbol{A}"'] \arrow[r, "\boldsymbol{A}/\boldsymbol{F}"'] & \mathit{inst}(\boldsymbol{B}) \arrow[u, "\boldsymbol{B}"'] \arrow[ul, "\boldsymbol{F}"']
\end{tikzcd}$$

Figure 6: Bond

$(\boldsymbol{A}/\boldsymbol{F})\backslash\boldsymbol{A} = \boldsymbol{F}$ and $\boldsymbol{B}/(\boldsymbol{F}\backslash\boldsymbol{B}) = \boldsymbol{F}$.

The first expression says that $\langle \boldsymbol{A}/\boldsymbol{F}, \boldsymbol{F} \rangle$ is an $\mathit{inst}(\boldsymbol{B})$-indexed collective $\boldsymbol{A}$-concept (or $\boldsymbol{F}$ is a collective $\boldsymbol{A}$-intent), and the second expression says that $\langle \boldsymbol{F}, \boldsymbol{F}\backslash\boldsymbol{B} \rangle$ is an $\mathit{typ}(\boldsymbol{A})$-indexed collective $\boldsymbol{B}$-concept (or that $\boldsymbol{F}$ is a collective $\boldsymbol{B}$-extent).

```
(1) (CNG$conglomeration bond)

(2) (CNG$function source)
    (CNG$signature source bond CLS$classification)

(3) (CNG$function target)
    (CNG$signature target bond CLS$classification)

(4) (CNG$function classification)
    (CNG$signature classification bond CLS$classification)
    (forall (?b (bond ?b))
        (and (= (CLS$instance (classification ?b)) (CLS$instance (target ?b)))
             (= (CLS$type (classification ?b)) (CLS$type (source ?b)))))
```

- Axiom (5) gives the KIF for the closure properties required of a bond.

```
(5) (forall (?b (bond ?b))
        (and (= (REL$left-residuation
                    (REL$right-residuation (classification ?b) (source ?b))
                    (source ?b))
                (classification ?b))
             (= (REL$right-residuation
                    (REL$left-residuation (classification ?b) (target ?b))
                    (target ?b))
                (classification ?b))))
```

- (The classification relation of) a bond is order-closed on left and right:

  $j' \leq_B j, jFt$ imply $j'Ft$, and $jFt, t \leq_A t'$ imply jFt'. Or,

  $j' \leq_B j$ implies $j'F \supseteq jF$, and $t \leq_A t'$ implies $Ft \subseteq Ft'$.

  The '`bimodule`' function defined in axiom (6) realizes these properties.

```
(6) (CNG$function bimodule)
    (CNG$signature bimodule bond ORD$bimodule)
    (forall (?b (bond ?b))
        (and (= (ORD$source (bimodule ?b))
                (CLS.CL$instance-order (target ?b)))
             (= (ORD$target (bimodule ?b))
                (CLS.CL$type-order (source ?b)))
             (= (ORD$relation (bimodule ?b))
                (classification ?b))))
```

- For any bond $\boldsymbol{F} : \boldsymbol{A} -\!\bullet\, \boldsymbol{B}$, the residuations in the fundamental closure property form a relation infomorphism $\mathit{info}(\boldsymbol{F}) = \langle (\boldsymbol{A}/\boldsymbol{F}), (\boldsymbol{F}\backslash\boldsymbol{B}) \rangle : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ from classification $\boldsymbol{A}$ to classification $\boldsymbol{B}$.

```
(7) (CNG$function infomorphism)
    (CNG$signature infomorphism bond CLS.REL$infomorphism)
```

```
        (forall (?b (bond ?b))
            (and (= (CLS.REL$source (infomorphism ?b)) (source ?b))
                 (= (CLS.REL$target (infomorphism ?b)) (target ?b))
                 (= (CLS.REL$instance (infomorphism ?b))
                    (REL$right-residuation (classification ?b) (source ?b)))
                 (= (CLS.REL$type (infomorphism ?b))
                    (REL$left-residuation (classification ?b) (target ?b)))))
```

- Moreover, the fundamental closure property implies that the bond of this relational infomorphism is the original bond.

```
        (forall (?b (bond ?b))
            (= (CLS.REL$bond (infomorphism ?b)) ?b))
```

- Associated with any bond is a morphism (*adjoint pair*) of the complete lattices of its source and target classifications.
  - By the first fundamental property, the two classifications $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$ and $\boldsymbol{F} = \langle \mathit{inst}(\boldsymbol{B}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{F}} \rangle$ have the same class of types, and the classification (binary relation) $\boldsymbol{F}$ is *type-closed* $(\boldsymbol{A}/\boldsymbol{F})\backslash\boldsymbol{A} = \boldsymbol{F}$ with respect to $\boldsymbol{A}$. Hence, there is an associated *coreflection* (adjoint pair) $\mathit{corefl}_{\boldsymbol{A},\boldsymbol{F}} = \langle \partial_0, \tilde{\partial}_0 \rangle : \mathit{latt}(\boldsymbol{F}) \rightleftarrows \mathit{latt}(\boldsymbol{A})$ between their complete lattices, where $\tilde{\partial}_0 : \mathit{latt}(\boldsymbol{A}) \rightarrow \mathit{latt}(\boldsymbol{F})$ is right adjoint right inverse (rari) to $\partial_0 : \mathit{latt}(\boldsymbol{F}) \rightarrow \mathit{latt}(\boldsymbol{A})$.
  - By the second fundamental property, the two classifications $\boldsymbol{F} = \langle \mathit{inst}(\boldsymbol{B}), \mathit{typ}(\boldsymbol{A}), \models_{\boldsymbol{F}} \rangle$ and $\boldsymbol{B} = \langle \mathit{inst}(\boldsymbol{B}), \mathit{typ}(\boldsymbol{B}), \models_{\boldsymbol{B}} \rangle$ have the same class of instances, and the classification (binary relation) $\boldsymbol{F}$ is *instance-closed* $\boldsymbol{B}/(\boldsymbol{F}\backslash\boldsymbol{B}) = \boldsymbol{F}$ with respect to $\boldsymbol{B}$. Hence, there is an associated *reflection* (adjoint pair) $\mathit{refl}_{\boldsymbol{F},\boldsymbol{B}} = \langle \tilde{\partial}_1, \partial_1 \rangle : \mathit{latt}(\boldsymbol{B}) \rightleftarrows \mathit{latt}(\boldsymbol{F})$ between their complete lattices, where $\tilde{\partial}_1 : \mathit{latt}(\boldsymbol{B}) \rightarrow \mathit{latt}(\boldsymbol{F})$ is left adjoint right inverse (lari) to $\partial_1 : \mathit{latt}(\boldsymbol{F}) \rightarrow \mathit{latt}(\boldsymbol{B})$.

  Define the associated adjoint pair as the composition of the reflection followed by the coreflection.

```
    (8) (CNG$function adjoint-pair)
        (CNG$signature adjoint-pair bond LAT.ADJ$adjoint-pair)
        (forall (?b (bond ?b))
            (and (= (LAT.ADJ$source (adjoint-pair ?b))
                    (CLS.CL$complete-lattice (target ?b)))
                 (= (LAT.ADJ$target (adjoint-pair ?b))
                    (CLS.CL$complete-lattice (source ?b)))
                 (= (LAT.ADJ$underlying (adjoint-pair ?b))
                    (LAT.ADJ$composition
                        (CLS.CL$reflection (target ?b) (classification ?b))
                        (CLS.CL$coreflection (classification ?b) (source ?b))))))
```

- Two bonds are composable when the target classification of the first is the source classification of the second. For any two composable bonds $\boldsymbol{F} : \boldsymbol{A} \multimap \boldsymbol{B}$ and $\boldsymbol{G} : \boldsymbol{B} \multimap \boldsymbol{C}$, the *composition* is the bond $\boldsymbol{F} \circ \boldsymbol{G} \triangleq (\boldsymbol{B}/\boldsymbol{G})\backslash\boldsymbol{F} : \boldsymbol{A} \multimap \boldsymbol{C}$ defined using left and right residuation. Since both $\boldsymbol{F}$ and $\boldsymbol{G}$ being bonds are closed with respect to $\boldsymbol{B}$, an equivalent expression for the composition is $\boldsymbol{F} \circ \boldsymbol{G} \triangleq \boldsymbol{G}/(\boldsymbol{F}\backslash\boldsymbol{B}) : \boldsymbol{A} \multimap \boldsymbol{C}$. Pointwise, the composition is $\boldsymbol{F} \circ \boldsymbol{G} = \{(c, \alpha) \mid \boldsymbol{F}\alpha \supseteq (c\boldsymbol{G})^{\boldsymbol{B}}\}$. To check closure, $(\boldsymbol{A}/(\boldsymbol{F}\circ\boldsymbol{G}))\backslash\boldsymbol{A} = (\boldsymbol{A}/((\boldsymbol{B}/\boldsymbol{G})\backslash\boldsymbol{F}))\backslash\boldsymbol{A} = (\boldsymbol{B}/\boldsymbol{G})\backslash\boldsymbol{F}$, since $\boldsymbol{F}$ being a collective $\boldsymbol{A}$-intent means that $(\boldsymbol{B}/\boldsymbol{G})\backslash\boldsymbol{F}$ is also a collective $\boldsymbol{A}$-intent.
- Composition, defined in axiom (8), produces a bond constructed according to the expressions above.

```
    (8) (CNG$function composition)
        (CNG$signature composition bond bond bond)
        (forall (?f (bond ?f) ?g (bond ?g))
            (<=> (exists (?h (bond ?h)) (= (composition ?f ?g) ?h))
                 (= (target ?f) (source ?g))))
        (forall (?f (bond ?f) ?g (bond ?g))
            (=> (= (target ?f) (source ?g))
                (and (= (source (composition ?f ?g)) (source ?f))
                     (= (target (composition ?f ?g)) (target ?g))
                     (= (classification (composition ?f ?g))
                        (REL$left-residuation (REL$right-residuation ?g (source ?g)) ?f)))))
```

- With respect to bond composition, the *identity* bond at any classification $\boldsymbol{A}$ is $\boldsymbol{A}$.

```
(9) (CNG$function identity)
    (CNG$signature identity CLS$classification bond)
    (forall (?a (CLS$classification ?a))
        (and (= (source (identity ?a)) ?a)
             (= (target (identity ?a)) ?a)
             (= (classification (identity ?a)) ?a)))
```

- For any given bond $\boldsymbol{F} : \boldsymbol{A} \multimap \boldsymbol{B}$ the *dual bond* $\boldsymbol{F}^{\mathrm{op}} : \boldsymbol{B}^{\mathrm{op}} \multimap \boldsymbol{A}^{\mathrm{op}}$ is the bond with source and target classifications switched, and source, target and classification relations transposed.

```
(10) (CNG$function opposite)
     (CNG$signature opposite bond bond)
     (forall (?f (bond ?f))
         (and (= (source (opposite ?f)) (CLS$opposite (target ?f)))
              (= (target (opposite ?f)) (CLS$opposite (source ?f)))
              (= (classification (opposite ?f)) (REL$opposite (classification ?f)))))
```

The fundamental property of bonds for composition, identity and involution follow from basic properties of residuation.

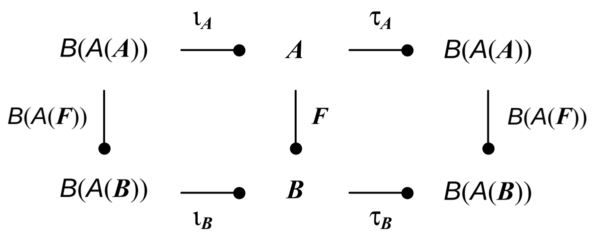
- The basic theorem of Formal Concept Analysis can be framed in terms of two fundamental bonds (relational infomorphisms) between any classification and its associated concept lattice.
  - For any classification $\boldsymbol{A}$ the instance embedding relation is a bond $\iota_{\boldsymbol{A}} : \mathsf{L}(\boldsymbol{A}) \multimap \boldsymbol{A}$ named *iota*, whose source classification is the (classification of the) concept lattice $\mathsf{L}(\boldsymbol{A})$ and whose target classification is $\boldsymbol{A}$. The classification relation of the instance embedding bond is the instance embedding relation. The pair $\langle \mathsf{L}(\boldsymbol{A})/\iota_{\boldsymbol{A}}, \tau_{\boldsymbol{A}} \rangle : \mathsf{L}(\boldsymbol{A}) \rightrightarrows \boldsymbol{A}$ is a relational infomorphism whose bond is the iota bond.
  - For any classification $\boldsymbol{A}$ the type embedding relation is a bond $\tau_{\boldsymbol{A}} : \boldsymbol{A} \multimap \mathsf{L}(\boldsymbol{A})$ named *tau*, whose source classification is $\boldsymbol{A}$ and whose target classification is the (classification of the) concept lattice $\mathsf{L}(\boldsymbol{A})$. The classification relation of the type embedding bond is the type embedding relation. The pair $\langle \iota_{\boldsymbol{A}}, \tau_{\boldsymbol{A}} \backslash \mathsf{L}(\boldsymbol{A}) \rangle : \boldsymbol{A} \rightrightarrows \mathsf{L}(\boldsymbol{A})$ is a relational infomorphism whose bond is the type embedding relation.

```
(11) (CNG$function iota)
     (CNG$signature iota CLS$classification bond)
     (forall (?a (CLS$classification ?a))
         (and (= (source (iota ?a)) (CL$classification (CLS.CL$concept-lattice ?a)))
              (= (type (iota ?a)) ?a)
              (= (classification (iota ?a)) CLS.CL$iota)))

(12) (CNG$function tau)
     (CNG$signature tau CLS$classification bond)
     (forall (?a (CLS$classification ?a))
         (and (= (source (iota ?a)) ?a)
              (= (type (iota ?a)) (CL$classification (CLS.CL$concept-lattice ?a)))
              (= (classification (tau ?a)) CLS.CL$tau)))
```

- The instance and type embedding bonds are inverse to each other: $\iota_{\boldsymbol{A}} \circ \tau_{\boldsymbol{A}} = Id_{\mathsf{L}(\boldsymbol{A})}$ and $\tau_{\boldsymbol{A}} \circ \iota_{\boldsymbol{A}} = Id_{\boldsymbol{A}}$.

```
     (forall (?a (CLS$classification ?a))
         (and (= (composition (iota ?a) (tau ?a))
                 (identity (CL$classification (CLS.CL$concept-lattice ?a))))
              (= (composition (tau ?a) (iota ?a))
                 (identity ?a))))
```

- The functor composition $\mathsf{A} \circ \mathsf{B}$ is naturally isomorphic to the identity functor $Id_{\mathbf{Bond}}$. To see this, let $\boldsymbol{A}$ be a classification with associated concept lattice $\mathit{lat}(\boldsymbol{A})$. The comments above demonstrate the bond isomorphism $\boldsymbol{A} \cong \mathit{cls}(\mathit{lat}(\boldsymbol{A}))$. Let $\boldsymbol{F} : \boldsymbol{A} \multimap \boldsymbol{B}$ be a bond between classifications $\boldsymbol{A}$ and $\boldsymbol{B}$ with associated complete adjoint



$$\begin{tikzcd}
\mathsf{B}(\mathsf{A}(\boldsymbol{A})) \arrow[r, "\iota_{\boldsymbol{A}}"] \arrow[d, "\mathsf{B}(\mathsf{A}(\boldsymbol{F}))"'] & \boldsymbol{A} \arrow[r, "\tau_{\boldsymbol{A}}"] \arrow[d, "\boldsymbol{F}"] & \mathsf{B}(\mathsf{A}(\boldsymbol{A})) \arrow[d, "\mathsf{B}(\mathsf{A}(\boldsymbol{F}))"] \\
\mathsf{B}(\mathsf{A}(\boldsymbol{B})) \arrow[r, "\iota_{\boldsymbol{B}}"'] & \boldsymbol{B} \arrow[r, "\tau_{\boldsymbol{B}}"'] & \mathsf{B}(\mathsf{A}(\boldsymbol{B}))
\end{tikzcd}$$

Diagram 2: Natural Isomorphism

$\mathit{adj}(\boldsymbol{F}) : \mathit{lat}(\boldsymbol{A}) \rightleftarrows \mathit{lat}(\boldsymbol{B})$. The classification relation of the bond $\mathit{bnd}(\mathit{adj}(\boldsymbol{F})) : \mathit{cls}(\mathit{lat}(\boldsymbol{A})) \multimap \mathit{cls}(\mathit{lat}(\boldsymbol{B}))$ contains a conceptual pair $(\boldsymbol{a}, \boldsymbol{b})$ of the form $\boldsymbol{a} = (A, \Gamma)$ and $\boldsymbol{b} = (B, \Delta)$ iff $B \circ \Gamma \subseteq \boldsymbol{F}$, where $B \circ \Gamma = B{\times}\Gamma$ a Cartesian product or rectangle, iff $B \subseteq \Gamma^{\boldsymbol{F}}$ iff $\Gamma \subseteq B^{\boldsymbol{F}}$. So, $(\iota_{\boldsymbol{A}} \circ \boldsymbol{F}) \circ \tau_{\boldsymbol{B}} = (\boldsymbol{F}/(\iota_{\boldsymbol{A}} \backslash \boldsymbol{A})) \circ \tau_{\boldsymbol{B}} = (\boldsymbol{F}/\tau_{\boldsymbol{A}}) \circ \tau_{\boldsymbol{B}} = (\boldsymbol{B}/\tau_{\boldsymbol{B}}) \backslash (\boldsymbol{F}/\tau_{\boldsymbol{A}}) = \iota_{\boldsymbol{B}} \backslash (\boldsymbol{F}/\tau_{\boldsymbol{A}}) = \mathit{bnd}(\mathit{adj}(\boldsymbol{F}))$, by bond composition and properties of the instance and type relations. Hence, $\mathit{bnd}(\mathit{adj}(\boldsymbol{F})) \circ \iota_{\boldsymbol{B}} = \iota_{\boldsymbol{A}} \circ \boldsymbol{F}$. This proves the required naturality condition.

```
(forall (?f (bond ?f))
    (= (composition (CL.MOR$bond (adjoint-pair ?f)) (iota (target ?f)))
       (composition (iota (source ?f)) ?f)))
```

Bonding Pairs

CLS.BNDPR

$$\begin{tikzcd}
L(A) \arrow[r, "\iota_A", -{Circle}] & A \arrow[d, "F", -{Circle}] \\
L(B) \arrow[u, "\tau_B", -{Circle}] & B \arrow[l, "G", -{Circle}]
\end{tikzcd}$$

A *complete* (*lattice*) *homomorphism* $\psi : \boldsymbol{L} \rightarrow \boldsymbol{K}$ between complete lattices $\boldsymbol{L}$ and $\boldsymbol{K}$ is a (monotonic) function that preserves both joins and meets. Being meet-preserving, ψ has a left adjoint $\varphi : \boldsymbol{K} \rightarrow \boldsymbol{L}$, and being join-preserving ψ has a right adjoint $\theta : \boldsymbol{K} \rightarrow \boldsymbol{L}$. Therefore, a complete homomorphism is the middle monotonic function in two adjunctions $\varphi \dashv \psi \dashv \theta$. Let Complete Lattice denote the quasi-category of complete lattices and complete homomorphisms.

The bond equivalent to a complete homomorphism would seem to be given by two bonds $\boldsymbol{F} : \boldsymbol{A} -\!\bullet\, \boldsymbol{B}$ and $\boldsymbol{G} : \boldsymbol{B} -\!\bullet\, \boldsymbol{A}$ where the right adjoint $\psi_{\boldsymbol{F}} : \boldsymbol{L}(\boldsymbol{A}) \rightarrow \boldsymbol{L}(\boldsymbol{B})$ of the complete adjoint $\boldsymbol{A}(\boldsymbol{F}) = \langle \varphi_{\boldsymbol{F}}, \psi_{\boldsymbol{F}} \rangle : \boldsymbol{L}(\boldsymbol{B}) \rightleftarrows \boldsymbol{L}(\boldsymbol{A})$ of one bond (say $\boldsymbol{F}$, without loss of generality) is equal to the left adjoint $\varphi_{\boldsymbol{G}} : \boldsymbol{L}(\boldsymbol{A}) \rightarrow \boldsymbol{L}(\boldsymbol{B})$ of the complete adjoint $\boldsymbol{A}(\boldsymbol{G}) = \langle \varphi_{\boldsymbol{G}}, \psi_{\boldsymbol{G}} \rangle : \boldsymbol{L}(\boldsymbol{A}) \rightleftarrows \boldsymbol{L}(\boldsymbol{B})$ of the other bond $\boldsymbol{G}$ with the resultant adjunctions, $\varphi_{\boldsymbol{F}} \dashv \psi_{\boldsymbol{F}} = \varphi_{\boldsymbol{G}} \dashv \psi_{\boldsymbol{G}}$, where the middle adjoint is the complete homomorphism. This is indeed the case, but the question is, what constraint should be placed on $\boldsymbol{F}$ and $\boldsymbol{G}$ in order for this to hold. The simple answer is to identify the actions of the two monotonic functions $\psi_{\boldsymbol{G}}$ and $\varphi_{\boldsymbol{F}}$. Let $(A, \Gamma) \in \boldsymbol{L}(\boldsymbol{B})$ be any formal concept in $\boldsymbol{L}(\boldsymbol{A})$. The action of the left adjoint $\varphi_{\boldsymbol{G}}$ on this concept is $(A, \Gamma) \mapsto (A^{\boldsymbol{GB}}, A^{\boldsymbol{G}})$, whereas the action of the right adjoint $\psi_{\boldsymbol{F}}$ on this concept is $(A, \Gamma) \mapsto (\Gamma^{\boldsymbol{F}}, \Gamma^{\boldsymbol{FB}})$. So the appropriate pointwise constraints are: $A^{\boldsymbol{GB}} = \Gamma^{\boldsymbol{F}}$ and $\Gamma^{\boldsymbol{FB}} = A^{\boldsymbol{G}}$, for every concept $(A, \Gamma) \in \boldsymbol{L}(\boldsymbol{A})$. We now give these pointwise constraints a relational formulation.

$$\begin{tikzcd}
& typ(\boldsymbol{A}) & typ(\boldsymbol{B}) \\
\boldsymbol{L}(\boldsymbol{A}) \arrow[ur, "\tau_{\boldsymbol{A}}"] \arrow[urr, "\iota_{\boldsymbol{A}}\backslash\boldsymbol{G}"] & & \\
& inst(\boldsymbol{A}) \arrow[uu] \arrow[ul, "\iota_{\boldsymbol{A}}"] \arrow[uur, "\boldsymbol{G}"'] & inst(\boldsymbol{B}) \arrow[uu] \arrow[uul, "\boldsymbol{F}"'] \arrow[ull, "\boldsymbol{F}/\tau_{\boldsymbol{A}}"]
\end{tikzcd}$$

Figure 7: Bonding Pair

○ A *bonding pair* $\langle \boldsymbol{F}, \boldsymbol{G} \rangle : \boldsymbol{A} -\!\blacklozenge\, \boldsymbol{B}$ between two classifications $\boldsymbol{A}$ and $\boldsymbol{B}$ is a contravariant pair of bonds, a bond $\boldsymbol{F} : \boldsymbol{A} -\!\bullet\, \boldsymbol{B}$ in the *forward* direction and a bond $\boldsymbol{G} : \boldsymbol{B} -\!\bullet\, \boldsymbol{A}$ in the *reverse* direction, satisfying the following *pairing constraints*:

$$\boldsymbol{F}/\tau_{\boldsymbol{A}} = \boldsymbol{B}/(\iota_{\boldsymbol{A}}\backslash\boldsymbol{G}) \text{ and } \iota_{\boldsymbol{A}}\backslash\boldsymbol{G} = (\boldsymbol{F}/\tau_{\boldsymbol{A}})\backslash\boldsymbol{B},$$

which state that $(\boldsymbol{F}/\tau_{\boldsymbol{A}}, \iota_{\boldsymbol{A}}\backslash\boldsymbol{G}) = (\iota_{\boldsymbol{A}} \circ \boldsymbol{F}, \boldsymbol{G} \circ \tau_{\boldsymbol{B}})$ is an $\boldsymbol{L}(\boldsymbol{A})$-indexed collective $\boldsymbol{B}$-concept. The definitions of the relations $\boldsymbol{F}/\tau_{\boldsymbol{A}}$ and $\iota_{\boldsymbol{A}}\backslash\boldsymbol{G}$ are given as follows: $\boldsymbol{F}/\tau_{\boldsymbol{A}} = \{(b, \boldsymbol{a}) \mid int(\boldsymbol{a}) \subseteq b\boldsymbol{F}\} = \{(b, \boldsymbol{a}) \mid ((b\boldsymbol{F})^{\boldsymbol{A}}, b\boldsymbol{F}) \leq_{\boldsymbol{B}} \boldsymbol{a}\}$ and $\iota_{\boldsymbol{A}}\backslash\boldsymbol{G} = \{(\boldsymbol{a}, \beta) \mid ext(\boldsymbol{a}) \subseteq \boldsymbol{G}\beta\} = \{(\boldsymbol{a}, \beta) \mid \boldsymbol{a} \leq_{\boldsymbol{B}} (\boldsymbol{G}\beta, (\boldsymbol{G}\beta)^{\boldsymbol{A}})\}$. Any concept $\boldsymbol{a} = (A, \Gamma) \in \boldsymbol{L}(\boldsymbol{A})$ is mapped by the relations as: $(\boldsymbol{F}/\tau_{\boldsymbol{A}})((A, \Gamma)) = \{b \mid \Gamma \subseteq b\boldsymbol{F}\} = \Gamma^{\boldsymbol{F}}$ and $(\iota_{\boldsymbol{A}}\backslash\boldsymbol{G})((A, \Gamma)) = \{\beta \mid A \subseteq \boldsymbol{G}\beta\} = A^{\boldsymbol{G}}$. Hence, pointwise the constraints are $\Gamma^{\boldsymbol{F}} = B^{\boldsymbol{GB}}$ and $A^{\boldsymbol{G}} = \Gamma^{\boldsymbol{FB}}$. These are the pointwise constraints discussed above.

```
(1) (CNG$conglomeration bonding-pair)

(2) (CNG$function source)
    (CNG$signature source bonding-pair CLS$classification)

(3) (CNG$function target)
    (CNG$signature target bonding-pair CLS$classification)

(4) (CNG$function forward)
    (CNG$signature forward bonding-pair bond)
    (forall (?bp (bonding-pair ?bp))
        (and (= (CLS.BND$source (forward ?bp)) (source ?bp))
             (= (CLS.BND$target (forward ?bp)) (target ?bp))))

(5) (CNG$function reverse)
    (CNG$signature reverse bonding-pair bond)
    (forall (?bp (bonding-pair ?bp))
        (and (= (CLS.BND$source (reverse ?bp)) (target ?bp))
             (= (CLS.BND$target (reverse ?bp)) (source ?bp))))

    (forall (?bp (bonding-pair ?bp))
        (and (= (REL$right-residuation (CLS.CL$tau (source ?bp)) (forward ?bp))
                (REL$right-residuation
                    (REL$left-residuation (CLS.CL$iota (source ?bp)) (reverse ?bp))
                    (target ?bp)))
             (= (REL$left-residuation (CLS.CL$iota (source ?bp)) (reverse ?bp))
                (REL$left-residuation
                    (REL$right-residuation (CLS.CL$tau (source ?bp)) (forward ?bp))
                    (target ?bp)))
```

- The pointwise constraints can be lifted to a collective setting – any bonding pair $\langle \boldsymbol{F}, \boldsymbol{G} \rangle : \boldsymbol{A} \multimap \boldsymbol{B}$ preserves collective concepts: for any A-indexed collective $\boldsymbol{A}$-concept (X, Y), $X \backslash \boldsymbol{A} = Y$ and $\boldsymbol{A}/Y = X$, the *conceptual image* $(\boldsymbol{F}/Y, X \backslash \boldsymbol{G})$ is an A-indexed collective $\boldsymbol{B}$-concept, since $\boldsymbol{B}/(X \backslash \boldsymbol{G}) = \boldsymbol{F}/Y$ and $(\boldsymbol{F}/Y) \backslash \boldsymbol{B} = X \backslash \boldsymbol{G}$. An important special case is the $L(\boldsymbol{A})$-indexed collective $\boldsymbol{A}$-concept $(\iota_{\boldsymbol{A}}, \tau_{\boldsymbol{A}})$. To state that the $\langle \boldsymbol{F}, \boldsymbol{G} \rangle$-image $(\boldsymbol{F}/\tau_{\boldsymbol{A}}, \iota_{\boldsymbol{A}} \backslash \boldsymbol{F})$ is an $L(\boldsymbol{A})$-indexed collective $\boldsymbol{B}$-concept, is to assert the pairing constraints $\boldsymbol{F}/\tau_{\boldsymbol{A}} = \boldsymbol{B}/(\iota_{\boldsymbol{A}} \backslash \boldsymbol{G})$ and $\iota_{\boldsymbol{A}} \backslash \boldsymbol{G} = (\boldsymbol{F}/\tau_{\boldsymbol{B}}) \backslash \boldsymbol{B}$. So, the concise definition in terms of pairing constraints, the original pointwise definition above, and the assertion that $\langle \boldsymbol{F}, \boldsymbol{G} \rangle$ preserves all collective concepts, are equivalent versions of the notion of a bonding pair.

```
(6) (CNG$function conceptual-image)
    (CNG$signature conceptual-image bonding-pair CNG$function)
    (forall (?bp (bonding-pair ?bp))
        (and (CNG$signature (conceptual-image ?bp)
                 (CLS.FIB$concept (source ?bp)) (CLS.FIB$concept (target ?bp)))
             (forall (?c ((CLS.FIB$concept (source ?bp)) ?c))
                 (and (= ((CLS.FIB$index (target ?bp)) ((conceptual-image ?bp) ?c))
                         ((CLS.FIB$index (source ?bp)) ?c))
                      (= ((CLS.FIB$extent (target ?bp)) ((conceptual-image ?bp) ?c))
                         ((CLS.FIB$right-derivation
                             (CLS.BND$classification (direct ?bp)))
                             ((CLS.FIB$intent (source ?bp)) ?c)))
                      (= ((CLS.FIB$intent (target ?bp)) ((conceptual-image ?bp) ?c))
                         ((CLS.FIB$left-derivation
                             (CLS.BND$classification (reverse ?bp)))
                             ((CLS.FIB$extent (source ?bp)) ?c)))))))
```

- Let $\langle \boldsymbol{F}, \boldsymbol{G} \rangle : \boldsymbol{A} \multimap \boldsymbol{B}$ and $\langle \boldsymbol{M}, \boldsymbol{N} \rangle : \boldsymbol{B} \multimap \boldsymbol{C}$ be two bonding pairs. Define the bonding pair composition $\langle \boldsymbol{F}, \boldsymbol{G} \rangle \circ \langle \boldsymbol{M}, \boldsymbol{N} \rangle \triangleq \langle \boldsymbol{F} \circ \boldsymbol{M}, \boldsymbol{N} \circ \boldsymbol{G} \rangle : \boldsymbol{A} \multimap \boldsymbol{C}$ in terms of bond composition.

```
(7) (CNG$function composition)
    (CNG$signature composition bonding-pair bonding-pair bonding-pair)
    (forall (?bp1 (bonding-pair ?bp1) ?bp2 (bonding-pair ?bp2))
        (<=> (exists (?bp (bonding-pair ?bp)) (= (composition ?bp1 ?bp2) ?bp))
             (= (target ?bp1) (source ?bp2))))
    (forall (?bp1 (bonding-pair ?bp1) ?bp2 (bonding-pair ?bp2))
        (=> (= (target ?bp1) (source ?bp2))
            (and (= (source (composition ?bp1 ?bp2)) (source ?bp1))
                 (= (target (composition ?bp1 ?bp2)) (target ?bp2))
                 (= (forward (composition ?bp1 ?bp2))
                    (CLS.BND$composition (forward ?bp1) (forward ?bp2)))
                 (= (reverse (composition ?bp1 ?bp2))
                    (CLS.BND$composition (reverse ?bp2) (reverse ?bp1))))))
```

- For any classification $\boldsymbol{A}$, define the bonding pair identity $\langle Id_{\boldsymbol{A}}, Id_{\boldsymbol{A}} \rangle : \boldsymbol{A} \multimap \boldsymbol{A}$ in terms of bond identity.

```
(8) (CNG$function identity)
    (CNG$signature identity CLS$classification bonding-pair)
    (forall (?a (CLS$classification ?a))
        (and (= (source (identity ?a)) ?a)
             (= (target (identity ?a)) ?a)
             (= (forward (identity ?a)) (CLS.BND$identity ?a))
             (= (reverse (identity ?a)) (CLS.BND$identity ?a))))
```

- For any classification $\boldsymbol{A}$ the type and instance embedding relations form bonding pairs in two different ways, $\langle \tau_{\boldsymbol{A}}, \iota_{\boldsymbol{A}} \rangle : \boldsymbol{A} \multimap L(\boldsymbol{A}) = A^2 \circ B^2(\boldsymbol{A})$ and $\langle \iota_{\boldsymbol{A}}, \tau_{\boldsymbol{A}} \rangle : A^2 \circ B^2(\boldsymbol{A}) = L(\boldsymbol{A}) \multimap \boldsymbol{A}$.

```
(9) (CNG$function tau-iota)
    (CNG$signature tau-iota CLS$classification bonding-pair)
    (forall (?a (CLS$classification ?a))
        (and (= (source (tau-iota ?a)) ?a)
                    (= (target (tau-iota ?a))
                       (CL$classification (CLS.CL$concept-lattice ?a)))
                    (= (forward (tau-iota ?a)) (CLS.BND$tau ?a))
                    (= (reverse (tau-iota ?a)) (CLS.BND$iota ?a))))

(10) (CNG$function iota-tau)
     (CNG$signature iota-tau CLS$classification bonding-pair)
      (forall (?a (CLS$classification ?a))
          (and (= (source (iota-tau ?a))
                  (CL$classification (CLS.CL$concept-lattice ?a)))
               (= (target (iota-tau ?a)) ?a)
               (= (forward (iota-tau ?a)) (CLS.BND$iota ?a))
               (= (reverse (iota-tau ?a)) (CLS.BND$tau ?a))))
```

- For any classification $\boldsymbol{A}$ the two bonding pairs, $\langle \tau_{\boldsymbol{A}}, \iota_{\boldsymbol{A}} \rangle : \boldsymbol{A} \multimap \mathsf{L}(\boldsymbol{A}) = \mathsf{A}^2{\circ}\mathsf{B}^2(\boldsymbol{A})$ and $\langle \iota_{\boldsymbol{A}}, \tau_{\boldsymbol{A}} \rangle : \mathsf{A}^2{\circ}\mathsf{B}^2(\boldsymbol{A}) = \mathsf{L}(\boldsymbol{A}) \multimap \boldsymbol{A}$, are inverse to each other: $\langle \tau_{\boldsymbol{A}}, \iota_{\boldsymbol{A}} \rangle \circ \langle \iota_{\boldsymbol{A}}, \tau_{\boldsymbol{A}} \rangle = Id_{\boldsymbol{A}}$ and $\langle \iota_{\boldsymbol{A}}, \tau_{\boldsymbol{A}} \rangle \circ \langle \tau_{\boldsymbol{A}}, \iota_{\boldsymbol{A}} \rangle = Id_{\mathsf{L}(\boldsymbol{A})}$. Therefore, each classification is isomorphic in the quasi-category Bonding Pair to its concept lattice: $\boldsymbol{A} \cong \mathsf{L}(\boldsymbol{A}) = \mathsf{A}^2{\circ}\mathsf{B}^2(\boldsymbol{A})$.

```
     (forall (?a (CLS$classification ?a))
         (and (= (composition (tau-iota ?a) (iota-tau ?a))
                 (identity ?a))
              (= (composition (iota-tau ?a) (tau-iota ?a))
                 (identity (CL$classification (CLS.CL$concept-lattice ?a))))))
```

- The functor composition $\mathsf{A}^2 \circ \mathsf{B}^2$ is naturally isomorphic to the identity functor $Id_{\mathsf{Bonding\ Pair}}$. Let $\langle F, G \rangle : A \multimap B$ be a bonding pair. As shown above, the naturality conditions for bonds F and G are expressed as $\iota_A \circ F \circ \tau_B = \mathsf{A}{\circ}\mathsf{B}(F)$ and $\iota_B \circ G \circ \tau_A = \mathsf{A}{\circ}\mathsf{B}(G)$. So $\langle \iota_A, \tau_A \rangle \circ \langle F, G \rangle \circ \langle \tau_A, \iota_A \rangle = \langle \iota_A \circ F \circ \tau_B, \iota_B \circ G \circ \tau_A \rangle = \langle \mathsf{A}{\circ}\mathsf{B}(F), \mathsf{A}{\circ}\mathsf{B}(G) \rangle = \mathsf{A}^2{\circ}\mathsf{B}^2(\langle F, G \rangle)$.

```
     (forall (?bp (bonding-pair ?bp))
         (= (composition (iota-tau (source ?bp)) ?bp)
            (composition
               (LAT.MOR$bonding-pair (CLS.BNDPR$homomorphism ?bp))
               (iota-tau (target ?bp)))))
```

- For any given bonding pair $\langle \boldsymbol{F}, \boldsymbol{G} \rangle : \boldsymbol{A} \multimap \boldsymbol{B}$ the *dual bonding pair* $\langle \boldsymbol{G}^{\mathrm{op}}, \boldsymbol{F}^{\mathrm{op}} \rangle : \boldsymbol{A}^{\mathrm{op}} \multimap \boldsymbol{B}^{\mathrm{op}}$ is the bonding pair with source/target classifications dualized, and forward/reverse bonds switched and dualized.

```
(11) (CNG$function opposite)
     (CNG$signature opposite bonding-pair bonding-pair)
     (forall (?bp (bonding-pair ?bp))
         (and (= (source (opposite ?bp)) (CLS$opposite (source ?bp)))
              (= (target (opposite ?bp)) (CLS$opposite (target ?bp)))
              (= (forward (opposite ?bp)) (CLS.BND$opposite (reverse ?bp)))
              (= (reverse (opposite ?bp)) (CLS.BND$opposite (forward ?bp)))))
```

- Let $\langle \boldsymbol{F}, \boldsymbol{G} \rangle : \boldsymbol{A} \multimap \boldsymbol{B}$ be any bonding pair. Then $\boldsymbol{F} : \boldsymbol{A} \multimap \boldsymbol{B}$ is a bond in the forward direction from classification $\boldsymbol{A}$ to classification $\boldsymbol{B}$, and $\boldsymbol{G} : \boldsymbol{A} \bullet\!\!- \boldsymbol{B}$ is a bond in the reverse direction to classification $\boldsymbol{A}$ from classification $\boldsymbol{B}$. Applying the complete adjoint functor A : Bond $\rightarrow$ Complete Adjoint, we get two adjoint pairs in opposite directions: an adjoint pair $\langle \varphi_F, \psi_F \rangle : \mathsf{L}(\boldsymbol{B}) \rightleftarrows \mathsf{L}(\boldsymbol{A})$ in the forward direction and an adjoint pair $\langle \varphi_G, \psi_G \rangle : \mathsf{L}(\boldsymbol{A}) \rightleftarrows \mathsf{L}(\boldsymbol{B})$ in the reverse direction. It was shown above that for bonding pairs the meet-preserving monotonic function $\psi_F : \mathsf{L}(\boldsymbol{A}) \rightarrow \mathsf{L}(\boldsymbol{B})$ is equal to the join-preserving monotonic function $\varphi_G : \mathsf{L}(\boldsymbol{A}) \rightarrow \mathsf{L}(\boldsymbol{B})$, giving a complete lattice homomorphism.

```
(12) (CNG$function homomorphism)
     (CNG$signature homomorphism bonding-pair CL.MOR$homomorphism)
     (forall (?bp (bonding-pair ?bp))
         (and (= (CL.MOR$source (homomorphism ?bp))
                 (CLS.CL$complete-lattice (source ?a)))
              (= (CL.MOR$target (homomorphism ?bp))
                 (CLS.CL$complete-lattice (target ?a)))
              (= (CL.MOR$forward (homomorphism ?bp))
                 (CLS.BND$adjoint-pair (forward ?h)))
              (= (CL.MOR$reverse (homomorphism ?bp))
                 (CLS.BND$adjoint-pair (reverse ?h)))))
```

This function is the unique mediating function for the $\mathsf{L}(A)$-indexed collective B-concept $(F/\tau_A, \iota_A\backslash G)$, the $\langle F, G\rangle$-image of the $\mathsf{L}(A)$-indexed collective A-concept (ι_A, τ_A), whose closure expressions define the pairing constraints.

Finite Colimits

CLS.COL

Classifications can be fused together and internalized using colimit operations. Here we present axioms that make CLASSIFICATION, the quasi-category of classifications and infomorphisms, finitely cocomplete. We assert the existence of initial classifications, binary coproducts of classifications, coequalizers of parallel pairs of infomorphisms and pushouts of spans of infomorphisms. Because of commonality, the terminology for binary coproducts, coequalizers and pushouts are put into sub-namespaces. The *diagrams*

$$\mathit{INST}^{-1}(\mathrm{SET}) = \mathrm{CLASSIFICATION} = \mathit{TYP}^{-1}(\mathrm{SET})$$

$$0 \dashv \mathit{INST} \dashv \wp \qquad\qquad \wp \dashv \mathit{TYP} \dashv 1$$

$$\begin{tikzcd}
 & \mathrm{CLASSIFICATION} \arrow[dl, "\mathit{INST}"] \arrow[dr, "\mathit{TYP}"] & \\
\mathrm{SET}^{\mathrm{op}} \arrow[ur, bend left, "0"] \arrow[ur, bend right, "\wp"'] & & \mathrm{SET} \arrow[ul, bend left, "\wp"] \arrow[ul, bend right, "1"']
\end{tikzcd}$$

Figure 1: Fibered Span

and *colimits* are denoted by both generic and specific terminology.

The following discussion refers to Figure 1 (where arrows denote functors).

The existence of colimits is mediated through the quasi-adjunction $\mathit{INST} \dashv \wp$ between

$\mathit{INST} : \mathrm{CLASSIFICATION} \rightarrow \mathrm{SET}^{\mathrm{op}}$ the underlying instance quasi-functor, and

$\wp : \mathrm{SET}^{\mathrm{op}} \rightarrow \mathrm{CLASSIFICATION}$ the instance power quasi-functor (see Figure 1),

and the (trivial) quasi-adjunction $\mathit{TYP} \dashv 1$ between

$\mathit{TYP} : \mathrm{CLASSIFICATION} \rightarrow \mathrm{SET}$ the underlying type quasi-functor, and

$1 : \mathrm{SET} \rightarrow \mathrm{CLASSIFICATION}$ the (trivial) terminal type quasi-functor (see Figure 1).

Since the quasi-functors *INST* and *TYP* are left adjoint, they preserve all colimits. Using preservation as a guide, all diagrams, cocones and colimits in CLASSIFICATION have (and use) underlying instance/type diagrams, instance cones, type cocones, instance limits and type colimits in SET.

The Initial Classification

$$\begin{tikzcd}
\boldsymbol{0} \arrow[r, shift left, "!_A"] & \boldsymbol{A} \arrow[l, shift left]
\end{tikzcd}$$

Figure 2: Initial Classification & Universality

- There is a special classification $\boldsymbol{0} = \langle 1 = \{0\}, \emptyset, \emptyset \rangle$ (see Figure 2, where arrows denote functions) called the *initial classification*, which has only one instance 0, no types, and empty incidence. The initial classification has the property that for any classification $\boldsymbol{A} = \langle \mathit{inst}(\boldsymbol{A}), \mathit{typ}(\boldsymbol{A}), \vDash_{\boldsymbol{A}} \rangle$ there is a *counique infomorphism* $!_A : \boldsymbol{0} \rightleftarrows \boldsymbol{A}$, the one and only infomorphism from $\boldsymbol{0}$ to $\boldsymbol{A}$. The instance function of this infomorphism is the unique function (the constant 0 function) from the instance class $\mathit{inst}(\boldsymbol{A})$ to the terminal class *1*. The type function of this infomorphism is the counique function (the empty function) from the initial (empty or null) class $\emptyset$ to the type class $\mathit{typ}(\boldsymbol{A})$. The fundamental constraint for the counique infomorphism is vacuous. Of course, this needs to be verified.

```
(1) (CLS$classification initial)
    (= (CLS$instance initial) SET.LIM$terminal)
    (= (CLS$type initial) SET.COL$initial)
    (= (CLS$incidence initial) SET.COL$empty)

(2) (CNG$function counique)
    (CNG$signature counique CLS$classification CLS.INFO$infomorphism)
    (forall (?a (CLS$classification ?a))
        (and (= (CLS.INFO$source (counique ?a)) initial)
             (= (CLS.INFO$target (counique ?a)) ?a)
             (= (CLS.INFO$instance (counique ?a)) (SET.LIM$unique (CLS$instance ?s)))
             (= (CLS.INFO$type (counique ?a)) (SET.COL$counique (CLS$type ?a)))))
```

Binary Coproducts

CLS.COL.COPRD

A *binary coproduct* is a finite colimit for a diagram of shape • •. Such a diagram (of classifications and infomorphisms) is called a *pair* of classifications. Given a pair of classifications $\boldsymbol{A}_1 = \langle inst(\boldsymbol{A}_1), typ(\boldsymbol{A}_1), \vDash_1\rangle$ and $\boldsymbol{A}_2 = \langle inst(\boldsymbol{A}_2), typ(\boldsymbol{A}_2), \vDash_2\rangle$, the *coproduct* or *sum* $\boldsymbol{A}_1 + \boldsymbol{A}_2$ (see top of Figure 3, where arrows denote functions) is the classification defined as follows:

- The set of instances is the Cartesian product $inst(\boldsymbol{A}_1+\boldsymbol{A}_2) = inst(\boldsymbol{A}_1)\times inst(\boldsymbol{A}_2)$. So, instances of $\boldsymbol{A}_1 + \boldsymbol{A}_2$ are pairs (i_1, i_2) of instances $i_1 \in inst(\boldsymbol{A}_1)$ and $i_2 \in inst(\boldsymbol{A}_2)$.
- The set of types is the disjoint union $typ(\boldsymbol{A}_1+\boldsymbol{A}_2) = typ(\boldsymbol{A}_1)+typ(\boldsymbol{A}_2)$. Concretely, types of $\boldsymbol{A}_1 + \boldsymbol{A}_2$ are either pairs $(1, t_1)$ where $t_1 \in typ(\boldsymbol{A}_1)$ or pairs $(2, t_2)$ where $t_2 \in inst(\boldsymbol{A}_2)$.
- The incidence $\vDash_{1+2}$ of $\boldsymbol{A}_1 + \boldsymbol{A}_2$ is defined by

  $(i_1, i_2) \vDash_{1+2} (1, t_1)$ when $i_1 \vDash_1 t_1$

  $(i_1, i_2) \vDash_{1+2} (2, t_2)$ when $i_2 \vDash_2 t_2$.

$$\begin{tikzcd}
\boldsymbol{A}_1 \arrow[r, shift left, "\iota_1"] \arrow[dr, shift left, "f_1"'] & \boldsymbol{A}_1+\boldsymbol{A}_2 \arrow[l, shift left] \arrow[r, shift right] \arrow[d, shift right] & \boldsymbol{A}_2 \arrow[l, shift right, "\iota_2"'] \arrow[dl, shift right] \\
& \boldsymbol{B} \arrow[ul, shift left] \arrow[u, shift right, "{[f_1,f_2]}"] \arrow[ur, shift right, "f_2"'] &
\end{tikzcd}$$

Figure 3: Binary Coproduct & Universality

- A *pair* (of classifications) is the appropriate base diagram for a binary coproduct. Each pair consists of a pair of classifications called *classification1* and *classification2*. Let either '`diagram`' or '`pair`' be the CLS namespace term that denotes the *Pair* collection. Pairs are determined by their two component classifications.

```
(1) (CNG$conglomerate diagram)
    (CNG$conglomerate pair)
    (= pair diagram)

(2) (CNG$function classification1)
    (CNG$signature classification1 diagram CLS$classification)

(3) (CNG$function classification2)
    (CNG$signature classification2 diagram CLS$classification)

    (forall (?p (diagram ?p) ?q (diagram ?q))
        (=> (and (= (classification1 ?p) (classification1 ?q))
                 (= (classification2 ?p) (classification2 ?q)))
            (= ?p ?q)))
```

- There is an *instance pair* or *instance diagram* function, which maps a pair of classifications to the underlying pair of instance classes. Similarly, there is a *type pair* function, which maps a pair of classifications to the underlying pair of type classes.

```
(4) (CNG$function instance-diagram)
    (CNG$function instance-pair)
    (= instance-pair instance-diagram)
    (CNG$signature instance-diagram diagram SET.LIM.PRD$diagram)
    (forall (?p (diagram ?p))
        (and (= (SET.LIM.PRD$class1 (instance-diagram ?p))
                (CLS$instance (classification1 ?p)))
             (= (SET.LIM.PRD$class2 (instance-diagram ?p))
                (CLS$instance (classification2 ?p)))))

(5) (CNG$function type-diagram)
    (CNG$function type-pair)
    (= type-pair type-diagram)
    (CNG$signature type-diagram diagram SET.COLIM.COPRD$diagram)
    (forall (?p (diagram ?p))
        (and (= (SET.COLIM.COPRD$class1 (type-diagram ?p))
                (CLS$type (classification1 ?p)))
             (= (SET.COLIM.COPRD$class2 (type-diagram ?p))
                (CLS$type (classification2 ?p)))))
```

- Every pair has an *opposite*.

```
(6) (CNG$function opposite)
    (CNG$signature opposite pair pair)
    (forall (?p (pair ?p))
        (and (= (classification1 (opposite ?p)) (classification2 ?p))
             (= (classification2 (opposite ?p)) (classification1 ?p))))
```

o The opposite of the opposite is the original pair – the following theorem can be proven.

```
    (forall (?p (pair ?p))
        (= (opposite (opposite ?p)) ?p))
```

o A *coproduct cocone* is the appropriate cocone for a binary coproduct. A coproduct cocone (Figure 4, where arrows denote functions) consists of a pair of infomorphisms called *opfirst* and *opsecond*. These are required to have a common source class called the *opvertex* of the cocone. Each coproduct cocone is over a pair. A coproduct cocone is the very special case of a cocone over a pair (of classifications). Let ‘cocone’ be the CLS term that denotes the *Coproduct Cocone* collection.

Figure 4: Coproduct Cocone

$$\begin{tikzcd}
\boldsymbol{A}_1 \arrow[dr, shift left] & & \boldsymbol{A}_2 \arrow[dl, shift right] \\
& \boldsymbol{A} \arrow[ul, shift left, "op1st"] \arrow[ur, shift right, "op2nd"'] &
\end{tikzcd}$$

```
(7) (CNG$conglomerate cocone)

(8) (CNG$function cocone-diagram)
    (CNG$signature cocone-diagram cocone diagram)

(9) (CNG$function opvertex)
    (CNG$signature opvertex cocone CLS$classification)

(10) (CNG$function opfirst)
     (CNG$signature opfirst cocone CLS.INFO$infomorphism)
     (forall (?s (cocone ?s))
         (and (= (CLS.INFO$source (opfirst ?s))
                 (classification1 (cocone-diagram ?s)))
              (= (CLS.INFO$target (opfirst ?s)) (opvertex ?s))))

(11) (CNG$function opsecond)
     (CNG$signature opsecond cocone CLS.INFO$infomorphism)
     (forall (?s (cocone ?s))
         (and (= (CLS.INFO$source (opsecond ?s))
                 (classification2 (cocone-diagram ?s)))
              (= (CLS.INFO$target (opsecond ?s)) (opvertex ?s))))
```

o There is an *instance cone* function, which maps a coproduct cocone of classifications and infomorphisms to the underlying product cone of instance classes and instance functions. Similarly, there is a *type cocone* function, which maps a coproduct cocone of classifications and infomorphisms to the underlying cocone of type classes and type functions.

```
(12) (CNG$function instance-cone)
     (CNG$signature instance-cone cocone SET.LIM.PRD$cone)
     (forall (?s (cocone ?s))
         (and (= (SET.LIM.PRD$cone-diagram (instance-cone ?s))
                 (instance-diagram (cocone-diagram ?s)))
              (= (SET.LIM.PRD$vertex (instance-cone ?s))
                 (CLS$instance (opvertex ?s)))
              (= (SET.LIM.PRD$first (instance-pair ?s))
                 (CLS.INFO$instance (opfirst ?s)))
              (= (SET.LIM.PRD$second (instance-pair ?s))
                 (CLS.INFO$instance (opsecond ?s)))))

(13) (CNG$function type-cocone)
     (CNG$signature type-cocone cocone SET.COLIM.COPRD$cocone)
     (forall (?s (cocone ?s))
         (and (= (SET.COLIM.COPRD$cocone-diagram (type-cocone ?s))
                 (type-diagram (cocone-diagram ?s)))
              (= (SET.COLIM.COPRD$opvertex (type-cocone ?s))
                 (CLS$type (opvertex ?s)))
              (= (SET.COLIM.COPRD$opfirst (type-cocone ?s))
                 (CLS.INFO$type (opfirst ?s)))
              (= (SET.COLIM.COPRD$opsecond (type-cocone ?s))
                 (CLS.INFO$type (opsecond ?s)))))
```

- There is a unary CNG function ‘`colimiting-cone`’ that maps a pair (of classifications) to its binary coproduct (colimiting binary coproduct cocone) (Figure 5, where arrows denote functions). Axiom (*) asserts that this function is total. This, along with the universality of the comediator infomorphism, implies that a binary coproduct exists for any pair of classifications. The opvertex of the colimiting binary coproduct cocone is a specific *Binary Coproduct* class given by the CNG function ‘`binary-coproduct`’. It comes equipped with two CNG injection infomorphisms ‘`injection1`’ and ‘`injection2`’. This notation is for convenience of reference. It is used for pushouts in general. Axiom (#) is the necessary condition that the instance and type quasi-functors preserve concrete colimits. This ensures that both this coproduct and its injection infomorphisms are specific – that its instance class is exactly the Cartesian product of the instance classes of the pair of classifications and that its type class is exactly the disjoint union of the type classes of the pair of classifications. The injection terms in axioms (16–17) are created for convenience of reference. That these are infomorphisms needs to be checked; that is, the instance and types of the source and target classifications need to be checked for correctness, and the fundamental property for these infomorphisms must be verified.

Figure 5: Colimiting Cocone

$$\begin{tikzcd}
\boldsymbol{A}_1 \arrow[dr, shift left] & & \boldsymbol{A}_2 \arrow[dl, shift left] \\
& \boldsymbol{A}_1{+}\boldsymbol{A}_2 \arrow[ul, shift left, "\iota_1"] \arrow[ur, shift left, "\iota_2"'] &
\end{tikzcd}$$

```
(14) (CNG$function colimiting-cocone)
     (CNG$signature colimiting-cocone diagram cocone)
 (*) (forall (?p (diagram ?p))
         (exists (?s (cocone ?s))
             (= (colimiting-cocone ?p) ?s)))

 (#) (forall (?p (diagram ?p))
         (and (= (cocone-diagram (colimiting-cocone ?p)) ?p)
              (= (instance-cone (colimiting-cocone ?p))
                 (SET.LIM.PRD$limiting-cone (instance-diagram ?p))
              (= (type-cocone (colimiting-cocone ?p))
                 (SET.COL.COPRD$colimiting-cocone (type-diagram ?p)))))

(15) (CNG$function colimit)
     (CNG$function binary-coproduct)
     (= binary-coproduct colimit)
     (CNG$signature colimit diagram CLS$classification)
     (forall (?p (diagram ?p))
         (= (colimit ?p) (opvertex (colimiting-cocone ?p))))

(16) (CNG$function injection1)
     (CNG$signature injection1 diagram CLS.INFO$infomorphism)
     (forall (?p (diagram ?p))
         (and (= (CLS.INFO$source (injection1 ?p)) (classification1 ?p))
              (= (CLS.INFO$target (injection1 ?p)) (colimit ?p))
              (= (injection1 ?p) (opfirst (colimiting-cocone ?p)))))

(17) (CNG$function injection2)
     (CNG$signature injection2 diagram CLS.INFO$infomorphism)
     (forall (?p (diagram ?p))
         (and (= (CLS.INFO$source (injection2 ?p)) (classification2 ?p))
              (= (CLS.INFO$target (injection2 ?p)) (colimit ?p))
              (= (injection2 ?p) (opsecond (colimiting-cocone ?p)))))

 (#) (forall (?p (diagram ?p))
         (forall (?i ((CLS$instance (colimit ?p)) ?i)
                  ?t ((CLS$type (colimit ?p)) ?t))
             (<=> (= ((CLS$incidence (colimit ?p)) [?i ?t])
                  (or ((CLS$incidence (classification1 ?p)) [(?i 1) (?t 2)])
                      ((CLS$incidence (classification2 ?p)) [(?i 2) (?t 2)])))))
```

- For any cocone, there is a *comediator* infomorphism (see Figure 6, where arrows denote infomorphisms) from the binary coproduct of the underlying diagram (pair of classifications) to the opvertex of the cocone. This is the unique infomorphism, which commutes with opfirst and opsecond. We use a KIF definite description abbreviation to define this. Existence and uniqueness represents the universality of the binary coproduct operator. That

Figure 6: Coproduct Comediator

$$\begin{tikzcd}
\boldsymbol{A}_1 \arrow[r, "\iota_1"] \arrow[dr, "op1st"'] & \boldsymbol{A}_1{+}\boldsymbol{A}_2 \arrow[d, "\gamma"] & \boldsymbol{A}_2 \arrow[l, "\iota_2"'] \arrow[dl, "op2n"] \\
& \boldsymbol{A} &
\end{tikzcd}$$

this is an infomorphism needs to be checked; that is, the instance and types of the source and target classifications need to be checked for correctness, and the fundamental property for this infomorphism must be verified.

```
(18) (CNG$function comediator)
     (CNG$signature comediator cocone CLS.INFO$infomorphism)
     (forall (?s (cocone ?s))
         (and (= (CLS.INFO$source (comediator ?s)) (colimit (cocone-diagram ?s)))
              (= (CLS.INFO$target (comediator ?s)) (opvertex ?s))
              (= (CLS.INFO$instance (comediator ?s))
                 (SET.LIM.PRD$mediator (instance-cone ?s)))
              (= (CLS.INFO$type (comediator ?s))
                 (SET.COL.COPRD$comediator (type-cocone ?s)))))
```

○ It can also be verified that the comediator is the unique infomorphism that makes the diagram in Figure 4 commutative.

```
     (forall (?s (cocone ?s))
         (= (comediator ?s)
            (the (?f (CLS.INFO$infomorphism ?f))
                (and (= (CLS.INFO$composition (injection1 (cocone-diagram ?s)) ?f)
                        (opfirst ?s))
                     (= (CLS.INFO$composition (injection2 (cocone-diagram ?s)) ?f)
                        (opsecond ?s))))))
```

Coinvariants and Coquotients

CLS.COL.COINV

○ Given a classification $\boldsymbol{A} = \langle inst(\boldsymbol{A}), typ(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$, a *coinvariant* (called a *dual invariant* in Barwise and Seligman, 1997) is a pair $J = \langle A, R \rangle$ consisting of a subclass of instances $A \subseteq inst(\boldsymbol{A})$ and a binary endorelation $\boldsymbol{R}$ on types $R \subseteq typ(\boldsymbol{A}) \times typ(\boldsymbol{A})$ that satisfies the fundamental constraint:

if $(t_0, t_1) \in R$ then for each $i \in A$, $i \models_{\boldsymbol{A}} t_0$ iff $i \models_{\boldsymbol{A}} t_1$.

The classification $\boldsymbol{A}$ is called the *base classification* of J – the classification on which J is based.

```
(1) (CNG$conglomerate coinvariant)

(2) (CNG$function classification)
    (CNG$function base)
    (= base classification)
    (CNG$signature base coinvariant CLS$classification)

(3) (CNG$function class)
    (CNG$signature class coinvariant SET$class)

(4) (CNG$function endorelation)
    (CNG$signature endorelation coinvariant REL.ENDO$endorelation)

    (forall (?j (coinvariant ?j))
        (and (SET$subclass (class ?j) (CLS$instance (base ?j))))
             (SET$subclass (REL.ENDO$class (endorelation ?j)) (CLS$type (base ?j))))
             (forall (?i ((class ?j) ?i)
                     ?t0 ?t1 (REL.ENDO$extent (endorelation ?j) [?t0 ?t1]))
                 (<=> ((CLS$incidence (base ?j)) [?i ?t0])
                      ((CLS$incidence (base ?j)) [?i ?t1])))))
```

○ Often, the relation $\boldsymbol{R}$ is an equivalence relation on the types. However, it is convenient not to require this. The endorelation $\boldsymbol{R}$ is contained in a smallest equivalence relation $\equiv_{\boldsymbol{R}}$ on types called the equivalence relation generated by $\boldsymbol{R}$. This is the reflexive, symmetric, transitive closure of $\boldsymbol{R}$. For any type $t \in typ(\boldsymbol{A})$, write $[t]_{\boldsymbol{R}}$ for the $\boldsymbol{R}$-equivalence class of t. The *coquotient* (called the *dual quotient* in Barwise and Seligman, 1997) $\boldsymbol{A}/J$ of a coinvariant J on a classification $\boldsymbol{A}$ (see Figure 7, where arrows denote functions) is the classification defined as follows:

– The set of instances is A, the given subset of $inst(\boldsymbol{A})$.

– The set of types is $typ(\boldsymbol{A})/\boldsymbol{R}$, the set of $\boldsymbol{R}$-equivalence classes of $typ(\boldsymbol{A})$.

$$\begin{tikzcd}
\boldsymbol{A} \arrow[r, shift left, "\boldsymbol{\tau}_J"] \arrow[rd, shift left] & \boldsymbol{A}/J \arrow[l, shift left] \arrow[d, shift left, "\tilde{\boldsymbol{f}}"] \\
 & \boldsymbol{B} \arrow[lu, shift left, "\boldsymbol{f}"] \arrow[u, shift left]
\end{tikzcd}$$

Figure 7: Universality for the Coquotient

– The incidence $\vDash_{A/J}$ of $\boldsymbol{A}/J$ is defined by

$i \vDash_{A/J} [t]_R$ when $i \vDash_{\boldsymbol{A}} t$

which is well-defined by the fundamental constraint.

There is a *canonical quotient infomorphism* $\boldsymbol{\tau}_J : \boldsymbol{A} \rightleftarrows \boldsymbol{A}/J$, whose instance function is the inclusion function $inc : A \rightarrow \mathit{inst}(\boldsymbol{A})$, and whose type function is the canonical quotient function and $[\text{-}]_{\boldsymbol{R}} : \mathit{typ}(\boldsymbol{A}) \rightarrow \mathit{typ}(\boldsymbol{A})/\boldsymbol{R}$. The fundamental property for this infomorphism is trivial, given the definition of the coquotient incidence above.

Here is the KIF formalism for coinvariants and coquotients. A coinvariant is determined by its base, class and endorelation triple.

```
(5) (CNG$function coquotient)
    (CNG$signature coquotient coinvariant CLS$classification)
    (forall (?j (coinvariant ?j))
        (and (= (CLS$instance (coquotient ?j))
                (class ?j))
             (= (CLS$type (coquotient ?j))
                (REL.ENDO$quotient (REL.ENDO$equivalence-closure (endorelation ?j))))
             (forall (?i ((class ?j) ?i)
                      ?t ((CLS$type (base ?j)) ?t))
                 (<=> ((CLS$incidence (coquotient ?j))
                          [?i ((REL.ENDO$canon (endorelation ?j)) ?t)])
                      ((CLS$incidence (base ?j)) [?i ?t])))))

(6) (CNG$function canon)
    (CNG$signature canon coinvariant CLS.INFO$infomorphism)
    (forall (?j (coinvariant ?j))
        (and (= (CLS.INFO$source (canon ?j)) (base ?j))
             (= (CLS.INFO$target (canon ?j)) (coquotient ?j))
             (= (CLS.INFO$instance (canon ?j))
                (SET.FTN$inclusion (class ?j) (CLS$instance (base ?j))))
             (= (CLS.INFO$type (canon ?j))
                (REL.ENDO$canon (endorelation ?j)))))
```

Let $J = \langle A, R\rangle$ be a coinvariant on a classification $\boldsymbol{A}$. An infomorphism $\boldsymbol{f} : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ *respects J* when:

1. For any instance $i \in \mathit{inst}(\boldsymbol{B})$, $\mathit{inst}(\boldsymbol{f})(i) \in A$; and
2. For any two types $t_0, t_1 \in \mathit{typ}(\boldsymbol{A})$, if $(t_0, t_1) \in R$ then $\mathit{typ}(\boldsymbol{f})(t_0) = \mathit{typ}(\boldsymbol{f})(t_1)$.

```
(7) (CNG$relation respects)
    (CNG$signature respects CLS.INFO$infomorphism coinvariant)
    (forall (?j (coinvariant ?j)
             ?f (CLS.INFO$infomorphism ?f))
        (<=> (respects ?f ?j)
             (and (= (CLS.INFO$source ?f) (base ?j))))
                  (forall (?i ((CLS.INFO$target ?f) ?i))
                      ((class ?j) ((CLS.INFO$instance ?f) ?i)))
                  (forall (?t0 ((CLS.INFO$source ?f) ?t0)
                           ?t1 ((CLS.INFO$source ?f) ?t1))
                      (=> ((REL.ENDO$extent (endorelation ?j)) [?t0 ?t1])
                          (= ((CLS.INFO$type ?f) ?t0) ((CLS.INFO$type ?f) ?t1))))))
```

Proposition. *Let J be a coinvariant on a classification* $\boldsymbol{A}$. *For every infomorphism* $\boldsymbol{f} : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ *that respects J, there is a unique comediating infomorphism* $\tilde{\boldsymbol{f}} : \boldsymbol{A}/J \rightleftarrows \boldsymbol{B}$ *such that* $\boldsymbol{\tau}_J \circ \tilde{\boldsymbol{f}} = \boldsymbol{f}$ *(the diagram in Figure 5 commutes).*

This proposition is used to define a *comediator* function, which maps a respectful infomorphism to its unique associate.

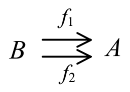
```
(8) (KIF$function comediator)
    (KIF$signature comediator coinvariant CNG$function)
    (forall (?j (coinvariant ?j)
        (and (CNG$signature (comediator ?j)
                 CLS.INFO$infomorphism CLS.INFO$infomorphism)
             (forall (?f (CLS.INFO$infomorphism ?f) (respects ?f ?j))
                  (= ((comediator ?j) ?f)
                     (the (?ft (CLS.INFO$infomorphism ?ft))
                        (and (= (CLS.INFO$source ?ft) (coquotient ?j))
                             (= (CLS.INFO$target ?ft) (CLS.INFO$target ?f))
                             (= (CLS.INFO$composition (coquotient ?j) ?ft) ?f)))))))
```

Coequalizers

`CLS.COL.COEQ`

A (binary) *coequalizer* is a finite colimit in CLASSIFICATION for a diagram of shape $\cdot \rightrightarrows \cdot$. Such a diagram (of classes and functions) is called a *parallel pair* of functions.

- A *parallel pair* (see Figure 8, where arrows denote infomorphisms) is the appropriate base diagram for a coequalizer. Each parallel pair consists of a pair of infomorphisms called *infomorphism1* and *infomorphism2* that share the same *source* and *target* classifications. Let either '`diagram`' or '`parallel-pair`' be the term that denotes the *Parallel Pair* collection. Parallel pairs are determined by their two component infomorphisms.



$$\begin{tikzcd} B \arrow[r, shift left, "f_1"] \arrow[r, shift right, "f_2"'] & A \end{tikzcd}$$

Figure 8: Parallel Pair

```
(1) (conglomerate diagram)
    (conglomerate parallel-pair)
    (= parallel-pair diagram)

(2) (CNG$function source)
    (CNG$signature source diagram CLS$classification)

(3) (CNG$function target)
    (CNG$signature target diagram CLS$classification)

(4) (CNG$function infomorphism1)
    (CNG$signature infomorphism1 diagram CLS.INFO$infomorphism)

(5) (CNG$function infomorphism2)
    (CNG$signature infomorphism2 diagram CLS.INFO$infomorphism)

    (forall (?p (diagram ?p))
        (and (= (SET.FTN$source (infomorphism1 ?p)) (source ?p))
             (= (SET.FTN$target (infomorphism1 ?p)) (target ?p))
             (= (SET.FTN$source (infomorphism2 ?p)) (source ?p))
             (= (SET.FTN$target (infomorphism2 ?p)) (target ?p))))

    (forall (?p (diagram ?p) ?q (diagram ?q))
        (=> (and (= (infomorphism1 ?p) (infomorphism1 ?q))
                 (= (infomorphism2 ?p) (infomorphism2 ?q)))
            (= ?p ?q)))
```

- There is an *instance parallel pair* or *instance diagram* function, which maps a parallel pair of infomorphisms to the underlying (`SET.LIM.EQU`) parallel pair of instance functions. Similarly, there is a *type parallel pair* or *type diagram* function, which maps a parallel pair of infomorphisms to the underlying (`SET.COLIM.COEQ`) parallel pair of type functions.

```
(6) (CNG$function instance-diagram)
    (CNG$function instance-parallel-pair)
    (= instance-parallel-pair instance-diagram)
    (CNG$signature instance-diagram diagram SET.LIM.EQU$diagram)
    (forall (?p (diagram ?p))
        (and (= (SET.LIM.EQU$source (instance-diagram ?p))
                (CLS$instance (target ?p)))
             (= (SET.LIM.EQU$target (instance-diagram ?p))
                (CLS$instance (source ?p)))
             (= (SET.LIM.EQU$function1 (instance-diagram ?p))
                (CLS.INFO$instance (infomorphism1 ?p)))
             (= (SET.LIM.EQU$function2 (instance-diagram ?p))
                (CLS.INFO$instance (infomorphism2 ?p)))))

(7) (CNG$function type-diagram)
    (CNG$function type-parallel-pair)
    (= type-parallel-pair type-diagram)
    (CNG$signature type-diagram diagram SET.COLIM.COEQ$diagram)
```

```
        (forall (?p (diagram ?p))
            (and (= (SET.COLIM.COEQ$source (type-diagram ?p))
                    (CLS$type (source ?p)))
                 (= (SET.COLIM.COEQ$target (type-diagram ?p))
                    (CLS$type (target ?p)))
                 (= (SET.COLIM.COEQ$function1 (type-diagram ?p))
                    (CLS.INFO$type (infomorphism1 ?p)))
                 (= (SET.COLIM.COEQ$function2 (type-diagram ?p))
                    (CLS.INFO$type (infomorphism2 ?p)))))
```

- Every parallel pair of infomorphisms has an associated coinvariant whose base classification is the target of the parallel pair. There is a *coinvariant* function, which maps a parallel pair of infomorphisms to the associated coinvariant.

```
(8) (CNG$function coinvariant)
    (CNG$signature coinvariant diagram CLS.COL.COINV$coinvariant)
    (forall (?p (diagram ?p))
        (and (= (CLS.COL.COINV$base (coinvariant ?p))
                (target ?p))
             (= (CLS.COL.COINV$class (coinvariant ?p))
                (SET.LIM.EQU$equalizer (instance-diagram ?p)))
             (= (CLS.COL.COINV$endorelation (coinvariant ?p))
                (SET.COL.COEQ$endorelation (type-diagram ?p)))))
```

- *Coequalizer Cocones* are used to specify and axiomatize coequalizers. Each coequalizer cocone (see Figure 9, where arrows denote infomorphisms) has an underlying *parallel-pair*, an *opvertex* class, and an infomorphism called *infomorphism*, whose source classification is the target classification of the infomorphisms in the parallel-pair and whose target classification is the opvertex. The composite infomorphism indicated in the diagram below is obviously not needed. An coequalizer cocone is the very special case of a cocone over an parallel-pair. Let 'cocone' be the term that denotes the *Coequalizer Cocone* collection.

$$\begin{tikzcd} B \arrow[r, shift left, "f_1"] \arrow[r, shift right, "f_2"'] \arrow[dr, "f_1 \cdot f = f_2 \cdot f"'] & A \arrow[d, "f"] \\ & C \end{tikzcd}$$

Figure 9: Coequalizer Cocone

```
(9) (CNG$conglomerate cocone)

(10) (CNG$function cocone-diagram)
     (CNG$signature cocone-diagram cocone diagram)

(11) (CNG$function opvertex)
     (CNG$signature opvertex cocone CLS$classification)

(12) (CNG$function infomorphism)
     (CNG$signature infomorphism cocone CLS.INFO$infomorphism)

     (forall (?s (cocone ?s))
        (and (= (CLS.INFO$source (infomorphism ?s)) (target (cocone-diagram ?s)))
             (= (CLS.INFO$target (infomorphism ?s)) (opvertex ?s))
             (= (CLS.INFO$composition
                    (infomorphism1 (cocone-diagram ?s))
                    (infomorphism ?s))
                (CLS.INFO$composition
                    (infomorphism2 (cocone-diagram ?s))
                    (infomorphism ?s)))))
```

- There is an *instance equalizer cone* function, which maps a coequalizer cocone of classifications and infomorphisms to the underlying equalizer cone of instance classes and instance functions. Similarly, there is a *type coequalizer cocone* function, which maps a coequalizer cocone of classifications and infomorphisms to the underlying coequalizer cocone of type classes and type functions.

```
(13) (CNG$function instance-cone)
     (CNG$signature instance-cone cocone SET.LIM.EQU$cone)
     (forall (?s (cocone ?s))
         (and (= (SET.LIM.EQU$cone-diagram (instance-cone ?s))
                 (instance-diagram (cocone-diagram ?s)))
              (= (SET.LIM.EQU$vertex (instance-cone ?s))
                 (CLS$instance (opvertex ?s)))
              (= (SET.LIM.EQU$function (instance-pair ?s))
                 (CLS.INFO$instance (infomorphism ?s)))))

(14) (CNG$function type-cocone)
     (CNG$signature type-cocone cocone SET.COLIM.COEQ$cocone)
     (forall (?s (cocone ?s))
         (and (= (SET.COLIM.COEQ$cocone-diagram (type-cocone ?s))
                 (type-diagram (cocone-diagram ?s)))
              (= (SET.COLIM.COEQ$opvertex (type-cocone ?s))
                 (CLS$type (opvertex ?s)))
              (= (SET.COLIM.COEQ$function (type-cocone ?s))
                 (CLS.INFO$type (infomorphism ?s)))))
```

- There is a unary CNG function '`limiting-cone`' that maps a parallel pair (of infomorphisms) to its coequalizer (colimiting coequalizer cocone) (see Figure 10, where arrows denote infomorphisms). Axiom (*) asserts that this function is total. This, along with the universality of the comediator infomorphism *gamma*, implies that a coequalizer exists for any parallel pair of infomorphisms. The opvertex of the colimiting coequalizer cocone is a specific *Coequalizer* class given by the CNG function '`coequalizer`'. It comes equipped with a CNG quotient infomorphism '`canon`'. This notation is for convenience of reference. Axiom (#) is the necessary condition that the instance and type quasi-functors preserve concrete colimits. This ensures that both this coequalizer and its canon infomorphism are specific. The canon term in axiom (17) is created for convenience of reference. That this is an infomorphism needs to be checked; that is, the instance and types of the source and target classifications need to be checked for correctness, and the fundamental property for this infomorphism must be verified.

Figure 10: Colimiting Cocone

$$\begin{tikzcd}
\boldsymbol{B} \arrow[r, shift left, "f_1"] \arrow[r, shift right, "f_2"'] \arrow[dr] & \boldsymbol{A} \arrow[d, "\gamma"] \\
& \hat{\boldsymbol{A}}
\end{tikzcd}$$

```
(15) (CNG$function colimiting-cocone)
     (CNG$signature colimiting-cocone diagram cocone)
 (*) (forall (?p (diagram ?p))
         (exists (?s (cocone ?s))
             (= (colimiting-cocone ?p) ?s)))

 (#) (forall (?p (diagram ?p))
         (and (= (cocone-diagram (colimiting-cocone ?p)) ?p)
              (= (instance-cone (colimiting-cocone ?p))
                 (SET.LIM.EQU$limiting-cone (instance-diagram ?p))
              (= (type-cocone (colimiting-cocone ?p))
                 (SET.COL.COEQ$colimiting-cocone (type-diagram ?p)))))

(16) (CNG$function colimit)
     (CNG$function coequalizer)
     (= coequalizer colimit)
     (CNG$signature colimit diagram CLS$classification)
     (forall (?p (diagram ?p))
        (= (colimit ?p) (opvertex (colimiting-cocone ?p))))

(17) (CNG$function canon)
     (CNG$signature canon diagram CLS.INFO$infomorphism)
     (forall (?p (diagram ?p))
         (and (= (CLS.INFO$source (canon ?p)) (target ?p))
              (= (CLS.INFO$target (canon ?p)) (colimit ?p))
              (= (canon ?p) (infomorphism (colimiting-cocone ?p)))))

 (#) (forall (?p (diagram ?p))
         (and (SET.FTN$surjection (CLS.INFO$type (canon ?p)))
              (forall (?i ((class ?j) ?i)
                       ?t ((CLS$type (target ?p)) ?t))
                  (<=> ((CLS$incidence (colimit ?p))
                           [?i ((CLS.INFO$type (canon ?p)) ?t)])
                       ((CLS$incidence (target ?p)) [?i ?t])))
```

- The coquotient classification of the coinvariant of a coequalizer diagram (parallel pair of infomorphisms) is (not just isomorphic but) equal to the coequalizer classification. Likewise, the canon infomorphism of the coinvariant of a coequalizer diagram (parallel pair of infomorphisms) is equal to the coequalizer canon infomorphism.

```
        (forall (?p (diagram ?p))
            (and (= (CLS.COL.COINV$coquotient (coinvariant ?p)) (coequalizer ?p))
                 (= (CLS.COL.COINV$canoon (coinvariant ?p)) (canon ?p))))
```

Pushouts

CLS.COL.PSH

Given a pair of infomorphisms $\boldsymbol{f}_1 : \boldsymbol{A} \rightleftarrows \boldsymbol{B}_1$ and $\boldsymbol{f}_2 : \boldsymbol{A} \rightleftarrows \boldsymbol{B}_2$ with a common source classification $\boldsymbol{A}$, a *pushout* of $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ (see Figure 11, where arrows denote functions) consists of a classification $\boldsymbol{P}$ and a pair of infomorphisms $\boldsymbol{m}_1 : \boldsymbol{B}_1 \rightleftarrows \boldsymbol{P}$ and $\boldsymbol{m}_2 : \boldsymbol{B}_2 \rightleftarrows \boldsymbol{P}$ with common target classification $\boldsymbol{P}$, that satisfy the following universality conditions.

1. $\boldsymbol{f}_1 \circ \boldsymbol{m}_1 = \boldsymbol{f}_2 \circ \boldsymbol{m}_2$
2. if $\boldsymbol{C}$ is any classification and $\boldsymbol{g}_1 : \boldsymbol{B}_1 \rightleftarrows \boldsymbol{C}$ and $\boldsymbol{g}_2 : \boldsymbol{B}_2 \rightleftarrows \boldsymbol{C}$ are a pair of infomorphisms with the common target classification $\boldsymbol{C}$, that satisfy $\boldsymbol{f}_1 \circ \boldsymbol{g}_1 = \boldsymbol{f}_2 \circ \boldsymbol{g}_2$, then there is a unique infomorphism $\boldsymbol{g} : \boldsymbol{P} \rightleftarrows \boldsymbol{C}$ satisfying $\boldsymbol{m}_1 \circ \boldsymbol{g} = \boldsymbol{g}_1$ and $\boldsymbol{m}_2 \circ \boldsymbol{g} = \boldsymbol{g}_2$.

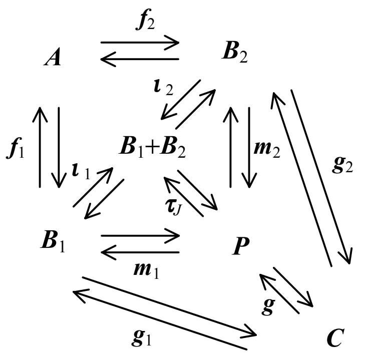


Figure 11: Pushout & Universality

In order to build the pushout we implicitly use a binary coproduct classification, a coinvariant on that coproduct, and a pushout classification that is the coquotient for that coinvariant. However, these elements are phrased in terms of a pullback of the instance component and a pushout of the type component of the infomorphism pair. Here is the definition in more detail.

1. Construct the Set pullback $inst(\boldsymbol{B}_1) \times_{inst(A)} inst(\boldsymbol{B}_2)$, whose vertex set is

   $A = \{(b_1, b_2) \mid b_1 \in inst(\boldsymbol{B}_1), b_2 \in inst(\boldsymbol{B}_2), inst(\boldsymbol{f}_1)(b_1) = inst(\boldsymbol{f}_2)(b_2)\}$
   = pullback vertex in the category Set of $inst(\boldsymbol{f}_1)$ and $inst(\boldsymbol{f}_2)$

   a subset of the binary Cartesian product $inst(\boldsymbol{B}_1) \times inst(\boldsymbol{B}_2)$. Let the pullback "projections" be denoted

   first : $inst(\boldsymbol{P}) \rightarrow inst(\boldsymbol{B}_1)$ and *second* : $inst(\boldsymbol{P}) \rightarrow inst(\boldsymbol{B}_2)$.

2. Construct the Set pushout $typ(\boldsymbol{B}_1) +_{typ(A)} typ(\boldsymbol{B}_2)$, whose endorelation is

   $R = \{((1, typ(\boldsymbol{f}_1)(t)), (2, typ(\boldsymbol{f}_2)(t))) \mid \text{some } t \in typ(\boldsymbol{A})\}$
   = pushout endorelation in the category Set of $typ(\boldsymbol{f}_1)$ and $typ(\boldsymbol{f}_2)$

   whose object set is the binary disjoint union $typ(\boldsymbol{B}_1) + typ(\boldsymbol{B}_2)$. Let the pushout "injections" be denoted

   opfirst : $typ(\boldsymbol{B}_1) \rightarrow typ(\boldsymbol{P})$ and *opsecond* : $typ(\boldsymbol{B}_2) \rightarrow typ(\boldsymbol{P})$.

3. Define the coinvariant $J = \langle A, R\rangle$ on the coproduct $\boldsymbol{B}_1 + \boldsymbol{B}_2$.

   Checked by:

   $(b_1, b_2) \vDash_{1+2} (1, typ(\boldsymbol{f}_1)(t))$ <u>iff</u> $b_1 \vDash_1 typ(\boldsymbol{f}_1)(t)$ <u>iff</u> $inst(\boldsymbol{f}_1)(b_1) \vDash_{\boldsymbol{A}} t$

   <u>iff</u> $inst(\boldsymbol{f}_2)(b_2) \vDash_{\boldsymbol{A}} t$ <u>iff</u> $b_2 \vDash_2 typ(\boldsymbol{f}_2)(t)$ <u>iff</u> $(b_1, b_2) \vDash_{1+2} (2, typ(\boldsymbol{f}_2)(t))$

4. Specify the pushout to be the associated coquotient.

   $\boldsymbol{P} = (\boldsymbol{B}_1+\boldsymbol{B}_2)/J$
   $\boldsymbol{m}_1 = \langle first, opfirst\rangle = \boldsymbol{\iota}_1 \circ \boldsymbol{\tau}_J$
   $\boldsymbol{m}_2 = \langle second, opsecond\rangle = \boldsymbol{\iota}_2 \circ \boldsymbol{\tau}_J$

   Checked by:

   $inst(\boldsymbol{m}_1)(b_1, b_2) \vDash_1 t_1$ <u>iff</u> $b_1 \vDash_1 t_1$ <u>iff</u> $(b_1, b_2) \vDash_{1+2} (1, t_1)$ <u>iff</u> $(b_1, b_2) \vDash_{(1+2)/J} [(1, t_1)]_R$

   <u>iff</u> $(b_1, b_2) \vDash_{(1+2)/J} inst(\boldsymbol{m}_1)(t_1)$

We demonstrate universality.

1. The equality $typ(f_1) \circ typ(m_1) = typ(f_2) \circ typ(m_2)$ holds, since the type function of the canonical quotient infomorphism maps equivalent types in $typ(B_1+B_2)$ to the same type in $typ(P) = typ((B_1+B_2)/J) = typ(B_1+B_2)/R$. The equality $inst(m_1) \circ inst(f_1) = inst(m_2) \circ inst(f_2)$ holds, since $(b_1, b_2) \in inst(P) = A$ implies $inst(f_1)(inst(m_1)(b_1, b_2)) = inst(f_1)(b_1) = inst(f_2)(b_2) = inst(f_2)(inst(m_2)(b_1, b_2))$. So, $f_1 \circ m_1 = f_2 \circ m_2$.
2. Suppose C is any classification and $g_1 : B_1 \rightleftarrows C$ and $g_2 : B_2 \rightleftarrows C$ are a pair of infomorphisms with the common target classification C, that satisfy $f_1 \circ g_1 = f_2 \circ g_2$. Consider the coproduct copairing $[g_1, g_2] : B_1+B_2 \rightleftarrows C$. It is straightforward to check that the infomorphism $[g_1, g_2]$ respects the coinvariant J. So there is a unique infomorphism $g : P \rightleftarrows C$ satisfying $\tau_J \circ g = [g_1, g_2]$. Hence, $g : P \rightleftarrows C$ is a unique infomorphism satisfying $m_1 \circ g = g_1$ and $m_2 \circ g = g_2$.
3. In summary, pushouts can be constructed from coproducts and coinvariants.

   coproducts & coinvariants ⇒ pushouts.

   In the other direction, given two classifications A and B, the pushout of the initial infomorphisms $!_A : 0 \rightleftarrows A$ and $!_B : 0 \rightleftarrows B$ is the coproduct. So, coproducts can be constructed from pushouts and the initial classification.

   pushouts & initial objects ⇒ coproducts (and more strongly, finite cocompleteness).

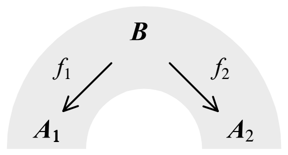


Figure 12: Pushout Diagram = Span

o A *pushout* is a finite limit for a diagram of shape • ← • → •. Such a diagram (of classifications and infomorphisms) is called an *span* (see Figure 12, where arrows denote infomorphisms) A *span* is the appropriate base diagram for a pushout. Each opspan consists of a pair of infomorphisms called *first* and *second*. These are required to have a common source classification B, denoted as the *vertex*. Let either ‘diagram’ or ‘span’ be the term that denotes the *Span* collection. Spans are determined by their pair of component infomorphisms.

```
(1) (CNG$conglomerate diagram)
    (CNG$conglomerate span)
    (= span diagram)

(2) (CNG$function classification1)
    (CNG$signature classification1 diagram CLS$classification)

(3) (CNG$function classification2)
    (CNG$signature classification2 diagram CLS$classification)

(4) (CNG$function vertex)
    (CNG$signature vertex diagram CLS$classification)

(5) (CNG$function first)
    (CNG$signature first diagram CLS.INFO$classification)

(6) (CNG$function second)
    (CNG$signature second diagram CLS.INFO$classification)

    (forall (?r (diagram ?r))
       (and (= (CLS.INFO$source (first ?r)) (vertex ?r))
            (= (CLS.INFO$source (second ?r)) (vertex ?r))
            (= (CLS.INFO$target (first ?r)) (classification1 ?r))
            (= (CLS.INFO$target (second ?r)) (classification2 ?r))))

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

- The *pair* of source classifications (suffixing discrete diagram) underlying any span (pushout diagram) is named.

```
(7) (CNG$function pair)
    (CNG$signature pair diagram CLS.COL.COPRD$diagram)
    (forall (?r (diagram ?r))
        (and (CLS.COL.COPRD$classification1 (pair ?r)) (classification1 ?r))
             (CLS.COL.COPRD$classification2 (pair ?r)) (classification2 ?r))))
```

- Every span has an opposite.

```
(8) (CNG$function opposite)
    (CNG$signature opposite span span)

    (forall (?r (span ?r))
       (and (= (classification1 (opposite ?r)) (classification2 ?r))
            (= (classification2 (opposite ?r)) (classification1 ?r))
            (= (vertex (opposite ?r)) (vertex ?r))
            (= (first (opposite ?r)) (second ?r))
            (= (second (opposite ?r)) (first ?r))))
```

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

```
    (forall (?r (span ?r))
       (= (opposite (opposite ?r)) ?r))
```

- There is an *instance opspan* or *instance diagram* function, which maps a span of infomorphisms to the underlying (`SET.LIM.PBK`) opspan of instance functions. Similarly, there is a *type span* or *type diagram* function, which maps a span of infomorphisms to the underlying (`SET.COLIM.COEQ`) span of type functions.

```
(9) (CNG$function instance-diagram)
    (CNG$function instance-opspan)
    (= instance-opspan instance-diagram)
    (CNG$signature instance-diagram diagram SET.LIM.PBK$diagram)
    (forall (?r (diagram ?r))
        (and (= (SET.LIM.PBK$class1 (instance-diagram ?r))
                (CLS$instance (classification1 ?r)))
             (= (SET.LIM.PBK$class2 (instance-diagram ?r))
                (CLS$instance (classification2 ?r)))
             (= (SET.LIM.PBK$opvertex (instance-diagram ?r))
                (CLS$instance (vertex ?r)))
             (= (SET.LIM.PBK$opfirst (instance-diagram ?r))
                (CLS.INFO$instance (first ?r)))
             (= (SET.LIM.PBK$opsecond (instance-diagram ?r))
                (CLS.INFO$instance (second ?r)))))

(10) (CNG$function type-diagram)
     (CNG$function type-span)
     (= type-span type-diagram)
     (CNG$signature type-diagram diagram SET.COL.PSH$diagram)
     (forall (?r (diagram ?r))
         (and (= (SET.COL.PSH$class1 (type-diagram ?r))
                 (CLS$type (classification1 ?r)))
              (= (SET.COL.PSH$class2 (type-diagram ?r))
                 (CLS$type (classification2 ?r)))
              (= (SET.COL.PSH$vertex (type-diagram ?r))
                 (CLS$type (vertex ?r)))
              (= (SET.COL.PSH$first (type-diagram ?r))
                 (CLS.INFO$type (first ?r)))
              (= (SET.COL.PSH$second (type-diagram ?r))
                 (CLS.INFO$type (second ?r)))))
```

- There is a *parallel pair* or *coequalizer diagram* function, which maps a span of infomorphisms to the associated (`CLS.COL.COEQ`) parallel pair of infomorphisms (see Figure 13, where arrows denote infomorphisms), which are the composite of the first and second infomorphisms of the span with the coproduct injection infomorphisms of the binary coproduct of the component

$$\begin{tikzcd} & \boldsymbol{B} \arrow[dl, "f_1"'] \arrow[d, shift left] \arrow[d, shift right] \arrow[dr, "f_2"] & \\ \boldsymbol{A_1} \arrow[r, "\iota_1"'] & \boldsymbol{A}_1+\boldsymbol{A}_2 & \boldsymbol{A}_2 \arrow[l, "\iota_2"] \end{tikzcd}$$

Figure 13: Coequalizer Diagram

classifications. The coequalizer and canon of the associated parallel pair will be used to define the pushout. This parallel-pair is represented by axiom (10).

```
(10) (CNG$function coequalizer-diagram)
     (CNG$function parallel-pair)
     (= parallel-pair coequalizer-diagram)
     (CNG$signature coequalizer-diagram diagram CLS.COL.COEQ$diagram)
     (forall (?r (diagram ?r))
         (and (= (CLS.COL.COEQ$source (coequalizer-diagram ?r))
                 (vertex ?r))
              (= (CLS.COL.COEQ$source (coequalizer-diagram ?r))
                 (CLS.COL.COPRD$binary-coproduct (pair ?r)))
              (= (CLS.COL.COEQ$infomorphism1 (coequalizer-diagram ?r))
                 (CLS.INFO$composition
                     (first ?r)
                     (CLS.COL.COPRD$injection1 (pair ?r))))
              (= (CLS.COL.COEQ$infomorphism2 (coequalizer-diagram ?r))
                 (CLS.INFO$composition
                     (second ?r)
                     (CLS.COL.COPRD$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.

```
     (forall (?r (diagram ?r))
         (and (= (CLS.COL.COEQ$instance-diagram
                     (CLS.COL.COEQ$colimiting-cocone (coequalizer-diagram ?r)))
                 (SET.LIM.EQU$limiting-cone
                     (SET.LIM.EQU$equalizer-diagram (instance-diagram ?r))))
              (= (CLS.COL.COEQ$type-diagram
                     (CLS.COL.COEQ$colimiting-cocone (coequalizer-diagram ?r)))
                 (SET.LIM.COEQ$colimiting-cocone
                     (SET.COL.COEQ$coequalizer-diagram (type-diagram ?r))))))
```

- *Pushout cocones* are used to specify and axiomatize pushouts. Each pushout cocone (Figure 14, where arrows denote infomorphisms) has an underlying *diagram* (the shaded part of Figure 14), an *opvertex* classification $\hat{A}$, and a pair of infomorphisms called *opfirst* and *opsecond*, whose common target classification is the opvertex and whose source classifications are the target classifications of the infomorphisms in the span. The opfirst and opsecond infomorphisms 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’ in axiom (9) is the term that denotes the *Pushout Cocone* collection. The term ‘cocone-diagram’ axiomatized in axiom (10) represents the underlying diagram.

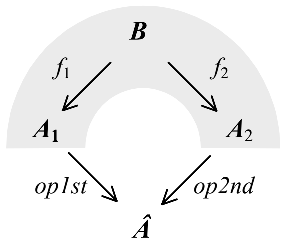

$$\begin{tikzcd} & B \arrow[dl, "f_1"'] \arrow[dr, "f_2"] & \\ A_1 \arrow[dr, "op1st"'] & & A_2 \arrow[dl, "op2nd"] \\ & \hat{A} & \end{tikzcd}$$

Figure 14: Pushout Cocone

```
(9) (CNG$conglomerate cocone)

(10) (CNG$function cocone-diagram)
     (CNG$signature cocone-diagram cocone diagram)

(11) (CNG$function opvertex)
     (CNG$signature opvertex cocone CLS$classification)

(12) (CNG$function opfirst)
     (CNG$signature opfirst cocone CLS.INFO$infomorphism)
     (forall (?s (cocone ?s))
        (and (= (CLS.INFO$source (opfirst ?s)) (classification1 (cocone-diagram ?s)))
             (= (CLS.INFO$target (opfirst ?s)) (opvertex ?s))))

(13) (CNG$function opsecond)
     (CNG$signature opsecond cocone CLS.INFO$infomorphism)
     (forall (?s (cocone ?s))
        (and (= (CLS.INFO$source (opsecond ?s)) (classification2 (cocone-diagram ?s)))
             (= (CLS.INFO$target (opsecond ?s)) (opvertex ?s))))

     (forall (?s (cocone ?s))
        (= (CLS.INFO$composition (first (cocone-diagram ?s)) (opfirst ?s))
           (CLS.INFO$composition (second (cocone-diagram ?s)) (opsecond ?s))))
```

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

```
(14) (CNG$function binary-coproduct-cocone)
     (CNG$signature binary-coproduct-cocone diagram CLS.COL.COPRD$cocone)
     (forall (?s (cocone ?s))
        (and (= (CLS.COL.COPRD$opvertex (binary-coproduct-cocone ?r)) (opvertex ?s))
             (= (CLS.COL.COPRD$opfirst (binary-coproduct-cocone ?r)) (opfirst ?s))
             (= (CLS.COL.COPRD$opsecond (binary-coproduct-cocone ?r)) (opsecond ?s))))
```

- There is a *coequalizer cocone* function, which maps a cocone of infomorphisms to the associated (`CLS.COL.COEQ`) coequalizer cocone of infomorphisms (see Figure 15, where arrows denote infomorphisms), which is the binary coproduct comediator of the opfirst and opsecond infomorphisms with respect to the coequalizer diagram of a cocone. This is the first step in the definition of the pushout comediator infomorphism. This coequalizer cocone is represented by axiom (14). The following string of equalities demonstrates that this cocone is well-defined.

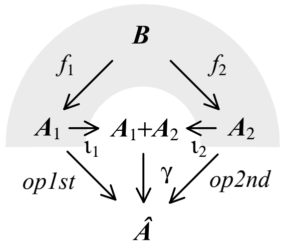


Figure 15: Coequalizer Cocone

$$f_1 \cdot \iota_1 \cdot \gamma = f_1 \cdot op1st = f_2 \cdot op1st = f_2 \cdot \iota_2 \cdot \gamma$$

```
(14) (CNG$function coequalizer-cocone)
     (CNG$signature coequalizer-cocone diagram CLS.COL.COEQ$cocone)
     (forall (?r (diagram ?r))
         (and (= (CLS.COL.COEQ$cocone-diagram (coequalizer-cocone ?r))
                 (coequalizer-diagram ?r))
              (= (CLS.COL.COEQ$opvertex (coequalizer-cocone ?r))
                 (opvertex ?r))
              (= (CLS.COL.COEQ$infomorphism (coequalizer-cocone ?r))
                 (CLS.COL.COPRD$comediator (binary-coproduct-cocone ?r)))))
```

- There is a unary CNG function '`colimiting-cocone`' that maps a span to its pushout (colimiting pushout cocone) (see Figure 16, where arrows denote infomorphisms). Axiom (*) gives the definition of this function in terms of the associated coequalizer diagram. This axiom ensures that this pushout is specific. 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* classification given by the CNG infomorphism '`pushout`'. It comes equipped with two CNG projection infomorphisms '`injection1`' and '`injection2`'.

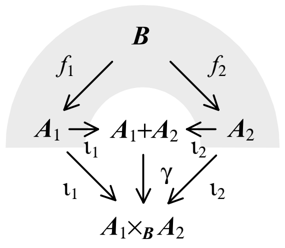


Figure 16: Colimiting Cocone

```
(15) (CNG$function colimiting-cocone)
     (CNG$signature colimiting-cocone diagram cocone)

 (*) (forall (?r (diagram ?r))
         (and (= (cocone-diagram (colimiting-cocone ?r)) ?r))
              (= (opvertex (colimiting-cocone ?r)) ?r))
                 (CLS.COL.COEQ$coequalizer (coequalizer-diagram ?r)))
              (= (opfirst (colimiting-cocone ?r)) ?r))
                 (CLS.INFO$composition
                     (CLS.COL.COPRD$injection1 (pair ?r))
                     (CLS.COL.COEQ$canon (coequalizer-diagram ?r))))
              (= (opsecond (colimiting-cocone ?r)) ?r))
                 (CLS.INFO$composition
                     (CLS.COL.COPRD$injection2 (pair ?r))
                     (CLS.COL.COEQ$canon (coequalizer-diagram ?r))))))

(16) (CNG$function colimit)
```

```
        (CNG$function pushout)
        (= pushout colimit)
        (CNG$signature colimit diagram CLS$classification)
        (forall (?r (diagram ?r))
           (= (colimit ?r) (opvertex (colimiting-cocone ?r))))

    (17) (CNG$function injection1)
        (CNG$signature injection1 diagram CLS.INFO$infomorphism)
        (forall (?r (diagram ?r))
           (and (= (CLS.INFO$source (injection1 ?r)) (classification1 ?r))
                (= (CLS.INFO$target (injection1 ?r)) (colimit ?r))
                (= (injection1 ?r) (opfirst (colimiting-cocone ?r)))))

    (18) (CNG$function injection2)
        (CNG$signature injection2 diagram CLS.INFO$infomorphism)
        (forall (?r (diagram ?r))
           (and (= (CLS.INFO$source (injection2 ?r)) (classification2 ?r))
                (= (CLS.INFO$target (injection2 ?r)) (colimit ?r))
                (= (injection2 ?r) (opsecond (colimiting-cocone ?r)))))
```

o There is a *comediator* infomorphism from the pushout of a span to the opvertex of a cocone over the span (see Figure 17, where arrows denote infomorphisms). This is the unique infomorphism that commutes with opfirst and opsecond.

```
    (19) (CNG$function comediator)
        (CNG$signature comediator cocone CLS.INFO$infomorphism)
        (forall (?s (cocone ?s))
            (and (= (CLS.INFO$source (comediator ?s))
                    (colimit (cocone-diagram ?s)))
                 (= (CLS.INFO$target (comediator ?s))
                    (opvertex ?s))
                 (= (comediator ?s)
                    (CLS.COL.COEQ$comediator
                        (coequalizer-cocone ?s)))))
```

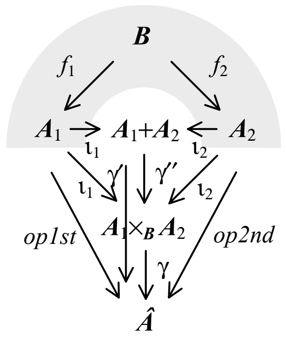

$$\begin{tikzcd}
 & \boldsymbol{B} \arrow[dl, "f_1"'] \arrow[dr, "f_2"] & \\
\boldsymbol{A}_1 \arrow[r, "\iota_1"'] \arrow[dr, "\iota_1"] \arrow[ddr, bend right, "op1st"'] & \boldsymbol{A}_1+\boldsymbol{A}_2 \arrow[d, "\gamma'"] \arrow[dd, bend right, "\gamma"'] & \boldsymbol{A}_2 \arrow[l, "\iota_2"] \arrow[dl, "\iota_2"'] \arrow[ddl, bend left, "op2nd"] \\
 & \boldsymbol{A}_1\times_{\boldsymbol{B}}\boldsymbol{A}_2 \arrow[d, "\gamma"] & \\
 & \hat{\boldsymbol{A}} &
\end{tikzcd}$$

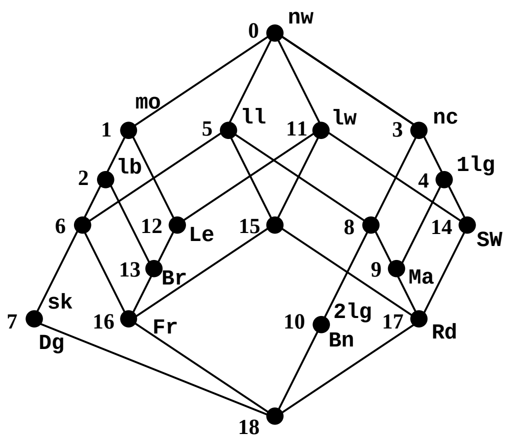
Figure 17: Comediator

Examples

The Living Classification

○ The *Living Classification* is a tiny dataset, which exists within a conceptual universe of living organisms. This classification listed below consists of eight organisms (plants and animals), and nine of their properties. The organisms are the *instances* of the classification, and the properties are the *types*. The classification relation is presented as a Boolean matrix in the following table. The living concept lattice, which contains 19 formal concepts, is presented in the following image.

Concept Lattice



Type Class

0	`nw`	needs water
1	`lw`	lives in water
2	`ll`	lives on land
3	`nc`	needs chlorophyll
4	`2lg`	2 leaf germination
5	`1lg`	1 leaf germination
6	`mo`	is motile
7	`lb`	has limbs
8	`sk`	suckles young

Instance Class

0	`Le`	Leech
1	`Br`	Bream
2	`Fr`	Frog
3	`Dg`	Dog
4	`SW`	Spike Weed
5	`Rd`	Reed
6	`Bn`	Bean
7	`Ma`	Maize

Classification Relation

	`nw`	`lw`	`ll`	`nc`	`2lg`	`1lg`	`mo`	`lb`	`sk`
`Le`	×	×					×		
`Br`	×	×					×	×	
`Fr`	×	×	×				×	×	
`Dg`	×		×				×	×	×
`SW`	×	×		×		×			
`Rd`	×	×	×	×		×			
`Bn`	×		×	×	×				
`Ma`	×		×	×		×			

The following table lists the formal concepts of the Living concept lattice in terms of their extent, intent, instance generators and type generators.

Formal Concepts				
	Generators			
Index	**Objects**	**Attributes**	**Extent**	**Intent**
0		needs water	{Leech, Bream, Frog, Dog, Spike Weed, Reed, Bean, Maize}	{needs water }
1		is motile	{Leech, Bream, Frog, Dog}	{needs water, is motile }
2		has limbs	{Bream, Frog, Dog}	{needs water, is motile, has limbs }

3		needs chlorophyll	{Spike Weed, Reed, Bean, Maize}	{needs water, needs chlorophyll }
4		1 leaf germination	{Spike Weed, Reed, Maize}	{needs water, needs chlorophyll, 1 leaf germination }
5		lives on land	{Frog, Dog, Reed, Bean, Maize}	{needs water, lives on land }
6			{Frog, Dog }	{needs water, lives on land, is motile, has limbs }
7	Dog	suckles young	{Dog }	{needs water, lives on land, is motile, has limbs, suckles young }
8			{Reed, Bean, Maize }	{needs water, lives on land, needs chlorophyll }
9	Maize		{Reed, Maize }	{needs water, lives on land, needs chlorophyll, 1 leaf germination }
10	Bean	2 leaf germination	{Bean }	{needs water, lives on land, needs chlorophyll, 2 leaf germination }
11		lives in water	{Leech, Bream, Frog, Spike Weed, Reed }	{needs water, lives in water }
12	Leech		{Leech, Bream, Frog }	{needs water, lives in water, is motile }
13	Bream		{Bream, Frog }	{needs water, lives in water, is motile, has limbs }
14	Spike Weed		{Spike Weed, Reed }	{needs water, lives in water, needs chlorophyll, 1 leaf germination }
15			{Frog, Reed }	{needs water, lives in water, lives on land }
16	Frog		{Frog}	{needs water, lives in water, lives on land, is motile, has limbs }
17	Reed		{Reed }	{needs water, lives in water, lives on land, needs chlorophyll, 1 leaf germination }
18			{} = ∅	{needs water, lives in water, lives on land, needs chlorophyll, 2 leaf germination, 1 leaf germination, is motile, has limbs, suckles young }

○ The following KIF represents the *Living Classification*.

```
(CLS$Classification Living)
((CLS$type Living) needs-water)
((CLS$type Living) lives-in-water)
((CLS$type Living) lives-on-land)
((CLS$type Living) needs-chlorophyll)
((CLS$type Living) 2-leaf-germination)
((CLS$type Living) 1-leaf-germination)
((CLS$type Living) is-motile)
((CLS$type Living) has-limbs)
((CLS$type Living) suckles-young)
((CLS$instance Living) Leech)
((CLS$instance Living) Bream)
((CLS$instance Living) Frog)
((CLS$instance Living) Dog)
((CLS$instance Living) Spike-Weed)
((CLS$instance Living) Reed)
```

```
((CLS$instance Living) Bean)
((CLS$instance Living) Maize)
...
((CLS$incidence Living) [Leech needs-water])
((CLS$incidence Living) [Leech lives-in-water])
(not ((CLS$incidence Living) [Leech lives-on-land]))
(not ((CLS$incidence Living) [Leech needs-chlorophyll]))
(not ((CLS$incidence Living) [Leech 2-leaf-germination]))
(not ((CLS$incidence Living) [Leech 1-leaf-germination]))
((CLS$incidence Living) [Leech is-motile])
(not ((CLS$incidence Living) [Leech has-limbs]))
(not ((CLS$incidence Living) [Leech suckles-young]))
...
```

○ The *Living Lattice* is the concept lattice of the Living Classification.

```
(CL$concept-lattice Living-Lattice)
(= Living-Lattice (CLS.CL$concept-lattice Living))
```

The Dictionary Classification

Here are examples of classifications and infomorphisms taken from the text *Information Flow: The Logic of Distributed Systems* by Barwise and Seligman.

○ The following KIF represents the *Webster Classification* on page 70 of Barwise and Seligman. This classification, which is (a small part of) the classification of English words according to parts of speech as given in Webster's dictionary, is diagrammed on the right.

Table 4: Webster Classification

Webster	Noun	Int-Vb	Tr-Vb	Adj
bet	1	1	1	0
eat	0	1	1	0
fit	1	1	1	1
friend	1	0	1	0
square	1	0	1	1
…			…	

```
(CLS$Classification Webster)
((CLS$type Webster) Noun)
((CLS$type Webster) Intransitive-Verb)
((CLS$type Webster) Transitive-Verb)
((CLS$type Webster) Adjective)
((CLS$instance Webster) bet)
((CLS$instance Webster) eat)
((CLS$instance Webster) fit)
((CLS$instance Webster) friend)
((CLS$instance Webster) square)
...
((CLS$incidence Webster) [bet Noun])
((CLS$incidence Webster) [bet Intransitive-Verb])
((CLS$incidence Webster) [bet Transitive-Verb])
(not ((CLS$incidence Webster) [bet Adjective]))
(not ((CLS$incidence Webster) [eat Noun]))
((CLS$incidence Webster) [eat Intransitive-Verb])
((CLS$incidence Webster) [eat Transitive-Verb])
(not ((CLS$incidence Webster) [fit Adjective]))
((CLS$incidence Webster) [fit Noun])
((CLS$incidence Webster) [fit Intransitive-Verb])
((CLS$incidence Webster) [fit Transitive-Verb])
((CLS$incidence Webster) [fit Adjective])
((CLS$incidence Webster) [friend Noun])
(not ((CLS$incidence Webster) [friend Intransitive-Verb]))
((CLS$incidence Webster) [friend Transitive-Verb])
(not ((CLS$incidence Webster) [friend Adjective]))
((CLS$incidence Webster) [square Noun])
(not ((CLS$incidence Webster) [square Intransitive-Verb]))
((CLS$incidence Webster) [square Transitive-Verb])
((CLS$incidence Webster) [square Adjective])
...
```

○ The following KIF represents the infomorphism defined on page 73 of Barwise and Seligman. This represents the way that punctuation at the end of a sentence carries information about the type of the sentence. The infomorphism is from a *Punctuation* classification to a *Sentence* classification. The instances of *Punctuation* are the inscriptions of the punctuation marks of English. These marks are classified by the terms ‘`Period`, ‘`Exclamation-Mark`’, ‘`Question-Mark`’, ‘`Comma`’, etc. The instances of *Sentence* are inscriptions of grammatical sentences of English. There are but three types of *Sentence*: ‘`Declarative`’, ‘`Question`’, and ‘`Other`’. The instance function of the infomorphism assigns to each sentence its own terminating punctuation mark. The type function of the infomorphism assigns ‘`Declarative`’ to ‘`Period`’ and ‘`Exclamation-Mark`’, ‘`Question`’ to ‘`Question-Mark`’, and ‘`Other`’ to other types of *Punctuation*. The fundamental property of this infomorphism is the requirement that a sentence be of the type indicated by its punctuation.

Let ‘`yakity-yak`’ denote the command “Take out the papers and the trash!” with its punctuation symbol ‘`yy-punc`’ being the exclamation symbol at the end of the sentence. Let ‘`gettysburg1`’ denote the statement that “Fourscore and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty and dedicated to the proposition that all men are created equal.” with its punctuation symbol ‘`g1-punc`’ being the period at the end of the sentence. Let ‘`angels`’ denote the question “How many angels can fit on the head of a pin?” with its punctuation symbol ‘`ag-punc`’ being the question mark at the end of the sentence.

```
(CLS$Classification Punctuation)
((CLS$type Punctuation) Period)
((CLS$type Punctuation) Exclamation-Mark)
((CLS$type Punctuation) Question-Mark)
((CLS$type Punctuation) Comma)
((CLS$instance Punctuation) yy-punc)
((CLS$instance Punctuation) g1-punc)
((CLS$instance Punctuation) ag-punc)
...
(not ((CLS$incidence Punctuation) [yy-punc Period]))
((CLS$incidence Punctuation) [yy-punc Exclamation-Mark])
(not ((CLS$incidence Punctuation) [yy-punc Question-Mark]))
...

(CLS$Classification Sentence)
((CLS$type Sentence) Declarative)
((CLS$type Sentence) Question)
((CLS$type Sentence) Other)
((CLS$instance Sentence) yakity-yak)
((CLS$instance Sentence) gettysburg1)
((CLS$instance Sentence) angels)
...
((CLS$incidence Sentence) [yakity-yak Declarative])
(not ((CLS$incidence Sentence) [yakity-yak Question]))
(not ((CLS$incidence Sentence) [yakity-yak Other]))
...
(CLS.INFO$Infomorphism punct-type)
(= (CLS.INFO$source punct-type) Punctuation)
(= (CLS.INFO$target punct-type) Sentence)

(= ((CLS.INFO$instance punct-type) yakity-yak) yy-punc)
(= ((CLS.INFO$instance punct-type) gettysburg1) g1-punc)
...
(= ((CLS.INFO$type punct-type) Period) Declarative)
(= ((CLS.INFO$type punct-type) Exclamation-Mark) Declarative)
(= ((CLS.INFO$type punct-type) Question-Mark) Question)
(= ((CLS.INFO$type punct-type) Comma) Other)
...
```

The Truth Classification

- The *truth classification* of a first-order language ***L*** is the large classification, whose instances are ***L***-structures (models), whose types are ***L***-sentences, and whose classification relation is satisfaction. Here we represent the truth classification in an external namespace. Note that the source is a class, whereas the target is a conglomerate – rather unusual. The image should then be just a class.

```
(CNG$function truth-classification)
(CNG$signature truth-classification lang$language CLS$classification)
(forall (?l (lang$language ?l))
    (and (= (CLS$instance (truth-classification ?l))  (MOD$fiber ?l))
         (= (CLS$type (truth-classification ?l))      (lang$sentence ?l))
         (= (CLS$incidence (truth-classification ?l)) (MOD$satisfaction ?l))))
```

- The *truth lattice* is the concept lattice of the truth meta-classification. This complete lattice functions as an appropriate “lattice of ontological theories.” A formal concept in this lattice has an intent that is a closed theory (set of sentences) and an extent that is the collection of all models for that theory. The join of two theories (concepts) is the intersection of the theories (intents), and the meet of two theories is the theory of the common models.

```
(CNG$function truth-lattice)
(CNG$signature truth-lattice lang$language CL$concept-lattice)
(forall (?l (lang$language ?l))
    (= (truth-lattice ?l)
       (CLS.CL$concept-lattice (truth-classification ?l))))
```