IKL

A LOGIC FOR INTEROPERATION

Pat Hayes
Florida IHMC
ONTOLOG invited presentation
2006-10-26

phayes@ihmc.us

IKL, CL, CLIF, KIF, etc.

see-ell

CL (Common Logic) is a draft ISO standard framework for describing first-order logic. It is not a single syntax: particular conforming syntaxes are called *dialects*. CL allows for a wide variety of surface syntax forms. CL syntax is unusually 'free', allowing a wide variety of expressions, and extends the semantics appropriately.

cliff

CLIF is one particular CL syntax, very 'lisp-like', modeled closely on KIF (though not identical). For more on CL and CLIF, see the ISO draft standard (available at http://common-logic.org/docs/cl/24707-21-June-2006.pdf Please do not cite until final ISO approval.) Many of the examples in this talk are written in CLIF syntax.

kiff

KIF is a venerable early 'standard' logic, the ancestor of CL. CLIF is very similar to a subset of KIF. Historically, the KIF project was the precursor of all the rest of this stuff, and many of the same people were involved. See http://logic.stanford.edu/kif/kif.html

ickle

eye-kay-ell

IKL is a recent extension of CL, very similar to CLIF, not yet 'standard' but in active use. This talk will focus on IKL, but will illustrate a lot about CLIF, and hence about CL, along the way. Development of IKL was supported by ARDA under the IKRIS program. For more on IKL, including many examples, see http://www.ihmc.us/users/phayes/IKL/GUIDE/GUIDE.html

IKL

IKL is a Common Logic (CL) extension which adds some *new kinds of naming expressions*, in particular a way to name propositions, but otherwise is a classical first-order logic.

It is expressive enough to represent content from a wide variety of exotic logics, all used by various systems to express ontological content. It is specifically designed for interoperation between knowledge systems.

One of the design principles of IKL was to keep the logic as syntactically simple and as unconstrained as possible. As a result, many different formalisms can be rendered or translated into IKL, and in many cases the translations between them can be stated as IKL axioms.

IKL DESIGN

Panoptic (one universe, containing every possible 'thing': *quantifiers work properly*) [IKL]

Transparent (meanings of expressions do not depend on the logical context: *equality works properly*)[CL, IKL]

Unrestricted (any name can be used for any logical purpose: *no type/sort checking, few 'legality' constraints*)[CL, IKL]

Uses **ordinary logic** (not modal, contextual, hybrid, partial, multivalued, free, whatever: *all the rest of the logic works properly*)[CL, IKL]

Is a **network logic** (meanings of expressions do not depend on their network location: *no lexical/syntactic negotiation*) (Horatio principle)[CL, IKL]

Network Logic

```
(Married Jack Jill)
(forall (x y)(if (and (Married x y)(Male x))
                 (Female y)
))
(Male Jack)
```

http://ex.PailOfWater

universe of people

universe of relations

http://ex:REL

```
(BinaryRelation Married)
(SymmetricRelation Married)
```

mathematical universe

```
(forall (r)(iff
   (SymmetricRelation r)
   (forall (x y)(iff (r x y)(r y x)))
))
```

http://ex:relalg

Network Logic

```
(Married Jill Jack)
(forall (x y)(if (and (Married x y)(Male x))
                 (Female y)
))
(Male Jack)
```

universe of people

http://ex.PailOfWater

universe of relations

http://ex:REL

```
(BinaryRelation Married)
(SymmetricRelation Married)
```

http://ex.MotherGoose

```
(Married Jill Jack)
(forall (x y)(if (and (Married x y)(Male x))
                 (Female y)
))
(Male Jack)

(BinaryRelation Married)
(SymmetricRelation Married)

(forall (r)(iff
   (SymmetricRelation r)
   (forall (x y)(iff (r x y)(r y x)))
))
```

mathematical universe

```
(forall (r)(iff
   (SymmetricRelation r)
   (forall (x y)(iff (r x y)(r y x)))
))
```

http://ex:relalg

universe contains people *and* relations over people

For all what, exactly?

When an ontology says

```
(forall (x ...
```

what exactly does this mean? For all *what*? Often some *particular* universe is intended, sometimes a 'closed world'. But the logic itself is panoptic, and readers of your ontology can see only the axioms, not the intention. So one should either restrict the quantifier:

```
(forall ((x person) ...
```

or else put the entire ontology into a CL *module*, which implicitly restricts all the quantifiers without your having to rewrite them all, and allows you to *exclude* some things from the 'local' universe:

```
(module http://ex.PailOfWater (exclude Married Male)(text
   (Married Jack Jill)
   (forall (x y)(if (and (Married x y)(Male x))
                    (Female y)
   ))
   (Male Jack)
))
```

Things in the universe called *http://ex.PailOfWater*, which, b.t.w., do not include *Married* and *Male*

For a full story, go elsewhere

Rather than survey the actual language(s) feature by feature, the rest this talk will focus on how to translate from other notations into IKL/CLIF. For details, background etc. see:

IKL:

IKRIS main site http://nrrc.mitre.org/NRRC/ikris.htm
overview http://www.ihmc.us/users/phayes/IKL/GUIDE/GUIDE.html
formal spec (for geeks) http://www.ihmc.us/users/phayes/IKL/SPEC/SPEC.html
slides http://www.ihmc.us/users/phayes/IKL/IKRIS_MV.ppt

CL:

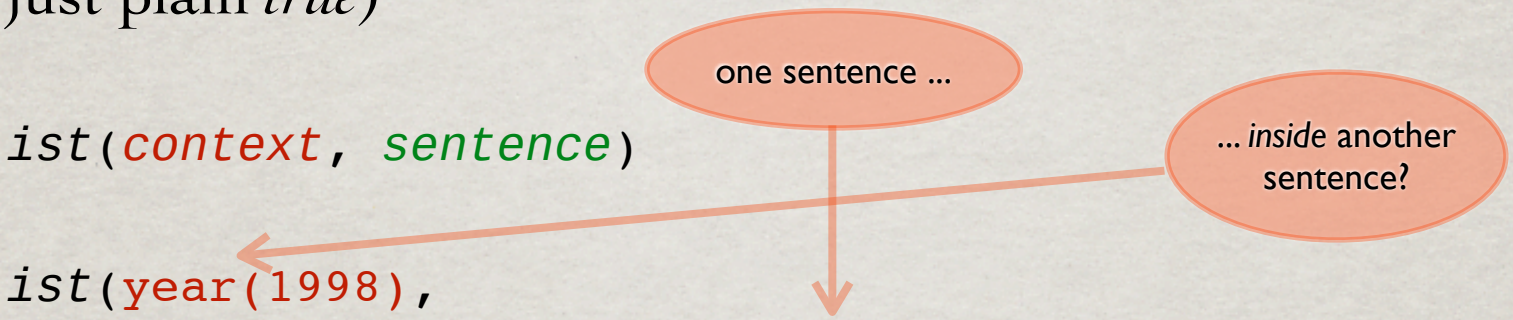
main website http://cl.tamu.edu/
ISO draft http://cl.tamu.edu/docs/cl/24707-21-June-2006.pdf
slide shows http://cl.tamu.edu/docs/cl/Common-Logic-Mar2005.ppt
http://cl.tamu.edu/docs/cl/Berlin_OpenForum_Delugach.ppt

Background papers

http://reliant.teknowledge.com/IJCAI01/HayesMenzel-SKIF-IJCAI2001.pdf
(This refers to SKIF, which was a very early CL draft)
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-82/SI_paper_12.pdf
(This refers to 'SCL' which was an early CL draft)

Example: Context Logic to IKL

Context logic assumes that sentences are true *in a context* (not just plain *true*)




```
ist(context, sentence)
```

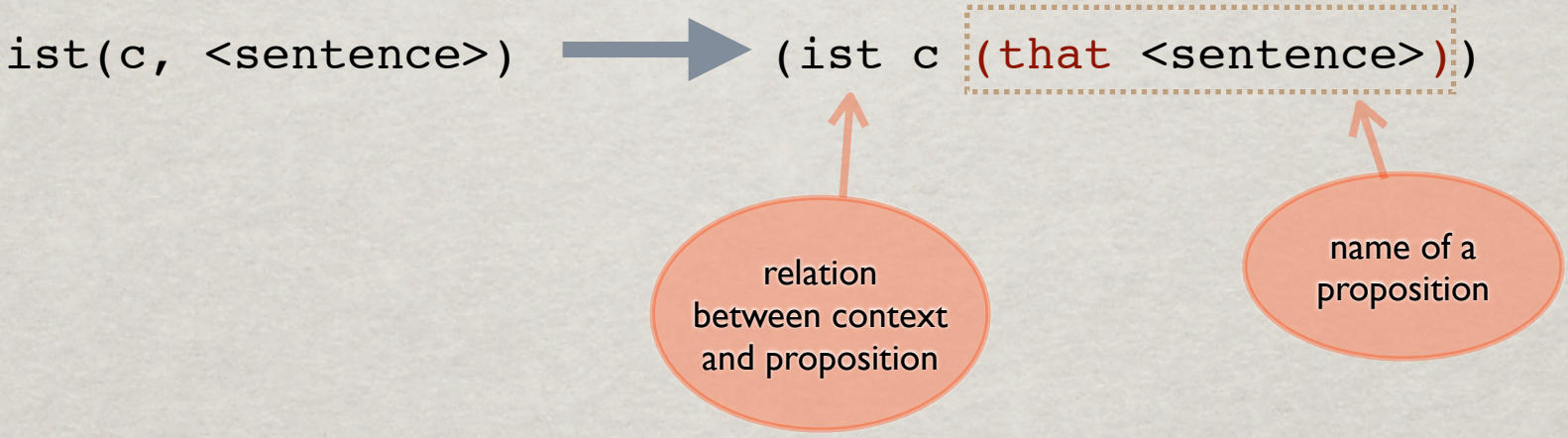
```
ist(year(1998),
    (forall ((?x DutchCitizen)(?y DutchCitizen))
    (if (Married(?x,?y) and Male(?x))
        Female(?y))) )
```

This kind of expression does not make sense in FO logic. Proponents of context logic claim it is a new approach to the foundations of logic itself.

Depending on the particular context logic, the meanings of sentences, names, quantifiers, etc., can change between contexts. This example changed in the `year(2000)`. *Context logic is therefore neither transparent nor a network logic.*

Example: Context Logic to IKL

IKL treats contextual assertions as relations between contexts and *propositions*, rather than as a special new logical form. The logic is then unchanged, but we have a new kind of naming expression, a **proposition name**.



Example: Context Logic to IKL

```
ist(year(1998),
    (forall ((?x DutchCitizen)(?y DutchCitizen))
    (if (and Married(?x,?y), Male(?x))
        Female(?y))) )
```

```
(ist (year 1998)
     (that
        (forall ((x DutchCitizen)(y DutchCitizen))
        (if (and (Married x y)(Male x)) (Female y))
     )
)
```

proposition name

A proposition name is a sentence with `(that ... )` wrapped around it. Any sentence will do, no matter how complicated. *Any* such name denotes a proposition of some kind, although in some 'paradoxical' cases it might not be what it seems to be (more on this later). The names inside proposition names still refer as usual, and logical rules apply to them as usual.

Proposition names are a new idea: but even though they are exotic names, they are still just *names*. They refer to new kinds of thing, but they don't change the actual logic. The actual logic of IKL is just common logic.

Example: Context Logic to IKL

```
(ist (year 1998)
     (that
        (forall ((x DutchCitizen)(y DutchCitizen))
        (if (and (Married x y)(Male x)) (Female y)))
))
```

Transparent (reference is independent of context; equality works properly)

so, we have to keep track of the fluents ...

```
(ist (year 1998)
     (that
        (forall (x y)(if
        (and (DutchCitizen x 1998)(DutchCitizen y 1998))
        (if (and (Married x y 1998)(Male x))
            (Female y))))
     )
)
```

Example: Context Logic to IKL

```
(ist (year 1998)
     (that
        (forall (x y)(if
        (and (DutchCitizen x 1998)(DutchCitizen y 1998))
        (if (and (Married x y 1998)(Male x))
            (Female y))))
     )
)
```

In fact, we no longer need the surrounding `ist` in this case, as the inner sentence has been completely *decontextualized*, so we can 'unwrap' the inner sentence from its proposition name:

```
(forall (x y)(if
        (and (DutchCitizen x 1998)(DutchCitizen y 1998))
        (if (and (Married x y 1998)(Male x))
            (Female y))))
```

The process of putting contextual content into a panoptic logic is basically one of decontextualizing it by properly incorporating the 'context' into the names.

All the art lies in knowing where to insert the context parameters. This case, for example, presumes that `Male` and `Female` are *not* fluents. In IKL such conditions can be stated as part of the ontology itself, or as a 'meta-ontology' used by a translator.

Many are the Ways

A fluent is a time-varying property... but there are many ways to put this idea into a logical formalism, and some of these come with heavy philosophical baggage. CLIF accommodates them all:

Fluent as a relation (on continuants) with an extra time parameter:

```
(Married x y 1998)
```

Fluent as a function from times to relations:

```
((Married 1998) x y)
```

Fluent as relation on temporal slices of enduring things ('4-d worms', histories, endurants)

```
(Married (x 1998)(y 1998))
```

And, for comparison, fluent as a relation between times and propositions, using IKL:

```
(istAtTime 1998 (that (Married x y)))
```

```
(Married x y)  +  1998
```

Axiomatizing syntax transforms in CLIF

Relations between different forms can be stated using CLIF/IKL axioms:

```
(forall (c r ...)(if
    (ContextParameterizableRelation r c)
    (iff (r ... c)((r c) ...))
))
```

Although in some cases you have to effectively embed LISP into CLIF (which you can indeed do) and so it might be simpler to just use a PERL script.

```
(forall (r ...)(iff (r (argseq ...))
(forall (x ...)(= (argseq x ...)(conseq x (argseq ...)))))
```

```
(forall (c)(= (sliceall c) (argseq)))
(forall (c x ...)(=
       (sliceall c x ...)
       (conseq (x c)(sliceall c ...))
))
```

```
(forall ((r sliceable) c ...)(iff
     (r ... c)
     (r (sliceall c ...))
))
```

Ontologies are more flexible than Logics

Contexts can be many things: times, time-intervals (in two senses), places (ditto), points of view, systems of belief, sources, documents, etc.. These all have different logical/ontological properties. For example: beliefs and document sources may 'contain' contradictions, but times and places cannot; conjunction, but not disjunction, is preserved under truth *throughout-a-time-interval*; the reverse under truth-*during-a-time-interval*.

Since IKL uses an **ontology** of contexts rather than a **logic** of contexts, it can describe all these variations explicitly, using for example different 'ist' relations, without requiring a change to the logic.

```
(isTrueThroughout 2002 (that (= (President USA) GWB)))

(isTrueDuring 2002 (that (exists ((x LesserKestrel))
                           (observed x IslesOfScilly))))

(Believes John (that (and (Human Superman)
                          (not (Human Superman))) ))
```

Axiomatizing syntax transforms in CLIF

The 'decontextualizing' rule which extracted the inner content from the 'ist' can itself be stated as an axiom:

```
(forall (c p)(if (decontext p)
(iff (ist c p)(p))
))
```

p here refers to the proposition. IKL can quantify over *all propositions*.

```
(forall (c (p decontext))(iff (ist c p)(p)))
```

Much of the utility of IKL (using CL 'wild west' syntactic freedom) lies in its ability to directly express what previously had to be treated as meta-mappings between different formalisms or different ontologies. Now, a variety of alternatives can be treated as *one large ontology*.

IKL's truth predicate

```
(forall (c (p decontext))(iff (ist c p)(p)))
```

Q: What does `(p)` mean?

A: It asserts that the proposition `p` is true: that is, it's just like writing the sentence of the proposition.

`((that <sentence>))` means the same as `<sentence>`

So the above says: if `p` is 'decontextualized', then `(ist c p)` is true iff `p` is true just by itself.

We could in fact define `ist` without a context argument as a truth predicate:

```
(forall (p)(iff (p)(ist p)))
```

Note, this quantifies over relations (legal in CL) and uses the same relation name with different arities (legal in CL).

OPAQUE NAMES

Sometimes, names used in a context change their meaning in more radical ways than just being time-sliced. For example, logics of belief (or, belief contexts) allow beliefs to contain misunderstandings about what names denote. Superman is Clark Kent, but Lois Lane doesn't believe it. She thinks there are two people when in fact there is only one. She knows who Clark Kent is, but her idea of Superman is imaginary. The name 'Superman' in her beliefs means something different from what it means in the real world.

To handle cases like this, IKL treats *quoted* names as functions from the context to the false referent.

```
('Superman' (belief Lois))
```

is what the name 'Superman' denotes *in the world of Lois' beliefs*, which might be quite different from Superman. In particular,

```
(not (= ('Superman' (belief Lois)) "Clark Kent"))
```

double quotes are CL's way of allowing names to contain spaces

OPAQUE NAMES

Just as with proposition names, if you leave out the context, you get the 'normal' case. Here, if you call a quoted name as a function with no arguments, then you get the normal referent:

```
<name> = ('<name>')
```

This allows one to write axioms about things by quantifying over their names, i.e. over character sequences. For example, we can define a 'transparent context' as one that uses all names correctly:

```
(forall (c)(iff (TransparentContext c)
                (forall ((s charseq))(= (s)(s
c)))
)
```

Note the restriction to character sequences: this is typical when quantifying over opaque names.

OPAQUE NAMES: FAQ

Q: How can a character string be a function??
A: In CL, *anything* in the universe of quantification is a function (and a relation). In IKL (and in CLIF), the universe always contains character strings. Ergo, they can be used as functions.

Q: That doesn't look like the kind of logic I'm used to, I don't like it.
A: OK, ignore the formal semantics and think of the context argument to the quoted-name-function as a kind of subscript to the quoted name, to indicate that its being used in a nonstandard 'contextual' way. That's really all that this boils down to in practice.

Q: What 'world' is this imaginary un-Superman really in?
A: An IKL panoptic universe. When you say 'forall' in IKL, you are quantifying over *everything*, even imaginary things. A basic rule is: if anyone could possibly refer to it, it can be in an IKL universe. Harry Potter is in an IKL universe, and so is his pet aardvark (which AK Rowling hasn't thought of, but I have.)

OPAQUE NAMES: FAQ

Q: Does that mean that IKL assumes that all this fantasy stuff is real?
A: No, things that actually exist in the real world are a small part of the IKL universe. To refer to them, use a predicate like isReal. Think of it as a context if you like.

Q: When Im writing my axioms, do I have to keep restricting my quantifiers to exclude all this nonsensical stuff?
A: No. Nor should you, indeed. Just write your axioms assuming that you are talking about some universe, put them in a Common Logic module and give it a name. The name of that module will be used by others as the name of your universe of discourse.

Q: Why 'opaque'?
A: I'm glad you asked that question. See the next slide.

OPAQUE NAMES

The traditional way is to use the name in a context to mean what it refers to *in that context*. Our example in a modal belief logic might then look like this:

```
Believes(Lois): (not (= Clark_Kent Superman))
```

where 'Superman' here denotes what Lois *thinks* it refers to, i.e. not the real Superman. The problem is, when different uses of a name refer differently, one cannot do equality reasoning:

```
(= Superman Clark_Kent)
(Believes(Lois): (not (= Clark_Kent Clark_Kent))) ??
```

A context (modality) which changes the referent of a name in this way is called *opaque* because the names inside it are invisible to normal logical inferences.

IKL incorporates this 'opacity' into the name itself, rather than relying implicitly on the syntactic context of use, thereby preserving the transparency of the logic:

```
(= Superman Clark_Kent)
(Believes Lois (that
     (not (= Clark_Kent ('Superman' Lois)))
))
```

Basically, IKL reduces all opacity to ordinary quotation, a single 'opaque' construction which is entirely unproblematic and has a trivial semantics.

OPAQUE NAMES

This has some costs - we have to be explicit about Lois' confusions - but also some benefits. For example, to say something 'normal' in an opaque logic, one has to resort to quantifiers:

```
(exists (x)(and
    (= x Superman)
    (Believes Lois (not (= x Superman)))
))
```

but this can be stated directly and simply in IKL:

```
(not (= Superman ('Superman' Lois)))
```

and by quantifying over the names, we can make general statements, for example that Lois is confused about some referent, without saying which:

```
(exists ((s charseq))(not (= (s)(s Lois))))
```

OPAQUE NAMES AS TYPED LITERALS

Although this opaque name construction is not conventional in logic, it is in fact already in wide use in the form of datatyped literals, which consist of a character string and a mapping, called a datatype, from strings to intended denotations. For example in RDF

```
ex:arthur  ex:ageInYears   "30"^^xsd:number  .
```

which maps to an IKL atomic sentence using an opaque name:

```
(ex:ageInYears  ex:arthur  ('32' xsd:number))
```

Example 2: Description Logics

Description logics don't look anything like conventional FO logic. They define *classes* in terms of other classes and 'restrictions' on *properties*, such as *the class of people whose parents are US citizens and were born after 1955*.

In CLIF/IKL, classes and properties are unary and binary relations, and the various DL class constructors are functions on these relations. For example

```
(= ChildOfUSCitizenPost1955
(AND (ALLARE parentOf (MUSTBE isCitizenOf  USA))
     (ALLARE dateOfBirth YearsSince1955)
)
```

which is the translation of the OWL-DL on the next slide. Then to say that someone is in this class, just use an atomic assertion:

```
(ChildOfUSCitizenPost1955 "Dorothy Möston")
```

Example 2: Description Logics

```
(= ChildOfUSCitizenPost1955
(AND (ALLARE parentOf (MUSTBE isCitizenOf  USA))
     (ALLARE dateOfBirth YearsSince1955)
)
```

```
<owl:Class rdf:id="#ChildOfUSCitizenPost1955">
  <owl:intersectionOf rdf:parseType="Collection">
   <owl:Restriction>
    <owl:onProperty rdf:resource="#parentOf" />
    <owl:allValuesFrom>
      <owl:Restriction>
       <owl:onProperty rdf:resource="#isCitizenOf" />
       <owl:hasValue rdf:resource="#USA" />
      </owl:Restriction>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#dateOfBirth />
    <owl:allvaluesFrom rdf:resource="#YearsSince1955" />
   </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Although, to be fair, much of the longwindedness is due to the RDF/XML in the OWL version. The CLIF translation style at the top mirrors the usual DL syntax quite closely.

For more on this topic, see http://www.ihmc.us/users/phayes/CL/SW2SCL.html
and http://philebus.tamu.edu/cmenzel/Papers/AxiomaticSemantics.pdf

Example 3: UNA and closed worlds

Since IKL can quantify over its own names, it can state general conditions on naming. These are often assumed to require a special non-monotonic logic, but that is a misapprehension.

A DB is a complete list of employees, so if a name isn't listed there then that person is not an employee (a *closed world assumption*).

```
(forall ((name charseq))(if
  (employee (name))
  (member name DBlist)
))
```

Each employee is listed there under a single name (a *unique name assumption*)

```
(forall ((n1 charseq)(n2 charseq))(if
   (and (= (n1)(n2))(member n1 DBlist)(member n2 DBlist))
   (= n1 n2)
))
```

Example 4: modal business rules

The general outlines of how to translate modal logics into FO have been known for decades. Temporal modalities are treated similarly to temporal context logics. For deontic modalities (permissions, obligations) its usually better to map them into an ontology of actions and states, with a 'badness' property to indicate prohibited actions. For example:

```
(forall ((e rentalEvent))(if
    (not (ValidDrivingLicense (licence (driver e))))
    (ImproperEvent e)
))
```

translates the deontic rule

```
(forall ((e rentalEvent))
[Obligatory: (ValidDrivingLicence (licence (driver e))) ]
)
```

This style also allows for more nuanced descriptions of *kinds* of improper or forbidden events, and the circumstances which surround them.

Some other things

CLIF/IKL syntax:

= allows *any* unicode character sequence to be used as a name

= allows arbitrary text to be attached as a comment to any expression, to any depth

= supports restricted and numerical quantifiers

= can be transmitted in XML using the CL conventions

= provides for naming of 'knowledge sources' and importing of IKL text, using Web conventions.

Bottom Line

IKL can express just about any content which can be written in any formalism, sometimes more compactly than the original.

In many cases it can also express relationships between different formalizations.

Writing CLIF/IKL axioms is easy as there are no restrictions on what can be said: one can directly state object data, data models, meta-models and structural axioms all in the same formalism, and it is all processed using the same logical rules.

It is the nearest thing yet to a *universal ontology solvent*.

Geek Time

Q: IKL can quantify over the propositions indicated by its own sentences, and it has an unrestricted truth predicate, so isn't it self-contradictory because of the paradoxes?

A: No. Most logics impose syntax constraints to prevent writing self-referential sentences, but in IKL they all just turn out to be contradictions, is all. For example

```
(= p (that (not (p)))
```

Liar paradox (no such p)

```
(forall (x)(iff (R x)(not (x x)) ))
```

Russell's paradox (no such R)

```
(forall (x)(iff (K x)
   (= x (that (forall (y)(iff (K y)(not (y)) )) )
))
```

Kripke's contingent paradox (no such K)

In IKL, these all assert that an impossible thing exists. Since this is always false, they are all contradictions. What makes them seem paradoxical is that they **look like** definitions. If they were, we would be in trouble.
There is a cost to this freedom, though. For users, be careful writing axioms; for logic designers, its harder to show that models exist; and for inference engines, there are new kinds of contradiction to consider.

Names are the Ontological key

Moral: It's all about having enough ways of naming things. CL lets you use names to refer to relations and functions (classes, properties, ...) . CLIF has names for numbers and character sequences (names), and CL modules supply names for local universes and for ontologies themselves. IKL adds opaque names for imaginary things and names for propositions. That seems to be enough.

First order logic, correctly understood, is adequate for all reasoning that anyone will ever want to do. You don't need a different logic: you need better ways to refer.

" If names be not correct, language is not in accordance with the truth of things. If language be not in accordance with the truth of things, affairs cannot be carried on to success. ... What the superior man requires is just that in his words there may be nothing incorrect. "

Confucious, Analects XIII, 3