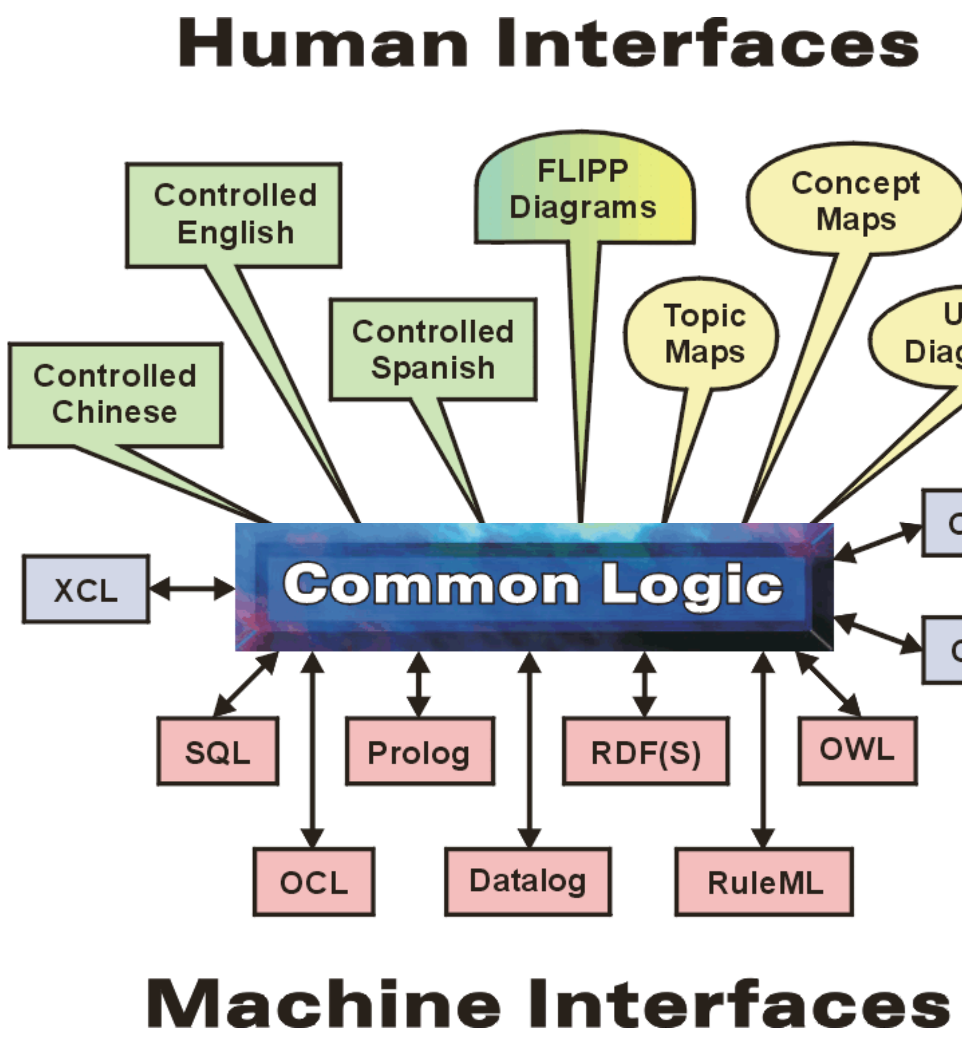
Using Common Logic

= Mapping other KR techniques to CL.
= Eliminating dongles.
= The zero case

Pat Hayes
Florida IHMC

John Sowa talked about this side.



Now we will look more closely at this side, using the CLIF dialect.

Wild West Syntax (CLIF)

Any character string can be a name, and any name can play any syntactic role, and any name can be quantified. Anything that can be named can also be the value of a function.

All this gives a lot of freedom to say exactly what we want, and to write axioms which convert between different notational conventions.

It also allows many other 'conventional' notations to be transcribed into CLIF directly and naturally.

It also makes CLIF into a genuine *network logic*, because CLIF texts retain their full meaning when transported across a network, archived, or freely combined with other CLIF texts. (Unlike OWL-DL and GOFOL)

Mapping other notations to CLIF.
1. Description Logics

Description logics provide ways to define *classes* in terms of other classes, *individuals* and *properties*.

All people who have at least two sons enlisted in the US Navy.

```
<this class> owl:intersectionOf [:Person _:x]
_:x rdf:type owl:Restriction
_:x owl:minCardinality "2"^^xsd:number
_:x owl:onProperty :hasSon
_:x owl:toClass _:y
_:y rdf:type owl:Restriction
_:y owl:hasValue :USNavy
_:y owl:onProperty :enlistedIn
```

Mapping other notations to CLIF.
1. Description Logics

Classes are CL unary relations, properties are CL binary relations.
So DL operators are *functions* from relations to other relations.

```
(owl:IntersectionOf Person (owl:minCardinality 2 hasSon (owl:valueIs enlistedIn USNavy)))
 (AND Person (MIN 2 hasSon (VAL enlistedIn USNavy)))
```

```
((AND Person (MIN 2 hasSon (VAL enlistedIn USNavy))) Harry)
(= 2ServingSons (AND Person (MIN 2 hasSon (VAL enlistedIn USNavy))))
(2ServingSons Harry)
(NavyPersonClassification 2ServingSons)
(BackgroundInfo 2ServingSons
        'Classification introduced in 2003 for public relation purposes.')
```

Mapping other notations to CLIF.
1. Description Logics

The *meanings* of these constructions can be axiomatized in CLIF:

```
(forall (x y ...)(iff
      ((AND y ...) x)
      (and (y x)((AND ...) y))
))
(forall (x)((AND) x))

(forall (x y p)(iff ((VAL p x) y) (p y x) ))

(forall (c y p)(iff ((SOMEARE p c) y) (exists (z)(and (p y z)(c z))) ))

(forall ((n integer) x p c)(iff ((MIN n p c) x)(exists n (z)(and (p x z)(c z))) ))
```

This is important to ensure semantic coherence, as a reference standard. Practical reasoners would probably not use these axioms directly.

For (lots) more detail, see
http://www.ihmc.us/users/phayes/cl/sw2scl.html
http://philebus.tamu.edu/cmenzel/Papers/AxiomaticSemantics.pdf

Mapping other notations to CLIF. 1. Description Logics

The *meanings* of these constructions can be axiomatized in CLIF:

```
(forall (x y ...)(iff
      ((AND y ...) x)
      (and (y x)((AND ...) y))
))
(forall (x)((AND) x))
```

Typical 'recursion' using sequence markers.

```
(forall (x y p)(iff ((VAL p x) y) (p y x) ))

(forall (c y p)(iff ((SOMEARE p c) y) (exists (z)(and (p y z)(c z))) ))

(forall ((n integer) x p c)(iff ((MIN n p c) x)(exists n (z)(and (p x z)(c z))) ))
```

This is important to ensure semantic coherence, as a reference standard. Practical reasoners would probably not use these axioms directly.

For (lots) more detail, see
http://www.ihmc.us/users/phayes/cl/sw2scl.html
http://philebus.tamu.edu/cmenzel/Papers/AxiomaticSemantics.pdf

Mapping other notations to CLIF.
1. Description Logics

Relatively minor changes to these axioms give CL theories corresponding to the various different description logics, and variants such as OWL-Full, OWL2 (forthcoming) and terHorst's intensional OWL (just use *if* instead of *iff*), as well as RDF and RDFS.

One *common syntax* and one *fixed logic* covers all these cases, which can be viewed as a selection of a vocabulary and a "logical ontology". By reserving a single namespace for each, reasoners can easily recognize sub-cases where efficient decidable DL reasoning can be used.

For interoperability, the key is that these translations all use a single syntax and semantics. The process of creating a new standard is vastly simplified. All the 'semantic' decisions can be recorded as CL axioms, and checked mechanically.

Mapping other notations to CLIF. 2. Modal notations

Several recent standards, notably *Semantics of Business Vocabulary and Rules* (SBVR) are based on modal logics; several ontology frameworks (*OntoClean*) use modal language; and some temporal frameworks use tense-like operators.

There are two approaches to mapping modal languages into CL.

Modal truth (necessity, temporal logic) is handled by introducing 'possible worlds' AKA "contexts".

Permissions and obligations (as in SBVR) are handled more intuitively by an ontology of actions and events.

Mapping other notations to CLIF. 2. Modal truth

Consider a simple temporal modal/hybrid language, in which sentences are asserted using tenses, as in "it *will rain* in Oaxacala", and suppose this is 'said' on Sunday 4 May 2008

(*Future* (Precipitation Oaxacala))

CL doesn't have a modal operator (such as *Future*) so we will talk about times explicitly and then *Future* translates to a quantifier: "*At some time t later than now, ...*". Need to know two things: the current time ("now"); and which of our relations and/or names are time-dependent (*Precipitation* is, *Oaxacala* isn't.)

Add a temporal parameter to those that are, add the relationship to *now*, then quantify:

```
(exists ((t time))(and
        (later t  05042008)
        (Precipitation Oaxacala t)
))
```

OR: ((Precipitation t) Oaxacala) OR (Precipitation (Oaxacala t))

Mapping other notations to CLIF.
2. Modal truth

Some names (role identifiers) are themselves time-dependent:

```
 (= (PresidentUSA  01051999)  WilliamClinton)
```

There are many points of view, debates, etc. about how best to conceptualize things extended in time. All of these boil down in practice to decisions about *where to put the time parameter*. Some frameworks prohibit role identifiers in certain cases, etc.. As long as the time parameter is placed *somewhere*, it is straightforward to translate between conventions which adopt these various restrictions and philosophical rules.

```
(forall ((t time)(c OBOContinuant) r )(iff
      (r c t)
      (r ((OBO2EPISTLE c) t))
))
```

The translation is represented here by a 'conversion function' between the vocabularies. In practice this might be a lookup table relating the names.

Mapping other notations to CLIF. 2. Modal truth

Notice, these times are 'pointlike' (perhaps not mathematical points); truth relative to an interval needs to be handled with more care (true *throughout*, or true only *somewhere* in the interval?)   The results can be complicated, but the translation is completely mechanical.

All last year there was someone guarding the house.

```
(forall ((t time)(if
    (in t [last year])
    (exists ((i timeInterval)(and
        (in t i)
        (exists ((x Person))(and
              (forall ((u time)(if
                    (in u i)
                    (guards x [the house] u)
         ))
     ))
))
```

Mapping other notations to CLIF.
3. SBVR and actions

SBVR rules conditionally permit or prohibit actions and states of affairs:
*If a car is rented to a driver then that driver **must** have a valid driver's license.*

CL has no notion of *must*, but it can describe states of affairs and actions, and then describe the conditions under which they are classified into 'permitted' or 'forbidden' categories.

*If a car is rented to a driver who does not have a valid license, then the rental is a **prohibited transaction**.*

```
(forall ((x CarRental))(if (not (ValidLicencedToDrive (driver x)))
(ProhibitedTransaction x) ))
```

This translation is semi-mechanical, but the CL version has several advantages, most notably allowing for different kinds of prohibition.

This can be used together with temporal-style language to handle time-dependent business rules. *All the necessary reasoning can be handled within CL.*

Wild West Syntax (CLIF)

= Eliminating dongles.

Any character string can be a name, and any name can play any syntactic role, and any name can be quantified. Anything that can be named can also be the value of a function.

All this gives a lot of freedom to say exactly what we want, and to write axioms which convert between different notational conventions.

Dongle vocabulary: terms that carry no meaning, but are there just to make the syntax "look right".

Pat *is* Human
Human *is* a biological category

(Human Pat)
(BiologicalCategory Human)

Pat *rdf:type* Human .
(*Instance* Human Pat)
(*hasType* Pat Human)
(*AppliesTo* Human Pat)
isa Pat Human
Pat *a* Human

An Objection

But surely there has to be some limit. I mean, dash it all, surely something like this:

('Pat' c)

would be ***silly****. Character sequences just aren't the kind of thing that can possibly be a predicate.*

Moral: If people are using it successfully, it probably makes some kind of sense.

An Objection

But surely there has to be some limit. I mean, dash it all, surely something like this:

('Pat' c)

*would be **silly**. Character sequences just aren't the kind of thing that can possibly be a predicate.*

Yet in fact, there are at least two very convincing uses for this kind of expression.

== RDF typed literals have exactly this form and this semantics.

Moral: If people are using it successfully, it probably makes some kind of sense.

== In the IKRIS project, we needed 'contextual names' which indicate what a name denotes in a context. We could have written

(MeaningInContext 'Pat' c)

but this is a classical dongle. It is simpler to just write

('Pat' c)

and think of it (if you like) as a name with a subscript: Pat_c

The zero case

The zero case

An 'atomic' sentence (atom) is a relation followed by a *sequence* of arguments:

```
(Human Pat)
(timeOrder
   (birth Pat) (marriage Pat Jackie) (graduation Pat) (grantedTenure Pat)(brokeCollarBone Pat)
)
```

and a term is a function followed by a *sequence* of arguments:

```
(Human (fatherOf Pat))
(= (+ 23 457 1219 27) 1926)
```

CLIF argument sequences can be any length, including zero:

```
(timeOrder)
(fatherOf)
```

so the question arises, what use are such things? Answer: ***whatever use you want them to have***.

Some examples...

The zero case

The zero case

Base case for recursive definitions.
Already seen this with AND in description logics, but some more examples:

```
(forall (x y ...)(iff
       ((OR y ...) x)
       (or (y x)((OR ...) y))
))
(forall (x)(not ((OR) x) ))

Eg ((OR mammal duck) Platypus3)
     (or (mammal Platypus3)((OR duck) Platypus3))
     (or (mammal Platypus3) (or (duck Platypus3) ((OR) Platypus3) ))
     (or (mammal Platypus3) (or (duck Platypus3)       False      ))
     (or (mammal Platypus3) (duck Platypus3))
-----------------

(forall (r)(if
   (Predicative r)
   (and
        (forall (x ...)(if (r x ...)(and (r x)(r ...)) ))
        (r)
    )
))

Eg (Predicative Human)
    (Human Pat John Chris Arthur Adam)
    (forall (x)(if (TypeClassifier x)(Predicative x) ))
    (Predicative Predicative)
    (Predicative Human Animal Mineral Vegetable)
```

The zero case

$\boxed{\text{If y is a character string, (y) is whatever y is the name of.}}$

The zero case

Indirect Names

Already mentioned 'contextual names' (actually, terms with a character string used as a function) such as
('Pat' c)   ('Pat' (beliefContext Joe))

Interesting idea is to use the zero case to mean the same as the name itself:

(= ('Pat') Pat)   (= Chris ('Chris')) etc..

This cannot be axiomatized, but it can be treated as a *semantic extension* of CLIF (IKL). Why bother? Because now we can relate things to their names, and make assertions about the relationship. For example:

```
(forall (R (L nameList))(iff
    (closedWorldFor R L)
    (forall ((y charseq)) (if (R (y))(member y L) ))
))
```

If y is a character string, (y) is whatever y is the name of.

axiomatizes the idea of a 'closed world' supporting *negation as failure*, as in Prolog and related systems. Note, this does **not** require a non-monotonic logic !

The zero case: propositions

IKL (a semantic extension of CLIF) has a built-in notation for this, written using 'that':

(= R (*that* (exists ((x FrenchCitizen)(y IsraeliCitizen))(Married x y))))

which is very convenient and natural to use, but is not essential.

The zero case: propositions

Propositions

A relation with no arguments gives an atomic sentence with just one name in it:

(R)

Being a sentence, this has a truth-value, i.e. it is true or false.

In this case, R represents a *proposition*, i.e. an *object with a truth-value*. To fix the truth-conditions for R, just connect it to a longer sentence which expresses them directly:

(iff (R) (exists ((x FrenchCitizen)(y IsraeliCitizen))(Married x y)))

IKL (a semantic extension of CLIF) has a built-in notation for this, written using 'that':

(= R (*that* (exists ((x FrenchCitizen)(y IsraeliCitizen))(Married x y))))

which is very convenient and natural to use, but is not essential.

Propositions allow even more translations into CLIF, including all of 'context logic'. They also allow CLIF ontologies to be *self-describing* in a strong sense, by making assertions about their own propositions.

The zero case: a moral

All it needed was to be set free.

The zero case: a moral

When CL was being designed, we had not thought of any of these uses. We included the 'zero case' simply because there was no reason not to, and we wanted the Wild West Syntax to be as free as possible from unnecessary restrictions, on general grounds of economy and elegance.

In ***every case***, the 'peculiar' constructions thus allowed have turned out to be practically useful. They provide for more compact, efficient ways to write axioms, more powerful reasoning, far more flexibility of expression for interoperability, and entire new areas of important expressivity. They surprised us.

We think that the logic is telling us something. First-order logic has long been seen as the foundational formalism for knowledge representation. But it is even better than it has traditionally seemed to be.

All it needed was to be set free.