The IFF Ur (meta) Ontology

Ur

UR

Axiomatization

Diagram 1 illustrates the IFF Metastack (the IFF core hierarchy), where the IFF Ur (meta) Ontology (**IFF-UR**) is colored in gray. Figure 1 illustrates the architecture for the **IFF-UR**. The IFF Ur (meta) Ontology (**IFF-UR**) is a tiny ontology at the very top of the IFF core metalevel hierarchy. A design constraint is that "things are opaque"; that is, the detailed state of affairs "inside" anything is unknown. Such detail is to be provided by the using ontology, which is the IFF Top Core (meta) Ontology (**IFF-TCO**). The Ur metalanguage and architecture is a simple framework. The **IFF-TCO** fills it out.

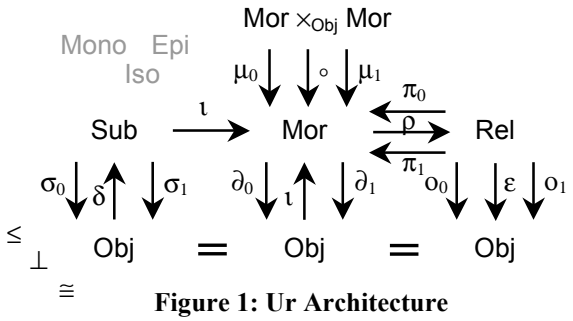
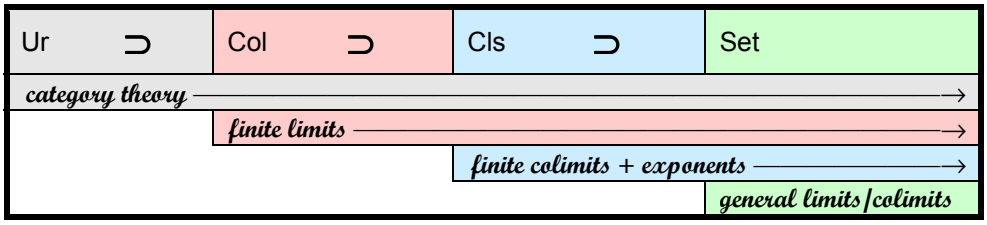


Figure 1: Ur Architecture

Diagram 1: The IFF Metastack



The terminology of the Ur metalanguage is listed in Table 1, which consists of only thirty terms: six generic sets; sixteen generic functions; and eight generic relations, consisting of three binary endorelation on objects, one binary and three unary relations on morphisms, and one binary relation on relations. The subobject, restriction and abridgment binary relations are important for the preservation properties of the IFF core hierarchy (the "IFF stack").

Table 1: The Ur Language; terms introduced in the Ur meta-ontology

	Set	Function	Relation
UR	**thing**		
	object		**subobject disjoint isomorphic**
	morphism **morphism-morphism**	**source target mor2rel** **morphism0 morphism1** **composition identity**	**restriction** **monomorphism epimorphism isomorphism**
	relation	**object0 object1 extent** **projection0 projection1**	**abridgment**
	subordinate	**lesser greater** **inclusion reflex**	

The complete Ur theory is expressed in the following axioms. The axioms specify the five basic kinds or sorts of *things*: *objects*, *subord*inate pairs of objects, *morphisms*, composable pairs of morphisms (*morphism-morphisms*) and *relations*. The axioms are partitioned accordingly.

Things

Things exist. Everything is either an *object*, a *morphism*, a composable-pair (*morphism-morphism*), a *relation* or a *subordinate*, but nothing can be any two of these.

```
(1) (forall (?x (thing ?x))
       (and (or (object ?x) (morphism ?x) (morphism-morphism ?x) (relation ?x) (subordinate ?x))
            (not (and (object ?x) (morphism ?x)))
            (not (and (object ?x) (morphism-morphism ?x)))
            (not (and (object ?x) (relation ?x)))
            (not (and (object ?x) (subordinate ?x)))
            (not (and (morphism ?x) (morphism-morphism ?x)))
            (not (and (morphism ?x) (relation ?x)))
            (not (and (morphism ?x) (subordinate ?x)))
            (not (and (morphism-morphism ?x) (relation ?x)))
            (not (and (morphism-morphism ?x) (subordinate ?x)))
            (not (and (relation ?x) (subordinate ?x)))))
```

Objects

Objects represent generic sets.

```
(1) (forall (?x (object ?x))
       (thing ?x))
```

There is a binary *subobject* preorder relationship between pairs of objects that are linked by a monomorphism. The subobject relation is reflexive and transitive, since identities are monomorphisms and monomorphisms are close under composition.

```
(2) (forall (?x (object ?x) ?y (object ?y))
       (<=> (subobject ?x ?y)
            (exists (?f (monomorphism ?f))
                (and (= ?x (source ?f)) (= ?y (target ?f)))))))

(3) (forall (?x (object ?x))
       (subobject ?x ?x))

(4) (forall (?x (object ?x) ?y (object ?y) ?z (object ?z))
       (=> (and (subobject ?x ?y) (subobject ?y ?z))
           (subobject ?x ?z)))
```

There is a binary *disjointness* relation between pairs of objects, that cannot be linked by a pair of monomorphisms. The disjointness relation is clearly symmetric.

```
(5) (forall (?x (object ?x) ?y (object ?y))
       (<=> (disjoint ?x ?y)
            (not (exists (?f (monomorphism ?f) ?g (monomorphism ?g))
                (and (= ?x (target ?f)) (= ?y (target ?g))
                     (= (source ?f) (source ?g)))))))

(6) (forall (?x (object ?x) ?y (object ?y))
       (=> (disjoint ?x ?y)
           (disjoint ?y ?x)))
```

There is a binary *isomorphism* equivalence relation between pairs of objects, that are linked by an isomorphism. The isomorphism relation is reflexive, symmetric and transitive, since identities are isomorphisms and monomorphisms are close under composition.

```
(7) (forall (?x (object ?x) ?y (object ?y))
       (<=> (isomorphic ?x ?y)
            (exists (?f (isomorphism ?f))
                (and (= ?x (source ?f)) (= ?y (target ?f)))))))

(8) (forall (?x (object ?x))
       (isomorphic ?x ?x))

(9) (forall (?x (object ?x) ?y (object ?y))
```

```
        (=> (isomorphic ?x ?y)
            (isomorphic ?y ?x)))

(10) (forall (?x (object ?x) ?y (object ?y) ?z (object ?z))
        (=> (and (isomorphic ?x ?y) (isomorphic ?y ?z))
            (isomorphic ?x ?z)))
```

Morphisms

Morphisms represent generic functions. Each morphism has a unique *source* object and a unique *target* object.

```
 (1) (forall (?x (morphism ?x))
        (thing ?x))

 (2) (forall (?f (morphism ?f))
        (and (exists (?y (object ?y))
                 (= (source ?f) ?y))
             (exists (?y0 (object ?y0) ?y1 (object ?y1)
                      (= (source ?f) ?y0) (= (source ?f) ?y1))
                 (= ?y0 ?y1))))

 (3) (forall (?f (morphism ?f))
        (and (exists (?y (object ?y))
                 (= (target ?f) ?y))
             (exists (?y0 (object ?y0) ?y1 (object ?y1)
                      (= (target ?f) ?y0) (= (target ?f) ?y1))
                 (= ?y0 ?y1))))
```

Two morphisms are *composable* when the target of the first is equal to the source of the second. The *composition* of two composable morphisms $f : A \to B$ and $g : B \to C$ is a morphism $f \cdot g : A \to C$. Hence, the source of the composite is the source of the first component, and the target of the composite is the target of the second component. Composition is associative.

```
 (4) (forall (?x (morphism-morphism ?x))
        (thing ?x))

 (5) (forall (?fg (morphism-morphism ?fg))
        (and (exists (?f (morphism ?f))
                 (= (morphism0 ?fg) ?f))
             (exists (?f0 (morphism ?f0) ?f1 (morphism ?f1)
                      (= (morphism0 ?fg) ?f0) (= (morphism0 ?fg) ?f1))
                 (= ?f0 ?f1))))

 (6) (forall (?fg (morphism-morphism ?fg))
        (and (exists (?g (morphism ?g))
                 (= (morphism1 ?fg) ?g))
             (exists (?g0 (morphism ?g0) ?g1 (morphism ?g1)
                      (= (morphism1 ?fg) ?g0) (= (morphism1 ?fg) ?g1))
                 (= ?g0 ?g1))))

 (7) (forall (?fg)
        (<=> (morphism-morphism ?fg))
             (= (target (morphism0 ?fg)) (source (morphism1 ?fg)))))

 (8) (forall (?fg (morphism-morphism ?fg))
        (and (exists (?h (morphism ?h))
                 (= (composition ?fg) ?h))
             (exists (?h0 (morphism ?h0) ?h1 (morphism ?h1)
                      (= (composition ?fg) ?h0) (= (composition ?fg) ?h1))
                 (= ?h0 ?h1))))

 (9) (forall (?fg (morphism-morphism ?fg))
        (and (= (source (composition ?fg))
                (source (morphism0 ?fg)))
             (= (target (composition ?fg))
                (target (morphism1 ?fg)))))

(10) (forall (?f (morphism ?f) ?g (morphism ?g) ?h (morphism ?h)
              (morphism-morphism ?fg) (morphism-morphism ?gh)
              (= ?f (morphism0 ?fg)) (= ?g (morphism1 ?fg))
```

```
            (= ?g (morphism0 ?gh)) (= ?h (morphism1 ?gh))
            (morphism-morphism ?f-gh)
            (= ?f (morphism0 ?f-gh)) (= (composition ?gh) (morphism1 ?f-gh))
            (morphism-morphism ?fg-h)
            (= (composition ?fg) (morphism0 ?fg-h)) (= ?h (morphism1 ?fg-h)))
        (= (composition ?f-gh)
           (composition ?fg-h)))
```

For every object there is an associated *identity* morphism. Composition with the identity of the source (target) of a morphism returns that morphism.

```
(11) (forall (?x (object ?x))
         (and (exists (?f (morphism ?f))
                  (= (identity ?x) ?f))
              (exists (?f0 (morphism ?f0) ?f1 (morphism ?f1)
                       (= (identity ?x) ?f0) (= (identity ?x) ?f1))
                  (= ?f0 ?f1))))

(12) (forall (?f (morphism ?f) (morphism-morphism ?f0) (morphism-morphism ?f1)
             (= (identity (source ?f)) (morphism0 ?f0))
             (= (identity (target ?f)) (morphism1 ?f1))
             (= ?f (morphism1 ?f0)) (= ?f (morphism0 ?f1))
         (and (= (composition ?f0) ?f)
              (= (composition ?f1) ?f)))
```

A morphism $f : A \to B$ is an *monomorphism* when for any two parallel morphisms $h_0, h_1 : D \to A$ the equality $h_0 \cdot f = h_1 \cdot f$ implies $h_0 = h_1$. A morphism $f : A \to B$ is an *epimorphism* when for any two parallel morphisms $g_0, g_1 : B \to C$ the equality $f \cdot g_0 = f \cdot g_1$ implies $g_0 = g_1$. A morphism $f : A \to B$ is an *isomorphism* when there is a morphism (in the opposite direction) $g : B \to A$ with $f \cdot g = 1_A$ and $g \cdot f = 1_B$. Monomorphisms, epimorphisms and isomorphisms are closed under composition. The identity is both a monomorphism and an epimorphism; hence also, an isomorphism.

```
(13) (forall (?f (morphism ?f))
         (<=> (monomorphism ?f)
              (forall (?z0 (morphism-morphism ?z0) ?z1 (morphism-morphism ?z1)
                       (= ?f (morphism1 ?z0)) (= ?f (morphism1 ?z1))
                       (= (composition ?z0) (composition ?z1)))
                  (= (morphism0 ?z0) (morphism0 ?z1))))))

(14) (forall (?f (morphism ?f))
         (<=> (epimorphism ?f)
              (forall (?z0 (morphism-morphism ?z0) ?z1 (morphism-morphism ?z1)
                       (= ?f (morphism0 ?z0)) (= ?f (morphism0 ?z1))
                       (= (composition ?z0) (composition ?z1)))
                  (= (morphism1 ?z0) (morphism1 ?z1))))))

(15) (forall (?f (morphism ?f))
         (<=> (isomorphism ?f)
              (exists (?g (morphism ?g))
                      ?fg (morphism-morphism ?fg) ?gf (morphism-morphism ?gf)
                       (= ?f (morphism0 ?fg)) (= ?g (morphism1 ?fg)
                       (= ?g (morphism0 ?gf)) (= ?f (morphism1 ?gf))
                  (and (= (composition ?fg) (identity (source ?f)))
                       (= (composition ?gf) (identity (target ?f)))))))))

(16) (forall (?f (monomorphism ?f) ?g (monomorphism ?g) (morphism-morphism ?fg)
             (= ?f (morphism0 ?fg)) (= ?g (morphism1 ?fg)))
         (monomorphism ?fg))

(17) (forall (?f (epimorphism ?f) ?g (epimorphism ?g) (morphism-morphism ?fg)
             (= ?f (morphism0 ?fg)) (= ?g (morphism1 ?fg)))
         (epimorphism ?fg))

(18) (forall (?f (isomorphism ?f) ?g (isomorphism ?g) (morphism-morphism ?fg)
             (= ?f (morphism0 ?fg)) (= ?g (morphism1 ?fg)))
         (isomorphism ?fg))

(19) (forall (?x (object ?x))
         (and (monomorphism (identity ?x))
              (epimorphism (identity ?x))
```

```
            (isomorphism (identity ?x))))
```

There is a *morphism to relation* injection that embeds morphisms as relations. The domain (codomain) of the relation of a morphism is its source (target). The domain projection is an isomorphism, and post-composition with the original morphism gives the codomain morphism.

```
(20) (forall (?f (morphism ?f))
         (and (exists (?r (relation ?r))
                  (= (mor2rel ?f) ?r))
              (exists (?r0 (relation ?r0) ?r1 (relation ?r1)
                       (= (mor2rel ?f) ?r0) (= (mor2rel ?f) ?r1))
                  (= ?r0 ?r1))
              (exists (?f0 (morphism ?f0) ?f1 (morphism ?f1)
                       (= (mor2rel ?f0) (mor2rel ?f1))
                  (= ?f0 ?f1))))

(21) (forall (?f (morphism ?f))
         (and (= (object0 (mor2rel ?f)) (source ?f))
              (= (object1 (mor2rel ?f)) (target ?f))
              (isomorphic (extent (mor2rel ?f)) (source ?f))
              (isomorphism (projection0 (mor2rel ?f)))
              (= (composition [(projection0 (mor2rel ?f)) ?f]) (projection1 (mor2rel ?f))))))
```

One morphism $m_1 : c_1 \to d_1$ is a *restriction* of another morphism $m_2 : c_2 \to d_2$ when the source (target) of m_1 is a subobject of the source (target) of m_2 and the morphisms agree (on source elements of m_1); that is, the morphisms commute with the domain/target inclusions. Restriction can be viewed as a constraint on the larger morphism – it says that the larger morphism maps the source object of the smaller morphism into the target object of the smaller morphism. There is a binary *restriction* preorder relationship between pairs of morphisms that are linked by source and target monomorphisms. The restriction relation is reflexive and transitive, since identities are monomorphisms and monomorphisms are close under composition.

```
(22) (forall (?m1 (morphism ?m1) ?m2 (morphism ?m2))
         (<=> (restriction ?m1 ?m2)
              (and (subobject (source ?m1) (source ?m1))
                   (subobject (target ?m1) (target ?m1))
                   (= (composition [?m1 (inclusion [(target ?m1) (target ?m1)])])
                      (composition [(inclusion [(source ?m1) (source ?m1)]) ?m2]))))))

(23) (forall (?m (morphism ?m))
         (restriction ?m ?m))

(24) (forall (?m1 (morphism ?m1) ?m2 (morphism ?m2) ?m3 (morphism ?m3))
         (=> (and (restriction ?m1 ?m2) (restriction ?m2 ?m3))
             (restriction ?m1 ?m3)))
```

Relations

Since the Ur ontology gives not just a morphic framework, but also a relational framework, we include some additional terminology and axiomatization.

Each relation has a unique *domain* object ($object_0$) and a unique *codomain* object ($object_1$).

```
 (1) (forall (?x (relation ?x))
         (thing ?x))

 (2) (forall (?r (relation ?r))
         (and (exists (?y (object ?y))
                  (= (object0 ?r) ?y))
              (exists (?y0 (object ?y0) ?y1 (object ?y1)
                       (= (object0 ?r) ?y0) (= (object0 ?r) ?y1))
                  (= ?y0 ?y1))))

 (3) (forall (?r (relation ?r))
         (and (exists (?y (object ?y))
                  (= (object1 ?r) ?y))
              (exists (?y0 (object ?y0) ?y1 (object ?y1)
                       (= (object1 ?r) ?y0) (= (object1 ?r) ?y1))
                  (= ?y0 ?y1))))
```

Each relation has a unique *extent* object.

```
(4) (forall (?r (relation ?r))
        (and (exists (?y (object ?y))
                 (= (extent ?r) ?y))
             (exists (?y0 (object ?y0) ?y1 (object ?y1)
                      (= (extent ?r) ?y0) (= (extent ?r) ?y1))
                 (= ?y0 ?y1))))
```

Each relation has a unique $projection_0$ morphism, whose source is the extent object and whose target is the domain $object_0$. Each relation has a unique $projection_1$ morphism, whose source is the extent object and whose target is the codomain $object_1$.

```
(5) (forall (?r (relation ?r))
        (and (= (source (projection0 ?r)) (extent ?r))
             (= (target (projection0 ?r)) (object0 ?r))
             (exists (?p (morphism ?p))
                 (= (projection0 ?r) ?p))
             (exists (?p0 (morphism ?p0) ?p1 (morphism ?p1)
                      (= (projection0 ?r) ?p0) (= (projection0 ?r) ?p1))
                 (= ?p0 ?p1))))

(6) (forall (?r (relation ?r))
        (and (= (source (projection1 ?r)) (extent ?r))
             (= (target (projection1 ?r)) (object1 ?r))
             (exists (?p (morphism ?p))
                 (= (projection1 ?r) ?p))
             (exists (?p0 (morphism ?p0) ?p1 (morphism ?p1)
                      (= (projection1 ?r) ?p0) (= (projection1 ?r) ?p1))
                 (= ?p0 ?p1))))
```

The notion of abridgment (Merriam-Webster) is that of "a shortened form of a work retaining the general sense and unity of the original". One relation *r* is an *abridgment* of another relation *s* when the domain $object_0$ and the codomain $object_1$ of *r* are subobjects of the domain object and the codomain object of *s*, respectively, and the extent of *r* is the "restriction" of the extent of *s* to the component objects of *r*. If relation *r* is an abridgment of relation *s*, then the extent of *r* is a subobject of the extent of *s*. There is a binary *abridgment* preorder relationship between pairs of relations, implicitly define in terms of the limit relationship between extents. The abridgment relation is reflexive and transitive.

```
(7) (forall (?r (relation ?r) ?s (relation ?s))
        (<=> (abridgment ?r ?s)
             (and (subobject (object0 ?r) (object0 ?s))
                  (subobject (object1 ?r) (object1 ?s))
                  (subobject (extent ?r) (extent ?s))
                  (forall (?o (object ?o)
                           ?m0 (morphism ?m0) ?m1 (morphism ?m1) ?m (morphism ?m)
                          (= (source ?mo) ?o) (= (target ?mo) (object0 ?r))
                          (= (source ?m1) ?o) (= (target ?m1) (object1 ?r))
                          (= (source ?m) ?o) (= (target ?m) (extent ?s))
                          (= (composition [?m0 (inclusion [(object0 ?r) (object0 ?s)])])
                             (composition [?m (projection0 ?s)]))
                          (= (composition [?m1 (inclusion [(object1 ?r) (object1 ?s)])])
                             (composition [?m (projection1 ?s)])))
                      (exists (?p (morphism ?p))
                          (= ?p (the (morphism ?n))
                                     (and (= (source ?n) ?o) (= (target ?n) (extent ?r))
                                          (= (composition [?n (projection0 ?r)]) ?m0)
                                          (= (composition [?n (projection1 ?r)]) ?m1)
                                          (= (composition
                                                 [?n (inclusion [(extent ?r) (extent ?s)])]) ?m)))))))))

(8) (forall (?r (relation ?r))
        (abridgment ?r ?r))

(9) (forall (?r (relation ?r) ?s (relation ?s) ?t (relation ?t))
        (=> (and (abridgment ?r ?s) (abridgment ?s ?t))
            (abridgment ?r ?t)))
```

Subordination

A pair of objects is *subordinate* when it satisfies the subobject relation. For each subordinate pair of objects, there is a unique *inclusion* morphism whose source is the *lesser object* and whose target is the *greater object*. Every object forms a subordinate pair with itself.

```
(1) (forall (?xy (subordinate ?xy))
        (and (thing ?xy)
             (subobject (lesser ?xy) (greater ?xy))))

(2) (forall (?x (object ?x) ?y (object ?y) (subobject ?x ?y))
        (exists (?xy (subordinate ?xy))
            (and (= ?x (lesser ?xy))
                 (= ?y (greater ?xy)))))

(3) (forall (?xy (subordinate ?xy))
        (and (exists (?x (object ?x))
                 (= (lesser ?xy) ?x))
             (exists (?x0 (object ?x0) ?x1 (object ?x1)
                      (= (lesser ?xy) ?x0) (= (lesser ?xy) ?x1))
                 (= ?x0 ?x1))))

(4) (forall (?xy (subordinate ?xy))
        (and (exists (?y (object ?y))
                 (= (greater ?xy) ?y))
             (exists (?y0 (object ?xy) ?y1 (object ?y1)
                      (= (greater ?xy) ?y0) (= (greater ?xy) ?y1))
                 (= ?y0 ?y1))))

(5) (forall (?x (object ?x))
        (and (exists (?xx (subordinate ?xx))
                 (= (reflex ?x) ?xx))
             (exists (?xx0 (subordinate ?x) ?xx1 (subordinate ?x)
                      (= (reflex ?x) ?xx0) (= (reflex ?x) ?xx1))
                 (= ?xx0 ?xx1))
             (= (lesser (reflex (?x)) ?x)
             (= (greater (reflex (?x)) ?x)))

(6) (forall (?xy (subordinate ?xy))
        (and (exists (?f (morphism ?f))
                 (= (inclusion ?xy) ?f))
             (exists (?f0 (morphism ?f0) ?f1 (morphism ?f1)
                      (= (inclusion ?xy) ?f0) (= (inclusion ?xy) ?f1))
                 (= ?f0 ?f1))
             (= (source (inclusion ?xy)) (lesser ?xy))
             (= (target (inclusion ?xy)) (greater ?xy))))
```