# Common Logic

## Motivations and Some Gentle Theory

Christopher Menzel
Philosophy Department
Texas A&M University
*cmenzel@tamu.edu*

# In Praise of FOL — Representation

- FOL is wonderfully expressive
  - If you can't say it in FOL, you can't say it!
- The simplest reason for this
  - Names for denoting things
    - '*PatHayes*', '*NGC1976*', '17', 'ω'
  - Predicates for describing the properties of, and relations among, things
    - $Effusive(PatHayes)$, $Nebula(NGC19786)$, $\omega < \omega+17$
  - Quantifiers for expressing generality
    - Nebulas exist: $(\exists x)Nebula(x)$
    - If everyone is effusive, Hayes is: $(\forall x)Effusive(x) \rightarrow Effusive(PatHayes)$
    - Some infinite ordinal is less than all other infinite ordinals: $(\exists x)(Ord(x) \,\&\, Inf(x) \,\&\, (\forall y)(Ord(y) \,\&\, Inf(y) \,\&\, x \neq y \rightarrow x<y))$

# In Praise of FOL — Theory

- Simple, rigorous syntax
- Clear, well-understood semantics (model theory)
- Semantically complete proof theory
  - Albeit only semi-decidable…
- For these reasons, it has become a virtually universal framework for formal representation and a standard (though obviously not unique) platform for automated reasoning.
  - Notably, OWL is basically a class theory of fragment of FOL
  - Otter, Prover9, Tau, E-SETHEO, Vampire, Waldmeister, etc are all first-order theorem provers

# Familiar Features of FOL

- Strict syntactic typing
  - Basic lexical elements divided strictly into disjoint classes
    - Predicate symbols, function symbols, individual constants
    - Predicates/Fn symbols can take only individual constants as arguments
    - Individual constants cannot take arguments
- Fixed signatures
  - Each predicate and function symbol takes a fixed number of arguments
- Strict semantic typing
  - Single domain of "individuals"
  - Individual constants only denote things in the domain
  - Predicate/function symbols denote things outside the domain
- Extensionality
  - The semantic values of predicate/function symbols are SETS of (n-tuples of) individuals

# But…

*These friendly, familiar features of FOL can be liabilities on OPEN NETWORKS!*

# Open Networks: Challenge

- Challenges arise when information is published on an open network such as the Web.
  - Publication typically places an arbitrarily large gap between context of assertion and context of use.
  - Neither context can assume any common state with the other, and the network need not support negotiation or communication between them (e.g. HTTP).
- The logical semantics determines entailment relationships between pieces of logical information.
  - In order to be useful, this should apply in the same way at all places on the network, without needing

# Open Networks: The Logical Solution

- The logical semantics (grounded in a common ontology) determines entailment relationships between pieces of logical information.
- But...

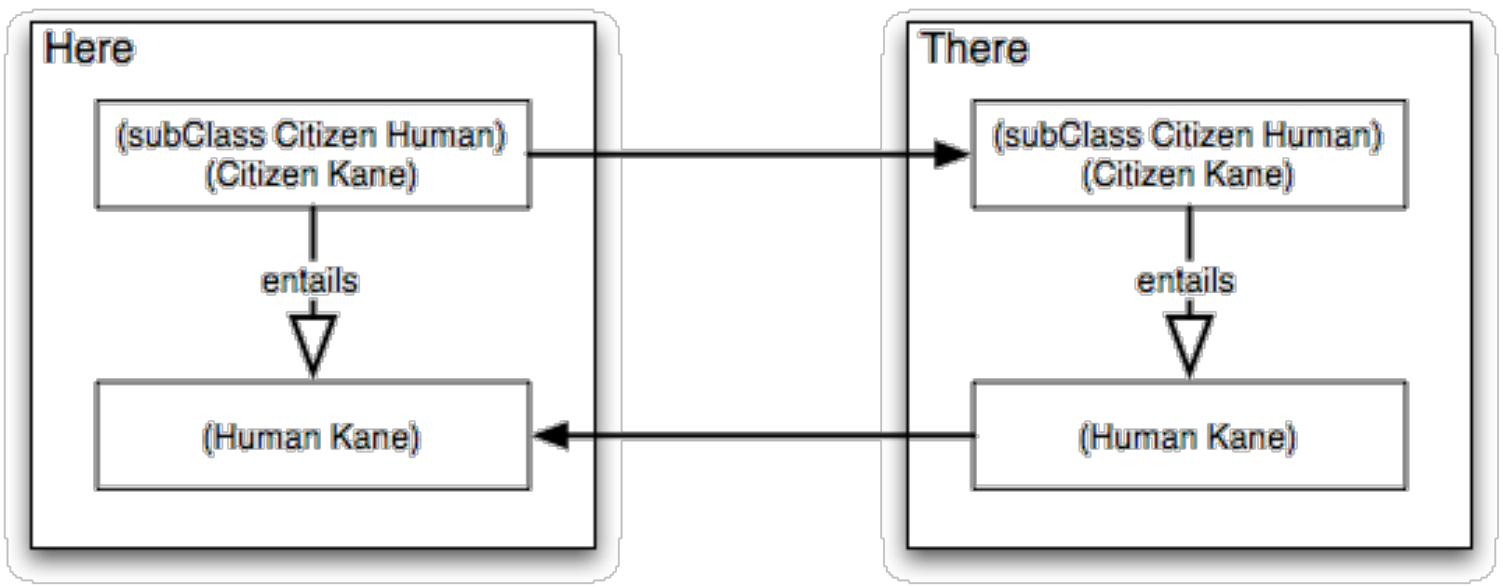
  *In order to be useful, entailment relationships should apply in the same way at all places on the network, without needing negotiation to check compatibility of assumptions.*

# Entailment and Open Networks

Basic property:

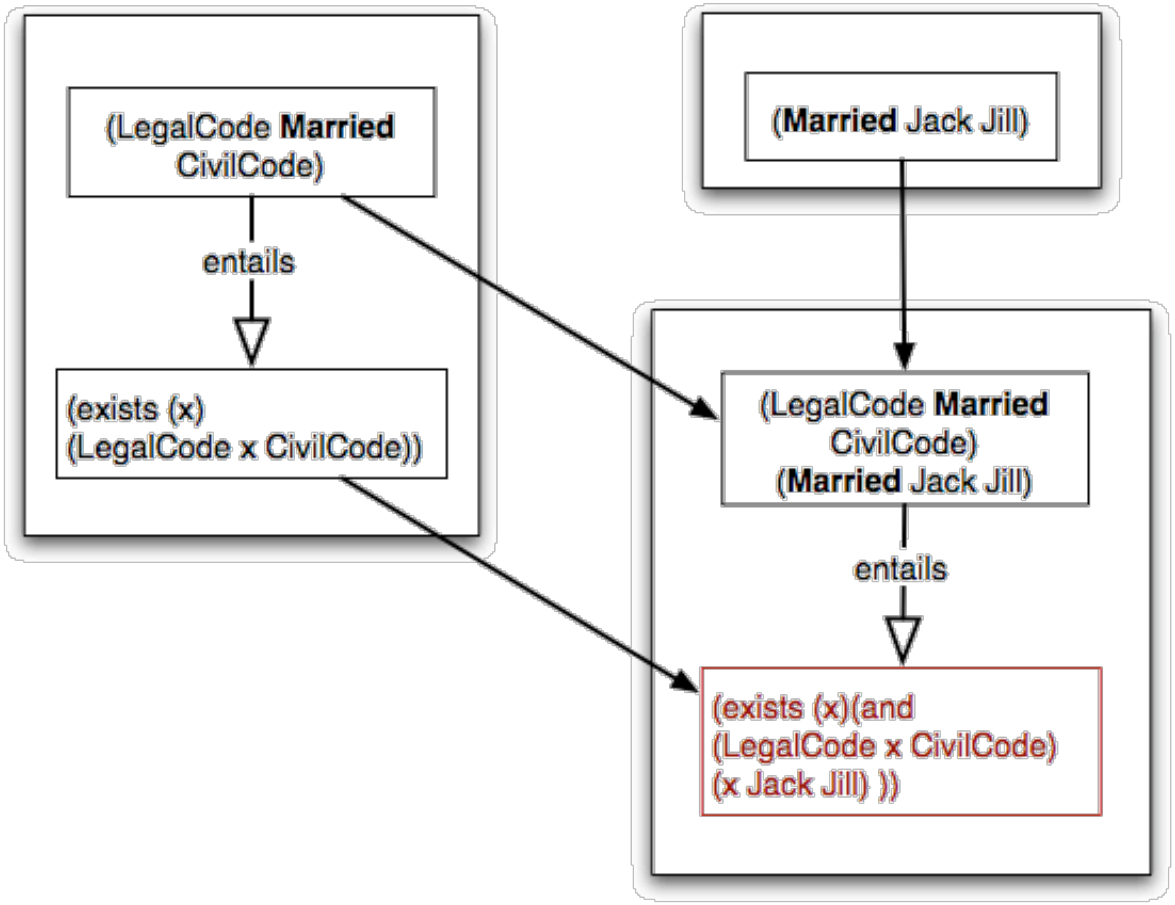
*entailment should commute with communication.*

Kind of obvious in simple cases…



# Entailment and Open Networks II

… but has some non-trivial consequences:



# Open Networks and Type/Signature Variability

- Assumptions can differ widely across frameworks about syntactic type and signature
  - *Mother*(*Mary*)
    - (Type of '*Mother*': PredSym; Sig: Object)
  - *Mother*(*Mary*,*Jesus*)
    - (Type of '*Mother*': PredSym; Sig: Object x Object)
  - *Mother*(*Mary*,*Jesus*,*0CE-33CE*)
    - (Type of '*Mother*': PredSym; Sig: Object x Object x Interval)
  - *Mother*(*Jesus*) = *Mary*
    - (Type of '*Mother*': FunSym; Sig: Object → Object)
  - *Holds*(*Mother*,*Mary*,*Jesus*)
    - (Type of '*Mother*': IndConstant)
      - NB: This usage might imply quantification over properties or classes

# Inference, Type Freedom, and Open Networks

- Inference rules must apply to names in a uniform way
  - We should be able to use *equality reasoning* on relations and functions, since we can do so for individuals:
    - If married=hitched and X and Y are married, then X and Y are hitched
      - $(Married{=}Hitched \;\&\; Married(x,y)) \rightarrow Hitched(x,y)$
  - We should be able to *quantify over* relations, since we can quantify over individuals:
    - If Socrates is wise, then there is something that Socrates is.
      - $Wise(Socrates) \rightarrow \exists FF(Socrates)$
  - And we should be able to *apply relations to* relations, since we can apply them to individuals:
    - Whenever he's running, John hates it
      - $\forall t\ (Running(t, john) \rightarrow hates(john, Running, t)$
    - $Symmetric(Married)$
    - `(rdf:type rdfs:Class rdfs:Class)`

# KIF — A Good Start

- KIF has some of the features we need
  - KIF is "signature-free"
  - KIF's predicates are "variably polyadic"
    - Can take any number of arguments (including zero)
    - All occurrences of "`Mother`" refer to the same entity
  - KIF has "sequence markers" (see below)
- *BUT...*

# KIF: Theoretical Problems

- Not type-free
- The *Formalization* Problem
  - No general model theory for all constructs of KIF
    - Hence no rigorous, general notion of meaning
- The *Conformance* Problem
  - KIF is itself Yet Another Representation Language
  - A broader notion of conformance is needed is needed that sanctions many languages that differ in both expressivity and surface form
- The *Soooo Last Century* Problem
  - KIF is dated – developed prior to the emergence of the Semantic Web
    - Not standarized
    - Doesn't follow current XML-based paradigm
    - Hence, connections to RDF/OWL/etc unclear

# Enter Common Logic

- CL is an abstract generalization and extension of KIF
- Full first-order expressibility *and then some*
- General purpose syntax for communicating declarative, logic-based information
- Designed for easy, natural use on the web
  - Flexibility for use on open networks
    - Type-free and signature free (*if you want!*)
  - No gratuitous assumptions about logical relationships between expressions (notably, as found in different ontologies)

# CL: Abstract Syntax

- A *text* is either a set or list or bag of phrases.
  - A piece of text may be identified by a name
- A *phrase* is either a comment, a module, a sentence, or an importation.
- A *comment* is a piece of data.
  - No particular restrictions are placed on comments.
  - Comments can be attached to other comments.
- A *module* consists of a name and a text called the "body text".
  - The module name indicates the local domain of discourse in which the text is to be understood
- An *importation* contains a name. (More below)

# CL: Abstract Syntax (cont.)

- A *sentence* is either an atom, a boolean sentence, or a quantified sentence.
  - A sentence may have an attached comment.
- A *boolean sentence* has a type, called a *connective*, and a number of sentences, called the *components* of the sentence.
  - The number depends on the type.
  - Every CL dialect must distinguish the following types: negation, conjunction, disjunction, conditional, and biconditional with, respectively, one, any number, any number, two and two components.
- A *quantified sentence* has (i) a type, called a *quantifier*, (ii) a finite, nonrepeating sequence of names and sequence markers called the *binding sequence*, each element of which is called a *binding* of the quantified sentence, and (iii) a sentence called the *body* of the quantified sentence.
  - Every CL dialect must distinguish the existential and universal quantifier types.
  - Bondage, freedom, and quantifier scope understood as usual.
  - Any name can serve as a binding.

# CL: Abstract Syntax (cont.)

- An *atom* is either an *equation* containing two *arguments*, which are terms, or an atomic sentence.
- An *atomic sentence* consists of a term, called the *predicate*, and a term sequence called the *argument sequence*.
  - Each term in the term sequence of an atomic sentence is called an *argument* of the sentence.
  - Any name can be the predicate in an atomic sentence.
- A *term* is either a name or a functional term.
  - Terms may have attached comments.
- A *functional term* consists of a term, called the "operator," and a term sequence called the "argument sequence".
  - Parallel qualifications to atomic sentences.
- A *term sequence* is a finite sequence of terms or sequence markers.
  - A term sequence may be empty.

# Features of the CL Abstract Syntax

- Well...abstraction!
  - No specification of any concrete syntactic forms
  - Specific form left to the KR designers
    - A given KR language needn't use all the features of CL
      - E.g., Description Logics lacking negation
      - Conformance defined flexibly enough to allow a side range of CL "dialects", including "traditional" first-order languages
  - "Every cloud has a silver lining" in GOFOL, CGs, and KIF
    - $\forall x(Cloud(x) \rightarrow \exists y(Lining(y) \wedge Silver(y) \wedge Has(x,y))$
    - [@every*x] [If: (Cloud ?x) [Then: [*y] (Lining ?y) (Silver ?y) (Has ?x ?y)]]
    - ```
      (forall (?x ?y)
         (implies (Cloud ?x))
                  (exists (?y)
                     (and (Lining ?y) (Silver ?y) (Has ?x ?y))))
      ```
      ```

# Features of the Abstract Syntax II

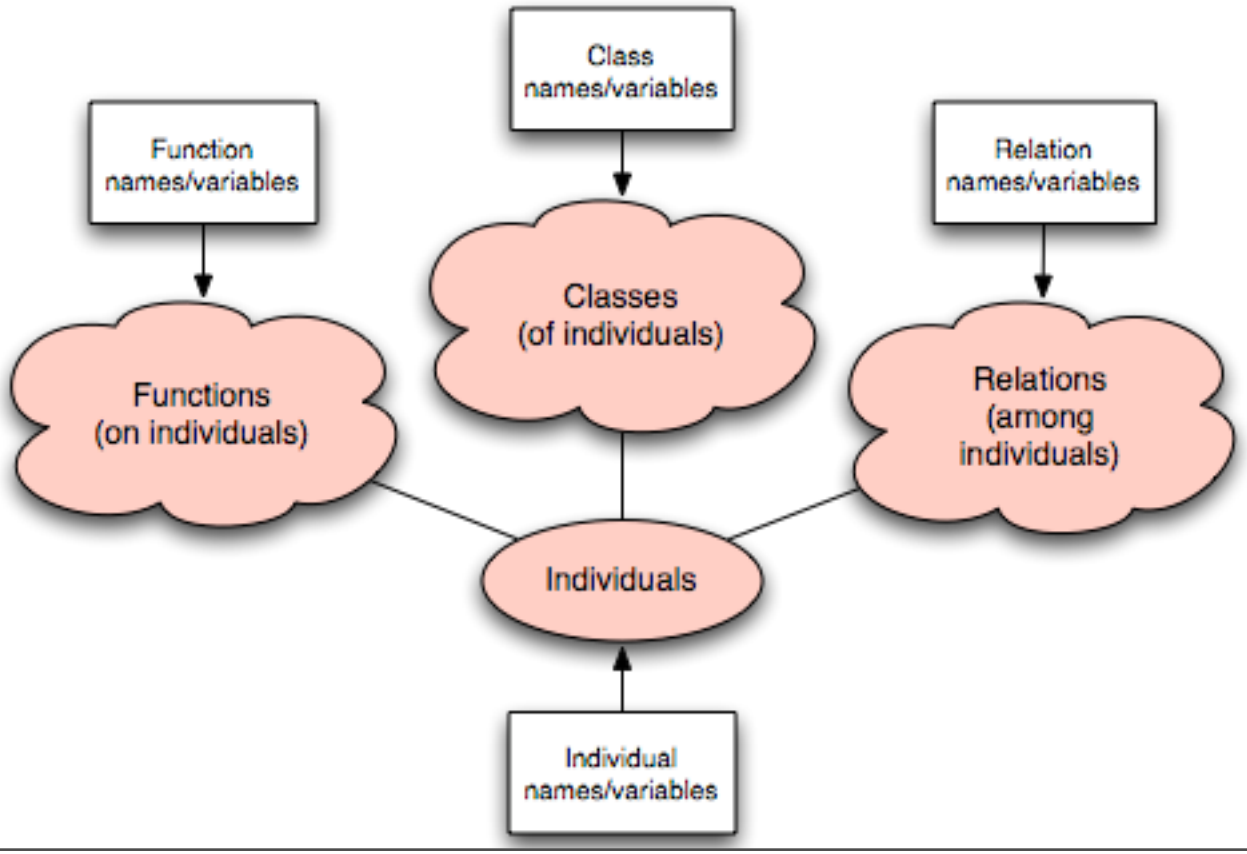
- Type freedom
  - There are only logical operators and names in the basic lexicon
  - Same name can play the role of individual constant, predicate symbol, or function symbol
    - *Symmetric*(*Married*) — OK!!
    - *Married*(*Bill*,*Hillary*) — OK!!
    - *Married*(*Bill*,*Hillary*) = 1 — OK!!
- Signature freedom
  - There are no predicates *per se* (***predicate*** is a *role*) hence no specification of arity (number of arguments a predicate can take)
  - There can be atomic sentences of different lengths with the same predicate
    - *Mother*(*Mary*,*Jesus*) and *Mother*(*Mary*,*Jesus*,*CE0-CE33*) — Both OK!!

# Features of the Abstract Syntax III

- "Higher-order" quantification
  - Quantification of variables (well, *names*, really) occurring predicate/function position
    - $\exists F \forall x(F(x) \leftrightarrow Loves(x,Chocolate) \wedge Hates(x,RedWine))$ — *OK!!*
- Properties and relations are "first-class citizens"
  - Properties and relations are in the domain of quantification (in the "freest" CL dialects)
  - Hence, can both predicate properties of them and predicate them of other things — perhaps in the same sentence
    - $\forall F\,(Symmetric(F) \rightarrow \forall x \forall y(F(x,y) \rightarrow F(y,x)))$ — *OK!!*
- A Consequence: Self-exemplification
  - Properties can be predicated of themselves
    - E.g., $Property(Property)$ – *OK!!*
    - $\exists F \forall G(F(G) \leftrightarrow \sim G(G))$ — *OK!! (But, of course, FALSE, on pain of paradox!)*

# Traditional Model Theory

- Traditional model theories for first-, second-, and higher-order logics map each name/variable type to a different semantic construct
  - First-order logic permits quantification over individuals only
  - Higher-order logics permit quantification over functions/classes/relations

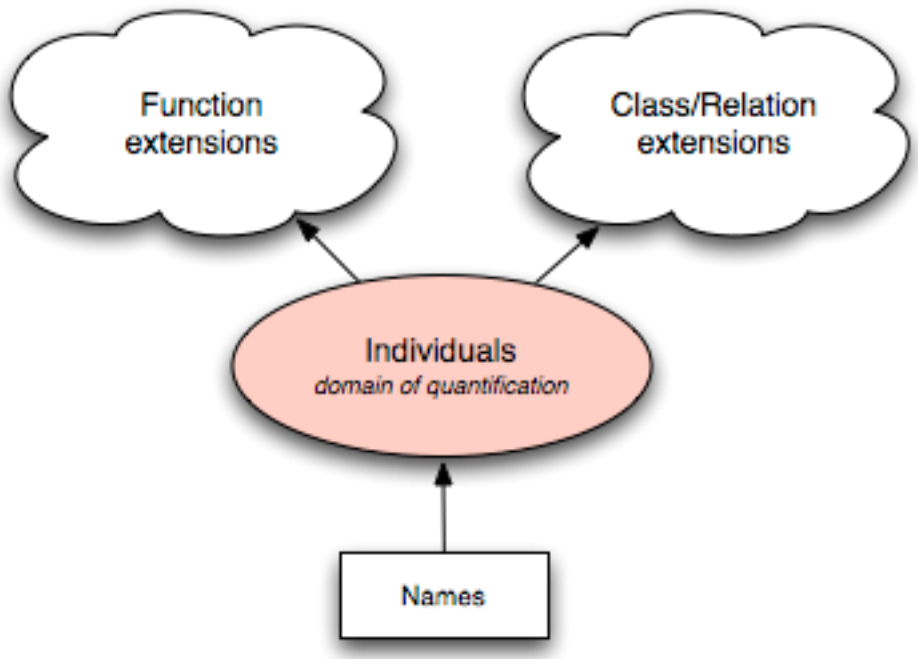


# Traditional Model Theory II

- Two things prevent "collapsing" individuals, functions, classes, relations into a single domain of logical individuals
  - 1. *Cardinality problems*: There are more functions/ classes/relations over individuals than there are individuals
  - 2. *Well-foundedness problems*: Functions/classes/relations in standard HO logic defined as sets of (ordered *n*-tuples of) individuals
    - Hence, if functions/classes/relations are treated as individuals, we have to be able to make sense of functions that apply to themselves, classes that contain themselves, and relations that relate themselves to other individuals.
    - This violates the well-foundedness condition in standard set theories

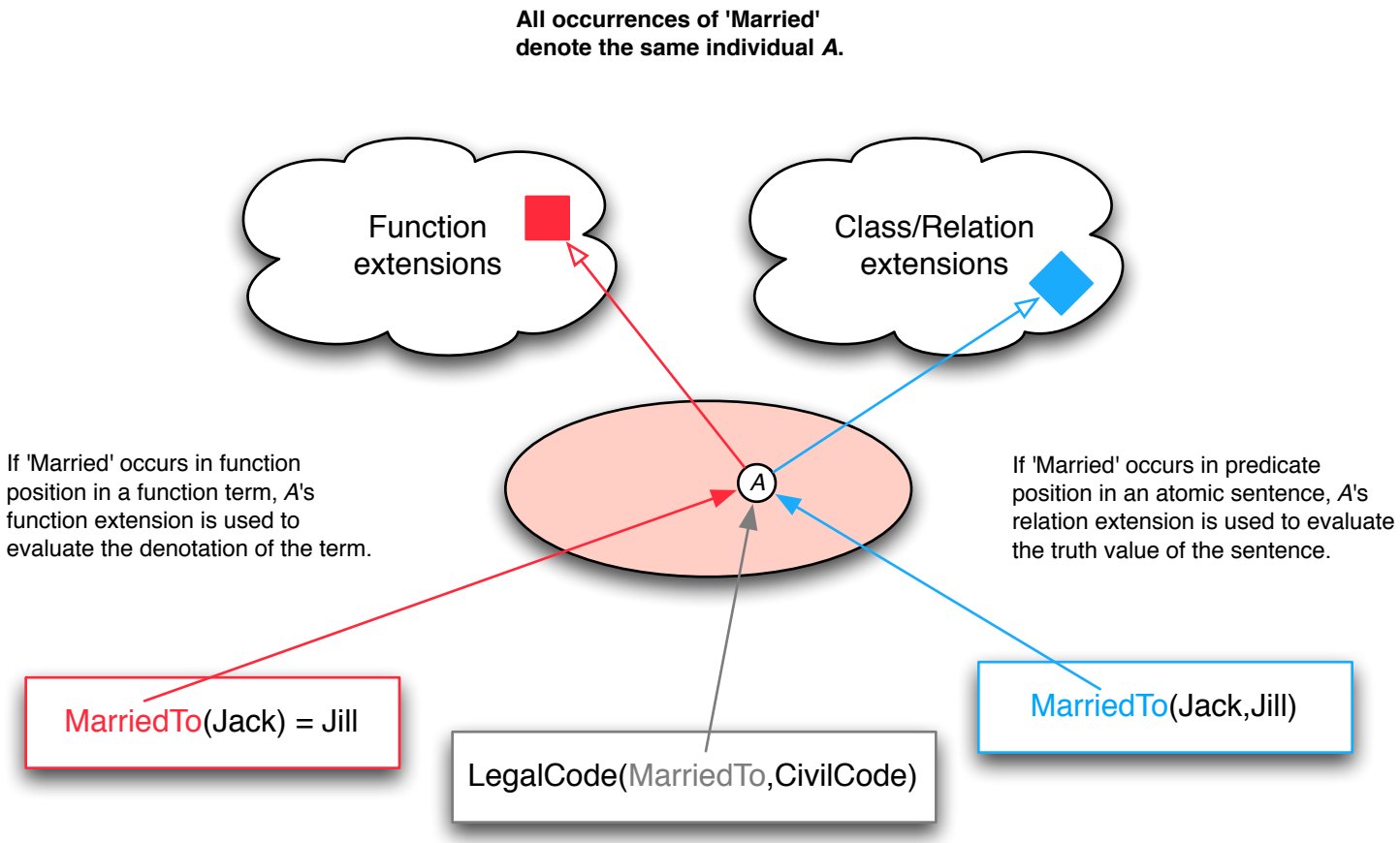
# CL Model Theory I

- In CL's model theory, everything is a logical individual
  - Only names in CL dialects; ***VARIABLE*** is a syntactic *role*
- Each individual has function and relation extensions
  - Classes treated as 1-place relations
    - No fixed "arity"
      - One relation extension can contain *n*-tuples, for different *n*
  - Cardinality and Well-foundedness problems avoided



# CL Model Theory II

- The syntactic context of name determines the appropriate semantic entity for evaluating the term or sentence in which is occurs.



# Translating into FOL

- First-order CL dialects can be thought of as notational variants of first-order theories.
  - Introduce predicate *Holds-n* and function symbol *App-n*, for each $n > 0$
  - Atomic sentences: $F(t1,\ldots,tn)^*$ = (*Holds-n F t1 … tn*)
  - Function terms $f(t1,\ldots,tn)^*$ = (*App-n f t1 … tn*)
    - For example

      $F(g(a),b,(g(f,f(a))))^* =$

      *Holds-3*(*F*,*App-1*(*g*,*a*),*b*,*App-2*(*g*,*f*,*App-1*(*f*,*a*)))

# CL and the Shortcomings of KIF

- Formalization
  - CL has a rigorous model theory for all of its constructs and a proof theory
  - Rigorous foundation for automated reasoning
    - OntologyWorks (http://ontologyworks.com)
    - H&S Information Systems (http://hsinfosystems.com)
- Specification
  - CL specifies structure rather than any particular form
  - This provides a general basis for determining/establishing conformance to CL syntax, hence a basis for meaning-preserving translation
- Soooo This Century
  - CL is an international ISO standard (ISO/IEC 24707)
  - CL framework includes XCL, an XML-based instance of CL
  - Includes numerous web-oriented features

# CLIF = the CL dialect closest to KIF

- CLIF syntax is 'LISP-like', based closely on KIF (but brought up to date re. character coding, IRIs, etc.; and simplified)
- CLIF incorporates simple extensions to facilitate frequently used content (argument roles, semantic sorts, quoted strings, common datatypes, numerical quantifiers)
- CLIF (in virtue of being a CL dialect) provides full web-based functionality for naming ontologies, importing ontologies, etc.

# Beyond FOL: Sequence Markers

- Sequence markers are a natural mechanism vis-à-vis signature-freedom
- *But:* They push CL beyond FOL in expressiveness
- Examples
  - Chaining
    - `(forall (F x) ((Chain F) x))`
      `(forall (F x y)`
      `(iff ((Chain F) ... x y)`
      `(and (F x y) ((Chain F) ... x)))))`
    - `(= AscendingOrder (Chain LessThan))`
    - `(AscendingOrder 2 5 17 25)`
  - Axioms for relations
    - `(iff (Unary F)`
    - `(and (not (F))`
    - `(not (exists (... x y) (F ... x y)))))`

# Sequence Markers II

- Chained Identity and Difference
  - `(AllEq x)`
  - `(iff (AllEq x y ...)`
    `(and (= x y) (AllEq y ...)))`
  - `(AllDiff x)  (Comment "a.k.a. 'NoRepeats'")`
  - `(iff (AllDiff x y ...)`
    `(and (not (= x y)) (AllDiff x ...) (AllDiff y ...)))`
- Finitude
  - `((seqOf F))  (Comment "'(seqOf F)' holds (only) of sequences of Fs")`
  - `(iff ((seqOf F) x ...) (and ((seqOf F) ...) (F x))`
  - `(iff (Finite F)`
    `(and (Unary F)`
    `(exists (...)`
    `(and ((seqOf F) ...)`
    `(AllDiff ...)`
    `(forall (x)`
    `(if (F x) (not (AllDiff x ...)))))))))`