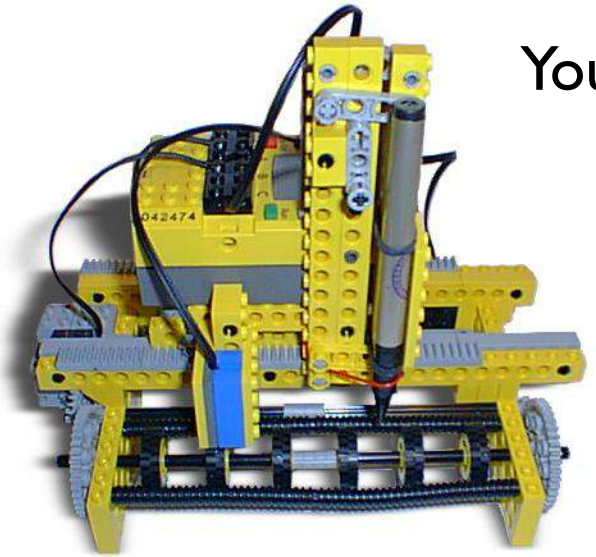
IKL

Common Logic on steroids

IKL is a logic

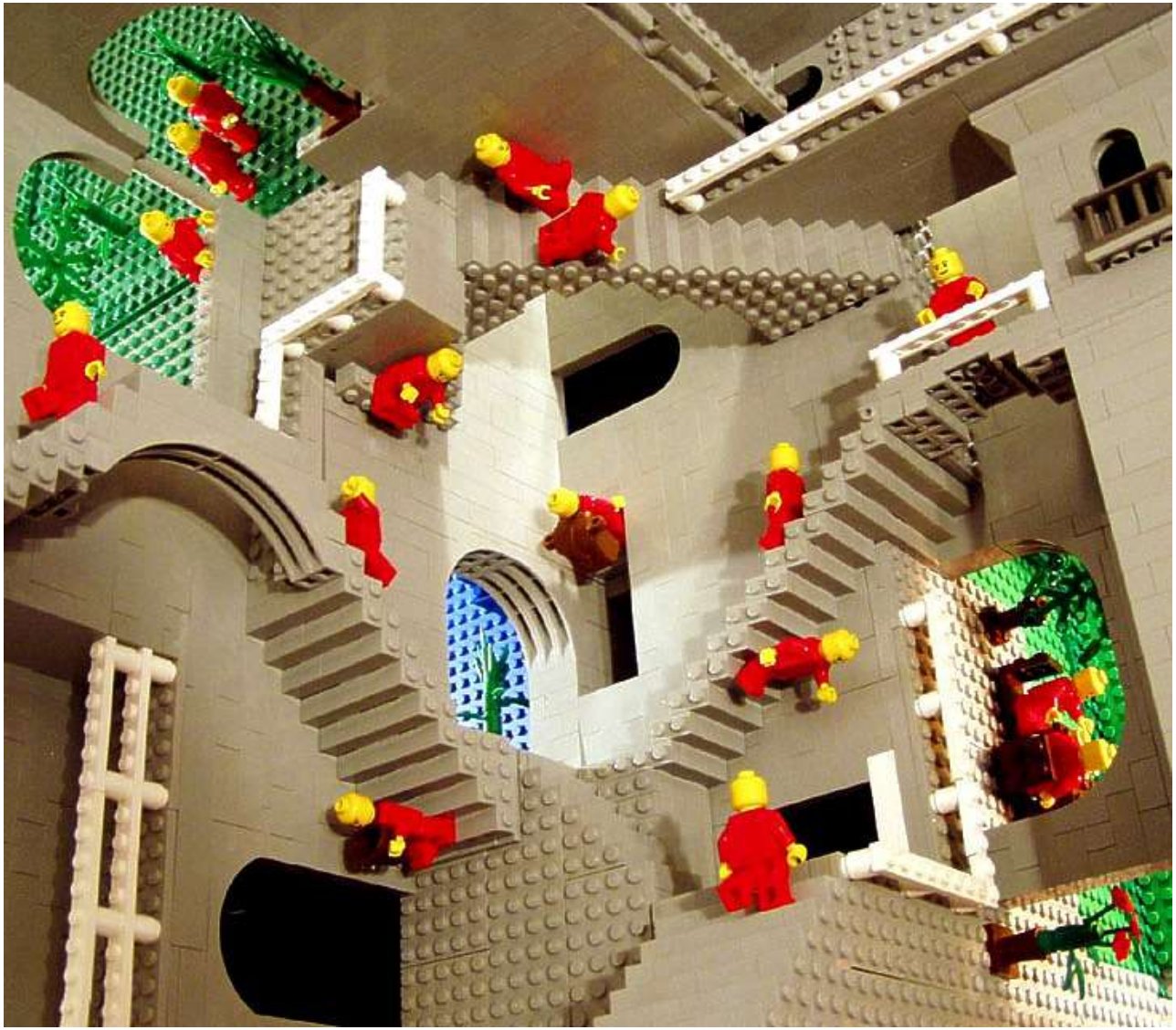
- A logic is a knowledge building *kit*
- Like LEGO, it only provides the building blocks. You do the actual building.



You can build something simple…

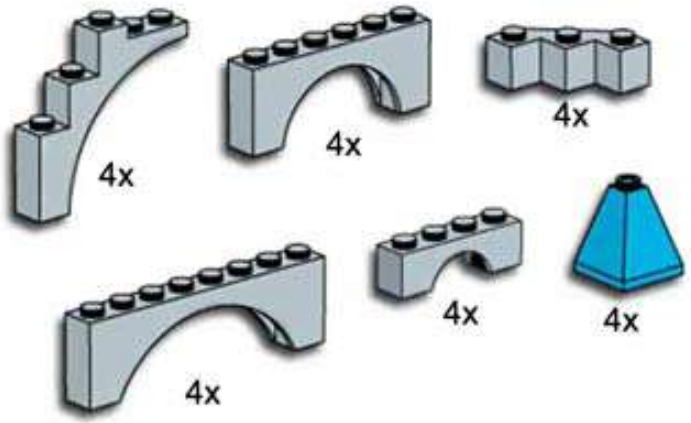
… or something more complicated

… or something completely insane



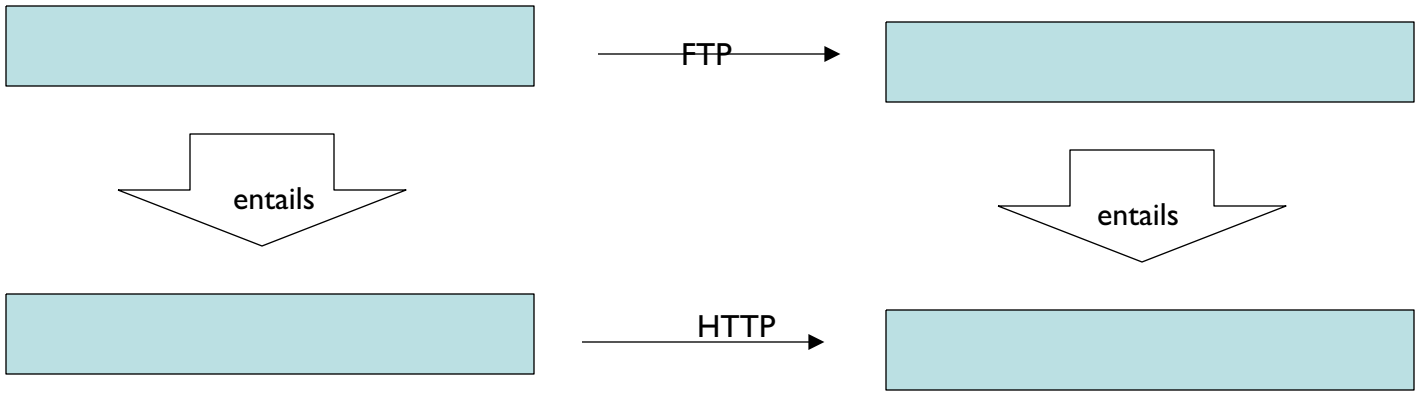
IKL is a logic

- Compared to traditional logic, Common Logic bricks fit together very easily.
- Compared to CL, IKL has some new kinds of brick.



IKL is a network logic

- IKL meanings do not change when IKL text is stored, transmitted, combined with other IKL text, or re-used
- IKL names refer and identify uniformly across a network
- IKL text from multiple sources can be combined freely without requiring negotiation.
- IKL entailment commutes with transfer protocols



IKL is a panoptic logic

- IKL describes a single ‘world’

- The IKL universe contains everything that can be described, referred to or hypothesized to exist

  including imaginary things, events that never happened but might have done, the times when they didn’t happen, etc..

So ‘forall’ says a lot in IKL, and is often restricted:

(forall ((x person)(y event)…

To say that something is 'real', say that explicitly.

IKL is a transparent logic

- Every occurrence of an IKL name has the same meaning.
- Equality substitution applies everywhere in IKL

  IKL is not indexical or referentially opaque (contextual, modal, tensed, belief, hybrid)

  Multiple referents for IKL names are handled by explicit functions on *quoted* names

  Descriptions which are relative to times, places, contexts, states of belief, points of view, etc., are made in IKL by relating *propositions* to the time, place, context, etc., explicitly.

  IKL both supports and requires *ontologies of context*.

IKL features from CL

- Names can be any Unicode character sequence

  John_Doe “John Doe” 08-21-1944 "John\'Doe\'"

  ÇZÜKJÖV "\00C7Z\00DCKJ\00D6V" הבפעצ صبئخحفغ I❤NY

- IKL can refer to and quantify over classes, properties, relations, functions, integers, character strings and ontologies.

- IKL texts can be named and imported into other texts; modules provide for a local universe of discourse

- Comments can be in any format and attached anywhere.

propositions

- can refer to, and quantify, over *propositions* expressed by sentences.

(exists (x)(and (Person x)(taller x Bill)))

(that (exists (x)(and (Person x)(taller x Bill))))

Proposition name

- propositions are real things that can be related to others

(Believes Mary (that (forall (x)(if (Person x)(smarter Bill x)))))

(LessLikelyThan (that (forall (x)(if (Person x)(smarter Bill x))))
(that ((AND Cheese Green)(materialOf Moon))))

and can be quantified, be the value of functions, etc.

(Believes Fred (Not (that (exists (p)(Believes Mary p)))))

function on propositions, not a connective.

- names *inside* proposition names work as usual, so you can quantify into them and reason about them
- (forall ((x Person))(Believes Mary (that (smarter Bill x))))

propositions

(forall ((x Person))(Believes Mary (that (smarter x Bill))))

Sentence *about* proposition

proposition name

Sentence *expressing* proposition

The inner sentence can be any IKL text.

Any name can refer to a proposition, but *proposition names* make the truth-conditions of the proposition explicit.

A proposition is *about* the things named by the free names in its proposition name.

propositions

- Propositions are identified with zero-ary relations, and handled similarly.

- Applying a proposition name to no arguments

  ((that (Married Bill Sue)))

  gives an atomic sentence which says the same thing as the sentence expressing the proposition.

  ((that (exists (x)(and (Person x)(taller x Bill)))))

  (exists (x)(and (Person x)(taller x Bill)))

- This works for *any term*, not just proposition names.

propositions

- Propositions are identified with zero-ary relations, and handled similarly.

- Applying a proposition name to no arguments

  ((that (Married Bill Sue)))

  gives an atomic sentence which says the same thing as the sentence expressing the proposition.

((that (exists (x)(and (Person x)(taller x Bill))))

Sentence

Proposition name

Sentence

(exists (x)(and (Person x)(taller x Bill))

propositions

- Propositions are identified with zero-ary relations, and handled similarly.

- Applying a proposition name to no arguments

  ((that (Married Bill Sue)))

  gives an atomic sentence which says the same thing as the sentence expressing the proposition.

  ~~((that~~ (exists (x)(and (Person x)(taller x Bill)) ~~)~~

  (exists (x)(and (Person x)(taller x Bill))

- The outer calling brackets can be read as 'it is true …'
  - *It is true that a person exists who is taller than Bill*

- Or, could say that the calling brackets *cancel that*.

propositions are very handy

Talk about beliefs (for example)

(forall (x)(if (Believes Mary x)(Believes Joe x)))

(forall (x)(if (Believes Mary x)(not (x))))

Assert properties of propositions

(forall (p (x agent))(if (less (secLevel x)(secLevel x))(prohibitedEvent (awareOf x p))))

Define functions on propositions

(forall (x)(= (Not x)(that (not (x)))))

Give names to complex propositions

(= hypothesis17 (that (member Joe Al_Quaeda)))

but beware of inherent contradictions

?? (= p (that (not (p)))) ??

Describe relations between propositions and contexts

(ist context39 (that (member Joe Al_Quaeda)))

Define classes of propositions.

identity of propositions

- When do two sentences express the same proposition?

- If two propositions are equal then their expressing sentences are equivalent:

  (if (= p q) (iff (p)(q)))

  … but maybe not the reverse.

- Identical propositions should be 'about the same things'

  A proposition expressed by a sentence is *about* the things referred to by the free names in the sentence.

identity of propositions

Logical equivalence of sentences is too inclusive

Syntactically same sentence is too fine-grained

- The special relation =P provides an intuitive notion of 'same proposition' for use in writing axioms [John Sowa]

```
(=P (that (and A B)) (that (and B  A)) )
(=P (that (or A  B C ... ))(that (not (and (not A)(not B)(not C)...))))
(=P (that (and A (and B C)))(that (and (and (A B) C))))
(=P (that (if A B)(that (or (not A) B)) )
(=P (that (iff A B))(that (and (if A B)(if B A))))
(=P (that (and A A) (that A))
(=P (that (forall (n) A)) (not (exists (n) (not A))) )
IF  (= n m)  THEN  (=P (that A) (that A[n/m] ))
```

- =P is decidable between sentences in n.log-n time

- If (=P x y) then x and y are 'about the same things'.

captured names

- Character sequences are *used* as names, and *denoted* by quoted strings. The string *captures* the name.

  'PatHayes' captures PatHayes

  'Chris Menzel' captures "Chris Menzel"

  'Sowa \(John\)' captures "Sowa \(John\)"

  '\00C7Z\00DCKJ\00D6V' captures ÇZÜKJÖV

- Any IKL name may be written as a quoted string which captures the same sequence, written inside brackets.

  PatHayes ==== ('PatHayes')

  (forall ((s string)(t string))(if (= s t)(= (s)(t))))

captured names

- Allow quantification over things referred to by names with various syntactic properties:

```
(forall ((s charseq) x y)(if
    (and (nationalSpelling s y)(culture x y) )
    (nationality (s) x)
))
```

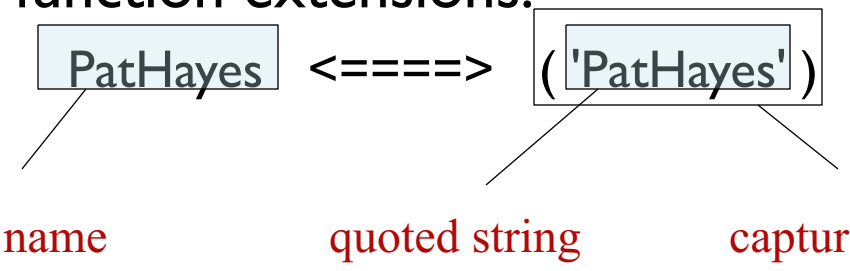
e.g. (nationalSpelling 'Llewelyn' (culture Wales))

entails

(nationality Llewelyn Wales)

captured names

- Captured names are zero-ary functions. IKL semantics requires that character sequences have corresponding zero-ary function extensions.



- this applies only to character sequences, so quantifiers must be restricted using *charseq*.

captured names can be captured

- This usage can be extended to describe alternative referents for names used in different contexts, by using the context as an argument to the captured-name function

  ('Fred') = Fred

  ('Fred' context13) = what 'Fred' refers to *in* context13

  ('Fred' interval17) = what 'Fred' refers to *during* interval17

  ('Fred' (beliefOf John)) = what John *believes that* 'Fred' refers to

  etc…

- Can think of this as an IKL version of subscripting a name:
  $Fred_{context13}$ $Fred_{interval17}$

- This device can represent subtle relationships between referents of names used in different contexts, beliefs, times, etc., without violating the transparency of IKL: every occurrence of an IKL name has the same meaning.

What's in a name?

Bill and Sue are married. Bill plays lacrosse. Robert knows them both but thinks they are not married. Mary does not know anything at all about Bill. She understands the name "Bill" to refer to Robert, and she knows that Robert and Sue are not married. Mary's sister Joan knows Bill personally, but also, for reasons that need not detain us here, believes that his name is "Robert". Finally, Joan's friend Wilma, a lacrosse fan, knows Bill as a lacrosse player and knows that Robert is a friend of Joan but, unlike Joan, she does not know that these are in fact the same person.

What's in a name?

Bill and Sue are married. Bill plays lacrosse. Robert knows them both but thinks they are not married. Mary does not know anything at all about Bill. She understands the name "Bill" to refer to Robert, and she knows that Robert and Sue are not married. Mary's sister Joan knows Bill personally, but also, for reasons that need not detain us here, believes that his name is "Robert". Finally, Joan's friend Wilma, a lacrosse fan, knows Bill as a lacrosse player and knows that Robert is a friend of Joan but, unlike Joan, she does not know that these are in fact the same person.

(married Bill Sue)

(believes Robert (that (not (married Bill Sue))))

(= Robert ('Bill' (BeliefsOf Mary)))

(believes Mary (that (not (married ???? Sue))))

All these names refer as usual. Nobody is confused about what the names mean, only about the facts.

What goes here? This in fact refers to Robert, but Mary thinks he is called 'Bill'.

Traditional opaque logics would use Mary's names when stating Mary's beliefs.

Based on the 'aggreement heuristic'.

What's in a name?

Bill and Sue are married. Bill plays lacrosse. Robert knows them both but thinks they are not married. Mary does not know anything at all about Bill. She understands the name "Bill" to refer to Robert, and she knows that Robert and Sue are not married. Mary's sister Joan knows Bill personally, but also, for reasons that need not detain us here, believes that his name is "Robert". Finally, Joan's friend Wilma, a lacrosse fan, knows Bill as a lacrosse player and knows that Robert is a friend of Joan but, unlike Joan, she does not know that these are in fact the same person.

(married Bill Sue)

(believes Robert (that (not (married Bill Sue))))

(= Robert ('Bill' (BeliefsOf Mary)))

(believes Mary (that (not (married ('Bill' (BeliefsOf Mary)) Sue))))

(forall ((s charseq) (p person))(= (s (Beliefs p))(s p)))

Structural axiom

(believes Mary (that (not (married ('Bill' Mary) Sue))))

What's in a name?

Bill and Sue are married. Bill plays lacrosse. Robert knows them both but thinks they are not married. Mary does not know anything at all about Bill. She understands the name "Bill" to refer to Robert, and she knows that Robert and Sue are not married. Mary's sister Joan knows Bill personally, but also, for reasons that need not detain us here, believes that his name is "Robert". Finally, Joan's friend Wilma, a lacrosse fan, knows Bill as a lacrosse player and knows that Robert is a friend of Joan but, unlike Joan, she does not know that these are in fact the same person.

(married Bill Sue)

(believes Robert (that (not (married Bill Sue))))

(= Robert ('Bill' (BeliefsOf Mary)))

(forall ((s charseq) (p person))(= (s (Beliefs p))(s p)))

(believes Mary (that (not (married ('Bill' Mary) Sue))))

(= Bill ('Robert' Joan))

(= Bill ('Bill' Wilma))

(not (= Bill ('Robert' Wilma)))

Who can this be? Nobody. Wilma is in a state of confusion. But that isn't our problem. IKL can have imaginary things in its universe.

Bill and Sue are married. Bill plays lacrosse. Robert knows them both but thinks they are not married. Mary does not know anything at all about Bill. She understands the name "Bill" to refer to Robert, and she knows that Robert and Sue are not married. Mary's sister Joan knows Bill personally, but also, for reasons that need not detain us here, believes that his name is "Robert". Finally, Joan's friend Wilma, a lacrosse fan, knows Bill as a lacrosse player and knows that Robert is a friend of Joan but, unlike Joan, she does not know that these are in fact the same person.

(married Bill Sue)

(believes Robert (that (not (married Bill Sue))))

(= Robert ('Bill' (BeliefsOf Mary)))

(forall ((s charseq) (p person))(= (s (Beliefs p))(s p)))

(believes Mary (that (not (married ('Bill' Mary) Sue))))

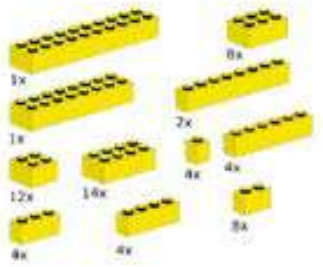
(= Bill ('Robert' Joan))

(= Bill ('Bill' Wilma))

(not (= Bill ('Robert' Wilma)))

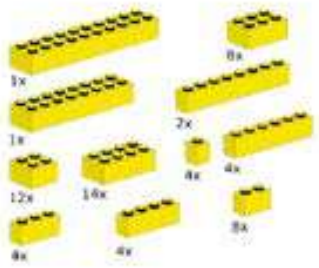
Jacob knows Bill and Sue, but does not consider them be married as their ceremony was not conducted by a minister of the true church.

(believes Jacob (not (('married' Jacob) Bill Sue)))



sequence… names vs. argument (lists)

- IKL follows CL in using quantified sequence names to express facts involving any number of arguments, using recursion.
- Many reasoners cannot handle this (yet), and other languages (RDF, OWL) use explicit argument lists.
- The RDF/OWL list technique can be axiomatized in IKL and then used consistently to replace sequence variables.
- This provides for interoperation between different techniques for handling multiple-arity relations.



sequence… names vs. argument (lists)

- (= nil (list))
- (forall (x …)(= (list x …)(cons x (list …)))
- (= (first (cons x y)) x)
- (= (rest (cons x y)) y)
- (forall (f)(if (listable f)
  (forall (…)(and (iff (f …)(f (list …))
  (= (f …)(f (list …))))
  ))

Now, if we say (listable R) we can freely go back and forth between

(R a b c …) <====> (R (list a b c …))

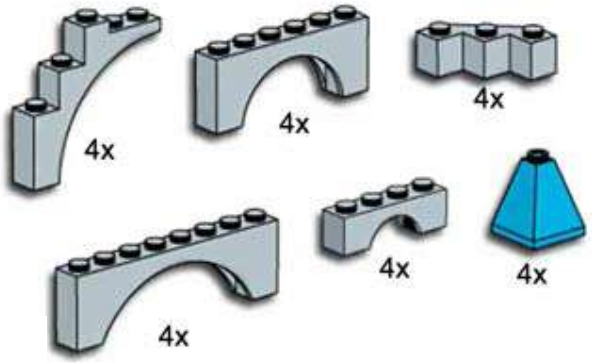
and can express recursions on lists by using *first* and *rest,* as in RDF

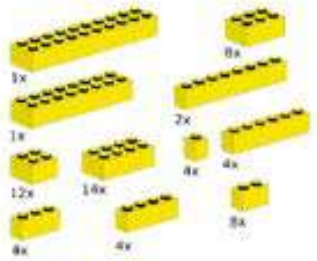
To change the vocabulary just write equations, e.g.

(= first hd)(= last tl)

structural axioms like this are easy to state in IKL and can be used to relate different vocabularies, styles, etc.

Now we have all the bricks





we can build some ontologies

- (forall (p) (= (Not p)(that (not (p)))))
  (= (And) (that (and)))
  (forall (p …)(= (And p …)(that (and (p)((And …)))))
  (forall (p q)(= (If p q)(that (if (p)(q)))))
  (forall (…)(= (Nand …)(Not (And …))))
- (forall (c x) (iff ((NOT c) x)(not (c x))))(= NOT owl:complementOf)
  (forall (x)((AND) x))
  (forall (x c …)(iff ((AND c …) x)(and (c x)((AND …) x))))
  (= AND owl:intersectionOf)
  (forall (c d)(
  (forall (c)(if (predicative c)(forall (x …)(iff (c x …)(and (c x)(c …))))))
  (forall (r x)((chain r) x))
  (forall (r x y …)(iff ((chain r) x y …)(and (r x y)((chain r) y …))))
- (forall (x)(allDiff x))
  (forall (x y …)(iff (allDiff x y …)(and (not (= x y)(allDiff x …)(allDiff y …))))
- (= charseq xsd:string)
  (predicative datatype)
  (datatype xsd:string charseq xsd;number xsd:date)
  (forall ((x datatype) (s string))(iff (legalStringOf x s)(x (x s))))

IKL can represent content expressed in many other kinds of logic

1. Modal and hybrid logics

   knows, believes

   future, past; true when

   should be, is prohibited

2. Temporal logic

   holds

3. Context logic

   ist

Other logics and IKL

- IKL can represent content expressed in many other kinds of logic

  But…translating these into IKL often requires more than a simple syntactic transformation in order to fully capture the intended meaning.

  Some logics are designed with particular ontological presumptions in mind, which must be made explicit in IKL ontologies.

For example, a tensed language assumes an underlying time structure of points or intervals, and modal languages assume a structure of 'alternative worlds'.

Temporal Logic in IKL

(holds T P)

(holds 1998 (PresidentOfUSA "William Clinton"))

Time-indexed sentence

But what does PresidentOfUSA mean in a transparent logic? Its not a simple property any more. Need to re-think it in some way to make its temporal nature explicit. Several possibilities have been tried.

It can be a relation involving time, a *fluent*:

(PresidentOfUSA "William Clinton" 1998)

or a function from times to properties:

((PresidentOfUSA 1998) "William Clinton")

or to things:

(= (PresidentOfUSA 1998) "William Clinton")

or even as a property of time-slices of things (*histories*):

(PresidentOfUSA (in "William Clinton" 1998))

or something that looks more like the original, but treats PresidentOfUSA as a function to a fluent, as in PSL:

((PresidentOfUSA "William Clinton") 1998)

This is not a sentence

Or these can be combined in various ways. But **the time parameter has to be in there *somewhere***.

Temporal Logic in IKL

(PresidentOfUSA "William Clinton" 1998)
((PresidentOfUSA 1998) "William Clinton")
(= (PresidentOfUSA 1998) "William Clinton")
(PresidentOfUSA (in "William Clinton" 1998))
((PresidentOfUSA "William Clinton") 1998)

IKL can describe relations between the various translation styles, using appropriate structural axioms:

(timeUnique PresidentOfUSA)
(forall ((r timeUnique) (t timeinterval) x)(iff (r x t)(= (r t) x)))
(forall (r (t timeinterval) …)(iff (r … t)((r t) …)))
(forall (r (t timeinterval) x)(iff (r x t)(r (x t))))
… histories are a bit more complicated…

Continuants vs. Roles

(holds 1998 (= PresidentOfUSA "William Clinton"))

This is trickier, since for example
(= PresidentOfUSA "William Clinton" 1998) **???**
does not make sense (isn't even syntactically legal IKL)

Need to distinguish names used in a temporal logic to refer to **continuants** from other names. Only the former should be translated into IKL names without being temporally qualified.

Continuants are things which are referred to in the same way at all times. (people, countries, … most things with proper names.) Other identifiers in a temporal logic are considered to define **roles**. When translating into IKL (or CL or FOL) it is important to *not* map both continuants and roles to simple individual names.

(= (PresidentOfUSA 1998) "William Clinton")
(PresidentOfUSA "William Clinton" 1998)

This relation is a continuant because it applies to all times

Temporal quantifiers

(holds 1974 (exists (x)(and (PresidentOfUSA x)(Crook x))))

what exactly does the 'i' quantify over?

A. Things that exist 'at that time'

B. Things that exist at all times (panoptic temporal logic, e.g. Cyc)

Case A requires a way to say 'exists at a time'. This requires some kind of *temporal ontology*. E.g.

(exists (x)(and

(During x 1974)(PresidentOfUSA x 1974)(Crook x)))

(forall (x t)(iff (During x t)(overlaps t (lifetime x)))) temporal ontology axiom

e.g. the conveyor belt between the coal crusher and the furnace

(holds I (exists (x)(and (coalPiece x)(on x conveyor))))

(forall ((J timeinterval))(if (subinterval J I)

(exists (x (K timeinterval))(and (subinterval K J)(on x conveyor K)

))))

Modal languages and IKL

- Classical semantics for modal languages due to Kripke, describes structure of *alternative possible worlds*. Different kinds of 'alternativeness' (transitive? reflexive?) give rise to different modal axioms.
- This translates into IKL (in fact, into GOFOL) using the same kind of techniques used for temporal logics, using explicit ontologies for possible worlds (e.g. situation calculus) *provided that names refer coherently*, so that they can be 'sliced'.
- If not, we have to be more subtle. Captured names with subscripts can be used to approach the general case.
- Modalities of obligation and permission can be expressed in IKL using ontologies of obligated and permitted *events* or *actions*.

Context logic and IKL

- (ist C SENT)in ICL maps to
  (ist C (that SENT)) in IKL ??

- Well, almost. Since ICL treats contextual sentences opaquely, while IKL is transparent, something has to be done about the names in the sentence.

The most general technique is to replace every name in ICL occurring in the context C by the captured name ('name' c) in IKL. This provides an accurate translation but is very unwieldy.

- (ist C SENT) maps to
- (ist C (that SENT'))
  where SENT' is SENT with every free name nnn replaced by ('nnn' C)

Context logic and IKL

- (ist C SENT) maps to
  (ist C (that SENT'))
  where SENT' is SENT with every free name nnn replaced by ('nnn' C)

eg (ist C (and (P a)(exists (x)(R x b))))
(ist C (that (and (('P' C) ('a' C))(exists (x)(('R' C) x ('b' C))))))

Nested ists are handled by applying the mapping recursively to the name of the context:

(ist C (ist D (P a)))

(ist C (that (ist ('D' C) (that (('P' ('D' C)) ('a' ('D' C)))))))

What 'P refers to in the context that C interprets 'D' to refer to

This is the translate of the atomic sentence (P a)

Context logic and IKL

eg (ist C (and (P a)(exists (x)(R x b))))

(ist C (that (and (('P' C) ('a' C))(exists (x)(('R' C) x ('b' C))))))

This can be simplified whenever we can safely assume that a context C uses a name nnn in the 'same way', i.e when (= nnn ('nnn' C)). (Continuants are an example for temporal contextualizations.) For example, if

(= D ('D' C)) (= P ('P' C)) (= R ('R' C))

then we get:

(ist C (and (P a)(exists (x)(R x b))))

(ist C (that (and (P ('a' C))(exists (x)(R x ('b' C))))))

(ist C (ist D (P a)))

(ist C (that (ist D (that (('P' D) ('a' D))))))

Context slicing in IKL

- (forall (c) (iff (AContext c)
  (forall (p ...)(iff
  (ist c (And p ...))
  (and (ist c (that (p))) (ist c (And ...)))
  ))
  ))
- (forall (c) (iff (OContext c)
  (forall (p ...)(iff
  (ist c (Or p ...))
  (or (ist c (that (p))) (ist c (Or ...)))
  ))
  ))
- (forall (c) (iff (NContext c)
  (forall (p)(iff
  (ist c (Not p))
  (not (ist c p))
  ))
  ))

It's amazing what you can build using bricks …