Formalization of the CRM: A first-order attempt

Carlo Meghini and Martin Doerr

Istituto di Scienza e Tecnologie della Informazione
Consiglio Nazionale delle Ricerche – Pisa

Iraklio, October 6th, 2015

Outline

- Introduction: Why, how, what
- Preliminaries
- A first-order theory of the CRM
- Post-reflections
- Conclusions

Introduction: Why doing it?

A representation language without formal semantics in incomplete.

Cannot define inference, *i.e.*, consistency checking and querying.

Cannot really evaluate whether the model correctly reflects and predicts reality.

Communication to other researchers:

- Understanding
- Computational Research
- Comparison
- Extension
- Experimentation

How to do it?

Choose your favourite theoretical tool:

- Mathematical Logic
- Computational Logic
- Set Theory
- Category Theory
- *you name it*

My choice: mathematical logic, because I know it (a bit)

But the are also advantages:

- discourse formalization (syntax and semantics)
- argument formalization (proof)
- many results to use (*e.g.*, description logics)

What did I do so far?

1. First-order logic translation of (parts of) the specs
2. Computational analysis of the resulting theory
   - Result: the theory is tractable
3. Definition of what a KB is (not really needed for an ontology, actually)
4. Theoretical implementation (in `datalog`)

Some things remain to be done:

1. Convince Martin and the SIG
2. Convince some journal editor
3. Practical implementation

Related Work

The work of formalizing in FO logic a semantic data model has been done more than thirty years ago by Ray Reiter.

- From this work we draw the basic principles of our formalization.
- The CRM specification includes some constraints, *e.g.*, shortcuts and quantifications, that have not been treated by Reiter.

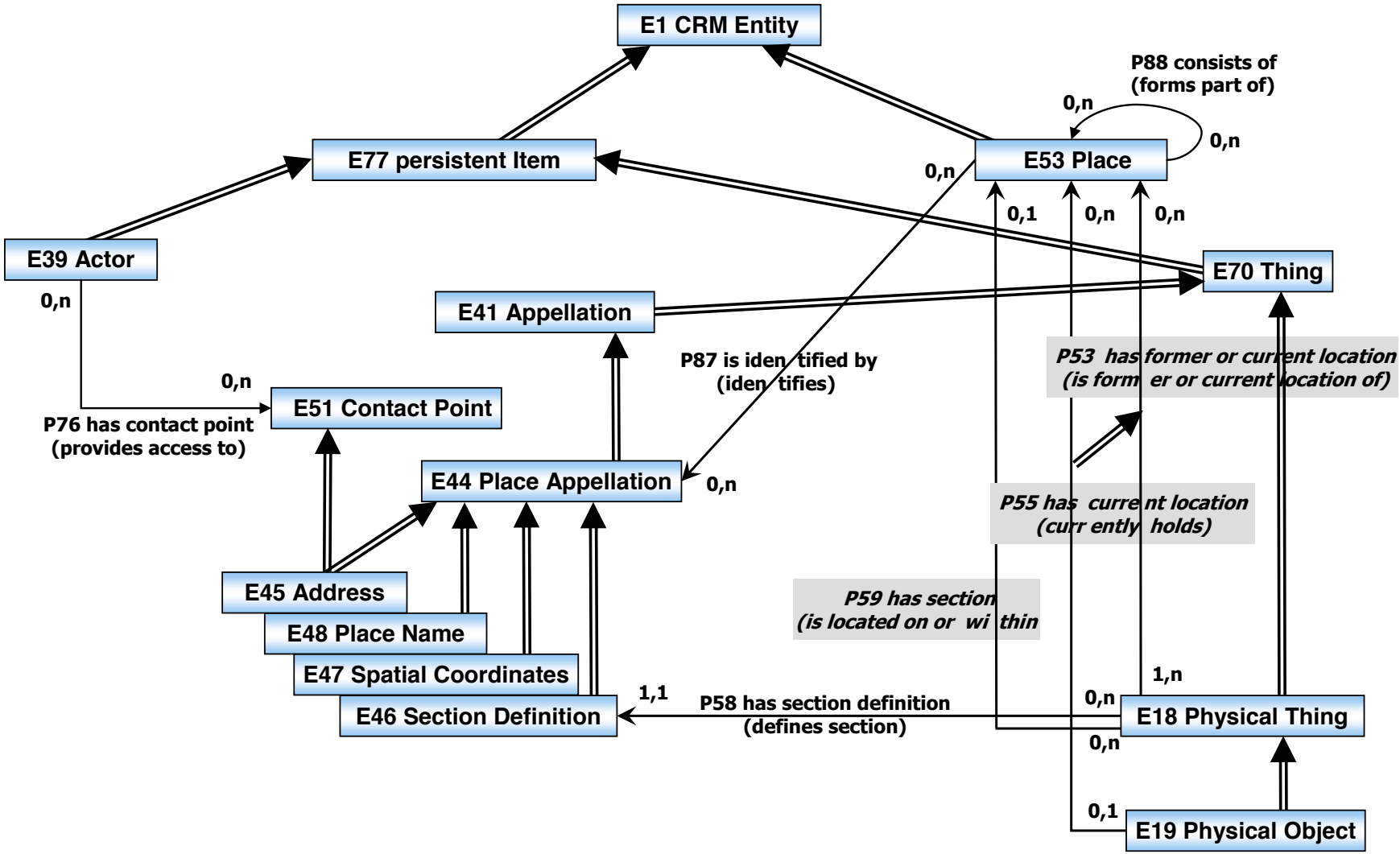
Shortcuts and quantifications (and much more) have been treated by Description Logics, today mostly used via the Ontology Web Language of the W3C.

We do NOT buy into DLs or OWL for several reasons:

1. OWL uses IRIs, we do not need to go that far
2. OWL does not cover strong shortcuts
3. OWL and DLs do not distinguish known from unknown individuals

We buy into Levesque & Lakemeyer $\mathcal{L}$ logic.

Preliminaries from Reiter's work



Classes

Classes are unary predicate symbols

Classes describe qualities of single individuals, have a time-less intension (*e.g.*, meaning) and a time-dependent extension (*e.g.*, individuals, the *instances* of the class).

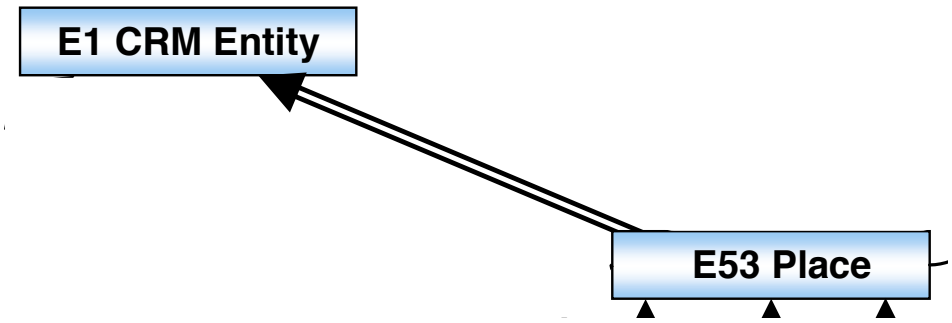
For each CRM class, we introduce in our FO language a unary predicate symbol, given by the class identifier.

- E53 Place $\Rightarrow$ *E53*

IsA links are conditionals:

- $(\forall x)\ [E53(x) \supset E1(x)]$

For all individuals x, if x is a $E53$, then x is a $E1$.



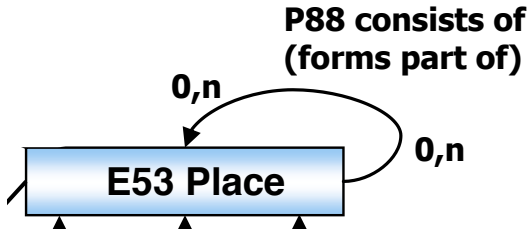
Disjointness constraints are negative conditionals:

E2 Temporal Entity is disjoint from E77 Persistent Item.

$(\forall x)[E2(x) \supset \neg E77(x)]$

Properties

Properties are binary predicate symbols



Properties describe qualities of pairs individuals (*Carlo* likes *Brunello*), have a time-less intension (*e.g.*, meaning) and a time-dependent extension (*e.g.*, pairs of individuals).

For each CRM property, we introduce in our FO language a binary predicate symbol, given by the property identifier.

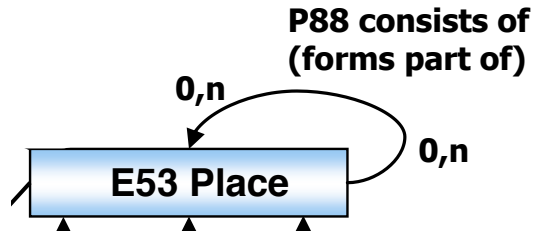
- P88 consists of $\Rightarrow$ *P88*

Property IsA links are also expressed as conditionals:

P12	occurred in the presence of (was present at)
P111	- added (was added by)

$$(\forall xy)[P111(x, y) \supset P12(x, y)]$$

Properties also have domain and range restrictions



These restrictions naturally translates as conditional axioms as well:

CRM Specification	*First-order logic*
P has Domain C	$(\forall xy)[P(x,y) \supset C(x)]$
P has Range D	$(\forall xy)[P(x,y) \supset D(y)]$

e.g.,

- $(\forall xy)[P88(x,y) \supset E53(x)]$
- $(\forall xy)[P88(x,y) \supset E53(y)]$

Properties may be:

- symmetric: P114 is equal in time to (of time periods)
- transitive: P86 falls within (of time periods)

Conditionals are also good for these axioms, as we learn at the lyceum (let's drop universal quantifiers to make our formulas lighter):

- symmetric: $P114(x, y) \supset P114(y, x)$
- transitive: $P86(x, y) \wedge P86(y, z) \supset P86(x, z)$

So far we have covered the basics.

Meta-Properties

A meta-property is a property whose domain is a property.

Meta-properties are modelled as 3-place predicate symbols:

- the first two places are given to the terms in the domain property,
- the last place is used for the type.

CRM Specification	*Translation into first-order logic*
P has Meta-Property P.n: C	$P.n(x, y, z) \supset [P(x, y) \wedge C(z)]$
P has Asymmetric Meta-Prop. P.n: C	$P.n(x, y, z) \supset [P(x, y) \wedge \neg P.n(y, x, z) \wedge C(z)]$

The corresponding axiom includes the assertion of the domain property in the consequent, thus making it possible to omit it whenever a typing statment is present.

Shortcuts

Property	*Type*	*Shortcut (from the CRM Specifications)*
P2 has type (is type of)	strong	From E1 CRM Entity through P41 classified (was classified), E17 Type Assignment, P42 assigned (was assigned by) to E55 Type
P43 has dimension (is dimension of)	weak	From E70 Thing through P39 measured (was measured by), E16 Measurement, P40 observed dimension (was observed in) to E54 Dimension
P53 has former or current location	inverse weak	From E18 Physical Thing through P161 has spatial projection, E53 Place, P121 overlaps with to E53 Place

CRM Specification	*Translation into first-order logic*
Weak Shortcut $P_1 \ldots P_n$	$[P_1(x, z_1) \wedge P_2(z_1, z_2) \wedge \ldots \wedge P_n(z_n, y)] \supset P(x, y)$
Weak Inverse Shortcut $P_1 \ldots P_n$	$P(x, y) \supset \exists z_1 \ldots z_n [P_1(x, z_1) \wedge \ldots \wedge P_n(z_n, y)]$
Strong Shortcut $P_1 \ldots P_n$	$P(x, y) \equiv \exists z_1 \ldots z_n [P_1(x, z_1) \wedge \ldots \wedge P_n(z_n, y)]$

Weak shortcut:

$$[P39(x, y) \wedge P40(y, z)] \supset P43(x, z)$$

Note that from domain and range axioms it follows that x, y and z are instances of $E70$, $E16$ and $E54$, respectively.

Likewise, inverse weak shortcut:

$$P53(x, y) \supset (\exists z)[P161(x, z) \wedge P121(z, y)]$$

Property quantification

The definition of quantifiers is given in terms of two features:

- *total property* and
- *functional property*

that can be applied to a property or to its inverse. Therefore a property or its inverse fall exactly into one of the following cases:

1. total and not functional, *i.e.*, defined on every element of its domain and can take up more than one value;
2. functional and not total, *i.e.*, at most one value is provided for any element of its domain;
3. the property is neither total, *i.e.*, some domain elements can miss it, nor functional, *i.e.*, can take up more than one value for any element of its domain;
4. both total and functional, *i.e.*all domain element must have one value for it, and no more than one.

Translation:

- P is functional: $[P(x,y) \wedge P(x,y')] \supset (y = y')$
- P is total (on domain A): $A(x) \supset \exists y P(x,y)$
- the inverse of P is functional: $[P(x,y) \wedge P(x',y)] \supset (x = x')$
- the inverse of P is total (having range A): $A(x) \supset \exists y P(y,x)$

The complete translation of each quantifier can be obtained by conjoining the translation of the corresponding features. For instance:

- *many to many* (0,n:0,n): P and its inverse are neither total nor functional: no axiom
- *one to one* (1,1:1,1): P and its inverse are total and functional:

  $A(x) \supset \exists y P(x,y)$
  $[P(x,y) \wedge P(x,y')] \supset (y = y')$
  $B(x) \supset \exists y P(y,x)$
  $[P(x,y) \wedge P(x',y)] \supset (x = x')$

Co-reference axioms

The well-known axioms for co-reference (or equality) are:

RefEq	$x = x$
SymEq	$(x = y) \supset (y = x)$
TransEq	$[(x = y) \wedge (y = z)] \supset (x = z)$
LLCI	$(x = y) \supset [C(x) \equiv C(y)]$
LLPr	$[(x_1 = y_1) \wedge (x_2 = y_2)] \supset [P(\vec{x}) \equiv P(\vec{y})]$
LLMP	$[(x_1 = y_1) \wedge (x_2 = y_2) \wedge (x_3 = y_3)] \supset [P.n(\vec{x}) \equiv P.n(\vec{y})]$

The last three sentences capture Leibnitz Law for the three kinds of predicate symbols in $\mathcal{L}_{\mathcal{C}}$ and, unlike the previous three sentences, are axiom schemas.

A FO theory of the CRM

By applying the rules above to the specification of classes and properties, we obtain a set of axioms that make up the $\mathcal{C}$ first-order theory.

Some pleasant consequences (based on standard logical notions):

- We can talk of the $\mathcal{C}$ language, as the set of predicate symbols that occur in the axioms
- We can *validate* the CRM:
  - we can *prove* that the $\mathcal{C}$ axioms are consistent (hopefully :-))
  - as well as any other property we think it's there
- We can *defend* the CRM: we can challenge the CRM's detractors to prove what they say
- We can *compare* the CRM, by formally testing whether a language is equivalent to, or less/more powerful than the CRM
- We can *check* whether an implementation is sound and complete with respect to the $\mathcal{C}$

Nothing particularly surprising, but a firm ground to start building.

But all this can be done ONLY with paper and pencil. Can we do something automatically? Let's take a retrospective look to the axiom schemes that we have used for capturing the CRM:

SubC	$A(x) \supset B(x)$
Dom	$P(x,y) \supset D_P(x)$
Ran	$P(x,y) \supset R_P(y)$
SubP	$P(x,y) \supset Q(x,y)$
SymP	$P(x,y) \supset P(y,x)$
TransP	$[P(x,y) \wedge P(y,z)] \supset P(x,z)$
MetaP	$P.n(x,y,z) \supset [P(x,y) \wedge E55(z)]$
WSCut	$[P_1(x,z_1) \wedge \ldots \wedge P_n(z_{n-1},y)] \supset P(x,y)$
FuncP	$[P(x,y) \wedge P(x,y')] \supset (y = y')$
FuncIP	$[P(x,y) \wedge P(x',y)] \supset (x = x')$
DisC	$A(x) \supset \neg B(x)$
AMetaP	$P.n(x,y,z) \supset \neg P.n(y,x,z)$
WICut	$P(x,y) \supset (\exists z_1 \ldots z_{n-1})[[P_1(x,z_1) \wedge \ldots \wedge P_n(z_{n-1},y)]$
TotP	$A(x) \supset (\exists y)P(x,y)$
TotIP	$B(x) \supset (\exists y)P(y,x)$
RefEq	$x = x$
SymEq	$(x = y) \supset (y = x)$
TransEq	$[(x = y) \wedge (y = z)] \supset (x = z)$
LLCl	$(x = y) \supset [C(x) \equiv C(y)]$
LLPr	$[(x_1 = y_1) \wedge (x_2 = y_2)] \supset [P(\vec{x}) \equiv P(\vec{y})]$
LLMP	$[(x_1 = y_1) \wedge (x_2 = y_2) \wedge (x_3 = y_3)] \supset [P.n(\vec{x}) \equiv P.n(\vec{y})]$

The structure of the $\mathcal{C}$ axioms is very close to that of definite program clauses (DPCs):

$$\forall x_1 \ldots x_n (B_1 \wedge \ldots \wedge B_k) \supset A$$

where each of the $A, B_1, \ldots, B_k$ is an atom.

This closeness suggests that a datalog implementation of $\mathcal{C}$ may be possible, as long as we can deal with:

- the *negation* in the axioms schemas in the middle group; and
- the *existential quantication* in the axioms schemas in the bottom group.

Removing negation

In order to remove negation, we introduce *complementary* classes and *complementary* meta-properties and state the disjointness between a class or a meta-property and its complement by using a special axiom leading to a contradiction.

For each unary predicate symbol B in $\mathcal{L}_{\mathcal{C}}$, we introduce a new unary predicate symbol $\overline{B}$ and replace the DisC axiom schema $A(x) \supset \neg B(x)$ by

$A(x) \supset \overline{B}(x)$
$B(x) \wedge \overline{B}(x) \supset \bot$

where $\bot$ is a contradiction; *e.g.*, replace $E2(x) \supset \neg E77(x)$ by:

$E2(x) \supset \overline{E77}(x)$
$E77(x) \wedge \overline{E77}(x) \supset \bot$

We will see soon how $\bot$ can be expressed.

Likewise, for each ternary predicate symbol $P.n$ in $\mathcal{L}_{\mathcal{C}}$, we introduce a new ternary predicate symbol $\overline{P.n}$ and replace each AMetaP axiom schema $P.n(x, y, z) \supset \neg P.n(y, x, z)$ by:

$P.n(x, y, z) \supset \overline{P.n}(y, x, z)$
$P.n(x, y, z) \wedge \overline{P.n}(x, y, z) \supset \bot$

A new set of axioms is obrained from $\mathcal{T}_{\mathcal{C}}$, which we denote as $\mathcal{T}_{\mathcal{C}}^{+}$.

$\mathcal{T}_{\mathcal{C}}^{+}$ is expressed in a new language that has no negation and a new set of predicate symbols.

Intuitively, $\mathcal{T}_{\mathcal{C}}$ and $\mathcal{T}_{\mathcal{C}}^{+}$ are equivalent sets of axioms, since they state the same constraints in different ways.

Formally, we have proved the equivalence.

Removing existential quantification

Skolemization: replacing each existential variable with a new constant, *i.e.*, a constant that does not occur in the KB.

This amounts to replace the axiom schemas of the third group by:

WICut	$[P(x,y) \wedge S_i(h_1,\ldots,h_{n-1})] \supset [P_1(x,h_1) \wedge \ldots \wedge P_n(h_{n-1},y)]$
TotP	$[A(x) \wedge T_i(h)] \supset P(x,h)$
TotIP	$[B(x) \wedge V_i(h)] \supset P(h,x)$

where $h, h_1, \ldots, h_{n-1}$ are new constants. Note that we use:

- one of the S_i for each instantiation of the WICut schema and for each strong shortcut
- one of the T_i for each instantiation of the TotP schema
- one of the V_i for each instantiation of the ToTIP schema.

S_i, T_i and V_i play the role of generators of new tuples of constants.

By instanting these schemas in place of the replaced ones, we obtain a new set of axioms that we denote as $\mathcal{T}_{\mathcal{C}}^{\star}$.

Re-writing the co-reference axioms

RefEq is not a DPC, but it states a mathematical property of co-reference that does not have any computational import, so we drop it.

SymEq and TransEq are clearly DPCs.

Each one of the three Leibnitz Laws can be restated into an equivalent DPC. For LLCI:

$$\begin{array}{ll} \text{LLCI1} & [(x = y) \wedge C(x)] \supset C(y) \\ \text{LLCI2} & [(x = y) \wedge C(y)] \supset C(x) \end{array}$$

Now, LLCI2 can be derived by SymEq and LLCI1, so it can be dispensed with. We are therefore left with LLCI1. So we replace LLCI, LLPr and LLMP by:

$$\begin{array}{ll} \text{LLCI1} & [(x = y) \wedge C(x)] \supset C(y) \\ \text{LLPr1} & [(x_1 = y_1) \wedge (x_2 = y_2) \wedge P(\vec{x})] \supset P(\vec{y}) \\ \text{LLMP1} & [(x_1 = y_1) \wedge (x_2 = y_2) \wedge (x_3 = y_3) \wedge P.n(\vec{x})] \supset P.n(\vec{y}) \end{array}$$

Knowledge Bases

Our job would be finished here, because we have reached an axiomatization of the CRM, and even one that can be computed with.

But why stop here?

After all, the CRM is created to be used in information systems.

So, we go on defining what a CRM knowledge base (KB) could be.

To begin with, a $\mathcal{C}$ KB is a set of sentences of the $\mathcal{C}$ language, including the axioms.

But what kind of sentences do we expect to find in a CRM KB?

Besides the axioms, we expect KB to hold a description of the state of the world, including the individuals in the domain of discourse.

But what kinds of individuals do we expect to find in a CRM KB?

Individuals and the CRM

The individuals in the domain of the CRM are:

- CRM-entities, which include appellations, and
- primitive values.

The CRM models these individuals as objects, identified by object identifiers. We note that the CRM object identifiers have the following features:

1. at any time, each identifier denotes only one object;
2. at any time, no two identifiers denote the same object;
3. each identifier denotes the same object throughout the whole KB lifetime.

The logical counterpart of objects identifiers are constant symbols, which satisfy the first feature above, because in any interepretation each of them denotes one individual of the domain.

However, constant symbols:

- do not satisfy the second feature because nothing prevents two constant symbols to co-refer in a specific model of the KB, that is to denote the same individual;
- do not satisfy the third feature above either, because a KB may have, at a given point during its lifetime, more than one model; and nothing prevents the same constant symbol to denote different individuals in two such models.

The problem posed by the second feature may be solved by introducing the *unique name axiom*. The problem posed by the last feature, however, remains.

Fortunately, a solution to this problem is presented by Levesque and Lakemeyer, in the form of a convention that consists in introducing a special category of symbols, called *standard names* and given by n_1, n_2, ...

Standard names

Standard names behave exactly as the CRM identifiers, they are one-to-one with the individuals in the domain of discourse *in all possible worlds.*

They denote known individuals.

So, to represent that the city appelled as *Pisa* is known, we use a standard name for it, say n_7. And to state that the city called *Pisa* is in fact the object that we know, we state co-reference:

$Pisa = n_7$

On the contrary, just knowing *Pisa*= *Vituperio_delle_genti* does not amount to have identified neither *Pisa* nor *Vituperio_delle_genti.*

We therefore use standard names to represent individuals in the domain of discourse of the CRM.

$n_i = n_j$ is a contradiction if $i \neq j$, so we can use it in place of $\perp$.

Do we need any other type of individuals?

Constants

In the KB lifetime, the user may need to represent knowledge about individuals whose identity is *presently* uncertain, in the sense that:

- it is not known whether these individuals have already been assigned a standard name in the KB, or
- which standard name that would be.

This uncertainty may be resolved at a later time, either by discovering the standard name that is used for these individuals, or by ascertaining that no standard name has yet been assigned to them.

But it is required that the KB be able to hold knowledge about these individuals until their identity is cleared and a standard name is available for them.

FO logic offers constant symbols for naming individuals whose identity may vary from interpretation to interpretation, therefore we also include constant symbols in our language.

Now that we have a language also for individuals, we turn to the contents of a CRM KBs.

First of all, a KB must include the axioms derived from the axiom schemas that provide a representation of the meaning of the terms in the $\mathcal{L}_{\mathcal{C}}$ vocabulary, *e.g.*:

$$E4(x) \supset E5(x)$$

Likewise, to capture that P4 has time-span is a functional property, we instantiate the FuncP axiom scheme and obtain the $\mathcal{C}$ axiom:

$$P4(x, y) \wedge P4(x, y') \supset (y = y')$$

and so on. Without these axioms, collectively called *ontological* knowledge, we cannot be sure that the KB exhibits the intended behavior, for instance when querying it.

Second, a KB must contain sentences representing the state of the world in the domain of discourse. These sentences form *domain* knowledge.

We envisage two kinds of domain knowledge

1. *instantiation literals*, representing the instantiation of classes and properties, *e.g.* $E81(n_4)$, $P1(Tom, \text{“}Tom'')$ or $\overline{P12}(n_{15}, bob)$
2. *co-reference literals*, representing the referential relationships between the constants and the standard names.

Co-reference literals come in two sorts:

- Positive co-reference literals:
  - $(n = a)$ asserting co-reference of a constant a and of a standard name n; this is a strong piece of knowledge, allowing to identify the individual named a.
  - $(a = b)$ asserting co-reference of two constants a and b; this atom does not give an equally vivid knowledge as the previous one, yet it allows to reduce the uncertainty in the KB by establishing co-reference of two constants.
- Negative co-reference literals:
  - $(n \neq a)$ asserting that the individual named a is not identified by n;
  - $(a \neq b)$ asserting non-coreference between constants.

CRM KB defined

We can now define formally a CRM KB.

A $\mathcal{C}$ KB $\mathcal{K}$ as a pair $\mathcal{K} = (\mathcal{T}_{\mathcal{C}}, \mathcal{A})$, where:

1. $\mathcal{T}_{\mathcal{C}}$, the *TBox* of $\mathcal{K}$, includes the CRM axioms, obtained by instantiating the axiom schemas introduced in the previous Section;
2. A, the *ABox* of $\mathcal{K}$, is a finite, possibly empty set of instantiation and co-reference literals, as discussed above.

And now . . .

- we can talk about a model of a KB, as any interpretation of the language that satisfies all the axioms and the sentences in the KB
- we can talk about the *consistency* of a KB
- we can talk about *reasoning* in CRM because we have an inference relation $KB \models \alpha$

We can define formally the *interaction* with a KB, thereby separating *once for all* the theory from its implementation:

- $\textsc{Tell}$(KB,s), where s is a sentence of our language
- we have a query language: the set of open formulas of the language
  - *e.g.*, $E55(x) \wedge \exists y[P27(x, y) \vee \neg P72(x, y)]$
- $\textsc{Ask}$(KB,α), wehre α is a query
- we can use the inference relation to define the answer to a query

Let's do it

The axioms in the set $\mathcal{T}_{\mathcal{C}}^{\star}$ are DPCs that can be expressed as datalog rules forming a datalog program that we call $\mathcal{P}_{\mathcal{C}}$.

$\mathcal{P}_{\mathcal{C}}$ is derived from *rule schemas*, in the same way the actual axioms of $\mathcal{C}$ are derived from the first-order axioms schemas.

For instance, rule scheme SubC gives raise to the actual $\mathcal{P}_{\mathcal{C}}$ rule:

$$E4(x) \leftarrow E5(x)$$

based on the $\mathcal{C}$ axiom $(\forall x)E5(x) \supset E4(x)$.

Likewise, rule scheme FuncP gives raise to the actual $\mathcal{P}_{\mathcal{C}}$ rule:

$$(y = y') \leftarrow P4(x, y), P4(x, y')$$

We can now apply the program $\mathcal{P}_{\mathcal{C}}$ to the literals in the ABox and derive all *positive* implicit literals in our KB.

For instance, if we have the literal $E5(n)$ in our KB, we derive $E4(n)$ from it, by applying the rule

$$E4(x) \leftarrow E5(x)$$

However, the users of a CRM KB are also interested in the implicit *negative* knowledge, and it is not difficult to see that $\mathcal{P}_{\mathcal{C}}$ is not sufficient to capture all such knowledge.

Let's see what kind of inference we want to be able to make, returning for a moment to first-order.

Suppose a KB includes $\neg E4(n_2)$ in its ABox. Together with rule $E4(x) \leftarrow E5(x)$, this implies $\neg E5(n_2)$.

A sound and complete first-order inference system, such as one based on resolution, would indeed derive $\neg E5(n_2)$.

But there is no way to obtain $\neg E5(n_2)$ (or its corresponding complement $\overline{E5}(n_2)$) from $\mathcal{P}_\mathcal{C}$. This is not surprising, since datalog aims at deriving positive atoms that can be seen as elements of the interpretation of a program.

We need to add more rules to those of $\mathcal{P}_\mathcal{C}$ in order to be able to derive the implicit negated atoms by means of a datalog-based inference system. In particular, we need to add the rule

$$\overline{E5}(x) \leftarrow \overline{E4}(x)$$

to our TBox in order to be able to derive $\overline{E5}(n_2)$.

However, we must be aware that not *all* negative knowledge is equally desirable.

Let's consider a scholar who is developing a KB K powered by a sound and complete inference engine about whether or not Dante Alighieri (standard name D) was present at the event (standard name b) of the birth of Francesco Petrarca.

This piece of knowledge can be represented in the CRM by using property P12 occurred in the presence of (was present at), linking an event (instance of $E5$) to a persistent item (instance of $E77$) that was present at the event.

Now, our scholar enters $E21(D)$, $E63(b)$ and $\neg P12(b, D)$ ("Dante was not present at the birth of Petrarca") in the ABox.

Now the scholar checks K out and finds that it contains the three assertions that he has inserted, but *in addition* it contains also the assertion $\neg P12(D, b)$ ("Dante did not occur in the presence of the birth of Petrarca") of no meaning and no use.

Why?

Well ...

- $E21(D)$ implies $E77(D)$ which implies $\neg E2(D)$ by disjointness
- if D is not an event, then D is not in the domain of $P12$, hence it cannot be true that $P12(D, b)$ (no matter what b is) and therefore its negation $\neg P12(D, b)$ is true.

In fact, $\neg P12(x, y)$ is true of all the x that are not events, or of all the y that are not persistent items (as b in the last example), or both.

And the same applies to every other property.

In other words, if the ABox of a KB contains all negated atoms that follow from the explicit knowledge, we may end up with a very large set of true but *totally irrelevant* facts.

The semantics of negation makes this unpleasant fact unavoidable.

However, in our language we do not use negation directly, but we simulate it through complements. This gives us the possibility of avoiding undesired negative knowlede in our KB.

Relevance criterion for negative knowledge: we accept negated property instantiation atoms $\overline{P}(i,j)$ or meta-property instantiation atoms $\overline{P.n}(i,j,t)$ in the ABox of a KB **only if** i is an instance of the domain of P and j is an instance of the range of P.

In other words, we consider relevant only negated atoms that involve instances of the proper classes, therefore sentences like $\overline{P12}(D,b)$ would generate an inconsistency if inserted into the KB.

For co-reference, the above criterion translates quite naturally as follows: we accept negated co-reference instantiation atoms ($i \neq j$) in the ABox of a KB only if i and j are instances of same class.

In order to implement this criterion, we introduce the following axiom schemas:

RDom	$\overline{P}(x, y) \supset D_P(x)$
RRan	$\overline{P}(x, y) \supset R_P(y)$
RMetaP	$\overline{P.x}(x, y, z) \supset \overline{P}(x, y)$
RCo	$(x \neq y) \wedge C(x) \supset C(y)$

Happily, all these axioms are DPCs.

Notice that the same axioms would create undesired results, if expressed through negation. For instance, RDom would be expressed as $\neg P(x, y) \supset D_P(x)$. On the other hand, Dom is $P(x, y) \supset D_P(x)$. Considered together, these axioms imply $(\forall x) D_P(x)$, a definitely undesired outcome.

The datalog program $\overline{\mathcal{P}_{\mathcal{C}}}$ is so obtained by introducing *complementary* rule schemas.

- the complementary rule schemas corresponding to Dom, Ran and the second MetaP would violate the relevance criterion stated above, and threfore they are subsistuted by rule schemas encoding RDom, RRan and RMetaP, respectively;
- WICut, TotP and TotIP do not have any corresponding rule schema, because the body of each such rule would contain a negated instantiation atom in which a new name $h, h_1, \ldots, h_{n-1}$ occurs. Such atoms can never be true, because new names are by definition used only in positive instantiation atoms;
- CIUn, CIBin, CITer and CICo are tautologies used solely for detecting inconsistencies and have therefore no corresponding complementary rule schema.

Now we can compute all implicit literals.

Computing implicit literals

The ABox $\mathcal{A}$ of a KB can be viewed as an instance of the symbols in the datalog program $\overline{\mathcal{P}_\mathcal{C}}$.

By applying $\overline{\mathcal{P}_\mathcal{C}}$ to $\mathcal{A}$, the minimal model $\mathcal{A}^\star$ of $\overline{\mathcal{P}_\mathcal{C}}$ is obtained in an efficient manner, that is using limited space and time resources.

$\mathcal{A}^\star$ includes the following types of atoms:

- The explicit instantiation atoms in $\mathcal{A}$ and those derived from $\mathcal{A}$ by applying the rules in $\overline{\mathcal{P}_\mathcal{C}}$. For instance, assuming that $E5(1)$ is an instantion atom in $\mathcal{A}$, the atom $E4(1)$ is in $\mathcal{A}^\star$.
- The explicit co-reference atoms in $\mathcal{A}$ and those derived by applying the rules in $\overline{\mathcal{P}_\mathcal{C}}$. For instance, assuming that $P4(1,2)$ and $P4(1,a)$ are instantiaton atoms in $\mathcal{A}$, due to rule for the functionality of $P4$, the atom $(2 = a)$ is in $\mathcal{A}^\star$.
- Inconsistent atoms of the form $(n_1 = n_2)$ with n_1 different from n_2.

Inconsistent atoms may result from two different types of derivation paths:

- from the application of one of the rules having the sentence in their heads, *i.e.*, either rule CIEq or an instance of one of CIUn, CIBin, CITer. In this case an individual is instance of a predicate symbol and of its complement, which is an obvious inconsistency;
- from the application of one of the rules having a co-reference atom in their heads, *i.e.*, either rules SymEq, TransEq or an instance of one of FuncP, FuncIP. In this case two different standard names, and possibly some constants, have been associated to the same individual through a functional property, and this too is an obvious inconsistency.

If an inconsistent atom is in $\mathcal{A}^\star$, then the application of $\overline{\mathcal{P}_\mathcal{C}}$ to the KB reveals an inconsistency in the KB.

Skolemization preserves satisfiability, so the algorithm just outlined offers a sound and complete method for the checking the consistency of any $\mathcal{L}_\mathcal{C}$ KB as defined above.

Otherwise, the the application of $\mathcal{P}_{\mathcal{C}}$ transforms a KB $\mathcal{K} = (\mathcal{T}_{\mathcal{C}}, \mathcal{A})$, into a new KB $\mathcal{K}^{\star} = (\mathcal{T}_{\mathcal{C}}, \mathcal{A}^{\star})$ that is an expansion of $\mathcal{K}$, the *closure* of $\mathcal{K}$, including all implicit literals in $\mathcal{K}$.

The closure of a KB is a natural candidate to compute the answers to the queries stated against the KB.

Conclusions

We have:

1. a first-order expression of the CRM, for documentation and analysis purposes
2. a definition of a CRM KB, able to handle elementary positive and negative knowledge, whose consistency can be checked in an efficient manner (with some adjustment on *unknown* individuals).
   - If all this is sound, we have the beginning of a possible implementation.

What's missing:

- Querying: we can use all the power of datalog to write recursive queries, which turn out very useful on graphs.
  - But these are just the simple queries.
  - We want the difficult ones :)

Thank you!

We want to see the hard ones . . .