

# The Model Theory of Onto Logic

Robert E. Kent

This paper seeks to develop a rigorous categorical model theory for first-order ontological languages by using the principles and techniques of Information Flow and Formal Concept Analysis. Ontological languages represent ontologies and the terminologies of description logic.

<b>1. REVIEW</b> .....	<b>2</b>
<b>2. HYPERGRAPHS</b> .....	<b>4</b>
LIMITS FOR HYPERGRAPHS.....	5
<i>Colimits</i> .....	5
<i>Limits</i> .....	6
LANGUAGES.....	7
<b>3. EXPRESSIONS</b> .....	<b>8</b>
THE EXPRESSION MONAD.....	10
DESCRIPTION LOGIC.....	11
EXAMPLES: ONTOLOGY AND TERMINOLOGY.....	11
<i>African Animals Ontology</i> .....	11
<b>4. MODELS</b> .....	<b>13</b>
MODEL MORPHISMS.....	14
A NON-FUNCTOR.....	15
EXPRESSION MODELS.....	16
EXAMPLES: ONTOLOGY AND TERMINOLOGY.....	18
<i>Wine Drinkers Terminology</i> .....	18
<i>Commerce and Industry Ontology</i> .....	19
<b>5. LIMITS FOR MODELS</b> .....	<b>21</b>
A NON-ADJOINTNESS.....	21
PRODUCTS.....	21
<i>Fiber Sum Models: Subposition</i> .....	22
EQUALIZERS AND PULLBACKS.....	23
<b>6. THE TRUTH CLASSIFICATION</b> .....	<b>25</b>
<b>7. KNOWLEDGE REPRESENTATION</b> .....	<b>29</b>
CYC.....	29
<b>8. PROBLEMS</b> .....	<b>32</b>
MODEL THEORIES.....	32
<b>9. FUTURE WORK</b> .....	<b>33</b>
<b>10. REFERENCES</b> .....	<b>34</b>

# 1. Review

According to the theory of Information Flow (Barwise and Seligman, 1997), information presupposes a system of classification. A classification  $A = \langle inst(A), typ(A), \vDash_A \rangle$  consists of

1. a set,  $inst(A)$ , of things to be classified, called the *instances* of  $A$ ,
2. a set,  $typ(A)$ , of things used to classify the instances, called the *types* of  $A$ , and
3. a binary *classification* relation,  $\vDash_A$ , between  $inst(A)$  and  $typ(A)$ .

The notation  $a \vDash_A \alpha$  is read “instance  $a$  is of type  $\alpha$  in  $A$ .” For any instance  $a \in inst(A)$ , the *intent* or *type set* of  $a$  is the set  $typ_A(a) = \{\alpha \in typ(A) \mid a \vDash_A \alpha\}$ . For any type  $\alpha \in typ(A)$ , the *extent* or *instance set* of  $\alpha$  is the set  $inst_A(\alpha) = \{a \in inst(A) \mid a \vDash_A \alpha\}$ . The classification relation and the instance set function contain the same information. This idea gives an equivalent definition: a *classification*  $A = \langle inst(A), typ(A), inst_A \rangle$  consists of a set instances  $inst(A)$ , a set of types  $typ(A)$ , and an *instance function*  $inst_A : typ(A) \rightarrow \wp inst(A)$  that interprets types as (domains) subsets of instances.

Classifications are known as *formal contexts* in Formal Concept Analysis. The following motivating example appears in (Barwise and Seligman, 1997): Given a first-order language  $L$ , the *truth classification* of  $L$  has  $L$ -structures as instances, sentences of  $L$  as types, and classification relation defined by  $M \vDash s$  if and only if sentence  $s$  is true in structure  $M$ . An ontology forms a classification with either explicit subtyping or subsumption being the classification relation. Systemic examples of classifications also abound. Given any set  $A$  (of instances), the *instance powerset classification*  $\wp A = \langle A, \wp A, \in \rangle$  associated with  $A$  has elements of  $A$  as instances and subsets of  $A$  as types with the membership relation serving as the classification relation. Given any classification  $A = \langle inst(A), typ(A), \vDash_A \rangle$ , the *dual classification*  $A^\infty = \langle typ(A), inst(A), \vDash_A^\infty \rangle$  is the involution of  $A$ . This is the classification, whose instances are types of  $A$ , whose types are instances of  $A$ , and whose classification is the transpose of the  $A$  classification.

Infomorphisms connect classifications and provide a way to move information back and forth between classifications. An *infomorphism*  $\langle f, g \rangle : A \rightleftharpoons B$  from classification  $A$  to classification  $B$  is a contravariant pair of functions, a function  $g : typ(A) \rightarrow typ(B)$  in the forward direction between types and a function  $f : inst(A) \leftarrow inst(B)$  in the reverse direction between instances, satisfying the following fundamental property  $f(b) \vDash_A \alpha$  iff  $b \vDash_B g(\alpha)$  for each instance  $b \in inst(B)$  and each type  $\alpha \in typ(A)$ . Given any classification  $A = \langle inst(A), typ(A), \vDash_A \rangle$  the *instance infomorphism*  $\eta_A = \langle Id, inst_A \rangle : A \rightleftharpoons \wp(inst(A))$  has identity instance function and instance set type function. For any two sets (of instances)  $A$  and  $B$ , any function  $f : A \leftarrow B$  and its inverse image function  $f^{-1} : \wp A \rightarrow \wp B$  form a *powerset infomorphism*  $\wp f = \langle f, f^{-1} \rangle : \wp A \rightleftharpoons \wp B$  between the instance powerset classifications. Any infomorphism  $\langle f, g \rangle : A \rightleftharpoons B$  commutes with the powerset infomorphism of its instance function:  $\langle f, g \rangle \circ \eta_B = \eta_A \circ \wp f$ . Restricting to the type parts of this commuting diagram gives an equivalent definition: an *infomorphism*  $\langle f, g \rangle : A \rightleftharpoons B$  from classification  $A = \langle inst(A), typ(A), inst_A \rangle$  to classification  $B = \langle inst(B), typ(B), inst_B \rangle$  is a contravariant pair of functions, a function  $g : typ(A) \rightarrow typ(B)$  in the forward direction between types and a function  $f : inst(A) \leftarrow inst(B)$  in the reverse direction between instances, satisfying the commuting diagram  $g \cdot inst_B = inst_A \cdot f^{-1}$ .

There is an infomorphism  $\langle f, g \rangle : PA \rightleftharpoons ZF$  from the standard interpretation of Peano Arithmetic to Zermelo-Frankel set theory ( $ZF$ ), whose type function is a map  $g$  of the sentences of number theory into the sentences of set theory, and whose instance function is a map  $f$  (in the other direction) is a map of the interpretations of  $ZF$  into those of  $PA$ :  $M \vDash_{PA} g(s)$  iff  $f(M) \vDash_{ZF} s$  for any sentence of number theory  $s$  and any model of set theory  $M$ . (Barwise and Seligman, 1997) A continuous map  $g : X \rightarrow Y$  of topological space  $X = \langle X, top(X) \rangle$  into topological space  $Y = \langle Y, top(Y) \rangle$  is an infomorphism  $\langle g^{-1}, g \rangle : X \rightleftharpoons Y$  from classification  $X = \langle X, top(X), \ni_X \rangle$  to classification  $Y = \langle Y, top(Y), \ni_Y \rangle$ :  $x \in g^{-1}(O)$  iff  $g(x) \in O$  for any point  $x \in X$  and any open set  $O \in top(Y)$ .

Although infomorphisms are of main interest, there is another morphism of classifications that has some limited uses. An *projection*  $\langle f, g \rangle : A \rightleftharpoons B$  from classification  $A$  to classification  $B$  is a covariant pair of functions, a function  $g : typ(A) \rightarrow typ(B)$  in the forward direction between types and a function  $f : inst(A) \rightarrow inst(B)$  in the forward direction between instances, satisfying the following fundamental

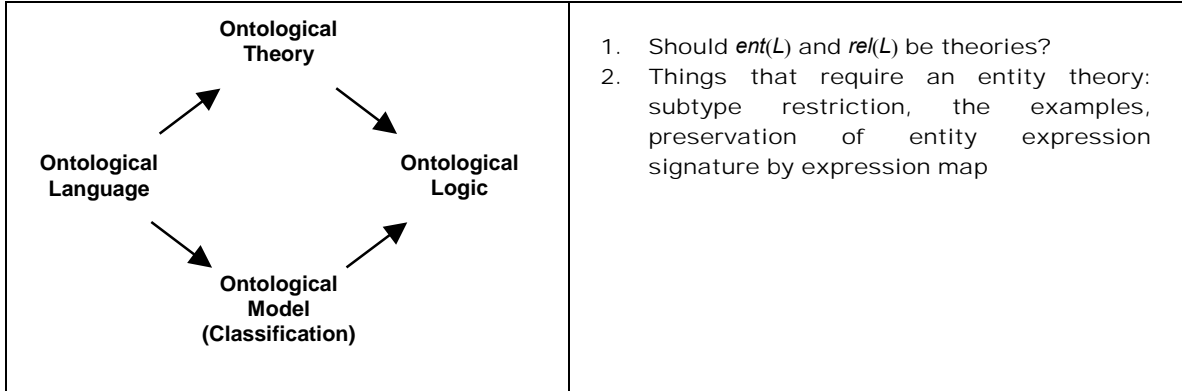
property  $a \models_A \alpha$  implies  $f(a) \models_B g(\alpha)$  for each instance  $a \in \text{inst}(A)$  and each type  $\alpha \in \text{typ}(A)$ . This notion of projection generalizes the notion of projection of state space as defined in Information Theory.

A *consistent subset*  $\Sigma \subseteq \text{typ}(T)$  of a (classification) theory  $T = \langle \text{typ}(T), \vdash_T \rangle$  is a subset of types that satisfies all constraints  $\Gamma \vdash_T \Delta$  in the theory  $T$ : if it contains all of the antecedent types  $\Gamma \subseteq \Sigma$ , then it contains some of the consequent types  $\Delta \cap \Sigma \neq \emptyset$ . Let  $\text{inst}(T)$  denote the set of all consistent subsets of  $T$ -types. A theory  $T$  determines a classification  $\text{Cla}(T) = \langle \text{inst}(T), \text{typ}(T), \exists_T \rangle$ , whose instances are consistent subsets of  $T$ -types, whose types are the types of  $T$ , and whose classification relation is defined by  $\Sigma \exists_T \alpha$  when  $\alpha \in \Sigma$ .  $\text{Cla}(T)$  is a sub-classification of the type power set classification  $\wp \text{typ}(T)$ .

When the theory functor **Th** : **Classification**  $\rightarrow$  **Theory** is applied to the powerset classification the result is the powerset theory **Th**( $\wp A$ ) =  $\langle \wp A, \vdash_A \rangle$  over  $A$ , where the meaning of a constraint  $\Gamma \vdash_A \Delta$  is given by the inclusion  $\cap \Gamma \subseteq \cup \Delta$ . For simplicity, we denote this theory by  $\wp A$  also. For any two sets connected by inclusion  $B \subseteq A$ , restriction to  $B$  is a theory interpretation  $\text{res} : \wp A \rightarrow \wp B$  that is left-adjoint-right-inverse to the inclusion theory interpretation  $\text{inc} : \wp B \rightarrow \wp A$ , between the underlying preorders. More generally, for any universal set  $A$ , there is a restriction functor  $\text{res} : \wp A \rightarrow \mathbf{Theory}^{\text{op}}$  and an inclusion functor  $\text{inc} : \mathbf{Theory} \rightarrow \wp A$  that are adjoint at each inclusion  $B \subseteq C$  of subsets of  $A$ .

Recall that the “free” logic functor **Log** : **Theory**  $\rightarrow$  **Logic** (add formal instances) is left adjoint  $\text{Log} \dashv \text{th}$  to the “underlying” theory functor  $\text{th} : \mathbf{Logic} \rightarrow \mathbf{Theory}$  (forget instances). Given any theory interpretation  $g : T \rightarrow \text{th}(\wp A)$  from a theory  $T = \langle \text{typ}(T), \vdash_T \rangle$  to the theory of a powerset logic  $\wp A = \langle A, \wp A, \vdash_A, \in_A \rangle$ , the adjoint logic infomorphism  $\langle \text{Id}, g \rangle : \mathbf{Log}(T) \simeq \wp A$  is the instance infomorphism from the logic  $\mathbf{Log}(T) = \langle A, \text{typ}(T), \vdash_T, \models_T \rangle$ , where  $a \models_T \alpha$  when  $a \in_A g(\alpha)$ .

## 2. Hypergraphs



In order to define ontological and information flow languages we assume an “adequate” (possibly denumerable) collection of variables  $var$ . Variables are used for indexing and naming the (functional) participants of relations. They correspond to the lines of identity and lambda variables of conceptual graphs. Tuples are important in what follows. For any set  $A$ , a *tuple*  $t$  of  $A$ -elements is a collection of (possibly duplicate)  $A$ -elements that are indexed by variable. The indexing elements of  $t$  form a set of variables. More precisely, a tuple  $t$  is a pair consisting of a set of variables  $arity(t) \subseteq var$  called the *arity* or *valence* of  $t$  and a map (function)  $pr(t) : arity(t) \rightarrow A$  called the *projector* of  $t$ . We use the exponential notation  $t \in A^{arity(t)}$  for tuples. A tuple morphism  $f: s \rightarrow t$  is a function  $f: arity(s) \rightarrow arity(t)$  satisfying  $pr(s) = f \cdot pr(t)$ . Although tuples are actually maps, we often use the tuple notation  $t = (t_x)_{x \in arity(t)}$  for a tuple. There is a *category of  $A$ -tuples*  $\mathbf{tuple}(A)$ . There is also a partial order on  $A$ -tuples:  $s \leq t$  when  $arity(s) \subseteq arity(t)$  and  $s_x = t_x$  for all  $x \in arity_L(s)$ . When two tuples match on their overlap  $arity(s) \cap arity(t)$  there is a *union tuple*  $s \cup t$  defined by:

$$\begin{aligned} (s \cup t)_x &= s_x \text{ for } x \in arity_L(s) - arity_L(t), \\ (s \cup t)_x &= t_x \text{ for } x \in arity_L(t) - arity_L(s), \text{ and} \\ (s \cup t)_x &= s_x = t_x \text{ for } x \in arity_L(s) \cap arity_L(t). \end{aligned}$$

If  $A^X$  is defined to be the set of  $A$ -tuples with arity  $X$ , then the collection of all  $A$ -tuples is the disjoint union  $\mathbf{tuple}(A) = \sum_{X \subseteq var} A^X$ . This is the object set of  $\mathbf{tuple}(A)$ . Let  $\mathbf{tuple}_A(t)$  denote the partial order of all  $A$ -tuples at or below  $t$ ; this is a finitely complete join semi-lattice with union and the empty tuple. For any classification  $A = \langle inst(A), typ(A), \models_A \rangle$  the *tuple classification over  $A$*  is the classification  $\mathbf{tuple}(A) = \langle \mathbf{tuple}(inst(A)), \mathbf{tuple}(typ(A)), \models_{\mathbf{tuple}(A)} \rangle$ , where  $a \models_A \alpha$  when  $arity(a) = arity(\alpha)$  and  $a_x \models \alpha_x$  for all variables  $x \in arity(\alpha)$ . For any infomorphism  $\langle f, g \rangle : A \rightleftharpoons B$  there is a tuple infomorphism  $\mathbf{tuple}(\langle f, g \rangle) : \mathbf{tuple}(A) \rightleftharpoons \mathbf{tuple}(B)$ . There is a tuple functor  $\mathbf{tuple} : \mathbf{Classification} \rightarrow \mathbf{Classification}$ . The notion of tuple allows us to define a notion that will play an important role in what follows – that of a hypergraph.

A *hypergraph* is a graph in which generalized edges (called hyperedges) connect multiple nodes. Formally, a *hypergraph*  $A = \langle node(A), edge(A), \partial_A \rangle$  consists of

- a set of nodes  $node(A)$ ,
- a set of hyperedges  $edge(A)$ , and
- a *vertex function*  $\partial_L : edge(A) \rightarrow \mathbf{tuple}(node(A))$  that maps a hyperedge  $\varepsilon \in edge(A)$  to its associated vertex  $\partial_A(\varepsilon) \in \mathbf{tuple}(node(A))$ .

A *hypergraph morphism*  $\langle n, e \rangle : A \rightarrow A'$  consists of a node function  $n : node(A) \rightarrow node(A')$  and an edge function  $e : edge(A) \rightarrow edge(A')$  that satisfy the hypergraph morphism condition expressed in terms of vertex functions as follows:  $n(\partial_x(\varepsilon)) = \partial_x'(e(\varepsilon))$  for each variable  $x \in arity_L(\partial(\varepsilon))$ . Or more concisely,  $\partial_x \cdot e = n \cdot \partial_x'$ . A hypergraph morphism  $\langle n, e \rangle : A \rightarrow A'$  has an associated

$$\begin{array}{ccc} edge(A) & \xrightarrow{e} & edge(A') \\ \partial_A \downarrow & & \downarrow \partial_{A'} \\ \mathbf{tuple}(node(A)) & \xrightarrow{\mathbf{tuple}(n)} & \mathbf{tuple}(node(A')) \\ node(A) & \xrightarrow{n} & node(A') \end{array}$$

vertex function  $\text{tuple}(n) : \text{tuple}(\text{node}(A)) \rightarrow \text{tuple}(\text{node}(A'))$  that preserves arity:  $\text{arity}_L(\text{tuple}(n)(v)) = \text{arity}_L(v)$ , and is defined by composition:  $\text{tuple}(n)(v)_x = n(v_x)$  for all  $x \in \text{arity}(v)$ . Or more concisely,  $\text{tuple}(n)(v) = v \cdot n$ . With vertex functions the hypergraph morphism condition can be expressed abstractly as  $\partial_A \cdot \text{tuple}(n) = e \cdot \partial_{A'}$ . The category **Hypergraph** has hypergraphs as objects and hypergraph morphisms as arrows. The *edge functor*  $\text{edge} : \mathbf{Hypergraph} \rightarrow \mathbf{Set}$  maps a hypergraph to its edge set, and maps a hypergraph morphism to its edge function.

## Limits for Hypergraphs

### Colimits

Let  $A$  and  $A'$  be two hypergraphs.

The *sum hypergraph*  $A+A' = \langle \text{node}(A)+\text{node}(A'), \text{edge}(A)+\text{edge}(A'), \partial_x \rangle$  is defined as follows.

- Its node set is the disjoint union of component node sets  $\text{node}(A)+\text{node}(A')$ .
- Its edge set is the disjoint union of component edge sets  $\text{edge}(A)+\text{edge}(A')$ .
- Its vertex map  $\partial_+ : \text{edge}(A)+\text{edge}(A') \rightarrow \text{tuple}(\text{node}(A)+\text{node}(A'))$  is defined in terms of the sum injections  $\text{in} : \text{node}(A) \rightarrow \text{node}(A)+\text{node}(A')$  and  $\text{in}' : \text{node}(A') \rightarrow \text{node}(A)+\text{node}(A')$  by  $\partial_+(\varepsilon) = \text{tuple}(\text{in})(\partial_A(\varepsilon)) = \partial_A(\varepsilon) \cdot \text{in}$  for edge  $\varepsilon \in \text{edge}(A)$  and  $\partial_+(\varepsilon') = \text{tuple}(\text{in}')(\partial_{A'}(\varepsilon')) = \partial_{A'}(\varepsilon') \cdot \text{in}'$  for edge  $\varepsilon' \in \text{edge}(A')$ .

The *sum injection hypergraph morphism*  $\text{in}_A = \langle \text{in}, \text{in} \rangle : A \rightarrow A+A'$  from the first component hypergraph  $A$  to the sum hypergraph  $A+A'$  has node sum injection as its node function and edge sum injection as its edge function. The hypergraph morphism condition holds, since  $\partial_+(\text{in}(\varepsilon))_x = \partial_A(\varepsilon)_x = \text{tuple}(\text{in})(\partial_A(\varepsilon))_x$  for any variable  $x \in \text{arity}(\varepsilon)$ . Let  $\langle n, e \rangle : A \rightarrow B$  and  $\langle n', e' \rangle : A' \rightarrow B$  be any two hypergraph morphisms with a common target. Then  $\langle [n, n'], [e, e'] \rangle : A+A' \rightarrow B$  is the unique mediating hypergraph morphism satisfying  $\text{in}_A \circ \langle n, n' \rangle, \langle e, e' \rangle = \langle n, e \rangle$  and  $\text{in}_{A'} \circ \langle n, n' \rangle, \langle e, e' \rangle = \langle n', e' \rangle$ , since it satisfies the hypergraph morphism  $[e, e'] \cdot \partial_B = \partial_{A+A'} \cdot \text{tuple}([n, n'])$ . So the sum  $A+A'$  is a coproduct in **Hypergraph**.

Given a hypergraph  $A$  a *hypergraph relation* on  $A$  is a pair  $J = \langle A, E \rangle$  consisting of a binary relation  $N \subseteq \text{node}(A) \times \text{node}(A)$  between nodes and a binary relation  $E \subseteq \text{edge}(A) \times \text{edge}(A)$  between edges, that satisfies the condition: if  $(\varepsilon_1, \varepsilon_2) \in T$ , then  $\text{arity}(\varepsilon_1) = \text{arity}(\varepsilon_2)$  and  $\partial_A(\varepsilon_1)_x \equiv_N \partial_A(\varepsilon_2)_x$  for all variables  $x \in \text{arity}(\varepsilon_1) = \text{arity}(\varepsilon_2)$ , where  $\equiv_N$  is the equivalence relation (reflexive, symmetric, transitive closure) generated by  $N$ . A simple inductive proof shows that the pair  $\langle \equiv_N, \equiv_E \rangle$  is also an invariant, where  $\equiv_E$  is the equivalence relation generated by  $E$ . The *hypergraph quotient of A*, written  $A/J = \langle \text{node}(A)/N, \text{edge}(A)/E, \partial_{A/J} \rangle$ , is the hypergraph with nodes the  $N$ -equivalence classes, edges the  $E$ -equivalence classes, and vertex function having the definition  $\partial_{A/J}([\varepsilon])_x = [\partial_A(\varepsilon)]_x$ , which is well-defined by the condition. The *canonical quotient hypergraph morphism*  $\tau_J = \langle [\ ]_N, [\ ]_E \rangle : A \rightarrow A/J$  consists of the canonical equivalence class quotient functions on nodes and edges. It satisfies the required condition:  $\partial_{A/J}([\varepsilon]_T) = \text{tuple}([\ ]_N)(\partial_A(\varepsilon))$ .

$$\begin{array}{ccc}
 \text{edge}(A) & \xrightarrow{[\ ]_E} & \text{edge}(A)/E \\
 \partial_A \downarrow & & \downarrow \partial_{A/J} \\
 \text{tuple}(\text{node}(A)) & \xrightarrow{\text{tuple}([\ ]_N)} & \text{tuple}(\text{node}(A)/N) \\
 \text{node}(A) & \xrightarrow{[\ ]_N} & \text{node}(A)/N
 \end{array}$$

Given a relation  $J = \langle A, T \rangle$  on a hypergraph  $A$ , a hypergraph morphism  $f = \langle n, e \rangle : A \rightarrow B$  respects  $J$  when: if  $(v_1, v_2) \in N$ , then  $n(v_1) = n(v_2)$ ; and if  $(\varepsilon_1, \varepsilon_2) \in T$ , then  $e(\varepsilon_1) = e(\varepsilon_2)$ . If  $f = \langle n, e \rangle : A \rightarrow B$  respects  $J$ , then it factors uniquely through the quotient: there is a unique hypergraph morphism  $\tilde{f} = \langle \tilde{n}, \tilde{e} \rangle : A/J \rightarrow B$  such that  $\tau_J \circ \tilde{f} = f$ . The necessary condition means that:  $[\ ]_N \cdot \tilde{n} = n$ , or  $\tilde{n}([\ ]_N) = n(v)$  for all nodes  $v \in \text{node}(A)$ ; and  $[\ ]_E \cdot \tilde{e} = e$ , or  $\tilde{e}([\varepsilon]_E) = e(\varepsilon)$  for all edges  $\varepsilon \in \text{edge}(A)$ . The constraints on  $f$  imply that  $\tilde{f}$  is a hypergraph morphism. The canonical quotient hypergraph morphism  $\tau_J : A/J \rightarrow A$  respects  $J$ , and its unique morphism is the identity.

Any hypergraph morphism  $f = \langle n, e \rangle : A \rightarrow A'$  determines a relation  $J(f) = \langle N, E \rangle$  on  $A$  defined as follows:

$N = \ker(n)$  is the kernel of  $n$  – the set of pairs of  $A$ -nodes  $(v_1, v_2)$  satisfying  $n(v_1) = n(v_2)$ ; and

$E = \ker(e)$  is the kernel of  $e$  – the set of pairs of  $A$ -edges  $(\varepsilon_1, \varepsilon_2)$  satisfying  $e(\varepsilon_1) = e(\varepsilon_2)$ .

The quotient hypergraph  $\text{img}(f) = A/J(f)$  is called the *image* of the hypergraph morphism  $f$ . The canonical morphism  $\varepsilon_f = \tau_{J(f)} : A \rightarrow \text{img}(f)$  is an epimorphism, and has a companion monomorphism  $\mu_f =$

$\langle \tilde{n}, \tilde{e} \rangle : \text{img}(f) \rightarrow A'$ , defined by  $\tilde{n}([v]_N) = n(v)$  and  $\tilde{e}([\varepsilon]_E) = e(\varepsilon)$ , which factor  $f: \mathbf{f} = \varepsilon_f \circ \mu_f$ . This is an image factorization for the category **Hypergraph**.

Two hypergraph morphisms  $f_1 = \langle n_1, e_1 \rangle : A' \rightarrow A$  and  $f_2 = \langle n_2, e_2 \rangle : A' \rightarrow A$  between the same models determine a relation  $J(f_1, f_2) = \langle N, E \rangle$  on  $A$  that is defined as follows:

- $N = \text{coeq}(n_1, n_2)$  is the coequalizer of  $n_1$  and  $n_2$ 
  - the set of pairs of  $A$ -nodes  $(n_1(v'), n_2(v'))$  for some  $A'$ -node  $v' \in \text{node}(A')$ ; and
- $E = \text{coeq}(e_1, e_2)$  is the coequalizer of  $e_1$  and  $e_2$ 
  - the set of pairs of  $A$ -edges  $(e_1(\varepsilon'), e_2(\varepsilon'))$  for some  $A'$ -edge  $\varepsilon' \in \text{edge}(A')$ .

The pair consisting of the quotient hypergraph  $\text{coeq}(f_1, f_2) = A/J(f_1, f_2)$  and the canonical morphism  $\tau_{J(f_1, f_2)} : A \rightarrow \text{coeq}(f_1, f_2)$  is called the *coequalizer* of the hypergraph morphism pair  $f_1$  and  $f_2$ . The canonical morphism is also called the “unique mediating hypergraph morphism” for the coequalizer. The coequalizer has the universal property that any hypergraph morphism  $f = \langle n, e \rangle : A \rightarrow B$  satisfying  $f_1 \circ f = f_2 \circ f$ , factors uniquely through the quotient: there is a unique hypergraph morphism  $\tilde{f} = \langle \tilde{n}, \tilde{e} \rangle : A/J \rightarrow B$  such that  $\tau_{J(f_1, f_2)} \circ \tilde{f} = f$ , since such a morphism respects the invariant  $J(f_1, f_2)$ . So,  $\text{coeq}(f_1, f_2)$  is the coequalizer in the category **Hypergraph**. Hence, **Hypergraph** has all colimits.

### Limits

Let  $A$  and  $A'$  be two hypergraphs.

The *product hypergraph*  $A \times A' = \langle \text{node}(A) \times \text{node}(A'), \text{edge}(A) \otimes \text{edge}(A'), \partial_x \rangle$  is defined as follows.

- Its node set is the Cartesian product of component node sets  $\text{node}(A) \times \text{node}(A')$ .
- Its edge set is the pullback of the component arity functions
 
$$\text{edge}(A) \otimes \text{edge}(A') = \{(\varepsilon, \varepsilon') \mid \varepsilon \in \text{edge}(A), \varepsilon' \in \text{edge}(A'), \text{arity}(\varepsilon) = \text{arity}(\varepsilon')\},$$
 which is the subset of those pairs of edges that have the same arity. Denote the inclusion function by
 
$$\text{inc} : \text{edge}(A) \otimes \text{edge}(A') \rightarrow \text{edge}(A) \times \text{edge}(A').$$
- Its vertex map  $\partial_x : \text{edge}(A) \otimes \text{edge}(A') \rightarrow \text{tuple}(\text{node}(A) \times \text{node}(A'))$  is defined componentwise by
 
$$\partial_x(\varepsilon, \varepsilon')_x = (\partial_A(\varepsilon)_x, \partial_{A'}(\varepsilon')_x)$$
 for any variable  $x \in \text{arity}(\varepsilon) = \text{arity}(\varepsilon')$ .

The *product projection hypergraph morphism*  $pr_A = \langle pr_A, \text{inc} \cdot pr_A \rangle : A \times A' \rightarrow A$  from the product hypergraph  $A \times A'$  to the first component hypergraph  $A$  has node product projection as its node function and the composition of pullback inclusion with edge product projection as its edge function. The hypergraph morphism condition holds, since  $\partial_A(pr_A(\text{inc}(\varepsilon, \varepsilon')))_x = \partial_A(\varepsilon)_x = \text{tuple}(pr_A)(\partial_A(\varepsilon, \varepsilon'))_x$  for any variable  $x \in \text{arity}(\varepsilon)$ . Let  $\langle n, e \rangle : B \rightarrow A$  and  $\langle n', e' \rangle : B \rightarrow A'$  be any two hypergraph morphisms with a common source. Define the hypergraph morphism  $\langle \tilde{n}, \tilde{e} \rangle : B \rightarrow A \times A'$  as follows:  $\tilde{n}(v) = v \cdot (n, n')$  for any node  $v \in \text{node}(B)$  and  $\tilde{e}(\zeta) = (e(\zeta), e'(\zeta))$  for any edge  $\zeta \in \text{edge}(B)$ . The edge function is well-formed since the equalities  $\partial_A(e(\zeta)) = \text{tuple}(n)(\partial_B(\zeta)) = \partial_B(\zeta) \cdot n$  and  $e'(\zeta) = \text{tuple}(n')(\partial_B(\zeta)) = \partial_B(\zeta) \cdot n'$  show that  $e(\zeta)$  and  $e'(\zeta)$  have the same arity. The hypergraph morphism condition  $\tilde{e} \cdot \partial_x = \partial_B \cdot \text{tuple}(\tilde{n})$  holds for  $\langle \tilde{n}, \tilde{e} \rangle$ , since the definition of the vertex map  $\partial_x$  implies  $\partial_x(\tilde{e}(\zeta))_x = (\partial_A(e(\zeta))_x, \partial_{A'}(e'(\zeta))_x) = (n(\partial_B(\zeta)_x), n'(\partial_B(\zeta)_x)) = \text{tuple}(\tilde{n})(\partial_B(\zeta))_x$  for any variable  $x \in \text{arity}(\zeta)$ . This is the unique mediating hypergraph morphism satisfying  $\langle \tilde{n}, \tilde{e} \rangle \circ pr_A = \langle n, e \rangle$  and  $\langle \tilde{n}, \tilde{e} \rangle \circ \text{inc}_{A'} = \langle n', e' \rangle$ . So the product  $A \times A'$  is a product in **Hypergraph**.

Given a hypergraph  $A$ , a *subhypergraph* of  $A$  is a pair  $I = \langle N, E \rangle$  consisting of a set  $N \subseteq \text{node}(A)$  of nodes and a set  $R \subseteq \text{edge}(A)$  of edges, that satisfies the following condition: if  $\varepsilon \in R$ , then  $\partial_A(\varepsilon)_x \in N$  for all variables  $x \in \text{arity}(\varepsilon)$ . The *subhypergraph*  $A/I = \langle N, E, \partial_{A/I} \rangle$  is the hypergraph with node set  $N$ , edge set  $E$ , and vertex function having the definition  $\partial_{A/I}(\varepsilon)_x = \partial_A(\varepsilon)_x$ , which is well-defined by the condition. The *canonical quotient hypergraph morphism*  $\tau_I = \langle \text{inc}_N, \text{inc}_E \rangle : A/I \rightarrow A$  consists of the inclusion functions on nodes and edges. It satisfies the required condition:  $\partial_A(\text{inc}_E(\varepsilon)) = \text{tuple}(\text{inc}_N)(\partial_{A/I}(\varepsilon))$ .

Given a subhypergraph  $I = \langle N, E \rangle$  of  $A$ , a hypergraph morphism  $f = \langle n, e \rangle : B \rightarrow A$  respects  $I$  when: for each  $v \in \text{node}(B)$ ,  $n(v) \in N$ ; and for each  $\varepsilon \in \text{edge}(B)$ ,  $e(\varepsilon) \in E$ . If  $f = \langle n, e \rangle : B \rightarrow A$  respects  $I$ , then it factors uniquely through the subhypergraph: there is a unique hypergraph morphism  $\tilde{f} = \langle \tilde{n}, \tilde{e} \rangle : B \rightarrow A/I$  such that  $\tilde{f} \circ \tau_I = f$ . The necessary condition means that  $\tilde{n} \cdot \text{inc}_N = n$ , or  $\tilde{n}(v) = n(v)$  for all nodes  $v \in N$ ; and  $\tilde{e} \cdot \text{inc}_E = e$ , or  $\tilde{e}(\varepsilon) = e(\varepsilon)$  for all edges  $\varepsilon \in E$ . The constraints on  $f$  imply that  $\tilde{f}$  is a hypergraph morphism.

$$\begin{array}{ccc}
 E & \xrightarrow{\text{inc}_E} & \text{edge}(A) \\
 \partial_{A/I} \downarrow & & \downarrow \partial_A \\
 \text{tuple}(N) & \xrightarrow{\text{tuple}(\text{inc}_N)} & \text{tuple}(\text{node}(A)) \\
 N & \xrightarrow{\text{inc}_N} & \text{node}(A')
 \end{array}$$

The canonical quotient hypergraph morphism  $\tau_I: A/I \rightarrow A$  respects  $I$ , and its unique morphism is the identity.

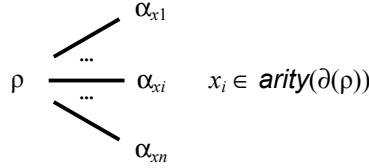
Two hypergraph morphisms  $f_1 = \langle n_1, e_1 \rangle: A \rightarrow A'$  and  $f_2 = \langle n_2, e_2 \rangle: A \rightarrow A'$  between the same models determine a subhypergraph  $I(f_1, f_2) = \langle N, R \rangle$  of  $A$  that is defined as follows:

- $N = \mathbf{eq}(n_1, n_2)$  is the equalizer of  $n_1$  and  $n_2$ 
  - the set of  $A$ -nodes  $v \in \mathbf{node}(A)$  such that  $n_1(v) = n_2(v)$ ; and
- $E = \mathbf{eq}(e_1, e_2)$  is the equalizer of  $e_1$  and  $e_2$ 
  - the set of  $A$ -edges  $\varepsilon \in \mathbf{edge}(A)$  such that  $e_1(\varepsilon) = e_2(\varepsilon)$ .

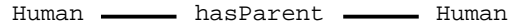
The pair consisting of the subhypergraph  $\mathbf{eq}(f_1, f_2) = A/I(f_1, f_2)$  and the canonical morphism  $\tau_{I(f_1, f_2)}: \mathbf{eq}(f_1, f_2) \rightarrow A$  is called the *equalizer* of the morphism pair  $f_1$  and  $f_2$ . The canonical quotient hypergraph morphism is also called the “unique mediating hypergraph morphism” for the equalizer. The equalizer has the universal property that any hypergraph morphism  $f = \langle n, e \rangle: B \rightarrow A$  satisfying  $f \circ f_1 = f \circ f_2$ , factors uniquely through the quotient: there is a unique hypergraph morphism  $\tilde{f} = \langle \tilde{n}, \tilde{e} \rangle: B \rightarrow A/I$  such that  $\tilde{f} \circ \tau_{I(f_1, f_2)} = f$ , since such a morphism respects the subhypergraph  $I(f_1, f_2)$ . So,  $\mathbf{eq}(f_1, f_2)$  is the equalizer in the category **Hypergraph**. Hence, **Hypergraph** has all limits.

## Languages

An *ontological language*  $L = \langle \mathbf{ent}(L), \mathbf{rel}(L), \partial_L \rangle$  is a hypergraph of types with entity types as nodes, relation types as hyperedges, and vertex function called the *signature function* of the language. Entity types are either object types (classes) or data types. Each relation type is further typed by its signature. The *external form* (*externalization* or *serialization*) for the relation type  $\rho \in \mathbf{rel}(L)$  with signature  $\partial_L(\rho) = \alpha$  is the type declaration string “ $\rho : (\alpha_x)_{x \in \mathbf{arity}(\partial(\rho))}$ ”. The existential graph for the relation type is given as follows.



A *language morphism*  $\langle e, r \rangle: L \rightarrow L'$  is a hypergraph morphism, consisting of an entity type function  $e: \mathbf{ent}(L) \rightarrow \mathbf{ent}(L')$  and a relation type function  $r: \mathbf{rel}(L) \rightarrow \mathbf{rel}(L')$  that satisfy the hypergraph morphism condition expressed in terms of signature functions as follows:  $\partial_{L'} \cdot \mathbf{tuple}(e) = r \cdot \partial_L$ . As an example, the following existential graph (Sowa, 2000) expresses the parenthood relationship.



Here there are words representing predicates (entity types) “Human” and relations (relation types) “hasParent”, and bars (lines of identity) representing individuals. The existential graph is rendered in the algebraic notation of logic as

$$(\exists x)(\exists y)(\text{Human}(x) \wedge \text{Human}(y) \wedge \text{hasParent}(x, y))$$

where each bar is represented by a variable  $x$  and  $y$  and an existential operator. Other variable names could have been chosen, such as *child* and *parent* which are more intuitive and which better represent the participation:

$$(\exists \text{child})(\exists \text{parent})(\text{Human}(\text{child}) \wedge \text{Human}(\text{parent}) \wedge \text{hasParent}(\text{child}, \text{parent})).$$

Variables can represent place-holding slots and also the participation functions of reification. In terms of ontological languages, there is an entity type  $\alpha = \text{Human} \in \mathbf{ent}(L)$  and a relation type  $\rho = \text{hasParent} \in \mathbf{rel}(L)$  that has arity  $\mathbf{arity}(\partial(\rho)) = \{x, y\}$  or  $\mathbf{arity}(\partial(\rho)) = \{\text{child}, \text{parent}\}$  and signature  $\partial(\rho) = (\text{Human}: x, \text{Human}: y)$ .

### 3. Expressions

An *expression* is the act, process, or instance of representing in a medium (as words); something that manifests, embodies, or symbolizes something else; a significant word or phrase; a mathematical or logical symbol or a meaningful combination of symbols. It can also be a mode, means, or use of significant representation or symbolism – especially a felicitous or vivid indication or depiction of mood or sentiment. In order to formalize a first-order ontological language  $L$ , we must define an adequate notion of expression. For this we use the following logical symbols: *parentheses* ‘()’, ‘{}’, ‘[]’; *propositional connectives* ‘∧’ (and), ‘∨’ (or), ‘¬’ (not), ‘→’ (implies), ‘↔’ (equivalence); *quantifiers* ‘∀’ (for all), ‘∃’ (exists); and a *type equivalence* symbol ‘≡’.

Signatures are used to facilitate the definition of expressions. The expressions that we consider here are type-oriented, since all the variables in any expression have a specified type. This very definitely restricts expressions to those that are well formed from a type-theoretic standpoint. An *expression*  $\varphi$  is a pair – it consists of an untyped expression *body*( $\varphi$ ) called the *body* (*internal form*) of  $\varphi$  and a signature  $\partial_L(\varphi) \in \mathit{ent}(L)^{\mathit{arity}(\varphi)}$  called the *sign* of  $\varphi$ . The arity of the signature is called the *arity* of the expression  $\mathit{arity}(\varphi) = \mathit{arity}(\partial_L(\varphi))$ . An expression also has an *external form* (*externalization* or *serialization*) *form*( $\varphi$ ), which is a string used for external communication. Expressions are used in the terminological definition of relation types; for example,  $\rho : (\alpha_x)_{x \in \mathit{arity}(\alpha)} \triangleq \mathit{form}(\varphi)$ . We distinguish propositional expressions from quantified expressions because of their effect upon model morphisms – propositional expressions have no restriction, whereas quantified expressions require model morphisms to have monomorphic (injective) instance functions. For simplicity of presentation we ambiguously use the symbol *expr* to denote either propositional or quantified expressions, but where the difference is important they will be distinguished. The set of expressions  $\mathit{expr}(L)$  and its extended *signature* map  $\partial_L : \mathit{expr}(L) \rightarrow \mathit{sign}(L)$  are defined inductively.

The most basic expressions are relational expressions (typed variable tuples). Unary relational expressions correspond to the *concepts* of Description Logics and Conceptual Graphs (except for description referents). (These concepts are also subsumed into the signatures of expressions.) Binary relational expressions correspond to the *roles* of Description Logic. Relational expressions with arbitrary arity correspond to the *conceptual relations* of Conceptual Graphs. Any relation type  $\rho \in \mathit{rel}(L)$  with signature  $\partial_L(\rho) = \alpha$  is a relational expression  $(\rho, \alpha) \in \mathit{expr}(L)$  with body  $\rho$  with signature  $\alpha$ . This relational expression represents the instance of classification  $(x)_{x \in \mathit{arity}(\partial_L(\rho))} \models_A \rho$  – relation instances are tuples of entity instances. The *canonical external form* (*externalization* or *serialization*) for this relational expression is the string “ $(\bigwedge_{x \in \mathit{arity}(\partial_L(\rho))} \alpha(x)) \wedge \rho(x)_{x \in \mathit{arity}(\partial_L(\rho))}$ ”. The *embedded external form* (*externalization* or *serialization*) is the string “ $\rho(\alpha(x))_{x \in \mathit{arity}(\partial_L(\rho))}$ ”. There is an injection  $r2e_L : \mathit{rel}(L) \rightarrow \mathit{expr}(L)$  that preserves signature:  $r2e_L \cdot \partial_L = \partial_L$ .

If  $\Phi$  is a collection of expressions with pairwise compatible signatures (they agree on their overlap), then the *conjunction* of  $\Phi$ , symbolized in this paper by  $\wedge\Phi$ , has a body that is the pair  $(\wedge, \{\mathit{body}(\varphi) \mid \varphi \in \Phi\})$  whose first element is a conjunction tag and whose second element is the set of component expression bodies; it has a signature  $\bigcup_{\varphi \in \Phi} \partial_L(\varphi)$  that is the union of the component signatures; and it has a form “ $\wedge \cdot \bullet_{\varphi \in \Phi} \mathit{form}(\varphi)$ ” whose first character is the conjunction symbol and whose remaining substring is the concatenation of the forms of the component expressions. The *disjunction* of  $\Phi$ , symbolized in this paper by  $\vee\Phi$ , is handled similarly. If  $\varphi$  and  $\psi$  are two expressions with compatible signatures (they agree on their overlap), then the *implication* with antecedent  $\varphi$  and consequent  $\psi$ , symbolized in this paper by  $(\varphi \rightarrow \psi)$ , has a body that is the pair  $(\rightarrow, (\mathit{body}(\varphi), \mathit{body}(\psi)))$  whose first element is an implication tag and whose second element is the antecedent and consequent body pair; it has a signature  $\partial_L(\varphi) \cup \partial_L(\psi)$  that is the union of the component signatures; and it has a form “ $\rightarrow \cdot \mathit{form}(\varphi) \cdot \mathit{form}(\psi)$ ” whose first character is the implication symbol and whose remaining substring is the concatenation of the antecedent and consequent forms. The *equivalence* of  $\varphi$  and  $\psi$ , symbolized in this paper by  $(\varphi \leftrightarrow \psi)$ , is handled similarly. If  $\varphi$  is an expression, then the *negation* of  $\varphi$ , symbolized in this paper by  $\neg\varphi$ , has body  $(\neg, \mathit{body}(\varphi))$ , signature  $\partial_L(\varphi)$ , and form “ $\neg \cdot \mathit{form}(\varphi)$ ”. Expressions can be restricted to subtypes: if  $\varphi$  is an expression and  $\alpha$  is a signature satisfying  $\alpha \vdash \partial_L(\varphi)$ , in the sense that  $\mathit{arity}_L(\alpha) = \mathit{arity}(\varphi)$  and  $\alpha_x \vdash \partial_L(\varphi)_x$  for all  $x \in \mathit{arity}_L(\alpha)$ , then the *subtype restriction* of  $\varphi$  with respect to  $\alpha$ , symbolized in this paper by  $\varphi[\alpha]$ , has a body  $([], (\alpha, \mathit{body}(\varphi)))$  whose first element is a subtype restriction tag and whose second element is a pair

consisting of the restricting signature and the expression body; it has the signature  $\alpha$ ; and it has a form “*form*( $\varphi$ )” · “[” ·  $\alpha$  · “]” that is the concatenation of brackets, the string representing the restricting subtype, and the form of the expression. Nothing else is an expression.

All expressions are quantified expressions (with no contained quantifiers). If  $\varphi$  is a quantified expression and  $x \in \text{arity}(\varphi)$ , then the *universal quantification* of  $\varphi$  with respect to  $x$ , symbolized in this paper by  $(\forall x)\varphi$ , has the meaning “for all instances  $x$ , if  $x$  is of type  $\partial_L(\varphi)(x)$  then  $\varphi(x)$  is true”; it has a body that is the pair  $(\forall, (x, \text{body}(\varphi)))$  whose first element is a universal quantification tag and whose second element the pair of quantified variable and expression body; it has a signature that is the restriction of  $\partial_L(\varphi)$  to  $\text{arity}_L(\varphi) - \{x\}$ ; and it has a form “(” · “ $\forall$ ” ·  $x$  · “)” · *form*( $\varphi$ ) that is the concatenation of parentheses, the universal quantification symbol, the string representing the quantified variable, and the form of the expression. The *existential quantification* of  $\varphi$  with respect to  $x$ , symbolized in this paper by  $(\exists x)\varphi$ , is handled similarly. Nothing else is a quantified expression.

Here are some examples of expressions.

- The first example, taken from the paper on “Subsumption and Taxonomy” by William Woods, illustrates quantified expressions. Consider the natural language expressions “*Every person lives in some place*” and “*Only people live in apartments*”. Here there are at least four entity types ‘Organism’, ‘Person’, ‘Place’ and ‘Apartment’ with the constraints ‘Person  $\vdash$  Organism’, ‘Apartment  $\vdash$  Place’ and ‘{Organism, Place}  $\vdash$  { }’. There is one binary relation type involved: ‘lives(Organism, Place)’. The two ontologic expressions representing these natural language expressions have empty signature – they are assertions.

$$\begin{aligned} & (\forall o:\text{Organism})(\exists p:\text{Place})(\text{Person}(o) \rightarrow \text{lives}(o,p)) \\ & (\forall o:\text{Organism})(\forall p:\text{Place})((\text{Apartment}(p) \wedge \text{lives}(o,p)) \rightarrow \text{Person}(o)) \end{aligned}$$

The next two examples come from the conceptual graphs standard.

- The second example is a simple disjunction of two predicates. We convert the disjunctive statement “*Either Sam has a car or Sam rides a bus*” into the predicate disjunction “*Sam is a car driver or Sam is a bus rider*”. This could be converted to the body of an expression by replacing the name “*Sam*” with the variable  $x$ : ‘Car driver’( $x$ ) $\vee$ ‘Bus rider’( $x$ ). Although the variable  $x$  links the predicate ‘Car driver’( $x$ ) with the predicate ‘Bus rider’( $x$ ), for expressions we also want a linking entity type. A natural one would be the type Person. We regard ‘Car Driver’ and ‘Bus Rider’ to be unary relation types with signature Person. These facts are represented by the assertions “*a car driver is a person*” and “*a bus rider is a person*”. We can construct an ontologic expression in a bottom-up fashion. There is an expression with body ‘Car driver’( $x$ ) and signature (Person:  $x$ ). Similarly, there is an expression with body ‘Bus rider’( $x$ ) and signature (Person:  $x$ ). Since the signatures of these two expressions agree on their overlap, there is an expression whose body is the above disjunction and whose signature is (Person:  $x$ ).
- The third example is the expression for the sentence “*John is going to Boston*”, which could be converted to the external form of an expression by replacing the name “*John*” with the variable  $x$  and the name “*Boston*” with the variable  $y$ , and reifying the binary multivalent relation type  $\text{go}(\text{Agent}, \text{Destination})$  into the entity type Go and the participant relation types  $\text{agent}(\text{Act}, \text{Animate})$  and  $\text{destination}(\text{Act}, \text{Place})$ :

$$(\exists z)(\text{Person}(x) \wedge \text{agent}(z, x) \wedge \text{Go}(z) \wedge \text{destination}(z, y) \wedge \text{City}(y)).$$

The body of this expression is more easily identified from its canonical form in conceptual graphs:

$$\begin{aligned} & ( \\ & \quad \text{lambda (Person *x, City *y)} \\ & \quad (\exists z)(\text{Go}(z) \wedge \text{agent}(z, x) \wedge \text{destination}(z, y)) \\ & ) \end{aligned}$$

This reveals the body  $(\exists z)(\text{Go}(z) \wedge \text{agent}(z, x) \wedge \text{destination}(z, y))$  and the signature (Person:  $x$ , City:  $y$ ). We can construct this expression in a bottom-up fashion. Consider the relation type  $\text{agent}(\text{Act}: z, \text{Animate}: x)$ . The type of the entity in its first argument must be “Act” or some subtype such as “Go  $\vdash$  Act”, and the type of the entity in its second argument must be “Animate” or some subtype such as “Person  $\vdash$  Animate”. Using subtype restriction of expressions, there is an expression whose body is the relation  $\text{agent}(z, x)$  and whose signature is (Go:  $z$ , Person:  $x$ ). Etc.

In the theory of conceptual graphs, lambda expressions are used to define new concepts and conceptual relations. In the conceptual graphs standard an n-adic lambda expression has the following canonical form:

$$\begin{aligned} & ( \\ & \quad \text{lambda } (\alpha_1 *x_1, \dots, \alpha_n *x_n) \end{aligned}$$

conceptual graph

where the keyword **lambda** is followed by the *signature* of the expression and then a conceptual graph called its *body*. The signature is a list of  $n$  formal parameters, each consisting of an entity type  $\alpha_i$  and a defining label  $x_i$ . Clearly, lambda expressions in conceptual graphs are closely related to expressions in onto logic.

## The Expression Monad

For any ontological language  $L = \langle \mathit{ent}(L), \mathit{rel}(L), \partial_L \rangle$  expressions of  $L$  form a new ontological language  $\mathit{expr}(L) = \langle \mathit{ent}(L), \mathit{expr}(L), \partial_L \rangle$ . There is a language morphism  $\eta_L = \langle \mathit{Id}, r2e_L \rangle : L \rightarrow \mathit{expr}(L)$  called the *expression unit at  $L$* , and a language morphism  $\mu_L = \langle \mathit{Id}, e^22e_L \rangle : \mathit{expr}(\mathit{expr}(L)) \rightarrow \mathit{expr}(L)$  called the *expression multiplication at  $L$* . The function  $e^22e_L : \mathit{expr}(\mathit{expr}(L)) \rightarrow \mathit{expr}(L)$  changes expressions of expressions into expressions by erasing the boundary between outer and inner expressions.

A language morphism  $\langle e, r \rangle : L \rightarrow L'$  determines an inductively defined *expression function*  $\mathit{expr}(e, r) : \mathit{expr}(L) \rightarrow \mathit{expr}(L')$ . At the base level it extends the relation function  $\mathit{expr}(e, r)(\rho) = r(\rho)$ . On Booleans, subtype restriction and quantifiers they are defined in terms of their component expressions:

$$\begin{aligned} \mathit{expr}(e, r)(\wedge\Phi) &= \wedge_{\Phi \in \Phi} \mathit{expr}(e, r)(\Phi); \\ \mathit{expr}(e, r)(\vee\Phi) &= \vee_{\Phi \in \Phi} \mathit{expr}(e, r)(\Phi); \\ \mathit{expr}(e, r)(\Phi \rightarrow \Psi) &= \mathit{expr}(e, r)(\Phi) \rightarrow \mathit{expr}(e, r)(\Psi); \\ \mathit{expr}(e, r)(\Phi \leftrightarrow \Psi) &= \mathit{expr}(e, r)(\Phi) \leftrightarrow \mathit{expr}(e, r)(\Psi); \\ \mathit{expr}(e, r)(\neg\Phi) &= \neg \mathit{expr}(e, r)(\Phi); \\ \mathit{expr}(e, r)(\Phi[\alpha]) &= \mathit{expr}(e, r)(\Phi)[\mathit{tuple}(e)(\alpha)]; \\ \mathit{expr}(e, r)((\forall x)\Phi) &= (\forall x)\mathit{expr}(e, r)(\Phi); \text{ and} \\ \mathit{expr}(e, r)((\exists x)\Phi) &= (\exists x)\mathit{expr}(e, r)(\Phi). \end{aligned}$$

$$\begin{array}{ccc} \mathit{rel}(L) & \xrightarrow{r} & \mathit{rel}(L') \\ \downarrow \mathit{r2e} & & \downarrow \mathit{r2e}' \\ \mathit{expr}(L) & \xrightarrow{\mathit{expr}(e, r)} & \mathit{expr}(L') \\ \downarrow \partial_L & & \downarrow \partial_{L'} \\ \mathit{tuple}(\mathit{ent}(L)) & \xrightarrow{\mathit{tuple}(e)} & \mathit{tuple}(\mathit{ent}(L')) \end{array}$$

The expression functions preserves signature  $\partial_{L'}(\mathit{expr}(e, r)(\Phi)) = \mathit{tuple}(e)(\partial_L(\Phi))$ . There is a language morphism  $\mathit{expr}(e, r) = \langle e, \mathit{expr}(e, r) \rangle : \mathit{expr}(L) \rightarrow \mathit{expr}(L')$ . This defines an *expression functor*  $\mathit{expr} : \mathbf{Language} \rightarrow \mathbf{Language}$  on languages (it will be lifted to theories after we define evaluation). The unit at  $L$  is the  $L$ -th component of an *expression unit* natural transformation  $\eta : \mathit{Id}_{\mathbf{Language}} \Rightarrow \mathit{expr}$ , and the multiplication at  $L$  is the  $L$ -th component of an *expression multiplication* natural transformation  $\mu : \mathit{expr} \cdot \mathit{expr} \Rightarrow \mathit{expr}$ . The triple  $\mathit{expr} = \langle \mathit{expr}, \eta, \mu \rangle$  is a monad over the category **Language** called the *expression monad*.

The following notion of an ontologic interpretation expands on the infomorphism example of ‘‘Interpretations in First-Order Logic’’ in (Barwise and Seligman, 1997). Let  $L$  and  $L'$  be ontological languages. An *ontologic interpretation* or *terminology*  $\langle e, r \rangle : L \rightarrow \mathit{expr}(L') = \langle \mathit{ent}(L'), \mathit{expr}(L'), \partial_{L'} \rangle$  is a language morphism from the language  $L$  to the expression language of  $L'$ .

This consists of an entity type function  $e : \mathit{ent}(L) \rightarrow \mathit{ent}(L')$  and a relation type function  $r : \mathit{rel}(L) \rightarrow \mathit{expr}(L')$  that interprets relation types of  $L$  as expressions of  $L'$ . These satisfy the hypergraph morphism condition expressed in terms of signature functions as follows:  $e(\partial_x(\rho)) = \partial_{x'}(r(\rho))$  for each relation type  $\rho \in \mathit{rel}(L)$  and each variable  $x \in \mathit{arity}_L(\partial(\rho))$ . Ontologic interpretations generalize language morphisms – one example of a useful ontologic interpretation is the composition  $\langle e, r \rangle^\Delta = \langle e, r \rangle \circ \eta_{L'} = \langle e, r \rangle \circ \langle \mathit{Id}, r2e_{L'} \rangle : L \rightarrow L' \rightarrow \mathit{expr}(L')$  for some language morphism  $\langle e, r \rangle : L \rightarrow L'$ . Using expression composition at  $L'$  (multiplication of the expression monad) an ontologic interpretation  $\langle e, r \rangle : L \rightarrow \mathit{expr}(L')$  lifts to a language morphism between expression languages  $\langle e, r \rangle^\# = \mathit{expr}(e, r) \circ \mu_{L'} : \mathit{expr}(L) \rightarrow \mathit{expr}(\mathit{expr}(L')) \rightarrow \mathit{expr}(L')$ .

$$\begin{array}{ccc} \mathit{rel}(L) & \xrightarrow{r} & \mathit{expr}(L') \\ \downarrow \partial_L & & \downarrow \partial_{L'} \\ \mathit{tuple}(\mathit{ent}(L)) & \xrightarrow{\mathit{tuple}(e)} & \mathit{tuple}(\mathit{ent}(L')) \\ \mathit{ent}(L) & \xrightarrow{e} & \mathit{ent}(L') \end{array}$$

**Diagram: Terminology**

There is a category  $\mathbf{Language}_{\mathit{Expr}}$ , called the *Kliesli category* of the expression monad  $\mathit{expr}$ , whose objects are ontologic languages and whose morphisms are ontologic interpretations. Ontologic interpretations compose: given two ontologic interpretations  $\langle e, r \rangle : L \rightarrow \mathit{expr}(L')$  and  $\langle e', r' \rangle : L' \rightarrow \mathit{expr}(L'')$ , define the composition  $\langle e, r \rangle \circ \langle e', r' \rangle = \langle e, r \rangle \circ \langle e', r' \rangle^\# : L \rightarrow \mathit{expr}(L'')$ . The unit  $\eta_L = \langle \mathit{Id}, r2e_L \rangle : L \rightarrow \mathit{expr}(L)$  is the identity ontologic interpretation at  $L$ .

## Description Logic

Description logic languages are sublanguages of ontological languages that are used as a means for expressing taxonomic knowledge by describing hierarchies of entity types (concepts). These descriptions of defined entity types are given in terms of primitive entity types and relations (roles, or attributes). A defined entity type is built up from the primitive entity and relation types by using various language constructs. Different description logics are distinguished by the constructs that they provide. A basic language for Description Logic is the language *ALC* (Schmidt-Schauss and Smolka, 1988) that offered the Boolean connectives and role quantification as constructs.

DL Construct	Onto Logic Expression	Meaning
$A$	$\alpha$	primitive entity type
$\top, \perp$	$\top, \perp$	top and bottom types
$A \sqcap B, A \sqcup B, \neg A$	$\alpha \wedge \beta, \alpha \vee \beta, \neg \alpha$	Boolean connectives
$\exists R.A, \forall R.A$	$(\exists y)(\rho(x, y) \wedge \alpha(y)), (\forall y)(\rho(x, y) \rightarrow \alpha(y))$	role quantification

## Examples: Ontology and Terminology

Ontologies correspond to knowledge bases. An ontology has two basic parts: a schema (**T-Box**) and a set of assertions (**A-Box**). In more detail, an ontology has the following components. There is a hierarchy of entity types structured as nested IF theories. There is always one generic theory at the top of the hierarchy. Other theories are anchored at particular entity types, with these as their root type. There are collections of binary relation types. There are terminologies consisting of coherent collections of defined types. There are classifications. There are collections of other assertions. These assertions can be grouped according to relations, functions or tables. The following examples illustrate ontologies. Several things are illustrated here: multi-inheritance of facets; operations of Description Logic; and intrinsic vs. extrinsic relations via binary relation, object and data types.

### African Animals Ontology

The following example from the paper (Fensel, Horrocks, et al, 2000) illustrates how description logic is represented in onto logic. The topic is African animals. This simple example was used to illustrate the expressive power of the XML application Ontology Interchange Language (OIL). The ontological language, ontological model and terminology components were extracted from the discussion of the paper. Note that instances (individuals and arguments) and classification relationships are not included in ontological languages, since expressions do not include instances.

### Symbols used

$\rightarrow$	Subtype
$:=$	Definition
$\langle \rangle$	Anonymous type defined expression
$\wedge \vee \neg$	Booleans
$\exists \forall$	Quantifiers
$\leftrightarrow$	Logical equivalence
$\equiv$	Type equivalence

### OIL Constructs

OIL Expression	Onto Logic Expression	Meaning
<b>Class-def</b> $\sigma$	$arity(\sigma) = \{x\}$ , a singleton	Declaration of the entity type $\sigma$ .
primitive $\sigma$	$\sigma \rightarrow$	This declares $\sigma$ to be a primitive entity type.
defined $\sigma$	$\sigma :=$	What follows is the definition of entity type $\sigma$ .
$\langle \dots \rangle$	Create an anonymous type.	$\dots$ is an expression that defines an anonymous type.
$\alpha$ subclass-of $\beta$	$(\forall x)(\sigma(x) \rightarrow \tau(x))$	Entity type $\alpha$ is a subtype of entity type $\beta$ .
slot-constraint $\rho$	$arity(\rho) = \{x, y\}$ , a doubleton	$\rho$ is a binary relation type.
$\rho$ has-value $\alpha$	$(\exists y)(\rho(x, y) \wedge \alpha(y))$	Binary relation type $\rho$ has possible entity type $\alpha$ for its second argument.
$\rho$ value-type $\alpha$	$(\forall y)(\rho(x, y) \rightarrow \alpha(y))$	Binary relation type $\rho$ has required entity type $\alpha$ for its second argument.
$\rho$ inverse $\zeta$	$(\forall x)(\zeta(x, y) \rightarrow \rho(y, x))$	Binary relation type $\zeta$ is the transpose of binary relation $\rho$ type.

*The Language  $L = L'$  and Terminology  $\langle e, r \rangle$*

The components are indexed according to the (left hand side of the) Terminology Diagram.

**Terminology  $\langle e, r \rangle : L \rightarrow \text{expr}(L)$**

**Example ontology**

Natural Language expression	Onto logic expression
“An animal is a thing.”	$\text{Animal}(x) \rightarrow \text{Thing}(x)$
“A plant is a thing that is not an animal.”	$\text{Plant}(x) \rightarrow \langle \text{Thing}(x) \wedge \neg \text{Animal}(x) \rangle$
“A tree is a plant.”	$\text{Tree}(x) \rightarrow \text{Plant}(x)$
“A branch is part of a tree.”	$\text{Branch}(x) \rightarrow \langle \text{Thing}(x) \wedge (\exists y)(\text{'is part of'}(x,y) \wedge \text{Tree}(y)) \rangle$
“A leaf is part of a branch.”	$\text{Leaf}(x) \rightarrow \langle \text{Thing}(x) \wedge (\exists y)(\text{'is part of'}(x,y) \wedge \text{Branch}(y)) \rangle$
“A carnivore is an animal that only eats animals.”	$\text{Carnivore}(x) := \text{Animal}(x) \wedge (\forall y)(\text{eats}(x,y) \rightarrow \text{Animal}(y))$
“A herbivore is an animal that is not a carnivore and that eats only plants or parts of plants.”	$\text{Herbivore}(x) := \text{Animal}(x) \wedge \neg \text{Carnivore}(x) \wedge (\forall y)(\text{eats}(x,y) \rightarrow (\text{Plant}(y) \vee (\exists z)(\text{'is part of'}(y,z))))$
“A giraffe is an animal that eats only leaves.”	$\text{Giraffe}(x) \rightarrow \langle \text{Animal}(x) \wedge (\forall y)(\text{eats}(x,y) \rightarrow \text{Leaves}(y)) \rangle$
“A lion is an animal that eats only herbivores.”	$\text{Lion}(x) \rightarrow \langle \text{Animal}(x) \wedge (\forall y)(\text{eats}(x,y) \rightarrow \text{Herbivore}(y)) \rangle$
“A tasty plant is a plant that is eaten by both herbivores and carnivores.”	$\text{'Tasty Plant'}(x) \rightarrow \langle \text{Plant}(x) \wedge (\exists y)(\text{'is eaten by'}(x,y) \wedge \text{Herbivore}(y)) \wedge (\exists z)(\text{'is eaten by'}(x,z) \wedge \text{Carnivore}(z)) \rangle$

## 4. Models

A “possible world” or model for an ontological language provides an interpretation for all types. It traditionally associates sets (domains) with entity types and relations with relation types. These associations must respect the various typings. The notion of model introduced in this paper is framed in terms of Information Flow. However, these ontological models can be placed in the traditional perspective.

Abstraction	Classification-projection diagram
$\begin{array}{c} \mathbf{rel}(A) \\ \Downarrow \langle \partial_0, \partial_1 \rangle \\ \mathbf{tuple}(\mathbf{ent}(L)) \end{array}$	$\begin{array}{ccc} & \partial_1 & \\ \mathbf{rel}(A) & \longrightarrow & \mathbf{tuple}(\mathbf{ent}(A)) \\ \vDash_A \downarrow & & \downarrow \mathbf{tuple}(\vDash_A) \\ \mathbf{arg}(A) & \xrightarrow{\partial_0} & \mathbf{tuple}(\mathbf{indiv}(A)) \end{array}$

A *ontological model*  $A = \langle \mathbf{ent}(A), \mathbf{rel}(A), \partial \rangle$  is either a hypergraph of classifications or a classification of hypergraphs. It is a two dimensional structure consisting of

- a *hypergraph of instances*  $\mathbf{inst}(A) = \langle \mathbf{indiv}(A), \mathbf{arg}(A), \partial_0 \rangle$ ;
- a *hypergraph of types*  $\mathbf{typ}(A) = \langle \mathbf{ent}(A), \mathbf{rel}(A), \partial_1 \rangle$ ;
- an *entity classification*  $\mathbf{ent}(A) = \langle \mathbf{indiv}(A), \mathbf{ent}(A), \vDash_A \rangle$ ;
- a *relation classification*  $\mathbf{rel}(A) = \langle \mathbf{arg}(A), \mathbf{rel}(A), \vDash_A \rangle$ ; and
- a *tuple projection*  $\partial = \langle \partial_0, \partial_1 \rangle : \mathbf{rel}(A) \Rightarrow \mathbf{tuple}(\mathbf{ent}(A))$ .

The set  $\mathbf{indiv}(A)$  is called the *universal domain*. Elements of  $\mathbf{indiv}(A)$  are called *individual designators*. Individual designators, thought of as identifiers for individuals, represent either objects or data values. A locator represents an object: a marker “ISBN-0-521-58386-1”, an indexical “you” or a name “K. Jon Barwise”. A data value is represented as a literal: a string “xyz”, a number “3.14159”, a date “1776/07/04”, etc. Elements of the set  $\mathbf{arg}(A)$  are called (*relational*) *arguments*. The type hypergraph is regarded as an ontological language. From the involutory view of classifications, the instance hypergraph is dual to the type hypergraph. The instance hypergraph consists of

1. a set of entity instances (individual designators)  $\mathbf{indiv}(A)$ ,
2. a set of (multivalent) relation instances (arguments)  $\mathbf{arg}(A)$ , and
3. an *entry map*  $\partial_0 : \mathbf{arg}(A) \rightarrow \mathbf{entry}(A)$  that maps an argument  $t \in \mathbf{arg}(A)$  to its associated entry  $\partial_0(t) \in \mathbf{entry}(A)$ .

An *entry*  $a \in \mathbf{indiv}(A)^{\mathbf{arity}(a)}$  is a tuple of  $A$ -individuals. Thus, it is an individual assignment. The collection of all entries of  $A$  is denoted  $\mathbf{entry}(A) = \mathbf{tuple}(\mathbf{indiv}(A))$ , with  $\mathbf{entry}_A(t)$  denoting the partial order of all entries at or below  $t$ . Entries by themselves are typeless and have no signatures; however, when a universal entity type  $\tau$  exists, the data in entries is *a priori* of only one type, that universal type. Entries are associated with arguments via the entry map, and each argument has an arity defined as the arity of its entry. Define  $A^X = \{t \in \mathbf{arg}(A) \mid \mathbf{arity}(t) = \mathbf{arity}(\partial_0(t)) = X\}$  for any set of variables  $X \subseteq \mathbf{var}$ , so that  $\mathbf{arg}(A)$  is the disjoint union  $\mathbf{arg}(A) = \sum_{X \subseteq \mathbf{var}} A^X$ . Arguments are abstractions of entries. We assume that the set of arguments is closed under projection: for any ordered pair of variable sets  $X \subseteq Y \subseteq \mathbf{var}$ , there is a projection function  $pr_{X,Y} : A^Y \rightarrow A^X$  that commutes with restricted tupling:  $pr_{X,Y} \cdot \partial_0 = \partial_0 \cdot pr_{X,Y}$ , where projection on the left hand side is over arguments and projection on the right hand side is over entries.

The entity classification  $\mathbf{ent}(A) = \langle \mathbf{indiv}(A), \mathbf{ent}(A), \vDash_A \rangle$  has the entity types of  $A$  as its types and the individuals of  $A$  as its instances. The relation classification  $\mathbf{rel}(A) = \langle \mathbf{arg}(A), \mathbf{rel}(A), \vDash_A \rangle$  has the relation types of  $A$  as its types and the arguments of  $A$  as its instances. In the relation classification the arguments (and their entries) have signatures (they are *records*) in the context of relations: when  $t \vDash_A \rho$ , we can regard  $\partial_1(\rho)$  to be the signature of  $t$ ; of course, although it has a fixed arity, for a different relation an argument could have a different signature. For example, an argument with the entry (sam, 70) could represent an age or a height; thus having two different signatures (Person, Years) and (Person, Inches).

## Model Morphisms

The instance infomorphisms of the entity and relation classifications show that models are traditional models. The *entity instance infomorphism*  $\langle Id, inst_A \rangle : \mathbf{ent}(A) = \wp \mathbf{indiv}(A)$  maps the entity classification to a powerset classification, where both logics have individuals of  $A$  as instances. The *relation instance infomorphism*  $\langle Id, inst_A \rangle : \mathbf{rel}(A) = \wp \mathbf{arg}(A)$  maps the relation classification to a powerset classification, where both classifications have arguments of  $A$  as instances. The definitions  $A_\alpha = inst_A(\alpha)$  and  $A_\rho = inst_A(\rho)$  provide the traditional interpretation for all types. In fact, we can take the instance infomorphisms as representing a *traditional model* of  $L$ .

Abstraction	Infomorphism-projection diagram
$  \begin{array}{ccc}  & \xrightarrow{\mathbf{rel}(f)} & \\  \mathbf{rel}(A) & \longleftrightarrow & \mathbf{rel}(A') \\  \downarrow \langle \partial_0, \partial_1 \rangle & & \downarrow \langle \partial_0, \partial_1 \rangle \\  & \xrightarrow{\mathbf{tuple}(\mathbf{ent}(f))} & \\  \mathbf{tuple}(\mathbf{ent}(L)) & \longleftrightarrow & \mathbf{tuple}(\mathbf{ent}(L')) \\  \downarrow \mathbf{ent}(f) & & \downarrow \mathbf{ent}(f) \\  \mathbf{ent}(A) & \longleftrightarrow & \mathbf{ent}(A')  \end{array}  $	

A *model morphism*  $f = \langle \mathbf{ent}(f), \mathbf{rel}(f) \rangle : A \simeq A'$  from model  $A$  to model  $A'$  consists of

1. an instance hypergraph morphism  $inst(f) = \langle i, o \rangle : inst(A) \leftarrow inst(A')$ ;
2. a type hypergraph morphism  $typ(f) = \langle e, r \rangle : typ(A) \rightarrow typ(A')$ ;
3. an entity infomorphism  $\mathbf{ent}(f) = \langle i, e \rangle : \mathbf{ent}(A) \simeq \mathbf{ent}(A')$ ; and
4. a relation infomorphism  $\mathbf{rel}(f) = \langle o, r \rangle : \mathbf{rel}(A) \simeq \mathbf{rel}(A')$ .

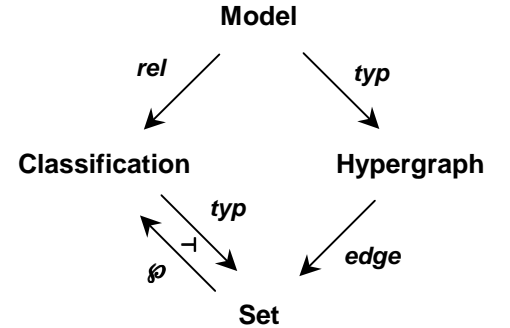
In more detail, a language-model morphism consists of

- an *entity instance (individual) function*  $i : \mathbf{indiv}(A) \leftarrow \mathbf{indiv}(A')$ ,
- an *relation instance (argument) function*  $o : \mathbf{arg}(A) \leftarrow \mathbf{arg}(A')$ ,
- an *entity type function*  $e : \mathbf{ent}(A) \rightarrow \mathbf{ent}(A')$ ,
- a *relation type function*  $r : \mathbf{rel}(A) \rightarrow \mathbf{rel}(A')$ ,

that satisfy the four conditions:

1.  $\partial_1(r(\rho)) = \mathbf{tuple}(e)(\partial_1(\rho))$  for each relation type  $\rho \in \mathbf{rel}(A)$ ;
2.  $\partial_0(o(t')) = \mathbf{tuple}(i)(\partial_0(t'))$  for each argument  $t' \in \mathbf{arg}(A')$ ;
3.  $i(a') \vDash_A \alpha$  iff  $a' \vDash_{A'} e(\alpha)$  for each individual  $a' \in \mathbf{indiv}(A')$  and each entity type  $\alpha \in \mathbf{ent}(A)$ ; and
4.  $o(t') \vDash_A \rho$  iff  $t' \vDash_{A'} r(\rho)$  for each argument  $t' \in \mathbf{arg}(A')$  and each relation type  $\rho \in \mathbf{rel}(A)$ .

A more general notion of model morphism might replace the equalities in conditions 1 and 2 with tuple morphisms in one direction or the other. There is a category **Model** whose objects are models and whose morphisms are model morphisms. This category has an involution  $\times$  between types and instances: flip the classifications, and interchange  $\mathbf{ent}(A) \leftrightarrow \mathbf{indiv}(A)$  and  $\mathbf{rel}(A) \leftrightarrow \mathbf{arg}(A)$ . The *relation functor*  $\mathbf{rel} : \mathbf{Model} \rightarrow \mathbf{Classification} \downarrow \wp \mathbf{var}$  maps a model to its relation classification over arity, and maps a model morphism to its relation infomorphism over arity. The *entity functor*  $\mathbf{ent} : \mathbf{Model} \rightarrow \mathbf{Classification}$  maps a model to its entity classification, and maps a model morphism to its entity infomorphism. The *instance (indexing) functor*  $\mathbf{inst} : \mathbf{Model}^{\text{op}} \rightarrow \mathbf{Hypergraph}$  maps a model to its instance hypergraph, and maps a model morphism to its instance hypergraph morphism. It forgets entity types, relation types, classifications and infomorphisms, yet retains instance hypergraph structures and morphisms. It is a contravariant functor. The *type (indexing) functor*  $\mathbf{typ} : \mathbf{Model} \rightarrow \mathbf{Hypergraph}$  maps a model to its type hypergraph, and maps a model morphism to its type hypergraph morphism. It forgets individuals (entity instances), arguments (relation instances), classifications and infomorphisms, yet retains



type hypergraph structures and morphisms. It is a covariant functor. **Model** is a category indexed by the **typ** functor with indexing category **Hypergraph**.

Let  $\mathbf{Model}(L) = \mathbf{typ}^{-1}(L)^{\text{op}}$  denote the *category of L-models* (for convenience, the opposite of the  $L$ -th fiber of **typ**), whose objects are models with type hypergraph  $L$  and whose arrows are (the reverse of) model morphisms with type hypergraph morphism  $Id_L = \langle Id, Id \rangle$ . A standard (rather large) example of an  $L$ -model is the *dual power hypergraph model*  $\mathcal{H}(L)^\infty$  with entity classification  $\langle \wp\text{ent}(L), \text{ent}(L), \varepsilon \rangle$ , relation classification  $\langle \mathcal{H}\text{rel}(L), \text{rel}(L), \varepsilon \rangle$  where  $\mathcal{H}\text{rel}(L)$  is the collection of all subsets of relation types with common arity, instance hypergraph  $\langle \wp\text{ent}(L), \mathcal{H}\text{rel}(L), \partial_1 \rangle$ , type hypergraph  $A = \langle \text{ent}(L), \text{rel}(L), \partial \rangle$ , instance vertex function  $\partial_0 : \mathcal{H}\text{rel}(L) \rightarrow \text{tuple}(\text{ent}(L))$  defined by  $\partial_0(R)_x = \{\partial_L(\rho)_x \mid \rho \in R\}$  for all  $x \in \text{arity}(R)$ , and type vertex function  $\partial_L : \text{rel}(L) \rightarrow \text{tuple}(\text{ent}(L))$ . We think of an element  $R \in \mathcal{H}\text{rel}(L)$  as a truth value assignment to relation types, with  $R(\rho) = \text{true}$  when  $\rho \in R$  and  $R(\rho) = \text{false}$  when  $\rho \notin R$  for all  $\rho \in \text{rel}(L)$ .

Any language morphism  $\langle e, r \rangle : L \rightarrow L'$  determines a *model functor*  $\mathbf{Model}(e, r) : \mathbf{Model}(L) \leftarrow \mathbf{Model}(L')$ : it maps an  $L'$ -model  $A'$  to the  $L$ -model  $\mathbf{Model}(e, r)(A') = \langle e^{-1}(\text{ent}(A')), r^{-1}(\text{rel}(A')) \rangle$  with entity classification  $e^{-1}(\text{ent}(A')) = \langle \text{indiv}(A'), \text{ent}(L), \vDash_A \rangle$  where  $a' \vDash_A \alpha$  when  $a' \vDash_{A'} e(\alpha)$  and with relation classification  $r^{-1}(\text{rel}(A')) = \langle \text{arg}(A'), \text{rel}(L), \vDash_A \rangle$  where  $t' \vDash_A \rho$  when  $t' \vDash_{A'} r(\rho)$ ; and it maps an  $L'$ -model morphism  $\langle i, o \rangle : A' \rightarrow A''$  to the  $L$ -model morphism  $\mathbf{Model}(e, r)(i, o) = \langle i, o \rangle : \mathbf{Model}(e, r)(A') \rightarrow \mathbf{Model}(e, r)(A'')$ . There is a model morphism  $\langle e, r, Id, Id \rangle : \mathbf{Model}(e, r)(A') \rightarrow A'$  with underlying type (language) morphism  $\langle e, r \rangle : L \rightarrow L'$ . Any model morphism  $\langle e, r, i, o \rangle : A \rightarrow A'$  with underlying type (language) morphism  $\langle e, r \rangle : L \rightarrow L'$  factors through the model  $\mathbf{Model}(e, r)(A')$  as  $\langle e, r, i, o \rangle = \langle Id, Id, i, o \rangle \cdot \langle e, r, Id, Id \rangle$ , where  $\langle i, o \rangle : A \rightarrow \mathbf{Model}(e, r)(A')$  is an  $L$ -model morphism.

## A non-Functor

The instance infomorphisms (entity and relation) can be used to generate a model from an instance hypergraph. This construction is inverse to the instance functor  $\mathbf{inst} : \mathbf{Model}^{\text{op}} \rightarrow \mathbf{Hypergraph}$ . For any hypergraph  $A = \langle \text{node}(A), \text{edge}(A), \partial \rangle$  define the *power model*  $\mathcal{H}(A)$  with entity classification  $\langle \text{node}(A), \wp\text{node}(A), \varepsilon \rangle$ , relation classification  $\langle \text{edge}(A), \mathcal{H}\text{edge}(A), \varepsilon \rangle$  where  $\mathcal{H}\text{edge}(A)$  is the collection of all coherent edge sets (subsets of edges with common arity), instance hypergraph  $A = \langle \text{node}(A), \text{edge}(A), \partial \rangle$ , type hypergraph  $\langle \wp\text{node}(A), \mathcal{H}\text{edge}(A), \partial_1 \rangle$ , instance vertex function  $\partial : \text{edge}(A) \rightarrow \text{tuple}(\text{node}(A))$ , and type vertex function  $\partial_1 : \mathcal{H}\text{edge}(A) \rightarrow \text{tuple}(\wp\text{node}(A))$  defined by  $\partial_1(E)_x = \{\partial(e)_x \mid e \in E\}$  for all  $x \in \text{arity}(E)$ . The tuple function pair is a tuple projection  $\langle \partial, \partial_1 \rangle : \text{rel}(\mathcal{H}A) \rightarrow \text{tuple}(\text{ent}(\mathcal{H}A))$ : if  $e \in E$  for edge  $e$  and edgeset  $E$  then  $\partial(e)_x \in \partial(E)_x$  for all  $x \in \text{arity}(e)$  by definition. The dual of this construction is the *dual power model*  $\mathcal{H}(A)^\infty$  with entity classification  $\langle \wp\text{node}(A), \text{node}(A), \varepsilon \rangle$ , relation classification  $\langle \mathcal{H}\text{edge}(A), \text{edge}(A), \varepsilon \rangle$ , instance hypergraph  $\langle \wp\text{node}(A), \mathcal{H}\text{edge}(A), \partial_1 \rangle$ , type hypergraph  $A = \langle \text{node}(A), \text{edge}(A), \partial \rangle$ , instance vertex function  $\partial_1 : \mathcal{H}\text{edge}(A) \rightarrow \text{tuple}(\wp\text{node}(A))$ , and type vertex function  $\partial : \text{edge}(A) \rightarrow \text{tuple}(\text{node}(A))$ . This construction is inverse to the type functor  $\mathbf{typ} : \mathbf{Model} \rightarrow \mathbf{Hypergraph}$ .

Let  $\mathcal{H}(A)$  and  $\mathcal{H}(B)$  be two power models over two (instance) hypergraphs  $A$  and  $B$ , respectively. Following the example of (Barwise and Seligman, 1997) let us examine the meaning of a model morphism conditions in this case. By definition, a model morphism  $f : \mathcal{H}(A) \rightarrow \mathcal{H}(B)$  consists of

- a *node function*  $i : \text{node}(A) \leftarrow \text{node}(B)$ ,
- an *edge function*  $o : \text{edge}(A) \leftarrow \text{arg}(B)$ ,
- an *entity type function*  $e : \wp\text{node}(A) \rightarrow \wp\text{node}(B)$ ,
- a *relation type function*  $r : \mathcal{H}\text{edge}(A) \rightarrow \mathcal{H}\text{edge}(B)$ ,

that satisfy the four conditions:

1.  $\partial_B(r(E)) = \text{tuple}(e)(\partial_A(E))$  for each coherent edge set  $E \in \mathcal{H}\text{edge}(A)$ ;
2.  $\partial_A(o(e')) = \text{tuple}(i)(\partial_B(e'))$  for each edge  $e' \in \text{edge}(B)$ ;
3.  $i(n') \vDash_A N$  iff  $n' \vDash_B e(N)$  for each node  $n' \in \text{node}(B)$  and each node set  $N \in \wp\text{node}(A)$ ; and
4.  $o(e') \vDash_A E$  iff  $e' \vDash_{A'} r(E)$  for each edge  $e' \in \text{edge}(B)$  and each coherent edge set  $E \in \mathcal{H}\text{edge}(A)$ .

As with classifications, conditions 3 and 4 imply that  $e$  and  $r$  are the inverse image functions,  $e = i^{-1}$  and  $r = o^{-1}$ , since  $e(N) = \{n' \in \mathbf{node}(\mathbf{B}) \mid i(n') \in_A N\} = i^{-1}(N)$  and  $r(E) = \{e' \in \mathbf{edge}(\mathbf{B}) \mid o(e') \in_A E\} = o^{-1}(E)$ . Condition 1 means that  $\{\partial_B(e')_x \mid o(e') \in_A E\} = \{\partial_B(e')_x \mid e' \in o^{-1}(E)\} = \partial_B(o^{-1}(E))_x = (\mathbf{tuple}(i^{-1})(\partial_A(E)))_x = i^{-1}(\partial_A(E))_x = \{n' \in \mathbf{node}(\mathbf{B}) \mid i(n') \in_A \partial_A(E)_x\}$  for each coherent edge set  $E \in \mathcal{H}\mathbf{edge}(\mathbf{A})$ . Condition 2 is just the hypergraph morphism condition – it means that  $\partial_A(o(e'))_x = i(\partial_B(e')_x)$  for each edge  $e' \in \mathbf{edge}(\mathbf{B})$ . Condition 2 implies half of condition 1,  $\partial_B(o^{-1}(E))_x \subseteq (\mathbf{tuple}(i^{-1})(\partial_A(E)))_x$ ; it implies the whole condition 1 under the constraints that  $o$  be a surjection and  $i$  be an injection. Under these constraints the construction  $\mathcal{H}$  is a functor.

## Expression Models

We regard the type hypergraph  $\mathbf{typ}(\mathbf{A})$  of a model  $\mathbf{A}$  to be an ontological language. We could regard the instance hypergraph  $\mathbf{inst}(\mathbf{A})$  to be a language also. However, we act here much as we act in Information Flow, where the type component, but not the instance component, of a classification is given the expression structure of a theory. An expression classification model can be defined for both propositional expressions and quantified expressions. Only the propositional expressions have a functor and monad.

The *expression classification*  $\mathbf{expr}(\mathbf{typ}(\mathbf{A})) = \langle \mathbf{arg}(\mathbf{A}), \mathbf{expr}(\mathbf{typ}(\mathbf{A})), \models_{\mathbf{A}} \rangle$  has arguments as instances, expressions as types, and classification defined inductively with respect to expressions. Let  $t \in \mathbf{arg}(\mathbf{A})$  be any argument of  $\mathbf{A}$ .

Relation:  $t \models_{\mathbf{A}} \rho$  when  $t \models_{\mathbf{A}} \rho$  in the relation classification  $\mathbf{rel}(\mathbf{A})$ ;

Conjunction:  $t \models_{\mathbf{A}} \phi \wedge \psi$  when  $\mathbf{arity}(t) = \mathbf{arity}(\phi) \cup \mathbf{arity}(\psi)$  and  $\mathit{pr}_{\mathbf{arity}(\phi)}(t) \models_{\mathbf{A}} \phi$  and  $\mathit{pr}_{\mathbf{arity}(\psi)}(t) \models_{\mathbf{A}} \psi$ ;

Disjunction:  $t \models_{\mathbf{A}} \phi \vee \psi$  when  $\mathbf{arity}(t) = \mathbf{arity}(\phi) \cup \mathbf{arity}(\psi)$  and  $(\mathit{pr}_{\mathbf{arity}(\phi)}(t) \models_{\mathbf{A}} \phi \text{ or } \mathit{pr}_{\mathbf{arity}(\psi)}(t) \models_{\mathbf{A}} \psi)$ ;

Negation:  $t \models_{\mathbf{A}} \neg \phi$  when  $\mathbf{arity}(t) = \mathbf{arity}(\phi)$  and  $t \not\models_{\mathbf{A}} \phi$ ;

Implication:  $t \models_{\mathbf{A}} \phi \rightarrow \psi$  when  $\mathbf{arity}(t) = \mathbf{arity}(\phi) \cup \mathbf{arity}(\psi)$  and  $(\mathit{pr}_{\mathbf{arity}(\phi)}(t) \models_{\mathbf{A}} \phi \text{ implies } \mathit{pr}_{\mathbf{arity}(\psi)}(t) \models_{\mathbf{A}} \psi)$ ;

Equivalence:  $t \models_{\mathbf{A}} \phi \leftrightarrow \psi$  when  $\mathbf{arity}(t) = \mathbf{arity}(\phi) \cup \mathbf{arity}(\psi)$  and  $(\mathit{pr}_{\mathbf{arity}(\phi)}(t) \models_{\mathbf{A}} \phi \text{ iff } \mathit{pr}_{\mathbf{arity}(\psi)}(t) \models_{\mathbf{A}} \psi)$ ;

Subtype restriction: If  $\phi$  is an expression and  $\alpha$  is a signature satisfying  $\alpha \vdash \partial_{\mathbf{typ}(\mathbf{A})}(\phi)$ ,

$t \models_{\mathbf{A}} \phi[\alpha]$  when  $\mathbf{arity}(t) = \mathbf{arity}(\phi)$ ,  $t \models_{\mathbf{A}} \phi$  and  $\partial_A(t) \models \alpha$  in  $\mathbf{entry}(\mathbf{A})$ ;

Quantification: If  $\phi$  is a expression and  $x \in \mathbf{arity}_{\mathbf{typ}(\mathbf{A})}(\phi)$ , then

$s \models_{\mathbf{A}} (\forall x)\phi$  when  $\mathbf{arity}(s) = \mathbf{arity}(\phi) - \{x\}$

and  $(\forall t \in \mathbf{arg}(\mathbf{A}))(\mathbf{arity}(t) = \mathbf{arity}(\phi) \text{ and } \mathit{pr}_{\mathbf{arity}(\phi) - \{x\}}(t) = s \text{ implies } t \models_{\mathbf{A}} \phi)$ ; and

$s \models_{\mathbf{A}} (\exists x)\phi$  when  $\mathbf{arity}(s) = \mathbf{arity}(\phi) - \{x\}$

and  $(\exists t \in \mathbf{arg}(\mathbf{A}))(\mathbf{arity}(t) = \mathbf{arity}(\phi) \text{ and } \mathit{pr}_{\mathbf{arity}(\phi) - \{x\}}(t) = s \text{ and } t \models_{\mathbf{A}} \phi)$ .

Abstraction	Classification-projection diagram
$\begin{array}{c} \mathbf{expr}(\mathbf{A}) \\ \langle \partial_0, \partial_1 \rangle \Downarrow \\ \mathbf{tuple}(\mathbf{ent}(\mathbf{L})) \end{array}$	$\begin{array}{ccc} & \xrightarrow{\partial_1} & \mathbf{tuple}(\mathbf{ent}(\mathbf{A})) \\ \mathbf{expr}(\mathbf{typ}(\mathbf{A})) & & \\ \models_{\mathbf{A}} \Big  & & \Big  \mathbf{tuple}(\models_{\mathbf{A}}) \\ \mathbf{arg}(\mathbf{A}) & \xrightarrow{\partial_0} & \mathbf{tuple}(\mathbf{indiv}(\mathbf{A})) \end{array}$

Any model  $\mathbf{A}$  determines an *expression model*  $\mathbf{expr}(\mathbf{A}) = \langle \mathbf{ent}(\mathbf{expr}(\mathbf{A})), \mathbf{rel}(\mathbf{expr}(\mathbf{A})), \partial \rangle$  consisting of

- the *entity classification*  $\mathbf{ent}(\mathbf{expr}(\mathbf{A})) = \mathbf{ent}(\mathbf{A}) = \langle \mathbf{indiv}(\mathbf{A}), \mathbf{ent}(\mathbf{A}), \models_{\mathbf{A}} \rangle$ ;
- the *expression classification*  $\mathbf{rel}(\mathbf{expr}(\mathbf{A})) = \mathbf{expr}(\mathbf{typ}(\mathbf{A})) = \langle \mathbf{arg}(\mathbf{A}), \mathbf{expr}(\mathbf{typ}(\mathbf{A})), \models_{\mathbf{A}} \rangle$ ;
- the *hypergraph of instances*  $\mathbf{inst}(\mathbf{expr}(\mathbf{A})) = \mathbf{inst}(\mathbf{A}) = \langle \mathbf{indiv}(\mathbf{A}), \mathbf{arg}(\mathbf{A}), \partial_0 \rangle$ ;
- the *hypergraph of types*  $\mathbf{typ}(\mathbf{expr}(\mathbf{A})) = \langle \mathbf{ent}(\mathbf{A}), \mathbf{expr}(\mathbf{typ}(\mathbf{A})), \partial_1 \rangle$ ; and
- the *tuple projection*  $\partial = \langle \partial_0, \partial_1 \rangle : \mathbf{rel}(\mathbf{expr}(\mathbf{A})) \Rightarrow \mathbf{tuple}(\mathbf{ent}(\mathbf{A}))$ .

The tuple projection condition is expressed as follows: the expression classification  $t \models_{\mathbf{A}} \phi$  implies  $\mathbf{arity}(a) = \mathbf{arity}(\phi)$  and the tuple classification  $\partial_A(t) \models_{\mathbf{A}} \partial_{\mathbf{typ}(\mathbf{A})}(\phi)$  in  $\mathbf{tuple}(\mathbf{ent}(\mathbf{A}))$ . A model and its expression are connected with the model morphism  $\langle Id, r2e, Id, Id \rangle : \mathbf{A} = \mathbf{expr}(\mathbf{A})$ .

**Theorem.** If  $\langle i, o \rangle : A \rightleftharpoons A'$  is a fiber model morphism in  $\mathbf{Model}(L)$ , then

$$o(t) \models_{A'} \Phi \text{ iff } t \models_A \Phi$$

for any argument  $t \in \mathbf{arg}(A)$  and any propositional expression  $\Phi \in \mathbf{expr0}(L)$ ; thus,  $\langle i, o \rangle : \mathbf{expr0}(A) \rightleftharpoons \mathbf{expr0}(A')$  is a fiber model morphism in  $\mathbf{Model}(\mathbf{expr0}(L))$ .

**Proof.**

- Relation:  $t \models_A \rho$  iff  $o(t) \models_{A'} \rho$  by assumption.
- Conjunction:  $t \models_A (\Phi \wedge \Psi)$  iff  $pr_{\partial(\Phi)}(t) \models_A \Phi$  and  $pr_{\partial(\Psi)}(t) \models_A \Psi$  iff (by induction)  $pr_{\partial(\Phi)}(o(t)) = o(pr_{\partial(\Phi)}(t)) \models_{A'} \Phi$  and  $pr_{\partial(\Psi)}(o(t)) = o(pr_{\partial(\Psi)}(t)) \models_{A'} \Psi$  iff  $o(t) \models_{A'} (\Phi \wedge \Psi)$ .
- Negation:  $t \models_A \neg\Phi$  iff not  $t \models_A \Phi$  iff (by induction) not  $o(t) \models_{A'} \Phi$  iff  $o(t) \models_{A'} \neg\Phi$ .
- Subtype restriction:  $t \models_A \Phi[\alpha]$  iff  $t \models_A \Phi$  and  $\partial_A(t) \models_{\text{tuple}(\text{indiv}(A))} \alpha$  iff (by induction)  $o(t) \models_{A'} \Phi$  and  $\partial_A(o(t)) \models_{\text{tuple}(\text{indiv}(A'))} \alpha$  iff  $o(t) \models_{A'} \Phi[\alpha]$ .

For each language  $L$  there is an expression functor between fibers

$$\xi_L : \mathbf{Model}(L) \rightarrow \mathbf{Model}(\mathbf{expr0}(L)) = \mathbf{expr0} \cdot \mathbf{Model}(L),$$

which is the  $L$ -th component of a natural transformation

$$\xi : \mathbf{Model} \Rightarrow \mathbf{expr0} \cdot \mathbf{Model} : \mathbf{Language}^{\text{op}} \rightarrow \mathbf{Cat}.$$

**Lemma.** If  $\langle i, o \rangle : A \rightleftharpoons A'$  is a fiber model morphism in  $\mathbf{Model}(L)$ , then for any argument  $t \in \mathbf{arg}(A)$  and any expression  $\Phi \in \mathbf{expr}(L)$

$$o(t) \models_{A'} \Phi \text{ implies } t \models_A \Phi$$

in the positive universal part of expressions, and,

$$t \models_A \Phi \text{ implies } o(t) \models_{A'} \Phi$$

in the positive existential part of expressions.

**Proof.** The propositional cases have been dealt with in the previous theorem, where equivalence is to be proved (and hence assumed by induction). However, here with just implication assumed, negation flips the implication direction.

- Universal Quantifier:  $o(s) \models_{A'} (\forall x)\Phi$   
iff  $(\forall t' \in \mathbf{arg}(A')) (pr(t') = o(s) \text{ implies } t' \models_{A'} \Phi)$   
implies (iff when  $o$  is epi)  $(\forall t \in \mathbf{arg}(A)) (pr(o(t)) = o(s) \text{ implies } o(t) \models_{A'} \Phi)$   
iff  $(\forall t \in \mathbf{arg}(A)) o(pr(t)) = o(s) \text{ implies } o(t) \models_{A'} \Phi$   
implies (by induction) (iff when  $o$  is mono)  $(\forall t \in \mathbf{arg}(A)) (pr(t) = s \text{ implies } t \models_A \Phi)$   
 $s \models_A (\forall x)\Phi$ .
- Existential Quantifier:  $o(s) \models_{A'} (\exists x)\Phi$   
iff  $(\exists t' \in \mathbf{arg}(A')) (pr(t') = o(s) \text{ and } t' \models_{A'} \Phi)$   
is implied by (iff when  $o$  is epi)  $(\exists t \in \mathbf{arg}(A)) (pr(o(t)) = o(s) \text{ and } o(t) \models_{A'} \Phi)$   
iff  $(\exists t \in \mathbf{arg}(A)) o(pr(t)) = o(s) \text{ and } o(t) \models_{A'} \Phi$   
is implied by (by induction) (iff when  $o$  is mono)  $(\exists t \in \mathbf{arg}(A)) (pr(t) = s \text{ and } t \models_A \Phi)$   
 $s \models_A (\exists x)\Phi$ .

In ontological languages constants correspond to individuals. Normally in logic and knowledge representation, constants are allowed (to be substituted) in(to) expressions. Let us call such expressions “formulas.” Formulas can be formalized – for any model  $A$ , a *formula*  $\Phi$  is an expression with substitution, and has an associated expression  $\mathbf{expr}(\Phi)$ , a subset of the arity of the expression  $\mathbf{index}(\Phi) \subseteq \mathbf{arity}(\mathbf{expr}(\Phi))$  and an assignment  $\mathbf{sub}(\Phi) \in \mathbf{indiv}(A)^{\mathbf{index}(\Phi)}$ . The fact that this formalization is wrong-headed shows up in the theory. Formulas can have any number of free variables. Formulas with no free variables are called *sentences*. Associated with a sentence  $\Phi$  is an expression  $\mathbf{expr}(\Phi)$  and an argument  $\mathbf{arg}(\Phi)$  that has been substituted into the expression to make the sentence. Asserting the validity of the sentence  $\models_A \Phi$  means the same as asserting the classification  $\mathbf{arg}(\Phi) \models_A \mathbf{expr}(\Phi)$ . In this paper expressions are truly abstract. They are not identified with their serialization, which is just a surface manifestation of the truly abstract expression. In this paper the classification relation of expressions more than adequately represents closed formula (sentences) – the substitution of constants into expressions.

Consider the following example from the Cyc documentation, where the Cyc prefix “#\$” has been removed for readability.

```
(and (isa Pittman HumanCyclist) (residesInRegion Pittman CityOfAustinTX))
```

This sentence  $\varphi$  contains an entity type “HumanCyclist”, a binary relation type “residesInRegion”, and two individuals “Pittman” and “CityOfAustinTX”. The expression  $expr(\varphi)$  of the sentence is the restriction of a relational expression with binary relation type  $\rho = \text{“residesInRegion”}$  and signature  $\partial_A(\rho) = (\text{Person: person, Location: location})$  to the subtype  $\alpha = (\text{HumanCyclist: person, City: location})$ . The classification of the sentence  $\varphi$  has argument  $arg(\varphi) = (\text{Pittman: person, CityOfAustinTX: location})$ .

## Examples: Ontology and Terminology

Ontologies correspond to knowledge bases. An ontology has two basic parts: a schema (**T-Box**) and a set of assertions (**A-Box**). In more detail, an ontology has the following components. There is a hierarchy of entity types structured as nested IF theories. There is always one generic theory at the top of the hierarchy. Other theories are anchored at particular entity types, with these as their root type. There are collections of binary relation types. There are terminologies consisting of coherent collections of defined types. There are classifications. There are collections of other assertions. These assertions can be grouped according to relations, functions or tables. The following examples illustrate ontologies. Several things are illustrated here: multi-inheritance of facets; operations of Description Logic; and intrinsic vs. extrinsic relations via binary relation, object and data types.

### Wine Drinkers Terminology

The following example is in the paper (Prediger and Stumme, 1999). The subject is wine. The ontological language, ontological model and terminology components were extracted from the discussion of the paper. Note that instances (individuals and arguments) and classification relationships are not included in ontological languages, since expressions do not include instances.

#### The Language $L'$

The components are indexed according to the (right hand side of the) Terminology Diagram.

##### Entity Types $ent(L')$

There is a (generic) theory of things partitioned into two central object types, ‘Person’ and ‘Wine’. There are also a subsidiary object type ‘Drinker.Wine’ and a data type ‘Natno’. Data types are implicitly assumed to be disjoint from all object types.

```
{Person; Wine; ‘Red Wine’; ‘White Wine’; Bordeaux; ‘Red Bordeaux’; Natno}
```

##### Relation Types $rel(L')$

There is a binary relation type ‘drinks’ with source entity type ‘Person’ and target entity type ‘Wine’. There is also a function ‘price’ from Wine to Natno.

```
{
    drinks(Person: person, Wine: wine);
    price(Wine: wine, Natno: value)
}
```

#### The $L'$ -Model $A'$

The components are indexed according to the Model Classification-projection Diagram.

##### Classification of Individuals (Entity Instances) $ent(A')$

There are individual persons and individual wines.

```
{Person(‘Mr. Smith’); Person(‘Mrs. Miller’); Person(‘Miss Cooper’); Person(‘Mr. Davis’)}
{‘Red Wine’(Figeac); ‘Bordeaux’(Figeac); ‘White Wine’(Staehe); ‘Red Wine’(‘Casa Solar’)}
```

##### Classification of Arguments (Relation Instances) $rel(A')$

```
{
    drinks(‘Mr. Smith’, ‘Casa Solar’);
    drinks(‘Mrs. Miller’, Staehe );
    drinks(‘Miss Cooper’, Figeac);
    drinks(‘Mr. Davis’, Figeac );
}
```

```

drinks('Mr. Davis', 'Casa Solar')
}
{
price(Figeac, 4990);
price(Staeble, 1490);
price('Casa Solar', 595)
}

```

### The $L'$ -model ontology $\text{ont}(A')$

Information Flow version	Onto Logic version
{Person, Wine} $\vdash$	
{Person, Natno} $\vdash$	
{Wine, Natno} $\vdash$	
'Red Wine' $\vdash$ Wine	
'White Wine' $\vdash$ Wine	
Bordeaux $\vdash$ Wine	
'Red Bordeaux' $\vdash$ Bordeaux	
'Red Bordeaux' $\vdash$ 'Red Wine'	

### The Language $L$ and Terminology $\langle e, r \rangle$

The components are indexed according to the (left hand side of the) Terminology Diagram.

### Terminology $\langle e, r \rangle : L \rightarrow \text{expr}(L')$

Description logic version	Onto logic version
'Wine Drinker' := Person $\wedge$ $\exists$ drinks.Wine;	'Wine Drinker'(Person: person) := ( $\exists$ wine: Wine)drinks(person, wine);
'Red Wine Drinker' := Person $\wedge$ $\exists$ drinks.'Red Wine';	
'White Wine Drinker' := Person $\wedge$ $\exists$ drinks.'White Wine';	
'Bordeaux Drinker' := Person $\wedge$ $\exists$ drinks.Bordeaux;	
'Red Bordeaux Drinker' := Person $\wedge$ $\exists$ drinks.('Red Wine' $\wedge$ Bordeaux);	'Red Bordeaux Drinker'(Person: person) := ( $\exists$ wine: 'Red Bordeaux')drinks(person, wine);

### Model Functor Classification Model(e, r)(A')

{Person('Mr. Smith'); 'Wine Drinker'('Mr. Smith'); 'Red Wine Drinker'('Mr. Smith'); Person('Mrs. Miller'); 'Wine Drinker'('Mrs. Miller'); 'White Wine Drinker'('Mrs. Miller'); Person('Miss Cooper'); 'Wine Drinker'('Miss Cooper'); 'Red Wine Drinker'('Miss Cooper'); 'Bordeaux Drinker'('Miss Cooper'); 'Red Bordeaux Drinker'('Miss Cooper'); Person('Mr. Davis'); 'Wine Drinker'('Mr. Davis'); 'Red Wine Drinker'('Mr. Davis'); 'Bordeaux Drinker'('Mr. Davis'); 'Red Bordeaux Drinker'('Mr. Davis')}

### Commerce and Industry Ontology

Here are some excerpts from the Commerce and Industry ontology that was used to help develop the WAVE networked information discovery and retrieval system. This example illustrates the handling of multivalent relations with tables. The logical formulation corresponds to object-oriented database structure. However, the relational database tables below show how the multivalent relations correspond to tables.

### The Language $L$

The components are indexed according to the Language Diagram.

### Entity Types $\text{ent}(L)$

There is an entity type hierarchy with the 'Company' type at the top and various subtypes, such as 'Aerospace' and 'Marketing', below it. There are several relevant data types, such as 'Date' and 'Natno'. Data types are implicitly assumed to be disjoint from all object types.

{Company; 'Aerospace Company'; Marketing; ..., Date; Natno}

**Relation Types rel(L)**

There are binary relation types such as 'competitor'. There are also multivalent relation types. The relation types 'employment' and 'revenues' are both ternary.

```
{
    competitor(Company: company1, Company: company2);
    employment(Company: company, Date: date, Natno: number);
    revenues(Company: company, Date: date, Natno: number);
    ...
}
```

**The L-Model A**

The components are indexed according to the Model Classification-projection Diagram.

**Classification of Individuals (Entity Instances) ent(A)**

There are individual persons and individual wines.

{'Aerospace Company'(Delta); 'Aerospace Company'(United); ...}

**Classification of Arguments (Relation Instances) rel(A)**

{competitor(Delta, United); ...}

{employment(Delta, 1996, 60289); ...}

{revenues(Delta, 1996, 12455); ...}

The argument classifications correspond to relational database tables.

employment	company	date	number
dal_empl_1996	Delta	1996	60289
...	...	...	...

revenues	company	date	number
dal_rev_1996	Delta	1996	12455
...	...	...	...

**The L-model ontology ont(A)**

Information Flow version	Onto Logic version
'Aerospace Company' ⊢ Company	
...	
'Marketing Company' ⊢ Company	
...	
{Company, Date} ⊢ {}	
{Company, Natno} ⊢ {}	
{Date, Natno} ⊢ {}	
...	
{Employment, Revenues} ⊢ {}	
...	

## 5. Limits for Models

### A Non-adjointness

Ordinary classifications have a unique infomorphisms to powerset classifications on instances or types. Is this true for models? Analogous to the indexing of the category **Model** along the type functor  $\mathbf{typ} : \mathbf{Model} \rightarrow \mathbf{Hypergraph}$ , the category **Classification** is indexed along the type functor  $\mathbf{typ} : \mathbf{Classification} \rightarrow \mathbf{Set}$ . For any set (of types)  $\Sigma$  let  $\mathbf{Classification}(\Sigma) = \mathbf{typ}^{-1}(\Sigma)^{\text{op}}$  denote the *category of  $\Sigma$ -classifications* (for convenience, the opposite of the  $\Sigma$ -th fiber of  $\mathbf{typ}$ ), whose objects are classifications with type set  $\Sigma$  and whose arrows are (the reverse of) infomorphisms with type function  $Id_{\Sigma}$ . For any  $\Sigma$ -classifications  $\langle A, \Sigma, \models \rangle$  the type function  $\mathbf{typ}_A : A \rightarrow \wp \Sigma$  is a unique  $\Sigma$ -infomorphism  $\mathbf{typ}_A : \langle A, \Sigma, \models \rangle \rightarrow \langle \wp \Sigma, \Sigma, \ni \rangle$ , since  $a \models \alpha$  iff  $\mathbf{typ}_A(a) \ni \alpha$ . By analogy, is it true that there is a unique  $L$ -model morphism  $\langle i, o \rangle : A \rightarrow \mathcal{H}(L)^{\infty}$  from any  $L$ -model  $A$  to the dual power hypergraph model  $\mathcal{H}(L)^{\infty}$ ? In more detail, a unique  $L$ -model morphism  $\langle i, o \rangle : A \rightarrow \mathcal{H}(L)^{\infty}$  consists of

- an entity instance (individual) function  $i : \mathit{indiv}(A) \rightarrow \mathit{indiv}(\mathcal{H}(L)^{\infty}) = \wp \mathit{ent}(L)$ , and
- an relation instance (argument) function  $o : \mathit{arg}(A) \rightarrow \mathit{arg}(\mathcal{H}(L)^{\infty}) = \mathcal{H} \mathit{rel}(L)$ ,

that satisfy the three conditions:

1.  $\partial_0(o(t)) = \mathit{tuple}(i)(\partial_0(t))$  for each argument  $t \in \mathit{arg}(A)$ ;
2.  $a \models_A \alpha$  for  $i(a) \ni \alpha$  for each individual  $a \in \mathit{indiv}(A)$  and each entity type  $\alpha \in \mathit{ent}(L)$ ; and
3.  $t \models_A \rho$  iff  $o(t) \ni \rho$  for each argument  $t \in \mathit{arg}(A)$  and each relation type  $\rho \in \mathit{rel}(L)$ .

Condition 2 requires the definition  $i(a) = \{\alpha \in \mathit{ent}(L) \mid a \models_A \alpha\} = \mathbf{typ}_A(a)$ , and condition 3 requires the definition  $o(t) = \{\rho \in \mathit{rel}(L) \mid t \models_A \rho\} = \mathbf{typ}_A(t)$ . The latter definition satisfies the requirement that all relation types in  $\mathbf{typ}_A(t)$  have a common arity, since the tuple projection condition means  $t \models_A \rho$  implies  $\partial_0(t) \models_{\mathit{tuple}(\mathit{ent}(A))} \partial_L(\rho)$  for each argument  $t \in \mathit{arg}(A)$  and each relation type  $\rho \in \mathit{rel}(L)$  – so that the argument  $t$  and the relation type  $\rho$  must have the same arity. The only condition that remains to be checked is the hypergraph morphism condition 1. Unfortunately equality here does not hold – we can only prove the inequality:  $\partial_0(o(t)) = \partial_0(\mathbf{typ}_A(t))_x = \{\partial_L(\rho)_x \mid \rho \in \mathbf{typ}_A(t)\} \subseteq \{\alpha \in \mathit{ent}(L) \mid \partial_0(t)_x \models_A \alpha\} = \mathbf{typ}_A(\partial_0(t)_x) = \mathit{tuple}(\mathbf{typ}_A)(\partial_0(t))_x = \mathit{tuple}(i)(\partial_0(t))$  for each argument  $t \in \mathit{arg}(A)$  and for all  $x \in \mathit{arity}(R)$ . For this to be an equality the relation  $A(\rho)$  would need to project to  $A(\partial_L(\rho)_x)$  along the  $x$ -th coordinate for each  $x \in \mathit{arity}(\rho)$ . However, most relations are not that big!

### Products

Let  $A$  and  $A'$  be two models.

Abstraction	Classification-projection diagram
$\begin{array}{c} \mathit{rel}(A) \otimes \mathit{rel}(A') \\ \langle \partial_0, \partial_1 \rangle \downarrow \downarrow \\ \mathit{tuple}(\mathit{ent}(A) \times \mathit{ent}(A')) \end{array}$	$\begin{array}{ccc} \mathit{rel}(A) \otimes \mathit{rel}(A') & \xrightarrow{\partial_x} & \mathit{tuple}(\mathit{ent}(A) \times \mathit{ent}(A')) \\ \models_{\otimes} \Big  & & \Big  \mathit{tuple}(\models_{\times}) \\ \mathit{arg}(A) + \mathit{arg}(A') & \xrightarrow[\partial_+]{\quad} & \mathit{tuple}(\mathit{indiv}(A) + \mathit{indiv}(A')) \end{array}$

The *product model*  $A \times A'$  has

- the sum hypergraph of instances  
 $\mathit{inst}(A \times A') = \mathit{inst}(A) + \mathit{inst}(A') = \langle \mathit{indiv}(A) + \mathit{indiv}(A'), \mathit{arg}(A) + \mathit{arg}(A'), \partial_+ \rangle$ ;
- the product hypergraph of types  
 $\mathbf{typ}(A \times A') = \mathbf{typ}(A) \times \mathbf{typ}(A') = \langle \mathit{ent}(A) \times \mathit{ent}(A'), \mathit{rel}(A) \otimes \mathit{rel}(A'), \partial_{\otimes} \rangle$ ;
- the product classification of entities  
 $\mathit{ent}(A \times A') = \mathit{ent}(A) \times \mathit{ent}(A') = \langle \mathit{indiv}(A) + \mathit{indiv}(A'), \mathit{ent}(A) \times \mathit{ent}(A'), \models_{\times} \rangle$ ;

- the subproduct classification of relations (the inverse image of the product relation classification)

$$\mathbf{rel}(A \times A') = \mathit{inc}^{-1}(\mathbf{rel}(A) \times \mathbf{rel}(A')) = \langle \mathit{arg}(A) + \mathit{arg}(A'), \mathbf{rel}(A) \otimes \mathbf{rel}(A'), \models_{\otimes} \rangle;$$

- the vertex classification projection  $\partial = \langle \partial_+, \partial_x \rangle : \mathbf{rel}(A) \otimes \mathbf{rel}(A') \Rightarrow \mathbf{tuple}(\mathbf{ent}(A) \times \mathbf{ent}(A'))$ .

The projection condition is expressed as follows: the expression classification  $t \models_{\otimes} (\rho, \rho')$  implies  $\mathit{arity}(t) =$

$\mathit{arity}(\rho, \rho')$  and the tuple classification  $\partial_0(t)_x \models_A \partial_1(\rho)_x$  in  $\mathbf{ent}(A)$ , hence  $\partial_0(t)_x \models_A (\partial_1(\rho)_x, \partial_1(\rho')_x)$  in

$\mathbf{ent}(A) \times \mathbf{ent}(A')$  for any variable  $x \in \mathit{arity}(\rho) = \mathit{arity}(\rho')$ . The projection model morphism  $pr_A =$

$\langle \mathit{in}_{\mathit{indiv}(A)}, \mathit{in}_{\mathit{arg}(A)}, pr_{\mathbf{ent}(A)}, \mathit{inc} \cdot pr_{\mathbf{rel}(A)} \rangle : A \times A' \Rightarrow A$  from the product model  $A \times A'$  to the first component model  $A$  has

- the instance sum injection hypergraph morphism  $\langle \mathit{in}_{\mathit{indiv}(A)}, \mathit{in}_{\mathit{arg}(A)} \rangle : \mathit{inst}(A) + \mathit{inst}(A') \leftarrow \mathit{inst}(A)$ ;
- the type product projection hypergraph morphism  $\langle pr_{\mathbf{ent}(A)}, \mathit{inc} \cdot pr_{\mathbf{rel}(A)} \rangle : \mathit{typ}(A) \times \mathit{typ}(A') \rightarrow \mathit{typ}(A)$ ;
- the entity product projection infomorphism  $\langle \mathit{in}_{\mathit{indiv}(A)}, pr_{\mathbf{ent}(A)} \rangle : \mathbf{ent}(A) \times \mathbf{ent}(A') \Rightarrow \mathbf{ent}(A)$ ; and
- the relation subproduct projection infomorphism  $\langle Id, \mathit{inc} \rangle \circ \langle \mathit{in}_{\mathit{arg}(A)}, pr_{\mathbf{rel}(A)} \rangle : \mathbf{rel}(A) \otimes \mathbf{rel}(A') \Rightarrow \mathbf{rel}(A)$ .

Let  $\langle i, o, e, r \rangle : B \Rightarrow A$  and  $\langle i', o', e', r' \rangle : B \Rightarrow A'$  be any two model morphisms with a common source. Then

$\langle i, o \rangle : \mathit{inst}(B) \leftarrow \mathit{inst}(A)$  and  $\langle i', o' \rangle : \mathit{inst}(B) \leftarrow \mathit{inst}(A')$  are instance hypergraph morphisms with a common target hypergraph. So there is a unique mediating hypergraph morphism

$\langle \tilde{i}, \tilde{o} \rangle : \mathit{inst}(B) \leftarrow \mathit{inst}(A) + \mathit{inst}(A')$ , defined by  $\langle \tilde{i}, \tilde{o} \rangle = [ \langle i, i' \rangle, \langle o, o' \rangle ] = \langle [i, i'], [o, o'] \rangle$ , and satisfying

$$\mathit{in}_A \circ \langle \tilde{i}, \tilde{o} \rangle = \langle \mathit{in}_{\mathit{indiv}(A)}, \mathit{in}_{\mathit{arg}(A)} \rangle \circ \langle \tilde{i}, \tilde{o} \rangle = \langle \mathit{in}_{\mathit{indiv}(A)} \cdot \tilde{i}, \mathit{in}_{\mathit{arg}(A)} \cdot \tilde{o} \rangle = \langle i, o \rangle \text{ and } \mathit{in}_{A'} \circ \langle \tilde{i}, \tilde{o} \rangle =$$

$$\langle \mathit{in}_{\mathit{indiv}(A')}, \mathit{in}_{\mathit{arg}(A')} \rangle \circ \langle \tilde{i}, \tilde{o} \rangle = \langle \mathit{in}_{\mathit{indiv}(A')} \cdot \tilde{i}, \mathit{in}_{\mathit{arg}(A')} \cdot \tilde{o} \rangle = \langle i', o' \rangle. \text{ Dually, } \langle e, r \rangle : \mathit{typ}(B) \rightarrow \mathit{typ}(A) \text{ is a type}$$

hypergraph morphism and  $\langle e', r' \rangle : \mathit{typ}(B) \rightarrow \mathit{typ}(A')$  is a type hypergraph morphism are type hypergraph morphisms with a common source hypergraph. So there is a unique mediating hypergraph morphism

$$\langle \tilde{e}, \tilde{r} \rangle : \mathit{typ}(B) \rightarrow \mathit{typ}(A) \times \mathit{typ}(A') \text{ satisfying } \langle \tilde{e}, \tilde{r} \rangle \circ pr_A = \langle \tilde{e}, \tilde{r} \rangle \circ \langle pr_{\mathbf{ent}(A)}, \mathit{inc} \cdot pr_{\mathbf{rel}(A)} \rangle =$$

$$\langle \tilde{e} \cdot pr_{\mathbf{ent}(A)}, \tilde{r} \cdot \mathit{inc} \cdot pr_{\mathbf{rel}(A)} \rangle = \langle e, r \rangle \text{ and } \langle \tilde{e}, \tilde{r} \rangle \circ \mathit{in}_{A'} = \langle \tilde{e}, \tilde{r} \rangle \circ \langle pr_{\mathbf{ent}(A')}, \mathit{inc} \cdot pr_{\mathbf{rel}(A')} \rangle =$$

$$\langle \tilde{e} \cdot pr_{\mathbf{ent}(A')}, \tilde{r} \cdot \mathit{inc} \cdot pr_{\mathbf{rel}(A')} \rangle = \langle e', r' \rangle. \text{ We want to show that } \langle \tilde{i}, \tilde{o}, \tilde{e}, \tilde{r} \rangle : B \Rightarrow A \times A' \text{ is the unique model}$$

morphism satisfying  $\langle \tilde{i}, \tilde{o}, \tilde{e}, \tilde{r} \rangle \circ pr_A = \langle i, o, e, r \rangle$  and  $\langle \tilde{i}, \tilde{o}, \tilde{e}, \tilde{r} \rangle \circ pr_{A'} = \langle i', o', e', r' \rangle$ . Existence and uniqueness are clear. We need to show well-formed-ness; that is, we need to verify that

$\langle \tilde{i}, \tilde{e} \rangle : \mathbf{ent}(B) \Rightarrow \mathbf{ent}(A) \times \mathbf{ent}(A')$  and  $\langle \tilde{o}, \tilde{r} \rangle : \mathbf{rel}(B) \Rightarrow \mathit{inc}^{-1}(\mathbf{rel}(A) \times \mathbf{rel}(A'))$  are infomorphisms. First,

choose instance  $a \in \mathit{indiv}(A) + \mathit{indiv}(A')$  and type  $\beta \in \mathit{typ}(B)$ ; then,  $\tilde{i}(a) \models_B \beta$  iff  $i(a) \models_B \beta$  iff  $a \models_A e(\beta)$  iff

$a \models_{\times} \tilde{e}(\beta) = (e(\beta), e'(\beta))$ . Second, choose instance  $t \in \mathit{arg}(A) + \mathit{arg}(A')$  and type  $\rho \in \mathbf{rel}(B)$ ; then,  $\tilde{o}(t) \models_B \rho$  iff

$o(t) \models_B \rho$  iff  $t \models_A r(\rho)$  iff  $t \models_{\otimes} \tilde{r}(\rho) = (r(\rho), r'(\rho))$ . So the product  $A \times A'$  is a product in **Model**, the **typ**

functor preserves products, and the **inst** functor maps products to coproducts.

### Fiber Sum Models: Subposition

Let  $L = \langle \mathbf{ent}(L), \mathbf{rel}(L), \partial_L \rangle$  be any ontological language. Let  $A$  and  $A'$  be two  $L$ -models.

Abstraction	Classification-projection diagram
$\mathbf{rel}(A) \& \mathbf{rel}(A')$ $\downarrow \langle \partial_0, \partial_L \rangle$ $\mathbf{tuple}(\mathbf{ent}(A) \& \mathbf{ent}(A'))$	$\begin{array}{ccc} \mathbf{rel}(L) & \xrightarrow{\partial_L} & \mathbf{tuple}(\mathbf{ent}(L)) \\ \models_{\&} \downarrow & & \downarrow \mathbf{tuple}(\models_{\&}) \\ \mathit{arg}(A) + \mathit{arg}(A') & \xrightarrow{\partial_+} & \mathbf{tuple}(\mathit{indiv}(A) + \mathit{indiv}(A')) \end{array}$

The fiber sum (or subposition) model  $A \& A'$  has

- the sum hypergraph of instances

$$\mathit{inst}(A \& A') = \mathit{inst}(A) + \mathit{inst}(A') = \langle \mathit{indiv}(A) + \mathit{indiv}(A'), \mathit{arg}(A) + \mathit{arg}(A'), \partial_+ \rangle;$$

- the subposition classification of entities

$$\mathbf{ent}(A \& A') = \mathbf{ent}(A) \& \mathbf{ent}(A') = \langle \mathit{indiv}(A) + \mathit{indiv}(A'), \mathbf{ent}(L), \models_{\&} \rangle;$$

- the subposition classification of relations

$$\mathbf{rel}(A \& A') = \mathbf{rel}(A) \& \mathbf{rel}(A') = \langle \mathit{arg}(A) + \mathit{arg}(A'), \mathbf{rel}(L), \models_{\&} \rangle;$$

- the vertex classification projection  $\partial = \langle \partial_+, \partial_- \rangle : \mathbf{rel}(A) \& \mathbf{rel}(A') \Rightarrow \mathbf{tuple}(\mathbf{ent}(A) \& \mathbf{ent}(A'))$ .

The projection condition is expressed as follows: the expression classification  $t \models_{\&} \rho$  implies  $\mathbf{arity}(t) = \mathbf{arity}(\rho)$  and the tuple classification  $\partial_0(t)_x \models_A \partial_L(\rho)_x$  in  $\mathbf{ent}(L)$  for any variable  $x \in \mathbf{arity}(\rho)$ . The *injection model morphism*  $in_A = \langle in_{\mathbf{indiv}(A)}, in_{\mathbf{arg}(A)} \rangle : A \Rightarrow A \& A'$  from the first component model  $A$  to the fiber sum model  $A \& A'$  has

- 1) the instance sum injection hypergraph morphism  $in_A = \langle in_{\mathbf{indiv}(A)}, in_{\mathbf{arg}(A)} \rangle : \mathbf{inst}(A) \rightarrow \mathbf{inst}(A) + \mathbf{inst}(A')$ ;
- 2) the entity subposition injection infomorphism  $\langle in_{\mathbf{indiv}(A)}, Id \rangle : \mathbf{ent}(A) \& \mathbf{ent}(A') \Rightarrow \mathbf{ent}(A)$ ; and
- 3) the relation subposition injection infomorphism  $\langle in_{\mathbf{arg}(A)}, Id \rangle : \mathbf{rel}(A) \& \mathbf{rel}(A') \Rightarrow \mathbf{rel}(A)$ .

Let  $\langle i, o \rangle : A \Rightarrow B$  and  $\langle i', o' \rangle : A' \Rightarrow B$  be any two  $L$ -model morphisms with a common target. Then  $\langle i, o \rangle : \mathbf{inst}(A) \rightarrow \mathbf{inst}(B)$  and  $\langle i', o' \rangle : \mathbf{inst}(A') \rightarrow \mathbf{inst}(B)$  are instance hypergraph morphisms with a common target hypergraph. So there is a unique mediating hypergraph morphism  $\langle \tilde{i}, \tilde{o} \rangle : \mathbf{inst}(A) + \mathbf{inst}(A') \rightarrow \mathbf{inst}(B)$ , defined by  $\langle \tilde{i}, \tilde{o} \rangle = [\langle i, i' \rangle, \langle o, o' \rangle] = \langle [i, i'], [o, o'] \rangle$ , and satisfying  $in_A \circ \langle \tilde{i}, \tilde{o} \rangle = \langle in_{\mathbf{indiv}(A)}, in_{\mathbf{arg}(A)} \rangle \circ \langle \tilde{i}, \tilde{o} \rangle = \langle in_{\mathbf{indiv}(A)} \cdot \tilde{i}, in_{\mathbf{arg}(A)} \cdot \tilde{o} \rangle = \langle i, o \rangle$  and  $in_{A'} \circ \langle \tilde{i}, \tilde{o} \rangle = \langle in_{\mathbf{indiv}(A')}, in_{\mathbf{arg}(A')} \rangle \circ \langle \tilde{i}, \tilde{o} \rangle = \langle in_{\mathbf{indiv}(A')} \cdot \tilde{i}, in_{\mathbf{arg}(A')} \cdot \tilde{o} \rangle = \langle i', o' \rangle$ . We want to show that  $\langle \tilde{i}, \tilde{o} \rangle : B \Rightarrow A \& A'$  is the unique model morphism satisfying  $\langle \tilde{i}, \tilde{o} \rangle \circ in_A = \langle i, o \rangle$  and  $\langle \tilde{i}, \tilde{o} \rangle \circ in_{A'} = \langle i', o' \rangle$ . Existence and uniqueness are clear. We need to show well-formed-ness; that is, we need to verify that  $\langle \tilde{i}, Id \rangle : \mathbf{ent}(B) \Rightarrow \mathbf{ent}(A) \& \mathbf{ent}(A')$  and  $\langle \tilde{o}, Id \rangle : \mathbf{rel}(B) \Rightarrow \mathbf{rel}(A) \& \mathbf{rel}(A')$  are infomorphisms. First, choose instance  $a \in \mathbf{indiv}(A) + \mathbf{indiv}(A')$  and type  $\beta \in \mathbf{typ}(L)$ ; then,  $\tilde{i}(a) \models_B \beta$  iff  $i(a) \models_B \beta$  iff  $a \models_A \beta$  iff  $a \models_{\&} \beta$ . Second, choose instance  $t \in \mathbf{arg}(A) + \mathbf{arg}(A')$  and type  $\rho \in \mathbf{rel}(L)$ ; then,  $\tilde{o}(t) \models_B \rho$  iff  $o(t) \models_B \rho$  iff  $t \models_A \rho$  iff  $t \models_{\&} \rho$ . So the fiber sum  $A \& A'$  is a coproduct in  $\mathbf{Model}(L)$ , and the  $\mathbf{inst}$  functor preserves coproducts.

## Equalizers and Pullbacks

Given a model  $A$ , a *model invariant* is a quadruple  $I = \langle A, T, E, R \rangle$  consisting of a binary relation  $A \subseteq \mathbf{indiv}(A) \times \mathbf{indiv}(A)$  between entity instances (individuals), a binary relation  $T \subseteq \mathbf{arg}(A) \times \mathbf{arg}(A)$  between relation instances (arguments), a set  $E \subseteq \mathbf{ent}(A)$  of entity types, and a set  $R \subseteq \mathbf{rel}(A)$  of relation types, that satisfies the following conditions:

1.  $\langle A, T \rangle$  is a hypergraph relation on  $\mathbf{inst}(A)$ ;
2.  $\langle E, R \rangle$  is a subhypergraph of  $\mathbf{typ}(A)$ ;
3.  $\langle A, E \rangle$  is a classification invariant on  $\mathbf{ent}(A)$ ; and
4.  $\langle T, R \rangle$  is a classification invariant on  $\mathbf{rel}(A)$ .

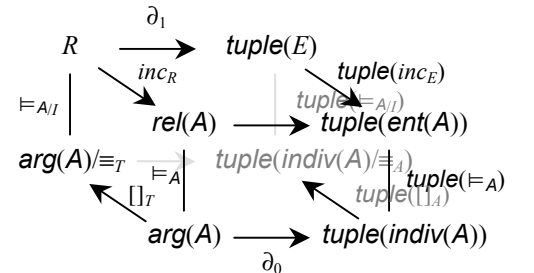
Or more specifically,

1. if  $(t_1, t_2) \in T$ , then  $\mathbf{arity}(t_1) = \mathbf{arity}(t_2)$  and  $\partial_A(t_1)_x \equiv_A \partial_A(t_2)_x$  for all variables  $x \in \mathbf{arity}(t_1) = \mathbf{arity}(t_2)$ ;
2. if  $\rho \in R$ , then  $\partial_A(\rho)_x \in E$  for all variables  $x \in \mathbf{arity}(\rho)$ ;
3. if  $(a_1, a_2) \in A$ , then for every  $\alpha \in E$ ,  $a_1 \models_A \alpha$  iff  $a_2 \models_A \alpha$  for every  $a_1, a_2 \in \mathbf{indiv}(A)$  and  $\alpha \in \mathbf{ent}(A)$ ; and
4. if  $(t_1, t_2) \in T$ , then for every  $\rho \in R$ ,  $t_1 \models_A \rho$  iff  $t_2 \models_A \rho$  for every  $t_1, t_2 \in \mathbf{arg}(A)$  and  $\rho \in \mathbf{rel}(A)$ .

Condition 1 includes, but is more general than, the case  $(\partial_A(t_1)_x, \partial_A(t_2)_x) \in A$ . Condition 3 and 4 state that  $\mathbf{ent}(I) = \langle A, E \rangle$  and  $\mathbf{rel}(I) = \langle T, R \rangle$  are classification invariants, respectively.

A simple inductive proof shows that the quadruple  $\langle \equiv_A, E, \equiv_T, R \rangle$  is also an invariant, where  $\equiv_A$  is the equivalence relation (reflexive, symmetric, transitive closure) generated by  $A$  and  $\equiv_T$  is the equivalence relation generated by  $T$ . The *model quotient of A*, written  $A/I$ , is the model with entity types  $E$ , relation types  $R$ , entity instances (note, no longer just individuals!) the  $A$ -equivalence classes with  $[a] \models_{A/I} \alpha$  when  $a \models_A \alpha$ , and relation instances (note, no longer just tuples of individuals!) the  $T$ -equivalence classes with  $[t] \models_{A/I} \rho$  when  $t \models_A \rho$ . The instance vertex function has the definition  $\partial_{A/I}([t])_x = [\partial_A(t)_x]$ , which is well-defined by condition 1. The type vertex function has the definition  $\partial_{A/I}(\rho) = \partial_A(\rho)$ , which is well-defined by condition 2. Finally, the condition for the tuple projection is satisfied:  $[t] \models_{A/I} \rho$  implies  $\partial_{A/I}([t])_x \models_A \partial_A(\rho)_x$  for every  $t \in \mathbf{arg}(A)$  and  $\rho \in R$ , and  $x \in \mathbf{arity}(t) = \mathbf{arity}(\rho)$ . So, the quotient  $A/I$  is a model, and consists of

- the *hypergraph of instances*  $\mathbf{inst}(A/I) = \langle \mathbf{indiv}(A)/\equiv_A, \mathbf{arg}(A)/\equiv_T, \partial_0 \rangle$ ;
- the *hypergraph of types*  $\mathbf{typ}(A/I) = \langle E, R, \partial_1 \rangle$ ;
- the *entity classification*  $\mathbf{ent}(A/I) = \langle \mathbf{indiv}(A)/\equiv_A, E, \models_{A/I} \rangle$ ;



- the relation classification  $\mathbf{rel}(A/I) = \langle \mathbf{arg}(A)/\equiv_T, R, \models_{A/I} \rangle$ ; and
- the tuple projection  $\partial = \langle \partial_0, \partial_1 \rangle : \mathbf{rel}(A/I) \Rightarrow \mathbf{tuple}(\mathbf{ent}(A/I))$ .

The canonical quotient model morphism  $\tau_I = \langle \llbracket \_ \rrbracket_A, \llbracket \_ \rrbracket_T, inc_E, inc_R \rangle : A/I \Rightarrow A$  is the inclusion on types and the canonical equivalence class quotient function on instances. It consists of

- an instance hypergraph morphism  $\mathbf{inst}(\tau_I) = \langle \llbracket \_ \rrbracket_A, \llbracket \_ \rrbracket_T \rangle : \mathbf{inst}(A/I) \leftarrow \mathbf{inst}(A)$ ;
- a type hypergraph morphism  $\mathbf{typ}(\tau_I) = \langle inc_E, inc_R \rangle : \mathbf{typ}(A/I) \rightarrow \mathbf{typ}(A)$ ;
- an entity infomorphism  $\mathbf{ent}(\tau_I) = \langle \llbracket \_ \rrbracket_A, inc_E \rangle : \mathbf{ent}(A/I) \Rightarrow \mathbf{ent}(A)$ ; and
- a relation infomorphism  $\mathbf{rel}(\tau_I) = \langle \llbracket \_ \rrbracket_T, inc_R \rangle : \mathbf{rel}(A/I) \Rightarrow \mathbf{rel}(A)$ .

that satisfies the required four conditions:

1.  $\partial_1(inc_R(\rho)) = \mathbf{tuple}(inc_E)(\partial_1(\rho))$ ;
2.  $\partial_{A/I}(\llbracket t \rrbracket_T) = \mathbf{tuple}(\llbracket \_ \rrbracket_A)(\partial_A(t))$ ;
3.  $\llbracket a \rrbracket_{A/I} \models_A \alpha$  iff  $a \models_A \alpha$  for each individual  $a \in \mathbf{indiv}(A)$  and each entity type  $\alpha \in E$ ; and
4.  $\llbracket t \rrbracket_T \models_{A/I} \rho$  iff  $t \models_A \rho$  for each argument  $t \in \mathbf{arg}(A)$  and each relation type  $\rho \in R$ .

Given an invariant  $I = \langle A, T, E, R \rangle$  on a model  $A$ , a model morphism  $f = \langle i, o, e, r \rangle : B \Rightarrow A$  respects  $I$  when

1. if  $(a_1, a_2) \in A$ , then  $i(a_1) = i(a_2)$ ;
2. if  $(t_1, t_2) \in T$ , then  $o(t_1) = o(t_2)$ ;
3. for each  $\alpha \in \mathbf{ent}(B)$ ,  $e(\alpha) \in E$ ; and
4. for each  $\rho \in \mathbf{rel}(B)$ ,  $r(\rho) \in R$ .

If  $f = \langle i, o, e, r \rangle : B \Rightarrow A$  respects  $I$ , then it factors uniquely through the quotient: there is a unique model morphism  $\tilde{f} = \langle \tilde{i}, \tilde{o}, \tilde{e}, \tilde{r} \rangle : B \Rightarrow A/I$  such that  $\tilde{f} \circ \tau_I = f$ . The necessary condition means that

1.  $\llbracket a \rrbracket_A \cdot \tilde{i} = i$ , or  $\tilde{i}(\llbracket a \rrbracket_A) = i(a)$  for all individuals  $a \in \mathbf{indiv}(A)$ ;
2.  $\llbracket t \rrbracket_T \cdot \tilde{o} = o$ , or  $\tilde{o}(\llbracket t \rrbracket_T) = o(t)$  for all arguments  $t \in \mathbf{arg}(A)$ ;
3.  $\tilde{e} \cdot inc_E = e$ , or  $\tilde{e}(\alpha) = e(\alpha)$  for all entity types  $\alpha \in E$ ; and
4.  $\tilde{r} \cdot inc_R = r$ , or  $\tilde{r}(\rho) = r(\rho)$  for all relation types  $\rho \in R$ .

The constraints on  $f$  imply that  $\tilde{f}$  is a model morphism. The canonical quotient model morphism  $\tau_I : A/I \Rightarrow A$  respects  $I$ , and its unique morphism is the identity.

Two model morphisms  $f_1 = \langle i_1, o_1, e_1, r_1 \rangle : A \Rightarrow A'$  and  $f_2 = \langle i_2, o_2, e_2, r_2 \rangle : A \Rightarrow A'$  between the same models determine an invariant  $I(f_1, f_2) = \langle A, T, E, R \rangle$  on  $A$  that is defined as follows:

$A = \mathbf{coeq}(i_1, i_2)$  is the coequalizer of  $i_1$  and  $i_2$

– the set of pairs of  $A$ -entity instances  $(i_1(a'), i_2(a'))$  for some  $A'$ -entity instance  $a' \in \mathbf{indiv}(A')$ ;

$T = \mathbf{coeq}(o_1, o_2)$  is the coequalizer of  $o_1$  and  $o_2$

– the set of pairs of  $A$ -relation instances  $(o_1(t'), o_2(t'))$  for some  $A'$ -relation instance  $t' \in \mathbf{arg}(A')$ ;

$E = \mathbf{eq}(e_1, e_2)$  is the equalizer of  $e_1$  and  $e_2$

– the set of  $A$ -entity types  $\alpha \in \mathbf{ent}(A)$  such that  $e_1(\alpha) = e_2(\alpha)$ ; and

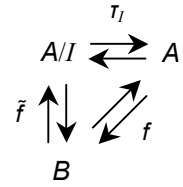
$R = \mathbf{eq}(r_1, r_2)$  is the equalizer of  $r_1$  and  $r_2$

– the set of  $A$ -relation types  $\rho \in \mathbf{rel}(A)$  such that  $r_1(\rho) = r_2(\rho)$ .

The pair consisting of the quotient model  $\mathbf{eq}(f_1, f_2) = A/I(f_1, f_2)$  and the canonical morphism

$\tau_{I(f_1, f_2)} : \mathbf{eq}(f_1, f_2) \Rightarrow A$  is called the *equalizer* of the morphism pair  $f_1$  and  $f_2$ . The canonical quotient model morphism  $\tau_I : \mathbf{eq}(f_1, f_2) \Rightarrow A$  is also called the “unique mediating model morphism” for the equalizer. The equalizer has the universal property that any model morphism  $f = \langle i, o, e, r \rangle : B \Rightarrow A$  satisfying  $f \circ f_1 = f \circ f_2$ ,

factors uniquely through the quotient: there is a unique model morphism  $\tilde{f} = \langle \tilde{i}, \tilde{o}, \tilde{e}, \tilde{r} \rangle : B \Rightarrow A/I$  such that  $\tilde{f} \circ \tau_{I(f_1, f_2)} = f$ , since such a morphism respects the invariant  $I(f_1, f_2)$ . So,  $\mathbf{eq}(f_1, f_2)$  is the equalizer in the category **Model**.



## 6. The Truth Classification

A fundamental satisfaction relation exists between models and expressions. Let  $L$  be any ontological language. An  $L$ -model  $A \in \mathbf{Model}(L)$  satisfies an  $L$ -expression  $\varphi \in \mathbf{expr}(L)$ , denoted  $A \models_L \varphi$ , when  $t \models_A \varphi$  for all arguments  $t \in \mathbf{arg}(A)$  (which is the same as all arguments  $t \in A^{\mathbf{arity}(\varphi)}$ ); that is, when  $\varphi$  generates the top concept in the  $A$ -expression concept lattice (The term “ $A$ -expression” is used here since there is a classification underlying the concept lattice, and classifications need more than a language, they need a model.). Other expressions for this relationship are:  $\varphi$  holds in  $A$ , or  $\varphi$  is satisfied in  $A$ , or  $A$  is a model of  $\varphi$ .

The *truth classification*  $\mathbf{truth}(L) = \langle \mathbf{Model}(L), \mathbf{expr}(L), \models_L \rangle$  is a “meta-classification” – it has  $L$ -models as instances,  $L$ -expressions as types and satisfaction as its classification relation. The type set or *intent*  $\mathbf{ont}(A) = \mathbf{typ}(A) = \{\varphi \in \mathbf{typ}(\mathbf{truth}(L)) \mid A \models_L \varphi\} = \{\varphi \in \mathbf{expr}(L) \mid A \models_L \varphi\}$  of an model ( $\mathbf{truth}(L)$  instance)  $A$  is called the *ontology* of  $A$ . The instance set or *extent*  $\mathbf{mod}(\varphi) = \mathbf{inst}(\varphi) = \{A \in \mathbf{inst}(\mathbf{truth}(L)) \mid A \models_L \varphi\} = \{A \in \mathbf{Model}(L) \mid A \models_L \varphi\}$  of an expression ( $\mathbf{truth}(L)$  type)  $\varphi$  is the collection of models of  $\varphi$ . The truth classification and ontology are closed under universal projection: if  $\varphi$  is a expression,  $A \models_L \varphi$ , and  $x \in \mathbf{arity}_L(\varphi)$ , then  $A \models_L (\forall x)\varphi$ ; that is,  $\varphi \in \mathbf{ont}(A)$  implies  $(\forall x)\varphi \in \mathbf{ont}(A)$ . A formal concept of the truth classification is important in model theory. The *truth concept lattice*  $\mathcal{L}(\mathbf{truth}(L))$  is the concept lattice of the truth classification. The *truth theory*  $\mathbf{Th}(\mathbf{truth}(L)) = \langle \mathbf{expr}(L), \vdash_L \rangle$  is the theory generated by of the truth classification. A constraint  $O \vdash_L O'$  holds in the truth consequence relation when for any  $L$ -model  $A$ , if  $A \models_L \varphi$  for every  $\varphi \in O$  then  $A \models_L \varphi$  for some  $\varphi \in O'$ .

Here we list some facts about the truth classification and its concept lattice. We have additional structure on the instances and types of the truth classification. So an obvious question is “How is this additional structure connected to the satisfaction relation or the order in the truth lattice?”

- An expression that holds in every model of  $L$  is called a *valid expression*, and symbolized by  $\models_L \varphi$ . Valid expressions form the intent of the top concept in the truth concept lattice.
- The concept generated by the subposition (fiber sum)  $A \& A'$  is the join of the concepts generated by the  $L$ -models  $A$  and  $A'$ , since the ontology of the subposition is the intersection  $\mathbf{ont}(A \& A') = \mathbf{ont}(A) \cap \mathbf{ont}(A')$ :  $\varphi \in \mathbf{ont}(A \& A')$  iff  $A \& A' \models_L \varphi$  iff  $t \models_{\&} \varphi$  for every argument  $t \in \mathbf{arg}(A \& A') = \mathbf{arg}(A) + \mathbf{arg}(A')$  iff  $t \models_A \varphi$  for every argument  $t \in \mathbf{arg}(A)$  and  $t \models_{A'} \varphi$  for every argument  $t \in \mathbf{arg}(A')$  iff  $A \models_L \varphi$  and  $A' \models_L \varphi$  iff  $\varphi \in \mathbf{ont}(A)$  and  $\varphi \in \mathbf{ont}(A')$  iff  $\varphi \in \mathbf{ont}(A) \cap \mathbf{ont}(A')$ .
- For any model morphism  $\langle e, r, i, o \rangle : A \Rightarrow A'$  with injective individual and hence argument functions, the type function of the associated expression infomorphism  $\mathbf{expr}(e, r, i, o) = \langle o, \mathbf{expr}(e, r) \rangle : \mathbf{expr}(A) = \mathbf{expr}(A')$  preserves valid expressions,  $A \models_L \varphi$  implies  $A' \models_L \mathbf{expr}(e, r)(\varphi)$ , since  $t' \in \mathbf{inst}_{A'}(\mathbf{expr}(e, r)(\varphi))$  iff  $o(t') \in \mathbf{inst}_A(\varphi)$ ; that is,  $\varphi \in \mathbf{ont}(A)$  implies  $\mathbf{expr}(e, r)(\varphi) \in \mathbf{ont}(A')$ ; or,  $\mathbf{expr}(e, r)(\mathbf{ont}(A)) \subseteq \mathbf{ont}(A')$ . Generalize this to the intent of all formal concepts of truth.
- If  $\langle i, o \rangle : A \rightarrow A'$  is an  $L$ -model morphism, and  $t_1, t_2 \in \mathbf{arg}(A')$  are two  $A'$ -arguments that map to the same  $A$ -argument  $o(t_1) = o(t_2)$ , then  $t_1$  and  $t_2$  are indistinguishable in  $A$ . Same for individuals. So on separable classification, the argument and individual functions are injective.
- The theory of truth  $\mathbf{Th}(\mathbf{truth}(L)) = \langle \mathbf{expr}(L), \vdash_L \rangle$  has as constraints those sequents of compatible expressions of onto logic that are valid in the usual sense.

An *ontological theory*  $O$  (or *ontology*) of  $L$  is a collection of expressions of  $L$ .

- The instance set or extent  $\mathbf{mod}(O) = \mathbf{ext}(O) = \{A \in \mathbf{Model}(L) \mid A \models_L \varphi \text{ for all } \varphi \in O\} = \{A \in \mathbf{Model}(L) \mid \mathbf{ont}(A) \supseteq O\}$  of an ontology  $O$  is the collection of models of  $O$ . This is a subcategory of  $\mathbf{Model}(L)$ .
- The closure of a set of types (an ontology) in the sense of Formal Concept Analysis is the logical closure (consequence) of the ontology. The closure of an ontology  $O$  is the intent of the extent of the ontology  $\mathbf{clo}(O) = \mathbf{ont}(\mathbf{mod}(O))$  in the truth concept lattice. An assertion  $\varphi$  is a consequence of an ontology  $O$ , symbolized  $O \vdash_L \varphi$ , when it is an element of the closure  $\varphi \in \mathbf{clo}(O)$ . So, the closure is the set of all consequences  $\mathbf{clo}(O) = \{\varphi \in \mathbf{expr}(L) \mid O \models \varphi\}$ . An ontology is *closed* when it is closed under the consequence  $\models$  relation,  $O = \mathbf{clo}(O)$ ; that is, when it is an intent in the truth concept lattice.

- An ontology is satisfiable when it has at least one model. Satisfiable ontologies are those that do not generate the bottom of the truth concept lattice.
- An ontology is *complete* when its closure is maximally consistent – minimal non-bottom in the truth concept lattice –  $\text{clo}(O)$  is the intent of a concept just above the bottom.
- Ontologies are normally presented by listing a set of axioms. A set of *axioms* of  $O$  is a set of expressions (we do not need assertions since we have universal closure) with the same consequences as  $O$ .

**Theorem.** For any  $L$ -ontology  $O$  there is an  $L$ -model  $A_O$  whose ontology is the closure of  $O$ :  $\text{ont}(A_O) = \text{clo}(O)$ .  $A_O$  is the terminal object in the subcategory  $\text{mod}(O)$ . Hence, a logical characterization of models will give a logical characterization of ontologies. Since model classification components are characterized by theories, only hypergraphs need to be logically characterized.

**Proof.** If  $\langle i, o \rangle : A \Rightarrow A'$  is a fiber model morphism in  $\text{Model}(L)$ , then  $o(t) \models_{A'} \varphi$  implies  $t \models_A \varphi$  for any argument  $t \in \text{arg}(A)$  and any expression  $\varphi \in \text{expr}(L)$ . Let  $A \rightarrow A'$  be a  $\text{Model}(L)$ -arrow. Then,  $A' \models_L \varphi$  implies  $A \models_L \varphi$ , since  $t' \models_{A'} \varphi$  for all  $t' \in \text{arg}(A')$  implies  $t \models_A \varphi$  for all  $t \in \text{arg}(A)$ . Thus,  $\text{ont}(A') \subseteq \text{ont}(A)$ . This means that the concept generated by  $A$  is lower in the truth concept lattice than the concept generated by  $A'$ ; or  $A$  is the more specific  $L$ -model and  $A'$  is the more generic  $L$ -model. In summary, if there is an  $\text{Model}(L)$ -arrow  $A \rightarrow A'$ , then  $A \leq A'$  in  $\mathcal{L}(\text{truth}(L))$ . Defining  $A_O = \&\text{mod}(O)$  to be the global subposition of all models in the extent of ontology  $O$ , then  $A_O \in \text{mod}(O)$ ,  $\text{mod}(O) = \{A' \in \text{Model}(L) \mid A' \rightarrow A_O\}$  and  $A_O$  is the terminal object in  $\text{mod}(O)$ .

**Theorem.** A language morphism  $\langle e, r \rangle : L \rightarrow L'$  determines a truth infomorphism

$$\text{truth}(e, r) = \langle \text{Model}(e, r), \text{expr}(e, r) \rangle : \text{truth}(L) \Rightarrow \text{truth}(L'),$$

whose type function is the expression function  $\text{expr}(e, r) : \text{expr}(L) \rightarrow \text{expr}(L')$ , and whose instance function is (the object function of) the model functor  $\text{Model}(e, r) : \text{Model}(L) \leftarrow \text{Model}(L')$  between fiber categories. So *truth* is a functor from languages to classifications

**truth** : Language  $\rightarrow$  Classification.

**Proof.** To show this, let  $A'$  be an  $L'$ -model and let  $\varphi \in \text{expr}(L)$  be any  $L$ -expression.  $\text{Model}(e, r)(A') \models_{\text{expr}(e, r)(\varphi)}$  Then,  $\text{Model}(e, r)(A') \models_L \varphi$  iff  $t \models_{\text{Model}(e, r)(A')} \varphi$  for all arguments  $t \in \text{Model}(e, r)(A')^{\text{arity}(\varphi)} = A'^{\text{arity}(\text{expr}(e, r)(\varphi))}$  iff  $t \models_{A'} \text{expr}(e, r)(\varphi)$  for all arguments  $t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi))} = \text{Model}(e, r)(A')^{\text{arity}(\varphi)}$  iff  $A' \models_L \text{expr}(e, r)(\varphi)$ , since

$$t \models_{\text{Model}(e, r)(A')} \varphi \text{ for } t \in \text{Model}(e, r)(A')^{\text{arity}(\varphi)} \text{ iff } t \models_{A'} \text{expr}(e, r)(\varphi) \text{ for } t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi))}$$

is here proven by induction:

- $t \models_{\text{Model}(e, r)(A')} p$  iff  $t \models_{A'} r(p)$  iff  $t \models_{A'} \text{expr}(e, r)(p)$ ;
- $t \models_{\text{Model}(e, r)(A')} \varphi \wedge \psi$  for  $t \in \text{Model}(e, r)(A')^{\text{arity}(\varphi) \cup \text{arity}(\psi)}$  iff  $pr_{\text{arity}(\varphi)}(t) \models_{\text{Model}(e, r)(A')} \varphi$  for  $pr_{\text{arity}(\varphi)}(t) \in \text{Model}(e, r)(A')^{\text{arity}(\varphi)}$  and  $pr_{\text{arity}(\psi)}(t) \models_{\text{Model}(e, r)(A')} \psi$  for  $pr_{\text{arity}(\psi)}(t) \in \text{Model}(e, r)(A')^{\text{arity}(\psi)}$  iff (by induction)  $pr_{\text{arity}(\text{expr}(e, r)(\varphi))}(t) \models_{A'} \text{expr}(e, r)(\varphi)$  for  $pr_{\text{arity}(\text{expr}(e, r)(\varphi))}(t) \in A'^{\text{arity}(\text{expr}(e, r)(\varphi))}$  and  $pr_{\text{arity}(\text{expr}(e, r)(\psi))}(t) \models_{A'} \text{expr}(e, r)(\psi)$  for  $pr_{\text{arity}(\text{expr}(e, r)(\psi))}(t) \in A'^{\text{arity}(\text{expr}(e, r)(\psi))}$  iff  $t \models_{A'} \text{expr}(e, r)(\varphi) \wedge \text{expr}(e, r)(\psi)$  for  $t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi) \wedge \text{expr}(e, r)(\psi))}$  iff  $t \models_{A'} \text{expr}(e, r)(\varphi \wedge \psi)$  for  $t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi \wedge \psi))}$ ;
- $t \models_{\text{Model}(e, r)(A')} \neg \varphi$  for  $t \in \text{Model}(e, r)(A')^{\text{arity}(\varphi)}$  iff  $t \not\models_{\text{Model}(e, r)(A')} \varphi$  for  $t \in \text{Model}(e, r)(A')^{\text{arity}(\varphi)}$  iff (by induction)  $t \not\models_{A'} \text{expr}(e, r)(\varphi)$  for  $t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi))}$  iff  $t \models_{A'} \neg \text{expr}(e, r)(\varphi)$  for  $t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi))}$  iff  $t \models_{A'} \text{expr}(e, r)(\neg \varphi)$  for  $t \in A'^{\text{arity}(\text{expr}(e, r)(\neg \varphi))}$ ;

If  $\varphi$  is an expression and  $\alpha$  is a signature satisfying  $\alpha \vdash \partial_L(\varphi)$ ,

- $t \models_{\text{Model}(e, r)(A')} \varphi[\alpha]$  for  $t \in \text{Model}(e, r)(A')^{\text{arity}(\varphi[\alpha])}$  iff  $t \models_{\text{Model}(e, r)(A')} \varphi$  and  $\partial \text{Model}(e, r)(A')(t) \models \alpha$  in  $\text{entry}(\text{Model}(e, r)(A'))$  for  $t \in \text{Model}(e, r)(A')^{\text{arity}(\varphi)}$  iff (by induction)  $t \models_{A'} \text{expr}(e, r)(\varphi)$  and  $\partial_{A'}(t) \models \alpha$  in  $\text{entry}(A')$  for  $t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi))}$  iff  $t \models_{A'} \text{expr}(e, r)(\varphi)[\alpha]$  for  $t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi)[\alpha])}$  iff  $t \models_{A'} \text{expr}(e, r)(\varphi[\alpha])$  for  $t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi[\alpha]))}$ ;

If  $\varphi$  is an expression and  $x \in \text{arity}_L(\varphi)$ ,

- $s \models_{\text{Model}(e, r)(A')} (\forall x)\varphi$  for  $s \in \text{Model}(e, r)(A')^{\text{arity}(\varphi) - \{x\}}$  iff  $(\forall t \in \text{Model}(e, r)(A')^{\text{arity}(\varphi)})(pr_{\text{arity}(\varphi) - \{x\}}(t) = s$  implies  $t \models_{\text{Model}(e, r)(A')} \varphi$  for  $s \in \text{Model}(e, r)(A')^{\text{arity}(\varphi) - \{x\}}$  iff (by induction)  $(\forall t \in A'^{\text{arity}(\text{expr}(e, r)(\varphi))})(pr_{\text{arity}(\text{expr}(e, r)(\varphi)) - \{x\}}(t) = s$  implies  $t \models_{A'} \text{expr}(e, r)(\varphi)$  for  $s \in A'^{\text{arity}(\text{expr}(e, r)(\varphi)) - \{x\}}$  iff  $s \models_{A'} (\forall x)\text{expr}(e, r)(\varphi)$  for  $s \in A'^{\text{arity}(\text{expr}(e, r)(\varphi)) - \{x\}}$  iff  $s \models_{A'} \text{expr}(e, r)((\forall x)\varphi)$  for  $s \in A'^{\text{arity}(\text{expr}(e, r)((\forall x)\varphi))}$ .

$$\begin{array}{ccccc}
& \text{expr}(e, r) & & \mu_{L'} & \\
\text{expr}(L) & \longrightarrow & \text{expr}(\text{expr}(L')) & \longrightarrow & \text{expr}(L') \\
\vdash_L \Big| & & \Big| \vdash_{\text{expr}(L')} & & \Big| \vdash_{L'} \\
\text{Model}(L) & \longleftarrow & \text{Model}(\text{expr}(L')) & \longleftarrow & \text{Model}(L') \\
& \text{Model}(e, r) & & \text{expr}_{L'} &
\end{array}$$

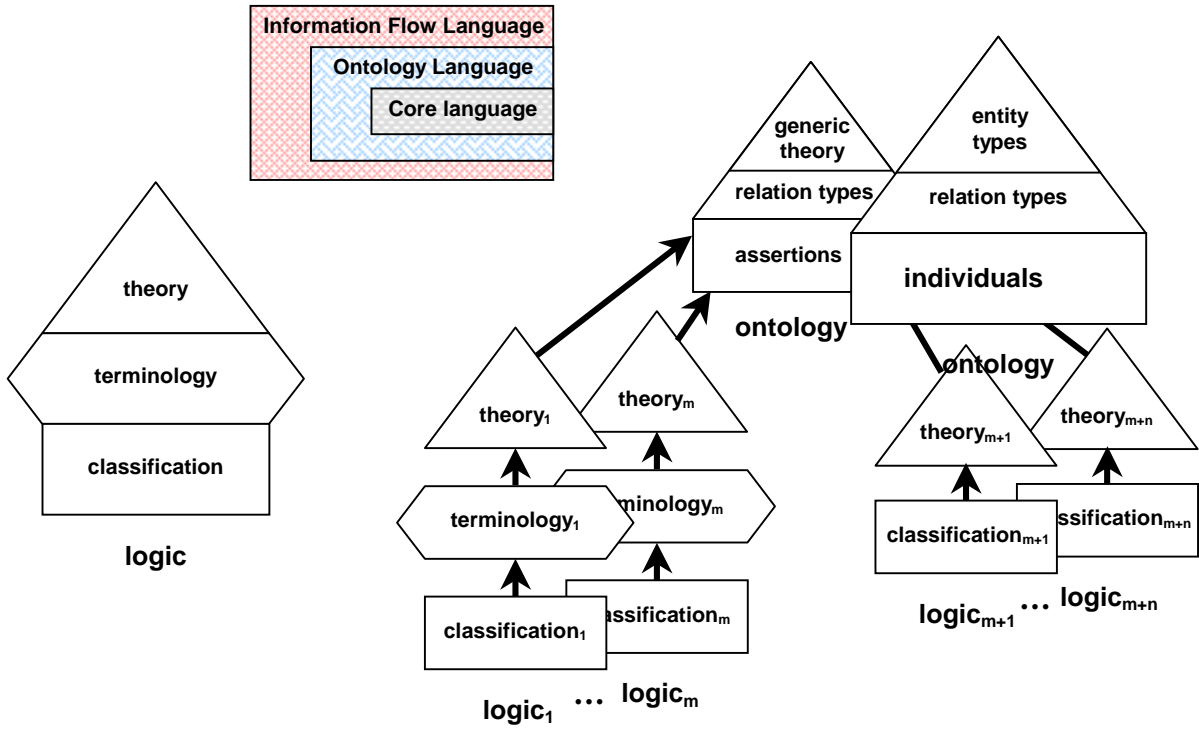
**Corollary.** A ontologic interpretation  $\langle e, r \rangle : L \rightarrow \text{expr}(L')$  determines an extended truth infomorphism

$$\mathbf{truth}^\#(e, r) = \mathbf{truth}(e, r) \circ \langle \text{expr}_{L'}, \mu_{L'} \rangle : \mathbf{truth}(L) \rightleftharpoons \mathbf{truth}(L'),$$

whose type function is the composite  $\text{expr}(e, r) \cdot \mu_{L'} : \text{expr}(L) \rightarrow \text{expr}(\text{expr}(L')) \rightarrow \text{expr}(L')$  and whose instance function is the composite  $\mathbf{Model}(e, r) \cdot \text{expr}_{L'} : \mathbf{Model}(L) \leftarrow \mathbf{Model}(\text{expr}(L')) \leftarrow \mathbf{Model}(L')$ . Extended truth is functorial on the Kliesli category of the expression monad. So extended truth is a functor from languages and terminologies to classifications

$$\mathbf{truth}^\# : \text{Language}_{\text{expr}} \rightarrow \text{Classification}.$$

**Proof.** To show this, we need to prove by induction that  $\langle \text{expr}_{L'}, \mu_{L'} \rangle : \mathbf{truth}(\text{expr}(L')) \rightleftharpoons \mathbf{truth}(L')$  is an infomorphism.



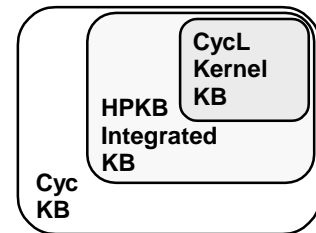
## 7. Knowledge Representation

### Cyc

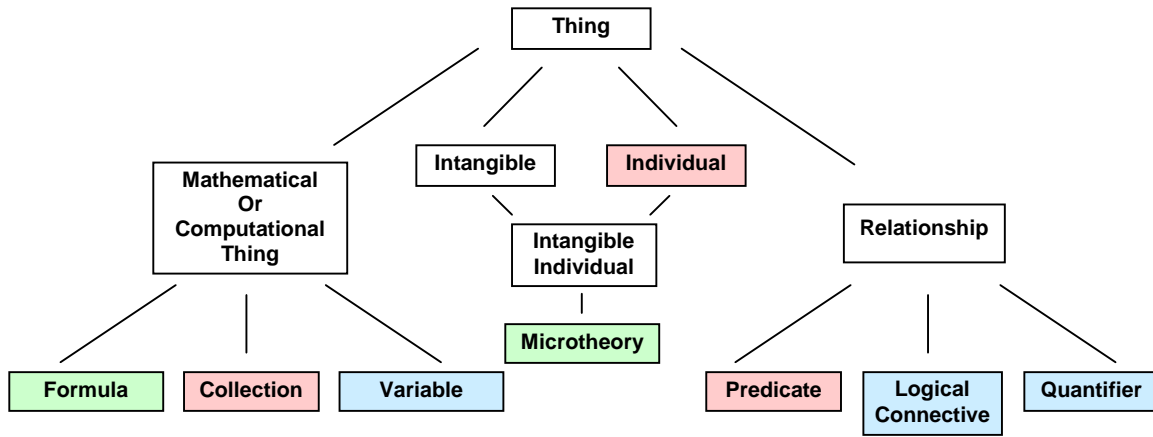
Cyc is the name of a very large, multi-contextual knowledge base and inference engine, the development of which started at the Microelectronics and Computer Technology Corporation (MCC) in Austin, Texas during the early 1980s. The Cyc project began as a dream to create a computerized encyclopedia. Cyc is an attempt to do symbolic artificial intelligence on a massive scale. Its avowed purpose is to break the software brittleness bottleneck once and for all by constructing a foundation of basic "common sense" knowledge. All of the knowledge in Cyc is represented declaratively in the form of logical assertions. Cyc assertions include simple statements of fact, rules about what conclusions to draw if certain statements of fact are satisfied (true), and rules about how to reason with certain types of facts and rules. The Cyc inference engine using deductive reasoning derives new conclusions.

Cycorp represents its knowledge in a form of predicate logic known as CycL, which is a superset of KIF. CycL is the representation language of Cyc. CycL, the Cyc representation language, is a form of first order predicate logic, with equality, augmentations for default reasoning, skolemization, and some second-order features (e.g., quantification over predicates is allowed in some circumstances). It uses a form of circumscription, includes the unique names assumption, and can make use of the closed world assumption where appropriate. The document: <http://www.cyc.com/cycl.html> gives an overview of the language.

A set of CycL sentences forms a knowledge base. There is a *kernel knowledge base* which is in common among all Cyc knowledge bases. The kernel knowledge base is important because it includes some axioms as well as the hierarchy of collections and individuals. A Cyc knowledge base is thought of as a large collection of Cyc *assertions*, with each assertion being no more about its first argument than its last one. Inference in Cyc is a general logical deduction. Outside the assertion collection are all the Cyc *constants*. The constants are linked to all the assertions in the knowledge base that reference the constant. Each of assertion in the knowledge base can itself be treated as a constant, having its own links to other assertions that reference it. In the following, the hierarchy on the left is a breakdown of the constant terms in the Cyc knowledge base, and the hierarchy on the right is a breakdown of the Cyc knowledge base by rules or formulae.



<ul style="list-style-type: none"> <li>• [40%] Categories           <ul style="list-style-type: none"> <li>◦ [5%] Categories of categories</li> <li>◦ Categories of individuals               <ul style="list-style-type: none"> <li>▪ [3.0%] Categories of intangible objects, information bearing objects, numbers, and physical attributes, etc.</li> <li>▪ [18.5%] Categories of tangible objects, living things, artifacts</li> </ul> </li> <li>◦ [18.0%] Categories of script types               <ul style="list-style-type: none"> <li>▪ Actions by one person: physiological actions, problem solving and planning, work/hobby/etc actions</li> <li>▪ Actions by more than one person: communication, rites of passage, trade and commerce, etc.</li> <li>▪ Actions of natural phenomena (weather etc.)</li> </ul> </li> </ul> </li> <li>• [15%] Predicates and Functions           <ul style="list-style-type: none"> <li>◦ Unary: see Categories and Attributes</li> <li>◦ [12.0%] Binary</li> <li>◦ [2.0%] Ternary</li> <li>◦ [1.0%] Quaternary or more</li> </ul> </li> <li>• [10%] Attributes</li> <li>• [15%] Lexical objects (words, parts of speech, tense, number, gender)</li> <li>• [15%] Proper nouns (specific people, places, languages, events, etc.)</li> <li>• [1.5%] Microtheories (long lived contexts)</li> <li>• [3.5%] Misc. and sundry</li> </ul>	<ul style="list-style-type: none"> <li>• [25%] Taxonomic information (e.g., type constraints on predicates)</li> <li>• [35%] Partonomic relations           <ul style="list-style-type: none"> <li>◦ [5%] General relationships</li> <li>◦ [15%] What kind of parts physical/anatomical/sub-events might various types of objects have?</li> <li>◦ [15%] What kind of actors are involved in various script types?</li> </ul> </li> <li>• [5%] Information about specific people, places, etc.</li> <li>• [10%] Lexical information           <ul style="list-style-type: none"> <li>◦ [8%] Linguistic properties of different word senses</li> <li>◦ [2%] Denotations of word senses</li> </ul> </li> <li>• [10%] More complex information interrelating script types, people and tangible objects.</li> <li>• [10%] General topics (time, space, intentions, stuff, numbers, etc.)</li> <li>• [5%] Misc. and sundry formulae</li> </ul>
---	---



CycL	Onto Logic
<b>Basic notions:</b>	
constant #Collection	type $ent(A)+indiv(A)+rel(A)$
	entity type $\alpha \in ent(A)$
individual #Individual	individual $a \in indiv(A)$
predicate #Predicate	relation type $\rho \in rel(A)$
predicate arity	relation arity $arity(\rho)$
predicate argument types	relation signature $\partial_A(\rho)$
variable	variable $x \in var$
formula #CycFormula	expression $\varphi \in expr(typ(A))$
ground atomic formula	relation classification (relationship) $t \models_A \rho$
microtheory or context	ontology (ontological theory) $O$
CycL knowledge base (KB)	ontological model $A$
<b>Structural relation types:</b>	
#isa(#ReifiableTerm, #Collection)	classification relation $\models_A$ of some model $A$
#\$genls(#Collection, #Collection)	entails (subtype) $\vdash$
#\$arity(#Relationship, #Integer)	card(arity)
#\$argnIsa(#Relationship, #Collection)	$\partial_A(\varphi)_n$ , the signature function
#\$ist(#Microtheory, #CycFormula)	satisfies, consequence $\vdash$ $O \vdash \varphi$ when $clo(O) \ni \varphi$
#\$genlMt(#Microtheory, #Microtheory)	ontology (truth lattice) order: $O_1 \leq O_2$ when $clo(O_1) \supseteq clo(O_2)$ $O_1 \leq O_2$ and $O_2 \vdash \varphi$ implies $O_1 \vdash \varphi$
#\$genlPreds(#Predicate, #Predicate)	expression order: $\varphi_1 \leq \varphi_2$ when $clo(\varphi_1) \ni \varphi_2$
<b>Logical connectives/quantifiers:</b>	
false, true	
and, or, not, implies, equivalent	$\wedge\Phi, \vee\Phi, \neg\varphi, (\varphi \rightarrow \psi), (\varphi \leftrightarrow \psi)$
—	$\varphi[\alpha]$ , subtype constraint
forall, exists	$(\forall x)\varphi, (\exists x)\varphi$
<b>Forms of Terms:</b>	
$(p a_1 \dots a_n)$	$\rho : (\alpha_x)_{x \in arity(\rho)}$
$(f a_1 \dots a_n)$	
(not $e_1$ )	
(and $e_1 \dots$ )	
(implies $e_2 e_2$ )	
(forall ?x e)	

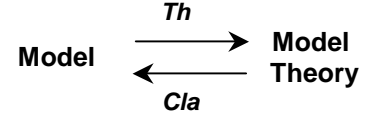
CycL constants represent: classes; individuals; predicates; logical connectives (and, or, not, equivalent, implies); and quantifier names (forAll, thereExists, thereExistAtLeast, ...). In the following table, on the left are some very basic Cyc entity types, and on the right are some of the subtyping between these basic types.

Assertion	ReifiableTerm < CycIndexedTerm < Thing
AttributeValue	Predicate < FunctionTheMathematicalType
Collection	ReifiableFunction < NonPredicateFunction < FunctionTheMathematicalType
CycELVariable	FunctionTheMathematicalType < Relationship < IndividualObject < Thing
CycExpression	Assertion < CycIndexedTerm
CycFormula	Assertion < IndividualObject
CycIndexedTerm	IndividualObject < Thing
FunctionTheMathematicalType	
IndividualObject	
Microtheory	
NonPredicateFunction	
Predicate	
ReifiableFunction	
ReifiableTerm	
Relationship	
Set-Mathematical	
SetOrCollection	
SingleEntry	
TheSet	
TheTerm	
Thing	

Here are some examples: on the left the Cyc syntax and on the right the onto logic notation.

(isa DougLenat Person)	DougLenat $\models_A$ Person
(government Canada)	government(Canada)
(likesAsFriend DougLenat KedithGoolsbey)	(DougLenat, KedithGoolsbey) $\models_A$ likesAsFriend

## 8. Problems



### Model Theories

We would like to characterize models in the same way that Information Flow characterizes classifications. Information Flow characterizes classifications in terms of (regular) Information Flow theories. Since models have components that are classifications, we would like the model characterization to be built out of the component classification characterizations. A *ontological model theory*  $T = \langle \mathbf{ent}(T), \mathbf{rel}(T), \partial_T \rangle$  is either a hypergraph of (classification) theories or a theory of hypergraphs. It consists of

- an *entity theory*  $\mathbf{ent}(T) = \langle \mathbf{ent}(T), \vdash_T \rangle$ ;
- a *relation theory*  $\mathbf{rel}(T) = \langle \mathbf{rel}(T), \vdash_T \rangle$ ; and
- a *hypergraph of types*  $\mathbf{typ}(T) = \langle \mathbf{ent}(T), \mathbf{rel}(T), \partial_T \rangle$ .

$$\mathbf{rel}(T) \xrightarrow{\partial_T} \mathbf{tuple}(\mathbf{ent}(T))$$

To require the function  $\partial_T$  to be a theory interpretation would be too strong: the fact that two relations are disjoint does not necessarily require their signatures to be disjoint. Any model  $A$  determines a model theory  $Th(A)$ , whose entity theory is determined by its entity classification  $\mathbf{ent}(Th(A)) = Th(\mathbf{ent}(A))$ , whose relation theory is determined by its relation classification  $\mathbf{rel}(Th(A)) = Th(\mathbf{rel}(A))$ , and whose type hypergraph is its own type hypergraph  $\mathbf{typ}(Th(A)) = \mathbf{typ}(A)$ . We know from Information Theory that the component theories are special theories called regular theories – they satisfy structural axioms such as identity, weakening, cut and partition. Are there any other structural axioms satisfied by model theories of the form  $Th(A)$ ? A (*model theory*) *interpretation*  $\langle e, r \rangle : T_1 \rightarrow T_2$ , is a pair of functions, where

- $e : \mathbf{ent}(T_1) \rightarrow \mathbf{ent}(T_2)$  is a theory interpretation;
- $r : \mathbf{rel}(T_1) \rightarrow \mathbf{rel}(T_2)$  is a theory interpretation; and
- $\langle e, r \rangle : \mathbf{typ}(T_1) \rightarrow \mathbf{typ}(T_2)$  is a hypergraph morphism.

Any model morphism determines a model theory interpretation. There is a theory functor **Th** : **Model** → **Model Theory**.

A model theory  $T$  determines an entity classification  $\mathbf{ent}(T) = \langle \mathbf{indiv}(T), \mathbf{ent}(T), \exists_T \rangle$ , whose instance set is the set of consistent subsets of entity types. We use the terminology *formal individual* and the notation  $\mathbf{indiv}(T)$  for the consistent subsets of  $T$ -types. A *formal argument*  $T \subseteq \mathbf{rel}(T)$  is a consistent subset of the relation theory  $\mathbf{rel}(T)$  that satisfies the following two constraints:

- **[coherent:]** There is a subset of variables  $\mathbf{arity}(T) \subseteq \mathbf{var}$  called the *arity* of  $T$ , and all types in  $T$  have that arity: if  $\rho \in T$ , then  $\mathbf{arity}(\rho) = \mathbf{arity}(T)$ .
- **[coordinated:]** For all  $x \in \mathbf{arity}(T)$  the set of entity types  $\partial_0(T)_x = \{\partial_T(\rho)_x \mid \rho \in T\}$  is a formal individual of  $T$ .

Let  $\mathbf{arg}(T)$  denote the set of all formal arguments. A model theory  $T$  determines a relation classification  $\mathbf{rel}(T) = \langle \mathbf{arg}(T), \mathbf{rel}(T), \exists_T \rangle$ , whose instance set is the set of formal arguments of  $T$ . In summary, a model theory  $T$  determines the model  $Mod(T) = \langle \mathbf{ent}(T), \mathbf{rel}(T), \partial \rangle$  consisting of

- the *hypergraph of instances*  $\mathbf{inst}(T) = \langle \mathbf{indiv}(A), \mathbf{arg}(T), \partial_0 \rangle$ ;
- the *hypergraph of types*  $\mathbf{typ}(T) = \langle \mathbf{ent}(T), \mathbf{rel}(T), \partial_T \rangle$ ;
- the *entity classification*  $\mathbf{ent}(T) = \langle \mathbf{indiv}(T), \mathbf{ent}(T), \exists_T \rangle$ ;
- the *relation classification*  $\mathbf{rel}(A) = \langle \mathbf{arg}(T), \mathbf{rel}(T), \exists_T \rangle$ ; and
- the *tuple projection*  $\partial = \langle \partial_0, \partial_T \rangle : \mathbf{rel}(T) \Rightarrow \mathbf{tuple}(\mathbf{ent}(T))$ .

$$\begin{array}{ccc} \mathbf{rel}(T_1) & \xrightarrow{r} & \mathbf{rel}(T_2) \\ \partial_L \downarrow & & \downarrow \partial_L \\ \mathbf{tuple}(\mathbf{ent}(T_1)) & \xrightarrow[\mathbf{tuple}(e)]{} & \mathbf{tuple}(\mathbf{ent}(T_2)) \\ \mathbf{ent}(T_1) & \xrightarrow[e]{} & \mathbf{ent}(T_2) \end{array}$$

$$\begin{array}{ccc} \mathbf{rel}(T) & \xrightarrow{\partial_T} & \mathbf{tuple}(\mathbf{ent}(T)) \\ \exists_T \Big| & & \Big| \mathbf{tuple}(\exists_T) \\ \mathbf{arg}(T) & \xrightarrow[\partial_0]{} & \mathbf{tuple}(\mathbf{indiv}(T)) \end{array}$$

### Problems:

- A model theory morphism does not determine a model morphism, since coordination is not preserved.
- If  $T = Th(A)$  is the model theory of some model  $A$  and  $t \in \mathbf{arg}(A)$ , then  $\mathbf{typ}_A(t) = \{\rho \in \mathbf{rel}(A) \mid t \vDash_A \rho\}$  is not necessarily a formal argument of  $Th(A)$  with  $\mathbf{arity}(\mathbf{typ}_A(t)) = \mathbf{arity}(t)$ .

## 9. Future Work

1. Define reification, mapping a relation type to an entity type, plus  $\partial_L(\rho)$  participate functions.
2. Show that **Model** has colimits. Show that **Model(L)** has colimits.
3. Show that **Model(L)** has equalizers; you need to define invariants for fixed type set.
4. Define ontology-sharing between two models  $A_0$  and  $A_1$  in terms of the functor **truth : Language  $\rightarrow$  Classification**.
5. Extend Information Flow using models in place of classifications.
6. Incorporate individuals into expressions (formulas). Call closed formulas “assertions”. Assertions are equivalent to individual-expression classifications. Interpret *CycL* in Onto Logic – do Cyc contexts (Microtheories) via this equivalence.
7. Replace “argument”  $arg(A)$  with “record”  $rec(A)$ .
8. Abstract the language, using “entity instance” rather than “individual” and “relation instance” instead of “record”; individuals and records are basic cases; limits/colimits/powerset give more general instances; incorporating types in instances and instances in types will also occur.  
 $ent(A) = \langle inst.ent(A), typ.ent(A), \models_A \rangle$  and  $rel(A) = \langle inst.rel(A), type.rel(A), \models_A \rangle$
9. Develop entity fibers in **Model**; develop corresponding model theories.
10. Map orders to classifications; incorporate types into instances using subposition.

## 10. References

Barwise, Jon & Jerry Seligman (1997). *Information Flow: the Logic of Distributed Systems*. Cambridge University Press, Cambridge 1997.

Kent, Robert (2000). The Information Flow Foundation for Conceptual Knowledge Organization. In: *Proceedings of ISKO 6: Dynamism and Stability in Knowledge Organization*. Toronto, 2000.

Prediger, Susanne & Gerd Stumme (1999). Theory-driven Logical Scaling: Conceptual Information Systems meet Description Logics. In: *Knowledge Representation and Data Bases*, 1999: pp. 46–49.

Schmidt-Schauss, Manfred & Gert Smolka (1991). Attribute concept descriptions with complements. In: *Artificial Intelligence* 48, 1991.