Describing Ontology Relationships

A brief guide to the expressive powers of ISO Common Logic

Pat Hayes, Florida IHMC
phayes@ihmc.us

ISO Common Logic

The recently published ISO Common Logic standard provides an unusually 'relaxed' syntax for FOL, one which **removes many of the commonly assumed constraints** on FOL syntax. This *wild west syntax* provides many new opportunities for describing **mappings between axioms in different ontologies**.

FOL syntax

In a conventional FOL language (here referred to as GOFOL), there are three syntactic *kinds* of symbols: individual, function and relation names, and usually the last two are further divided by *arity* (number of arguments) and sometimes they are further subdivided into types or *sorts*. All this classification of the vocabulary is called a **signature**, and GOFOL languages are required to have a signature.

CL does away with all this. The only 'signature' required is a single set of names.

Wild West syntax

In CL, there are two basic assumptions: *any name can be used for any purpose*; and *anything that has a name is in the universe of discourse*. So we can use a name to refer to, say, a relation with one argument and a function with seven arguments and an individual. Anything at all can be treated as an individual (of course) but also as a function and as a relation, and this always makes logical sense.

We call this 'wild west syntax' because, just like in the movies, there are no laws on the frontier. Anything goes.

Silly Example

These are syntactically legal CL (in the CLIF dialect of CL, which we will use from now on):

```
(P (P P))
((P P)(P P))
(exists (x)((x x)(x x)))
```

And are even satisfiable. The first says that `P` is a function which, when applied to itself, has a value of which it, itself, is true. For example (using the lambda-calculus), *[lambda x. true]* would satisfy this CL sentence.

CL really is FOL

```
(exists (x)((x x)(x x)) )
```

doesn't look *at all* like conventional GOFOL. If anything, it looks like a warped kind of higher-order logic, since it seems to allow quantification over functions and relations. And indeed, it does; and yet, it *really is first-order*.

If you like, think of it as an alternative way of writing:

```
(exists (x)
   (holds (apply x x)(apply x x)) )
```

which is of course perfectly correct GOFOL.

A pattern-maker's logic

The practical effect of all this is that one can write arbitrary 'patterns' in CL and quantify over arbitrary 'parts' of them with complete freedom, without worrying too hard about what exactly the names denote.

Moreover, the sequence markers allow one to recursively define LISP-like operations on the syntax itself.

This is extremely useful.

The same relation can have different numbers of arguments:

```
(Married Jill)      (Married Jill Jack)
(Married Jill Jack 2003) ...
```

so CL allows quantification over arbitrary *finite sequences*, to cover all the cases:

```
(forall (r s ...)(iff
    ((conj r s) ...)
    (and (r ...)(s ...))
))
```

We know now that these are equivalent:

```
((conj Married InLove) Jill Jack)
(and (Married Jill Jack)(InLove Jill Jack))
```

and, since we can use names for anything, we can give this a name:

```
(= (conj Married InLove) TrueRomance)
```

First Real Example

Transforming continuant language to '4-d' language.

```
(forall ((c Continuant)(t Time) R )(if
     (R c t)
     ((transf R) (slice c t))
))
```

and then *in the 4d ontology* one can safely add

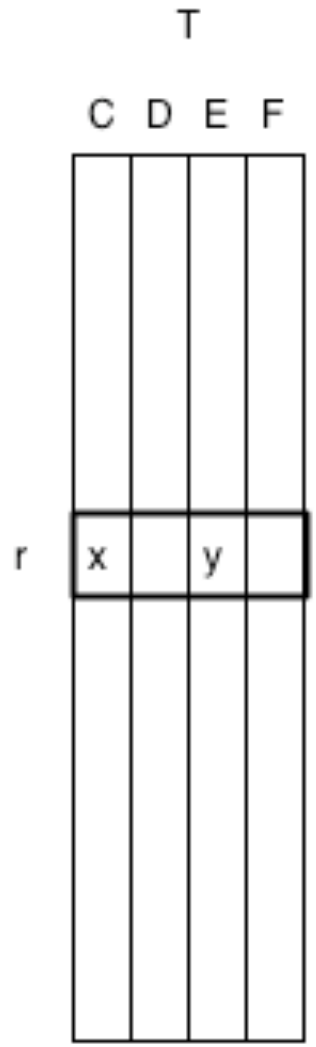
```
(forall (x)(= x (transf x))
```

which means that one can there use the same name for the continuant and its sister occurrent (because there, they are the same thing).

This trick of applying a 'transfer function' to one vocabulary is widely applicable.

Another real example: database mappings

A database table can be treated as a predicate (class) which is true of all the rows in the table, and each column as a function (selection function, property) from the row to a value.



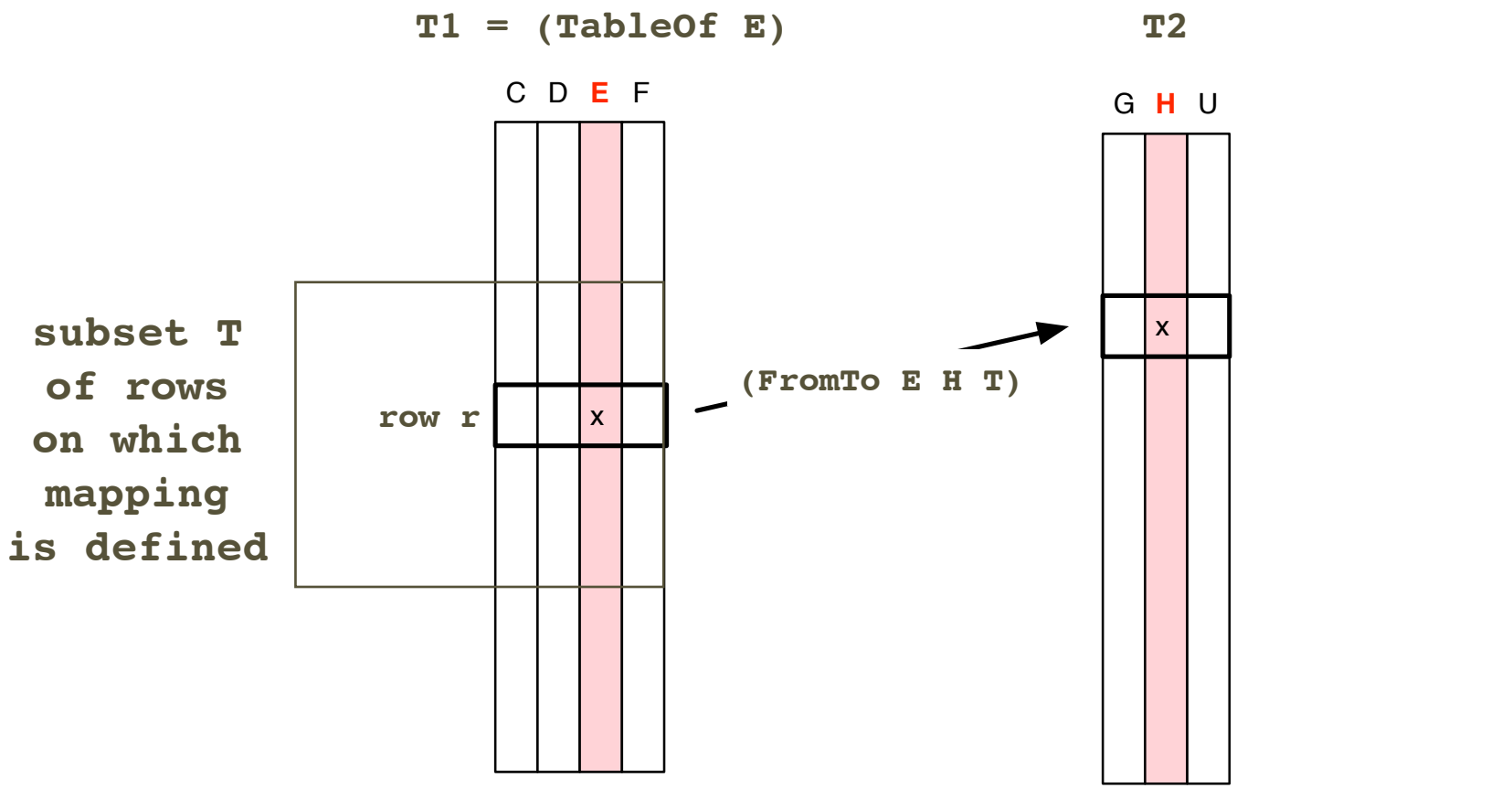
(T r)

(= (C r) x)

(= (E r) y)

Another real example: database mappings

Mappings between tables can then be described directly in CL syntax, even though they may seem to involve 'higher-order' constructions.




```
(forall ((r T))(= (E r)(H ((FromTo E H T) r)) ))
```

Another real example: database mappings

Cases like mappings on subsets of rows, compositional mappings, etc. are straightforward:

```
(forall (S T)(iff (SubTable S T)(forall (x)(if (S x)(T x))) ))
(forall ((r S) E H)(= (E r)(H ((FromTo E H) r)) ))
(forall ((r S) a b)(= ((FromTo a b) r)((FromTo a b S) r) ))
(forall (x)(= ((comp f g) x)(f (g x)) ))
```

These axioms are somewhat naive in that they assume that the relevant mappings always exist and are unique. A more careful approach could treat `(FromTo a b S)` as a *predicate* on mapping functions:

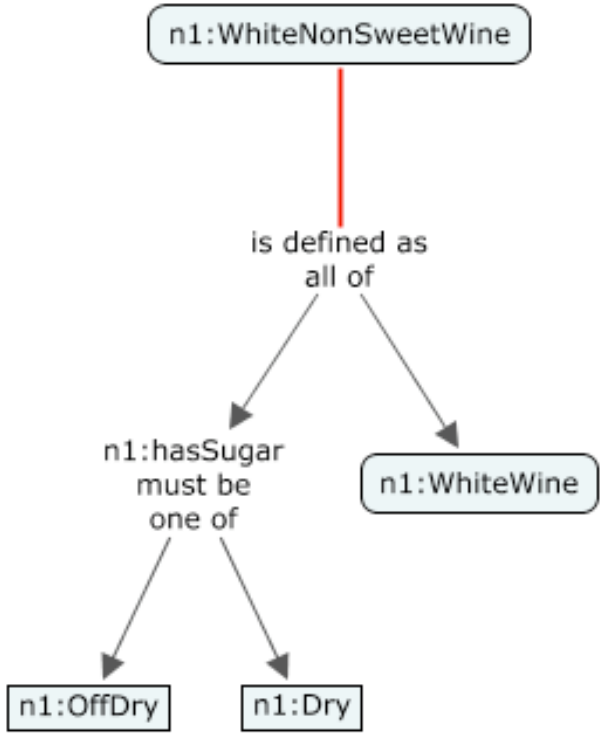
```
(forall (S E H m)(iff
       ((FromTo E H S) m)
       (forall ((r S))(= (E r)(H (m r))))    ))

(forall (a b)(= (FromTo a b)(FromTo a b (TableOf a)) ))

(forall (f g A B S)(if
         (and ((FromTo A B S) f)((FromTo B C) g))
         ((FromTo A C S) (comp g f))                ))
```

OWL-DL in CL

```
(forall (x y ...)(iff (owl:allDifferent x y ...)(and (not (= x y)(owl:allDifferent y ...)
(owl:allDifferent x ...)) ))
(forall (x)(owl:allDifferent x))
(forall (f)(iff (owl:FunctionalProperty f)(forall (x y z)(if (and (f x y)(f x z))(= y z) )) ))
(forall (f)(iff (owl:InverseFunctionalProperty f)(forall (x y z)(if (and (f y x)(f z x))(= y z) )) ))
(forall (x)(and (not (owl:Nothing x))(owl:Thing x) ))
(forall (f)(iff (owl:SymmetricProperty f)(forall (x y)(iff (f x y)(f y x))) ))
(forall (f)(iff (owl:TransitiveProperty f)(forall (x y z)(if (and (f x y)(f y z))(f x z) )) ))
(forall (p c x)(iff ((owl:AllValuesFrom p c) x)(forall (y)(if (p x y)(c y)) ))
(forall (c x)(iff ((owl:complementOf c) x)(not (c x)) ))
(forall (c d)(iff (owl:disjointWith c d)(forall (x)(iff (c x)(not (d x)))) ))
(forall (c d)(iff (owl:equivalentClass c d)(forall (x)(iff (c x)(d x))) ))
(forall (c d)(iff (owl:equivalentProperty c d)(forall (x y)(iff (c x y)(d x y))) ))
(forall (p v x)(iff ((owl:hasvalue p v) x)(p x v) ))
(forall (c ...)(iff ((owl:intersectionOf c ...) x)(and (c x)((owl:intersectionOf ...) x)) ))
(forall (x)((owl:intersectionOf) x)
(forall (p x y)(iff ((owl:inverseOf p) x y)(p y x) ))
(forall (... x y)(iff ((owl:oneOf x ...) y)(or (= x y)((owl:oneOf ...) y)) ))
(forall (x)(not ((Owl:oneOf) x) ))
(forall (p c x)(iff ((owl:somevaluesFrom p c) x)(exists (y)(and (p x y)(c y))) ))
(forall (p (n Number) x)(iff
     ((owl:maxCardinality p n) x)
     (not (exists (y ...)(and (owl:allDifferent y ...)(number n ...)((TrueOfAll p x y) ...)) ))
))
(forall (p (n Number) x)(iff
     ((owl:minCardinality p n) x)
     (exists (...)(and (owl:allDifferent ...)(number n ...)((TrueOfAll p x) ...)) )
))
(forall (n x ...)(iff (number (succ n) x ...)(number  n ...) ))
(number 0)
(forall (p x ...1 ...2)(iff ((TrueOfAll p ...1) x ...2)(and (p ...1 x)((TrueOfAll p ...1) ...2)) ))
(forall (p ...)((TrueOfAll p ...)))
```

OWL-DL in CL

```
<owl:Class rdf:ID="WhiteNonSweetWine">
   <owl:intersectionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#WhiteWine" />
     <owl:Restriction>
       <owl:onProperty rdf:resource="#hasSugar" />
       <owl:allValuesFrom>
         <owl:Class>
           <owl:oneOf rdf:parseType="Collection">
             <owl:Thing rdf:about="#Dry" />
             <owl:Thing rdf:about="#OffDry" />
           </owl:oneOf>
         </owl:Class>
       </owl:allValuesFrom>
     </owl:Restriction>
   </owl:intersectionOf>
 </owl:Class>
```




```
(= WhiteNonSweetWine
  (owl:intersectionOf #WhiteWine (owl:allValuesFrom #hasSugar (owl:oneOf #Dry #OffDry))))
)
```

```
(= WhiteNonSweetWine (AND #WhiteWine (MustBe #hasSugar (OneOf #Dry #OffDry))) )
```