Common Logic

A noble ambition, long in gestation, soon to be eased into ISO reality

What CL is and isn't

CL is a family of *first-order* logics which share a common *abstract syntax* and *model theory*, and an XML framework for encoding and transmitting them, or their content, on an open network.

CL syntax is very relaxed in the expressions it allows, in some ways going beyond classical FO logic.

CL is not a modal, free, hybrid, context, temporal, non-monotonic, logic programming, description, etc. logic.

On the other hand, CL syntax does try to be generous with non-FO syntax, so that non-FO content can be transmitted through CL-compliant engines. And some of these can be translated or embedded into CL expressions.

What CL is and isn't

CL is a family of *first-order* logics which share a common *abstract syntax* and *model theory*, and an XML framework for encoding and transmitting them, or their content, on an open network.

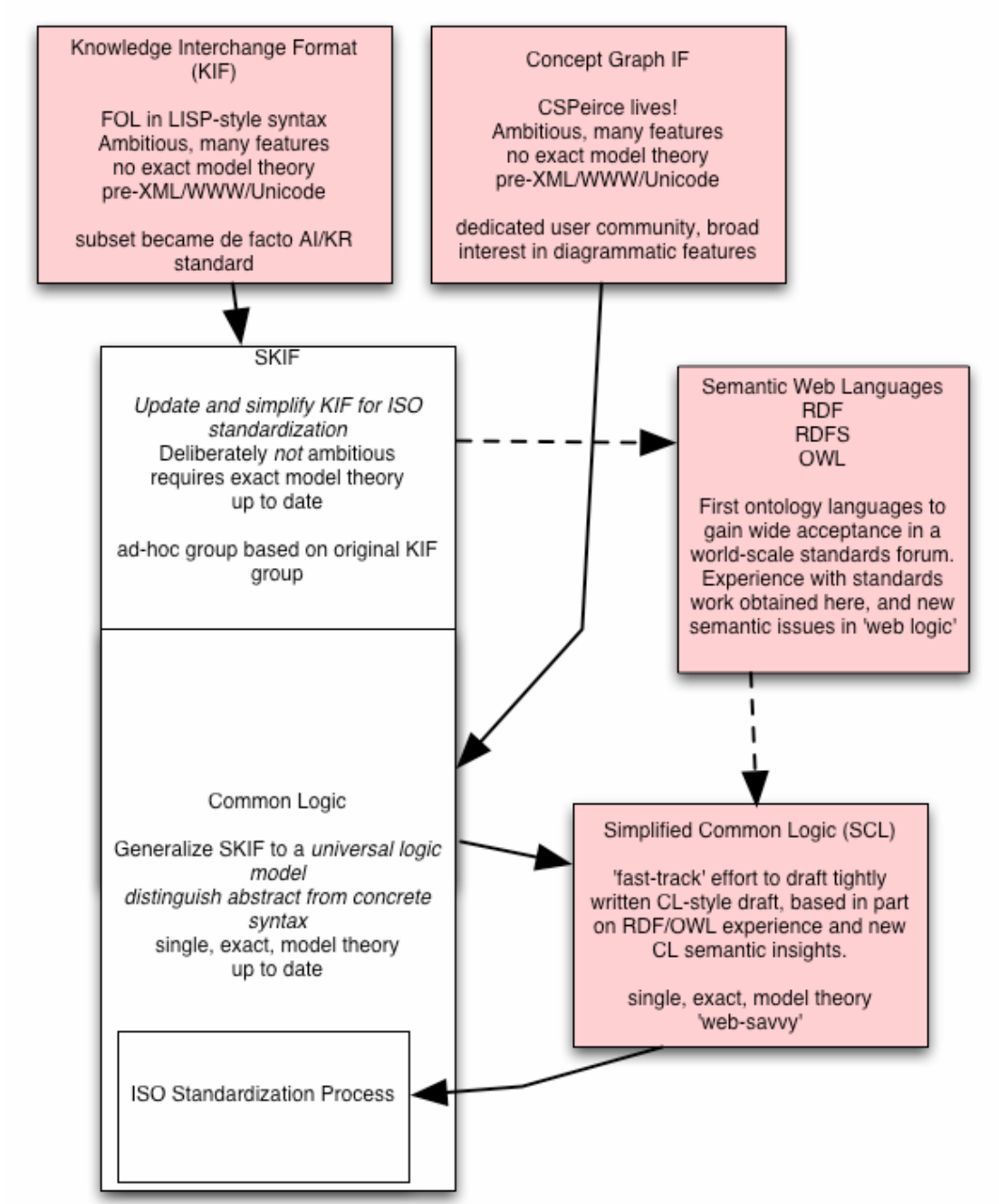
CL syntax is very relaxed in the expressions it allows, in some ways going beyond classical FO logic.

CL is not a modal, free, hybrid, context, temporal, non-monotonic, logic programming, description, etc. logic.

On the other hand, CL syntax does try to be generous with non-FO syntax, so that non-FO content can be transmitted through CL-compliant engines. And some of these can be translated or embedded into CL expressions.

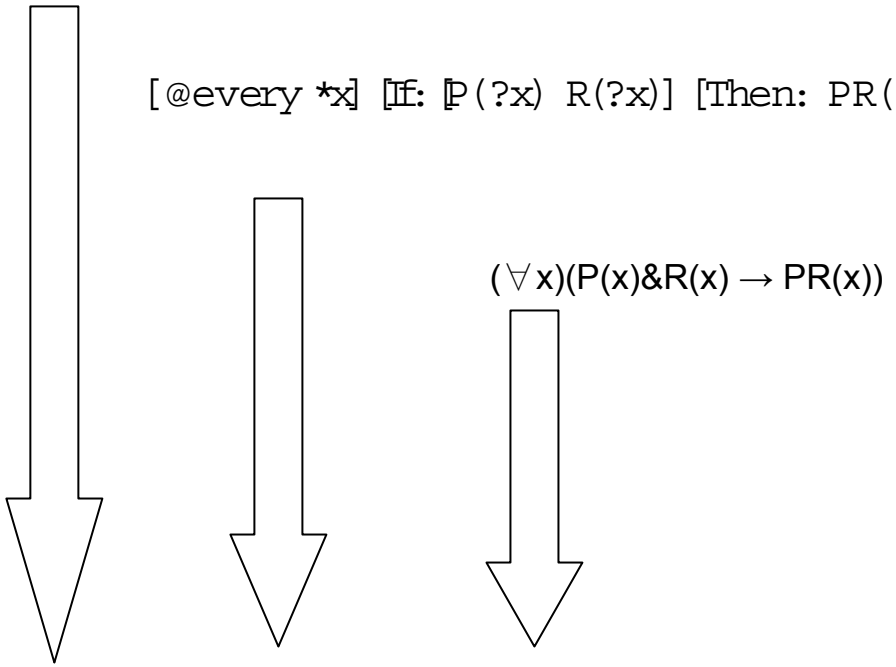
Design goals for CL

1. Common interlingua for variety of KR notations
2. Syntactically as unconstrained as possible
3. Semantically as simple and conventional as possible
4. Full first-order logic with equality, at least.
5. web-savvy, up-to-date
6. Historical origins in KIF. (Current version is CLIF.)


Knowledge Interchange Format (KIF)
FOL in LISP-style syntax
Ambitious, many features
no exact model theory
pre-XML/WWW/Unicode
subset became de facto AI/KR standard
Concept Graph IF
CSPeirce lives!
Ambitious, many features
no exact model theory
pre-XML/WWW/Unicode
dedicated user community, broad interest in diagrammatic features
SKIF
Update and simplify KIF for ISO standardization
Deliberately not ambitious
requires exact model theory
up to date
ad-hoc group based on original KIF group
Semantic Web Languages
RDF
RDFS
OWL
First ontology languages to gain wide acceptance in a world-scale standards forum. Experience with standards work obtained here, and new semantic issues in 'web logic'
Common Logic
Generalize SKIF to a universal logic model
distinguish abstract from concrete syntax
single, exact, model theory
up to date
Simplified Common Logic (SCL)
'fast-track' effort to draft tightly written CL-style draft, based in part on RDF/OWL experience and new CL semantic insights.
single, exact, model theory
'web-savvy'
ISO Standardization Process

CL dialects

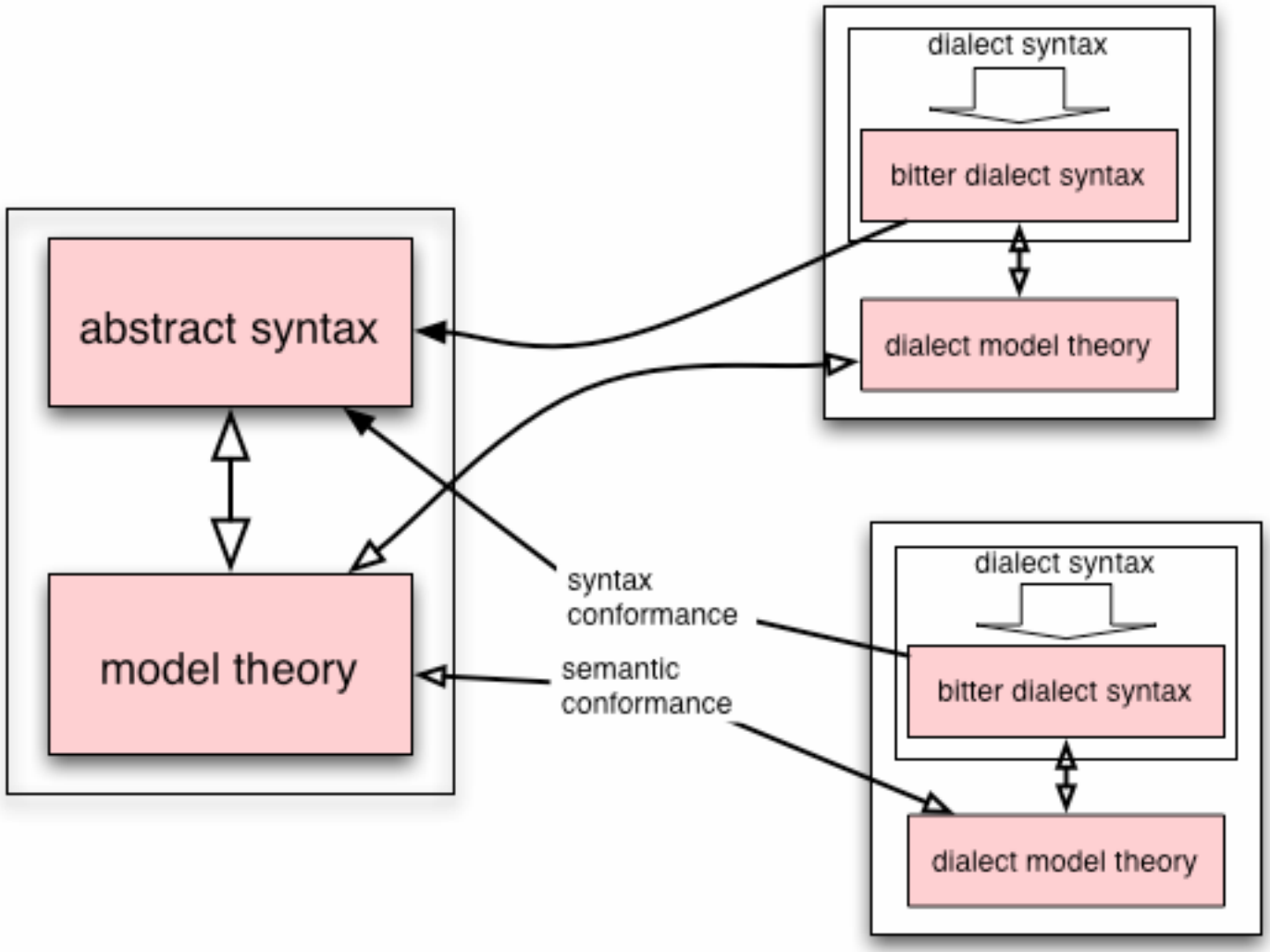
(forall (?x)(implies (and (P ?x) (R ?x)) (PR ?x))))



Abstract syntax with associated semantics.

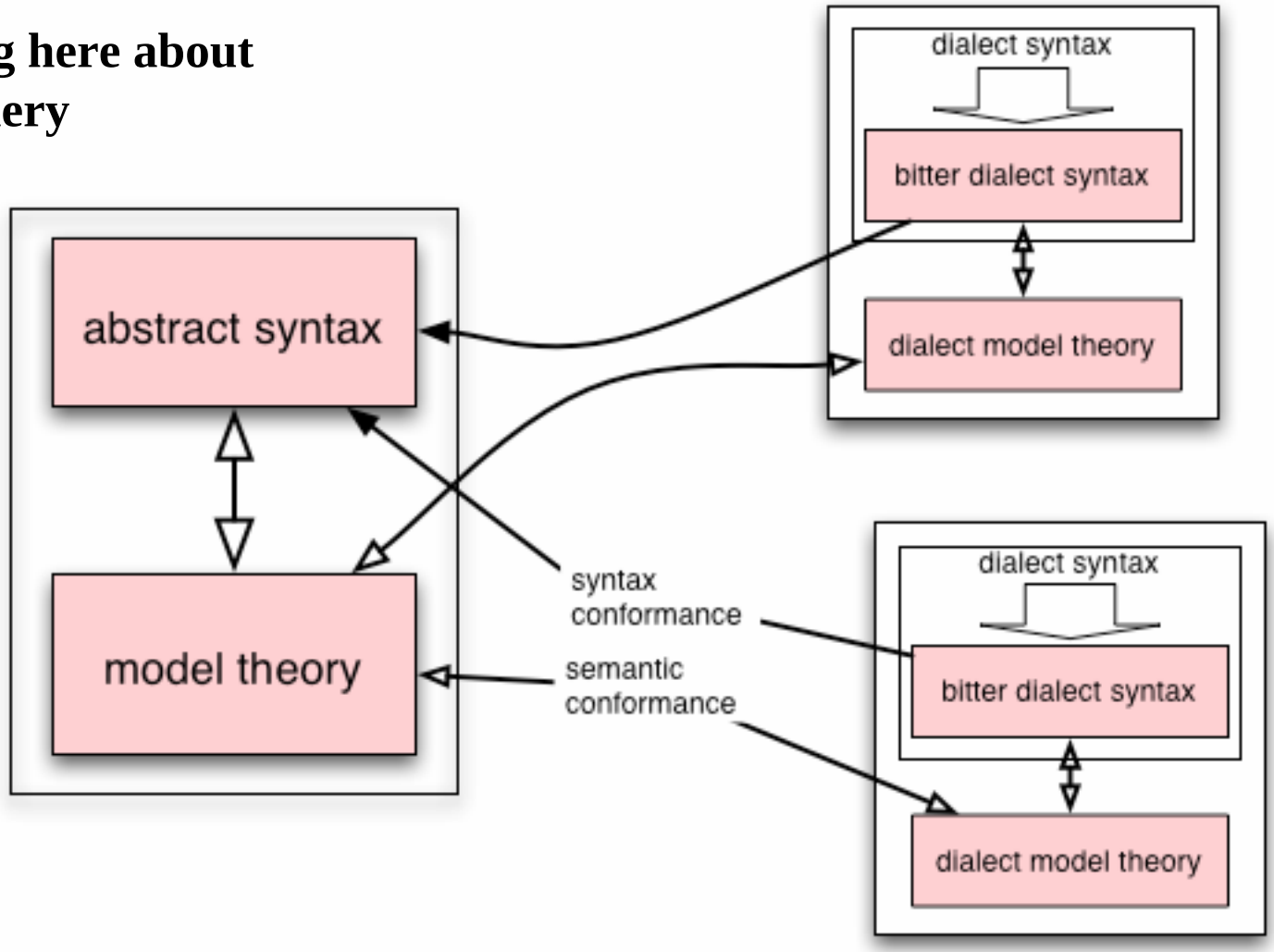
Different surface syntax forms all map to the abstract syntax, which provides a common semantic reference.

Abstract syntax, dialects, compliance



Abstract syntax, dialects, compliance

Nothing here about machinery



Abstract syntax and compliance

Every soldier carries their own rifle.

KIF:

```
(forall ((?x soldier)(?y rifle))(=> (owns ?x ?y)(carries ?x ?y) ))
```

Bitter KIF:

```
(forall (?x ?y)(=> (and (soldier ?x)(rifle ?y))
   (=> (owns ?x ?y)(carries ?x ?y) )) )
```

CL abstract syntax

A text is a set, list or bag of phrases. It may be identified by a name.
A phrase is either a comment, or a module, or a sentence, or an importation, or a phrase with an attached comment.
A comment is a piece of data.
A module consists of a name, an optional set of names called the exclusion set, and a text called the body text.
An importation contains a name.
A sentence is either a quantified sentence or a Boolean sentence or an atom, or a sentence with an attached comment, or an irregular sentence.
A quantified sentence has a type, called a quantifier, and a set of names called the bound names, and a sentence called the body of the quantified sentence. CL recognizes the existential and universal quantifier.
A Boolean sentence has a type, called a connective, and a number of sentences called the components of the Boolean sentence. The number depends on the particular type. CL recognizes the conjunction, disjunction, negation, implication and biconditional types with respectively any number, any number, one, two and two components.
An irregular sentence may have as immediate components any number of sentences, terms or names.
An atom is either an equation containing two arguments which are terms, or consists of a term, called the predicate, and a term sequence called the argument sequence, containing terms called arguments of the atom.
A term is either a name or a functional term, or a term with an attached comment.
A functional term consists of a term, called the operator, and a term sequence called the argument sequence, containing terms called arguments of the functional term.
An term sequence is a finite sequence of terms and an optional sequence variable.

Abstract syntax and compliance

Dialects **need not correspond exactly** to CL abstract syntax, as long as they can be embedded into it. Dialects can also extend the CL syntax and count as partially conformant.

There is a special category of "irregular sentence" in the abstract syntax, as a safety net to catch things like modalities or contextual assertions. The CL semantics treats irregular sentences as opaque sentential variables.

This allows CL to treat sentences with such extensions as logical sentences and to recognize some inferences.

Eg consider a modal extension to CLIF with [Nec] as a modality, then

(implies ([Nec] (foo baz)) ([Nec] (foo baz)))

Is a CL tautology even though the full meaning of [Nec] is invisible to the Cl model theory.

This also allows CL processors to 'pass along' notations which have extended sentential types without being obliged to report syntax errors.

CL Wild West Syntax

The most startling feature of CL to most FOL-savvy readers is its freewheeling lack of concern with the usual division between individual, relation and function names. CL makes no such distinctions (just like RDF): a name can be used anywhere. It can be used to name a thing, a relation (= OWL/RDF classes and properties) and a function (gensyms). It can also be used as a 'variable', i.e. can be bound by a quantifier.

```
(married Jack Jill)
(= (when (married Jack Jill)) (hour 3 (pm (thursday (week 12
          (year 1997))))) )
(exists (x) (x Jack Jill))
```

And yet, CL is a *first order* logic ?!

CL Wild West Syntax

(married Jack Jill)
(= (when (married Jack Jill)) (hour 3 (pm (thursday (week 12 (year 1997))))))
(ConjugalRelation married)
(exists (x) (and (x Jack Jill) (ConjugalRelation x))

And yet, CL is a *first order* logic ?!

Yes, because these quantifiers always range over a **single first-order universe**. There are no comprehension assumptions in CL (unlike in type theory or higher-order logic.) The only semantic presumption is that all names denote something, and that any name that is used as a relation or function name must denote something that has a relation or functional extension; and these are normal first-order semantic assumptions.

CL Wild West Syntax

(thisproperty Jill)
(thatproperty Jack)
??entails??
(exists (property)(and (property Jill)(property Jack)))

In higher-order logic, yes, because property could be
(lambda (x)(or (this property x)(thatproperty x)))

In CL, no. There are models in which two properties exist but their 'union' doesn't.
If you want it to follow, you can axiomatize the necessary construction:

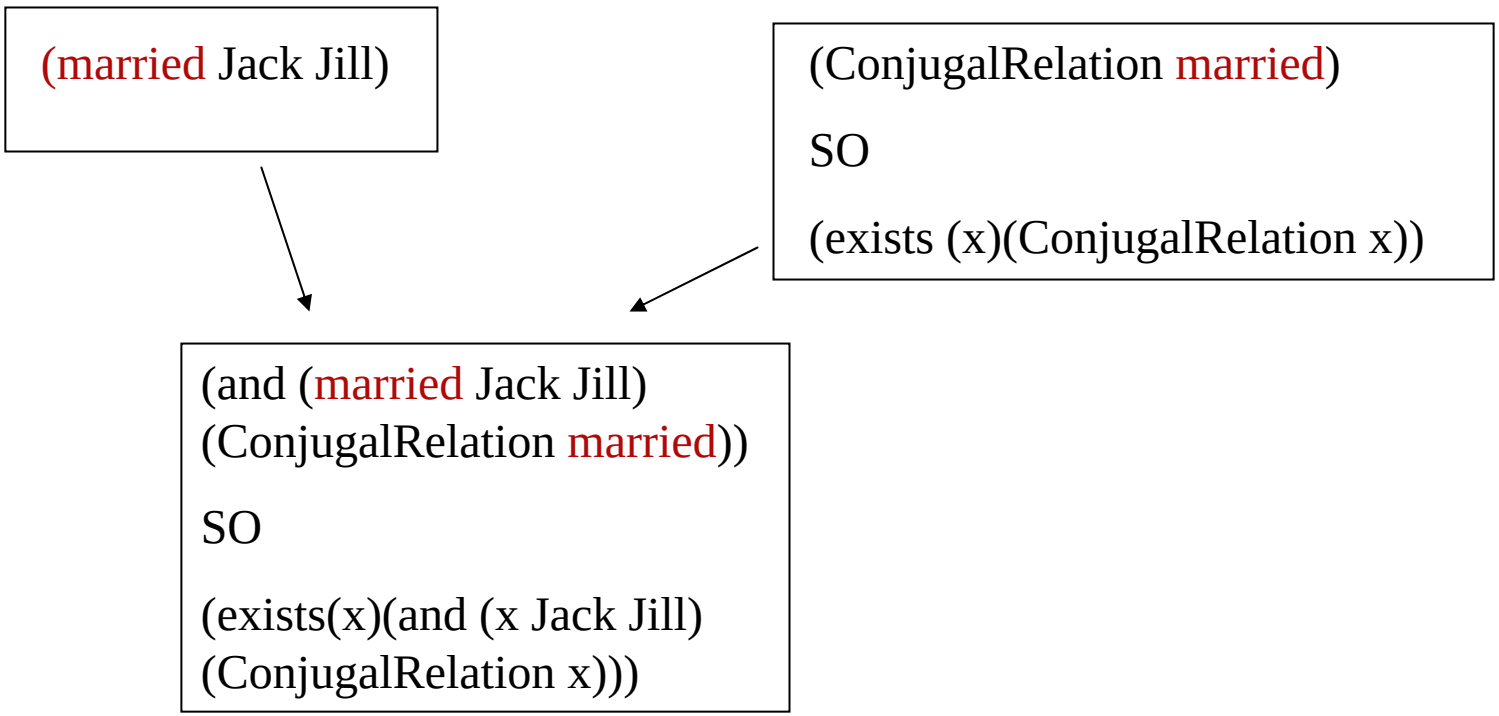
(forall (x y)(iff ((owl:Union x y) …) (or (x …)(y …))))

And now the conclusion above does follow. So you see, it's just up to you.

CL as a network logic

Agents on a network send logic expressions to one another. Suppose they send *just* the sentences to one another. Then the same logical inference principles should work at any node.

Communication and entailment should *commute*.



CL (no seq vars) embedded in GOFOL

Treat all the names as individuals, and insert a new relation name *holds* in front of every atom, and a new function name *app* in front of every term:

(married Jack Jill)	holds(married Jack Jill)
(ConjugalRelation married)	app(ConjugalRelation married)

And everything is magically transformed back into conventional FO syntax, and all the inferences are conventional FO entailments.

This is a quick and dirty way to implement a CL engine from a conventional FO tool. (Some care is needed with equality.) Note, the 'new' relations and functions are not in the universe of quantification, as required by GOFOL.

CL (no seq vars) embedded in GOFOL

Treat all the names as individuals, and insert a new relation name *holds* in front of every atom, and a new function name *app* in front of every term:

(married Jack Jill)	holds(married Jack Jill)
(ConjugalRelation married)	app(ConjugalRelation married)

And everything is magically transformed back into conventional FO syntax, and all the inferences are conventional FO entailments.

Which means that the **semantics** is accurately captured by this translation as well. One can view CL as *being* GOFOL, written in this odd way, with the *holds* and *app* simply erased from the surface syntax everywhere.

CLIF syntax

(:text rifle (:import http://www.ikris/ont36/military_personnel)
(forall ((x)(implies ((:comment 'is mp the right namespace?-PJH' mp:soldier) x))(exists (y)(and (mp:rifle y)(owns x y))))(:comment 'end of rifle text'))

(forall ((x rdfs:Class)) (and (:comment 'example of self-application' (x x)) (rdfs:subClassOf x x)))

(:text lists&collections (:comment 'Defines cons-nil and RDF collection vocabulary styles for describing lists.')(:comment '深港澳 考察□修班')(= nil (list))
(forall (x) (= (list x …)(cons x (list …))))
(forall (x y z)(iff (List x) (exists y z)(= x (list y z))))
(= List http://www.w3.org/1999/02/22-rdf-syntax-ns#List)
(forall (x (y List))(and (= (rdf:first (cons x y)) x)
(= (rdf:rest (cons x y)) y)))
(forall ((r "relation which takes argument lists"))(iff (r …)(r (list …))))
(forall (x)(= (concat (list) x) x))
(forall (x z) (= (concat (list x …) z)(cons x (concat (list …) z))))
)

CL$_{IF}$ sequence variables

(forall (x) (= (list x …)(cons x (list …))))

Means:

(forall (x)(= (list x)(cons x (list))))
(forall (x x1) (= (list x x1)(cons x (list x1))))
(forall (x x1 x2) (= (list x x1 x2)(cons x (list x1 x2))))
(forall (x x1 x2 x3) (= (list x x1 x2 x3)(cons x (list x1 x2 x3))))
….
A phrase with a seqvar in it stands for an *infinite* (RE) set of sentences. In practice, this can be implemented using a recursive call-out mechanism like Prolog.

CL with seqvars is therefore not compact, and therefore not first-order. However if we think of such phrases as axiom schema, then the logic is first-order. And this is usually all anyone wants.

CLIF extras

CLIF has a few extra features, including decimal numerals, quoted character strings, and the ability to use declared datatype names as predicates and functions. For example,

```
(= 345 (xsd:integer '345'))
(not (xsd:integer '3a'))
```

Are both logically true in CLIF. (Modeled directly on RDF/OWL treatment.)

CLIF includes plus and times on integers. Added to the lists machinery, this allows CLIF to express numerical quantifiers and other exotica.

These all correspond to *RE sets of ground sentences*. One lesson of the CL foundational work has been that **all ways of introducing computable constructions are fundamentally equivalent.**

CL and RDF, OWL, etc

The semantic web languages (RDF, RDFS, varieties of OWL) all map into CL or into CL ontologies. Most of it is straightforward: rdf triples become binary atoms, rdf:type is application.

s p o . goes to (p s o)

s rdf:type c . goes to (c s)

```
(= owl:intersectionOf AND)
(forall (P Q x)(iff ( (AND P Q) x)(and (P x)(Q x)) ))
(forall (P Q x)(iff ( (OR P Q) x)(or (P x)(Q x)) ))
(forall (P Q x)(iff ( (NOT P) x)(not (P x)) ))
(forall (R P x) (iff ( (ALLARE R P) x)(forall (y)(implies (R x y)(P y)) ))
(forall (R P x) (iff ( (SOMEARE R P) x)(exists (y)(and (R x y)(P y)) ))
(= (bigRedRubberBall)(AND Big MadeOfRubber Red Ball))
```

CL and RDF, OWL, etc

Cardinality restrictions are the most complicated.

```
(forall (r p n)(implies
   (and (owl:onProperty r p) (owl:minCardinality r n))
   (forall (x)(iff
      (r x)
      (exists ((L NOREPEATSLIST))(and
         (forall (y)(implies (member y L)(p x y) ))
         (lesseq n (length L))
))))))
(= (LENGTH nil) 0)
(forall (x)(= (LENGTH (list x ...)) (plus 1 (LENGTH (list ...))))
(iff (ALLDIFFERENT ...)(NOREPEATSLIST (list ...)))
(forall (x)(not (MEMBER x nil)))
(forall (x y)(iff (MEMBER x (list y ...))(or (= x y)(MEMBER x (list ...)))))
```

New stuff

ISO possibility: integrate with ISO Prolog

IKRIS extensions:

1. explicit context assertions, (p holds in context c (of type C))
2. things, relations, functions *and propositions*.
3. providing for provenance and source (meta?)data
4. providing for information flow control/secrecy.
5. importance of standardizing generally useful content as well as notation.