A first-order logic expression of the CIDOC Conceptual Reference Model

Carlo Meghini and Martin Doerr

1 Introduction

The CIDOC Conceptual Reference Model[1] (hereafter CRM) is a well-known conceptual modelling language for documenting cultural heritage artifacts, with a special attention to museum objects. CRM is an ISO standard since 2006 (ISO21127:2006).

The CRM is specified in a semantic data modelling style and relies on consolidated notions for the representation of knowledge such as classes, properties, IsA hierachies, domain and range constraints and cardinality restrictions. These notions are considered sufficiently clear and unambiguous both in the documentation and in the standardization communities. The CRM specification has been used several times for implementing the model, and some of these implementations support large knoweldge bases, such as the one at the British Museum. The status of the CRM as a data definition language is therefore quite solid, from a practical point of view.

However, the CRM still lacks a formal specification of its semantical and inferential apparatus. This lack makes it difficult to clearly define fundamental operations on a CRM knowledge base, such as querying or consistency checking, while preventing any investigation on the computational properties of the language.

The lack of formalization of semantic data models has been one of the two main motivations for the development of Description Logics [2], the other motivation being computational amenability. Description Logics are the theoretical counterpart of the Ontology Web Language (OWL) [4], a distinguished memeber of the Semantic web suite. This suite includes also the Resource Description Framework (RDF), a rather basic knowledge representation language endowed with a model-theoretic semantics [5]. RDF offers a vocabulary for semantic modelling known as RDF Schema [3]. It would seem natural, then, to resort to one of these languages to express the inferential apparatus of the CRM. There are several reasons why we will not follow this route.

First, both RDF and OWL have a syntax that is closer to that of implementation language rather than to a specification language; for instance, they use International Resource Identifiers as non-logical symbols, which are hard to read and unnecessary for our purposes. Second, and more important, our study will show that none of these languaes has the expressive power required by CRM: RDF Schema does not allow to capture property quantification, while strong shortcuts are not expressible in OWL. Third, it turns out that the treatment of individuals at the basis of the CRM cannot be captured even by standard first-order logic, let alone Description Logics, which are contractions of standard first-order logic.

For these reasons, we have chosen to express CRM as an instance of the first-order logic $\mathcal{L}$, defined in [6]. This logic extends the classical predicate calculus with a referential apparatus based on standard names that adequately captures the treatment of individuals of the CRM. Happily, this choice does not prevent to achieve computational tractability.

The logical formalization is expected to bring several advantages to the CRM, from a scientific point of view. In particular, the first-order logic expression of the model may serve as a better communication medium with other researchers, allowing for a better understanding of the CRM that can serve several purposes:

[1] Version 5.1.2 [11]. The current CRM specification is being used by the ISO working group ISO/TC46/SC4/WG9 as community draft for the regular revision of ISO21127 due after 5 years and expected for 2015.

– appreciating the CRM's underlying ontological decisions;

– enriching the CRM with knowledge representation mechanisms that are well understood and as such transferrable with no special effort, such as n-ary relations, tractable forms of negation, or simple role constructors, just to mention a few;

– extending the CRM to new domains or to aspects of knowledge representation that are not dealt with by the current version.

Finally, logic provides a language for expressing constraints that go beyond the representation machinery of object-oriented modelling, and which may turn out to be useful in more refined definitions of the CRM. Logic also provides two useful notions, such as consistency and implication; consistency is important for the integrity of a CRM knowledge base, while implication is important for extracting knowledge from a CRM knowledge base.

From a knowledge representation point of view, the present study can be viewed as an attempt at defining the *knoledge level* of a CRM knowledge base [9].

The paper is structured as follows: we will start by examining the expressive requirements of the CRM (Section 2). Based on the results of this examination, we will define the first-order language $\mathcal{L}_\mathcal{C}$ for the logical expression of the CRM (Section 3). Next, we will introduce the axioms that capture the CRM ontology. We will then define the notion of knowledge base and query, thus concluding the logical formulation of the CRM (Section 5). The remaining part of the paper (Section 6) is devoted to design a datalog-based implementation of a knowledge base. Section **??** concludes.

2 Expressive Requirements of the CRM

In this Section, we examine the basic principles of the CRM in order to obtain a clear understanding of its expressive requirements. We start by outlining a general correspondence between semantic data models and logical languages. Subsequently, we will consider the specific features of the CRM. As a notational convention, CRM terms are written in Sans Serif, *e.g.* E1 CRM Entity, while first-order symbols are written in italics, *e.g. E1 CRM Entity.*

2.1 From the CRM to logic: first principles

The correspondence between semantic data modeling and first-order logic has been defined in detail in [10] and will be used thorughout the paper for the logical capturing of the representational primitives of the CRM and of some of its constraints. For the remaining constraints, we extend the work in [10] by presenting the corresponding axioms.

The CRM specification is given in terms of the primitives of semantic data modelling. As such, it uses:

– *objects* to represent the individuals in the domain of discourse in a one-to-one fashion, such as the object pisa standing for the city of Pisa; in the CRM, objects are called "items" in order not to confuse them with physical objects; we will stick to "objects" for alignment with the standard practice in semantic data modelling;

– *classes* to represent general notions in the domain of discourse, such as for instance the CRM class E53 Place to represent the notion of a place;

– *properties* to represent the binary relations that link to each other the individuals in the domain of discourse, such as for instance the CRM property P89 falls within linking a place to the region where it belongs.

Ontological knowledge is expressed in semantic data models by means of various kinds of constraints, such as IsA hierarchies. We will examine in detail the CRM constraints in Section 4. Factual knowledge, on the other hand, is expressed in semantic data bases by means of *instantiation.* In particular,

– *Class instantiation* makes an object an instance of a class, thereby expressing the knowledge that the corresponding individual belongs to the corresponding notion; for instance, in order to express that the city of Pisa is a place, the object pisa is made an instance of class E53 Place.

– *Property instantiation* makes a pair of objects an instance of a property, thereby expressing the knowledge that the corresponding relationship holds between the corresponding individuals; for instance, in order to express that Pisa is in Italy, the CRM property P89 falls within is instantiated with the pair of objects (pisa, italy).

In contrast to semantic data modelling, first-order logic-based knowledge representation relies on a language for formally encoding knowledge in *sentences.* This language can be directly put in correspondence with the elements of semantic data models as follows:

– objects are named by the *constant symbols* of the language, such as *pisa*;

– classes are named by *unary predicate symbols,* such as *Place*;

– properties are named by *binary predicate symbols,* such as *falls within;*

Ontological knowledge is expressed in logic by means of *logical axioms,* which correspond to the constraints of semantic modelling. Factual knowledge is expressed in logic by means of ground atoms, such as *Place(pisa)* and *falls within(pisa,italy),* which represent the same kind of knowledge expressed by the class and property instances in semantic modelling.

These basic considerations can be used as general principles to establish a correspondence between the CRM and a first-order logical language. However, our task requires to *identify a specific first-order language* $\mathcal{L}_\mathcal{C}$ that is able to capture the intended meaning of the CRM vocabualry. In order to achieve this goal, we will build on the correspondence outlined above and carry out a detailed analysis of the features of the CRM, to the end of determining the specific alphabet and syntax of $\mathcal{L}_\mathcal{C}$ that will be needed to express such features.

2.2 The individuals in the domain of discourse of the CRM

The *Intended Scope* of the CRM is[2] "all information required for the exchange and integration of heterogeneous scientific documentation of museum collections", where "the documentation of collections includes the detailed description of individual items within collections, groups of items and collections as a whole". This amounts to say that in a CRM KB we expect to find statements about individual items within museum collections, other groups of such items and collections themselves. We will refer to these individuals as *CRM-entities.*

Amongst the CRM-entities, a very important role is played by appellations. Appellations are the names that CRM-entities are called, or have been called in their context of existence, and are included in a CRM KB because the use and assignment of appellations are historical facts of great relevance. Indeed, the provenance of museum objects can be verified by tracing appellation use without reference to any other feature. The registration of all previous identifiers of a museum object is a requirement of the CIDOC[3] documentation guidelines. This requirement is common to other domains: the *Union List of Artist Names* is a resource by the Getty registering all names ever in use for artists; similarly, the Getty Thesaurus of Geographic Names registers all names ever in use for places. Since museum objects, artists and places are all in the CRM scope, appellations have been introduced to document the names these resource have had at any time. Technically, appellations are instances of the CRM class E41 Appellation, which has several sub-classes, each capturing a specific name type (*e.g.*, place, time or agent appellations, titles, and others). An appellation is therefore a CRM-entity of a special kind: it is a piece of language, entirely identified by its lexical form. This feature of appellations will be taken into account in the development of the terms of $\mathcal{L}_\mathcal{C}$.

As it turns out, CRM-entities are not the only individuals in the domain of discourse of a CRM KB. There are also individuals that lie outside the intended scope of the CRM but are nevertheless related to the CRM

[2] All quotations in this Section are from Section "Scope of the CIDOC CRM" of [11]

[3] CIDOC is the International Committee for Documentation of the International Council of Museums

entities and as such they must be documented in a CRM KB. Examples of these individuals are numbers, temporal intervals, and geometric regions. In the CRM, these individuals are called *primitive values* and are instances of the CRM class E59 Primitive Value which is the *only* CRM class that is not a sub-class of E1 CRM Entity. In this sense, primitive values make up the complement of the CRM domain proper. However, since a KB needs to refer to primitive values, they are modelled in the CRM as first-class objects, although no knowledge about primitive values is expected to be represented in a CRM KB. Indeed, no CMR property has E59 Primitive Value as domain, while a few classes have it as range. In the present version of the CRM, there are just a few sub-classes of E59 Primitive Value, but these sub-classes may be expected to vary, in particular: the set of these classes will shrink as the CRM is extended and some primitive values become citizens of the CRM, or will enlarge as the CRM is applied to new domains and new primitive values are needed.

In sum, the individuals in the domain of the CRM are: CRM-entities, which include appellations, and primitive values. As pointed out in the previous Section, 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, as pointed out in the previous Section. Indeed, constant symbols satisfy the first feature above, because in any interepretation each of them denotes one individual of the domain. However, they do not satisfy the second nor the third feature. They 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. They 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* [10]. The problem posed by the last feature, however, remains. We conclude that constant symbols do not provide an adequate representation of the intended meaning and use of CRM object identifiers.

Fortunately, a solution to this problem is presented in [6], in the form of a convention that consists in introducing[4] a special category of symbols, called *standard names* and given by n_1, n_2, ... Standard names behave exactly as the CRM object identifiers, that is they are one-to-one with the individuals in the domain of discourse *in all possible worlds.* Technically, this amounts to consider the space of all terms in the language as partitioned into equivalence classes, the equivalence relation being *co-reference:* two terms are in the same equivalence class if and only if they refer to the same individual. The resulting equivalence classes are named by the standard names n_1, n_2, ... and it is established, as a convention, that *a term is identified just in case it is possible to name which equivalence class it belongs to.* So, to know that *Pisa* co-refers with standard name n_7, written *Pisa*=n_7, means by convention to know which individual in the domain *Pisa* is. On the contrary, just knowing that *Pisa*=*Vituperio_delle_genti* does not amount to have identified neither *Pisa* nor *Vituperio_delle_genti.*

We therefore introduce a countably infinite set $D_{\mathcal{C}}$ of standard names that are in one-to-one correspondence with the individuals in the domain of discourse of the CRM. $D_{\mathcal{C}}$ will play both a linguistic and a semantic role.

Notice that the standard names include all appellations one may want to use in a CRM KB. This does not pose any problem, since $D_{\mathcal{C}}$ is a countably infinite set, and can therefore accommodate any finite number of countably infinite subsets. From a more theoretical point of view, some CRM classes, such as E60 Number, have an extension that has the cardinality of real numbers, so a countable domain might seem indaquate to account for such classes. But, as it is well known[5], any consistent theory with an infinite domain has a model at each infinite cardinality, therefore the restriction to a countable domain is not a limitation even from a theoretical point of view.

[4]The text in this paragraph is taken from [6], page 22. We omit direct quotation for readability.

[5]The result is known as the Loewenheim-Skolem theorem.

The last requirement concerning individuals is the necessity of representing 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 current KB and 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. As discussed above, first-order logic offers constant symbols for naming individuals whose identity may vary from interpretation to interpretation, therefore we will also include constant symbols in our language.

We are now ready to introduce the first-order language for capturing the inferential apparatus of the CRM.

3 The language $\mathcal{L}_\mathcal{C}$

$\mathcal{L}_\mathcal{C}$ is derived from the language $\mathcal{L}$ presented in [6]. It includes the sentences that are required in order to axiomatize the CRM while at the same time allowing users to build KBs that realize the documentation purposes reviewed in the previous Section.

3.1 Syntax

As customary in logic, the alphabet of $\mathcal{L}_\mathcal{C}$ includes two kinds of symbols: logical and non-logical symbols.

The *logical symbols* are the symbols whose usage and interpretation are fixed. The logical symbols of $\mathcal{L}_\mathcal{C}$ are:

- countably many variables $x, y, z \ldots$;
- countably many standard names n_1, n_2, ... For readability, we will also use strings of lowercase characters between quotes as standard names for appellations;
- the co-reference symbol $=$ naming the co-reference relation;
- the connectives $\neg$ and $\vee$ and the existential quantifier $\exists$.

The *non-logical symbols* are the domain-dependent symbols. The non-logical symbols of $\mathcal{L}_\mathcal{C}$ are:

- countably many constant symbols, or simply constants, a, b, $\ldots$;
- a unary predicate symbol for each CRM class, given by the unique identifier of the class name; *e.g.*, *E2* is the predicate symbol representing class E2 Temporal Entity;
- a binary predicate symbol Pn for each CRM property, given by the unique identifier of the property name; *e.g.*, *P16* is the predicate symbol representing property P16 used specific object (was used for);
- a 3-place predicate symbol for each CRM meta-property, that is property having a property as domain. The first two places are given to the instance of the domain property, the last place is used for stating the type of that instance. The predicate symbol representing each meta-property is derived in the same way as the predicate symbol representing each property.

The *terms* of $\mathcal{L}_\mathcal{C}$ are constants, variables and standard names.

The *atoms* of $\mathcal{L}_\mathcal{C}$ are expressions of the form $P(t_1, \ldots, t_k)$ where each t_i is a term.

A *ground* atom is an atom $P(t_1, \ldots, t_k)$ where each t_i is a constant or a standard name.

A *primitive* atom is a ground atom $P(n_1, \ldots, n_k)$ whose arguments are all standard names.

A *formula* of $\mathcal{L}_\mathcal{C}$ is one of the following:

- an atom;

– a co-reference formula of the form $(t_1 = t_2)$, where t_1 and t_2 are terms;

– the negation of a formula $\neg\alpha$;

– the disjunction of two formulas $(\alpha \vee \beta)$;

– an existential quantification of the form $\exists x.\alpha$

A *sentence* of $\mathcal{L}_\mathcal{C}$ is a formula each variable of which is bound to one quantifier, *i.e.*, a formula with no free variables. As already pointed out, sentences are the elements of a logical KB, since they are used to express both axioms (ontological knowledge) and ground atoms (factual knowledge).

As customary, we will consider sentences including the universal quantifier $\forall$ and the connectives $\wedge$ ("and"), $\supset$ ("implies") and $\equiv$ ("if and only if") as part of $\mathcal{L}_\mathcal{C}$, obtained as abbreviations of the equivalent sentences using the previously introduced symbols. For instance, the sentence $\forall x.E70(x) \supset E77(x)$, stating that every E70 Thing IsA E77 Persistent Item, is an abbrevation of the sentence $\neg(\exists x.E70(x) \wedge \neg E77(x))$, stating that no thing is not a persistent item and which in turn abbreviates $\neg(\exists x.\neg(\neg E70(x) \vee E77(x)))$ built with the official connectives and quantifier. This last sentence is very hard to read, and this is why abbreviations are introduced.

3.2 Semantics

The semantics of the language defines in a mathematical way the form of reality, that is the *Universe of Discourse* of the language. The mathematical notion of reality is a *world state,* traditionally known as *interpretation.* We prefer to use the former term, to keep with [6]. Moreover, semantics provides rules for establishing the truth or falsity of the sentences of the language in a given world state.

A world state w consists of two elements:

– the *domain of interpretation,* which fixes the set of individuals that are part of the reality. In our case, the domain of interpretation is fixed once for all to be the set of standard names $D_\mathcal{C}$. In other words, following [6] we make standard names both part of the language and part of reality.

– the *interpretation function,* which fixes the denotation of each syntactic construct of the language, and uses that denotation to assign a truth value to every sentence of the language. For convenience, a world state w is equated with its interpretation function, what in our case makes much sense, since the domain of interpretation is for us fixed.

The denotation of standard names and constants is quite obvious:

1. to every standard name n, w assigns itself: $w(n) = n$;
2. to every constant a, w assigns a standard name: $w(a) \in D_\mathcal{C}$.

According to the first point, each standard name denotes itself in every world state. Thanks to this choice a one-to-one correspondence between standard names and domain objects is established, satisfying at once all three requirements on CRM individuals spelled out in Section 2.2. In fact, every occurrence of a standard name in a sentence can be seen as an occurrence of the "real thing", immediately revealing what the sentence is referring to. In contrast, the interpretation of constants is not fixed, so that different interpretations may assign different standard names to the same constant. This behavior respects the intuition that is at the basis of constants: as already pointed out, constants are used whenever the identity of an individual is not certain, which is to say that the individual at hand may be anyone of the known standard names.

The denotation of predicate symbols amounts to say which standard names belong to each class (technically, this is the extension of unary predicate symbols), and which individuals are linked by each property and meta-property (the extension of binary and ternary predicate symbols, respectively). This can be done with a single rule as follows:

3. w assigns to every primitive atom $\alpha = P(n_1, \ldots, n_k)$, either 1 or 0.

We recall that the primitive atoms are the atoms that have only standard names as arguments, and that 1 and 0 in the last definition represent *true* and *false,* respectively. The rule only considers primitive atoms, because w maps every constant to a standard name, so all we need to consider is standard names.

In sum, in each world state w standard names always denote themselves, while the denotation of constant and prodicate symbols varies from world state to world state; this accounts for the fact that there may be several *possible world states* that comply with a knowledge base, due to the incompleteness of the knowledge. Based on this information, a world state w can assign a truth value to each sentence of $\mathcal{L}_\mathcal{C}$ by means of the following rules, which are the same for every world state. As customary, we use the notation $w \models \alpha$ to mean that α is true in world state w. The $\models$ relation is defined on all sentences of $\mathcal{L}_\mathcal{C}$ as follows, for every world state w :

1. for every ground atom $P(t_1, \ldots, t_k)$, $w \models P(t_1, \ldots, t_k)$ iff $w[P(n_1, \ldots, n_k)] = 1$, where $n_i = w[t_i]$ for all $1 \leq i \leq k$. In words, for each argument t_i in the given ground atom, we use the function w to determine the standard name n_i that t_i denotes in w. Once all arguments are reduced to standard names, the given ground atom becomes a primitive atom $P(n_1, \ldots, n_k)$, and we use again the function w to determine whether this primitive atom is true.
2. $w \models (t_1 = t_2)$ iff $w(t_1)$ is the same name as $w(t_2)$. We use the same method as in the previous point to reduce each argument in a co-reference statement to the corresponding standard name. Then, the co-reference statement is true just in case the resulting standard names are the same.
3. $w \models \neg\alpha$ iff it is not the case that $w \models \alpha$. This is the standard definition of truth for negation: the negation of a sentence is true in a world state just in case the sentence is false in the same state, and vice-versa.
4. $w \models \alpha \vee \beta$ iff either $w \models \alpha$ or $w \models \beta$. This is the standard definition of truth for disjunction: the disjunction of two sentences is true in a world state just in case at least one of the two sentences is true in the same state.
5. $w \models \exists x.\alpha$ iff for some standard name n, $w \models \alpha^x_n$, where α^x_n is α with all occurrences of x replaced by occurrences of n. An existentially quantified sentence is true in a world state just in case it is possible to find a standard name n as a replacement of the variable that makes true the sentence resulting from the replacement.

We can now use the classical definitions to define the inference relation of $\mathcal{L}_\mathcal{C}$. A world state w is a *model* of a sentence α just in case α is true in w, *i.e.*, $w \models \alpha$. A world state w is a model of a set of sentences S if it is a model of every sentence in S. A set of sentences S is *consistent* if it has a model. A sentence α *logically follows* from a set of sentences S, $S \models \alpha$, just in case every model of S is also a model of α.

Now that we have a language and a semantics, we can encode the CRM ontology as axioms of the language, and then use the inference relation just defined to establish basic properties of the CRM ontology.

4 The axioms of $\mathcal{C}$

In order to derive the axioms of our CRM theory, in the next two Sections we will examine the definitions of the classes and of the properties of the CRM, capturing (the formalizable part of) these definitions into logical form.

4.1 Class axioms

A definition of a CRM class provides:

– the class name;

– the super-classes and the the sub-classes of the class; we will use only the latter information as the former is redundant;

– a textual definition of the concept the class represents and some examples; this information is given in natural language and will not be considered;

– a list of properties the class is the domain of; this information is also provided in the definition of properties and will be considered in the next Section.

Translation of sub-class statements is obvious: the constraint A *Subclass of* B is captured by the sentence:

$$(\forall x)A(x) \supset B(x) \tag{1}$$

The sentence reads: for each individual x, if x is an A, then it is also a B. To exemplify, the constraint that E70 Thing is a sub-class of E77 Persistent Item is captured by the sentence

$$(\forall x)E70(x) \supset E77(x)$$

which we have already seen previously.

For each subclass constraint in the CRM specification, an axiom of the form (1) goes into $T_{\mathcal{C}}$. For this reason, a sentence like (1) is called an *axiom schema*. For simplicity, from now on we will omit the universal quantifiers in sentences and tacitly understand free variables as universally quantified.

Disjointness constraints capture incompatibility between classes or properties, making inconsistent any KB in which such classes or properties share a common instance. The CRM makes the following class disjointness statements[6]:

– E2 Temporal Entity is disjoint from E77 Persistent Item.

– E18 Physical Thing is disjoint from E28 Conceptual Object.

In order to capture disjointness between classes A and B, we use the axiom schema:

$$A(x) \supset \neg B(x)$$

which reads: for each individual x, if x is an A, then it is not a B. This sentence is weaker than:

$$A(x) \equiv \neg B(x)$$

which reads: for each individual x, x is an A if and only if it is not a B. For every interpretation, the last axiom forces every individual in the domain of the interpretation to be either in the extension of A or in the extension of B. The weaker form, instead, allows individuals to be neither in the extension of A nor in the extension of B, while keeping these extensions disjoint. We note that the weaker form is also the interpretation of class disjointness in OWL [8].

So we have the following two axioms in $T_{\mathcal{C}}$:

$$E2(x) \supset \neg E77(x)$$
$$E18(x) \supset \neg E28(x)$$

Since disjointness axioms propagate down the IsA hierarchy, these two axioms sanction the disjointness of many classes. But of course we do not need to state any of the corresponding axioms, because they logically follow (in the sense defined in Section 3.2) from the instances of (1) and the last two axioms.

[6] pag. xv

4.2 Property axioms

A definition of a CRM property provides:

- the property name;
- domain and range specification;
- the super-properties and the sub-properties of the property; we will use only the latter information as the former is redundant;
- the meta-properties, that is properties having the defined property as domain and a specified range;
- quantification, that is the possible number of occurrences for domain and range class instances for the property;
- a declaration whether the property is symmetric or transitive;
- a textual definition of the concept the property represents and some examples; this information is given in natural language and will not be considered, except if itindicates the shortcut that the property defines.

All these aspects of a property are expressed as shown in Table 1. To illustrate, the domain axiom schema reads: for each pair of individuals x and y, if x and y are linked by P, then x is an instance of class D_P. Throughout the paper, we will use the symbol D_P to denote the class that is the domain of property P.

The range axiom reads in a similar way, it has the same premise and concludes that y is an instance of class R_P. Similarly to domain, we will use the symbol R_P to denote the class that is the range of property P.

The sub-property axiom schema is analogous to the sub-class schema. Simmetry and transitivity axiom schemas are easy to read.

CRM Specification	*Axiom schema*
P has Domain D_P	$P(x,y) \supset D_P(x)$
P has Range R_P	$P(x,y) \supset R_P(y)$
P is a Subproperty of Q	$P(x,y) \supset Q(x,y)$
P is Symmetric	$P(x,y) \supset P(y,x)$
P is Transitive	$[P(x,y) \wedge P(y,z)] \supset P(x,z)$

Table 1: Axiom schemas derived from the specification of CRM property P.

Meta-properties, quantification and shortcuts are dealt with in a separate subsection below.

4.2.1 Meta-Properties

As already mentioned, a meta-property is a property whose domain is a property (hence the name): any instance of a property associates an instance of the domain property with a value which is always an E55 Type. "Their purpose is to simulate a specialization of their parent property through the use of property subtypes declared as instances of E55 Type[7]".

In the CRM definition, meta-properties are specified in the context of their parent properties. The definition indicates the parent property and the type, as follows:

P has Meta-Property P.n: E55 Type

[7] page xvi

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

Table 2: Examples of shortcuts in the CRM

The axiom schema capturing the above definition is:

$$P.n(x, y, z) \supset [P(x, y) \wedge E55(z)]$$

which reads: if the individuals x, y and z are linked by $P.n$, then x and y are linked by P and z is a E55 Type. For example, meta-property P62.1 mode of depiction specializes the property P62 depicts (is depicted by) by indicating the type of the depiction, which is a E55 Type. The instance of the above axiom schema capturing this meta-property is therefore

$$P62.1(x, y, z) \supset [P62(x, y) \wedge E55(z)]$$

Some meta-properties are defined to be asymmetric, as in:

P has Asymmetric Meta-Property P.n: E55 Type

In this case, the following additional axiom schema is required to capture asymmetry:

$$P.n(x, y, z) \supset \neg P.n(y, x, z)$$

4.2.2 Shortcuts

Some properties realize *shortcuts,* in the sense that an instance (a, b) of the property implies, or is implied by a set of other instances involving other properties and classes, and forming a chain from a to b : (a, c_1) $(c_1, c_2) \ldots (c_n, b)$. Table 2 presents some text fragments from the CRM Specifications describing the three types of shortcuts in the CRM.

In general, a shortcut can be seen as a pair (P, S) where P is a property, called the *shortcut property,* and S is a sequence of properties, called the *shortcut path.* The shortcut property of the top shortcut in Table 2 is P2, while its path is ((P41,P42). The CRM specification of a shortcut includes also the class where the shortcut path originates and the classes met along the path. We ignore these classes as they can be derived as the domains and ranges of the properties on the shortcut path.

As Table 2 shows, shortcuts come in three sorts:

- *weak* shortcuts, in which an instance of each of the properties on the shortcut path implies an instance of the shortcut property;
- *inverse weak* shortcuts, that is weak shortcuts on the inverse properties, and therefore in them an instance of the shortcut property implies an instance of each of the properties on the shortcut path ;
- *strong* shortcuts, which are both weak and weak inverse, and therefore establish an equivalence between the shortcut property and the shortcut path.

A weak shortcut with property P and path $(P_1, P_2, \ldots, P_n)$ is captured by the following axiom schema:

$$(\forall x, y)(\exists z_1, \ldots, z_{n-1})[P_1(x, z_1) \wedge P_2(z_1, z_2) \wedge \ldots \wedge P_n(z_{n-1}, y)] \supset P(x, y) \tag{2}$$

which reads: for each x and y, if there exist $z_1, , z_2, \ldots z_{n-1}$ such that: x is linked to z_1 by P_1 and z_1 is linked to z_2 by P_2 and $\ldots$ and z_{n-1} is linked to y by P_n, then x is linked to y by P. It is a known fact in logic that (2) is equivalent to

$$(\forall x, z_1, \ldots, z_{n-1}, y)[P_1(x, z_1) \wedge P_2(z_1, z_2) \wedge \ldots \wedge P_n(z_{n-1}, y)] \supset P(x, y)$$

which we write as

$$[P_1(x, z_1) \wedge P_2(z_1, z_2) \wedge \ldots \wedge P_n(z_{n-1}, y)] \supset P(x, y) \tag{3}$$

following the convention of omitting universal quantifiers. For example, the weak shotcut in the middle of Table 2 yields the following axiom:

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

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

Similarly, a weak inverse shotcut is formalized as the following axiom schema:

$$P(x, y) \supset (\exists z_1 z_2 \ldots z_{n-1})[P_1(x, z_1) \wedge P_2(z_1, z_2) \wedge \ldots \wedge P_n(z_{n-1}, y)] \tag{4}$$

The shortcut shown in the bottom of Table 2 is captured by the following instance of the (4) schema:

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

Finally, a strong shortcut is captured by using both the two axiom schemas (3) and (4) above.

4.2.3 Property quantification

The CRM specification includes constraints on the cardinality of properties, named *quantification* statements, or simply CRM quantifier. For self-containedness, the definitions of the CRM quantifiers are reported in appendix, in Table 5[8].

For the capture in $\mathcal{L}_\mathcal{C}$ of the property quantifiers, we follow a three step approach:

1. the definition of each CRM quantifier is reduced to an equivalent set of simpler statements. This is done in Table 6, also given in the appendix. As a result of this step, we have that each quantification statement can be expressed in terms of two simpler statements, *total property* and *functional property*, that can be applied to the property or to its inverse. Therefore a property or its inverse falls into one of the following cases:
   (a) the property is only total, *i.e.*, defined on every element of its domain, but can take up more than one value;
   (b) the property is only functional, *i.e.*, at most one value is provided for any element of its domain;
   (c) the property is neither total, *i.e.*, some domain elements can miss it, nor functional, *i.e.*, more than one value can be provided for any element of its domain;
   (d) the property is both total and functional, *i.e.*all domain element must have one value for it, and no more than one.
2. each of the simpler statements obtained in the previous step is expressed as an axiom schema, as follows:

[8] pag. xii-xiii

- P is a functional property: $[P(x,y) \wedge P(x,y')] \supset (y = y')$ (for each individual x, y and y', if x is linked by P to y and y', then y and y' are the same individual).
- P is a total property $D_P(x) \supset (\exists y)P(x,y)$ (for each individual x, if x is an D_P, then it is linked by P to some y).
- the inverse of P is a functional property: $[P(x,y) \wedge P(x',y)] \supset (x = x')$ (for each individual x, x' and y, if x and x' are linked by P to y, then x and x' are the same individual).
- the inverse of P is a total property: $R_P(x) \supset (\exists y)P(y,x)$ for each individual x, if x is an A, then some y is linked by P to x).

3. the definition of each CRM quantifier is captured as a set of axiom schemas by conjoining the capture of the simpler statements as shown in the previous point. We show how it is done for two CRM quantifiers:
   - *many to many* (0,n:0,n): no axiom schema is required;
   - *one to one* (1,1:1,1) means that both P and its inverse are total and functional (here A and B are the domain and the range pf property P, respectively). We then have the following axiom schemas:

$$
\begin{aligned}
&D_P(x) \supset (\exists y)P(x,y)\\
&[P(x,y) \wedge P(x,y')] \supset (y = y')\\
&R_P(x) \supset (\exists y)P(y,x)\\
&[P(x,y) \wedge P(x',y)] \supset (x = x')
\end{aligned}
$$

   where the first two axioms capture totality and functionality of P, respectively, and the last two axioms do the same for the inverse of P.

4.3 Co-reference axioms

In order to axiomatize quantification statements we have introduced co-reference, which will also play an important role in ABoxes, as it will be showed later. We have therefore to introduce axioms to capture the basic characteristics of co-reference. Co-reference axioms are identical to equality axioms, which are well-known (see, *e.g.*, [6]):

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 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. We use the notation $\vec{x}$ to denote the variables $x_1, x_2, \ldots, x_n$ in an n-ary term whenever these variables are just placeholders.

4.4 Recap: the axiom schemas capturing the CRM specification

Table 3 presents all the axioms schemas introduced thus far and the axioms for co-reference. Each axiom schema is named after the CRM construct or the co-reference property that it captures, indicated on the third column of the Table. The Table is also horizontally partitioned in four sections that will be used later on.

For convenience, we will denote as $T_\mathcal{C}$ the instances of the axiom schemas in Table 3, obtained by capturing the CRM constructs as indicated in the previous part of this Section. Clearly, $T_\mathcal{C}$ is a formulation of the CRM ontology in the logic $\mathcal{L}_\mathcal{C}$.

Name	Axiom schema	
SubC	$A(x) \supset B(x)$	Subclass
Dom	$P(x,y) \supset D_P(x)$	Property domain
Ran	$P(x,y) \supset R_P(y)$	Property range
SubP	$P(x,y) \supset Q(x,y)$	Subproperty
SymP	$P(x,y) \supset P(y,x)$	Symmetric property
TransP	$[P(x,y) \wedge P(y,z)] \supset P(x,z)$	Transitive property
MetaP	$P.n(x,y,z) \supset [P(x,y) \wedge E55(z)]$	Meta-property
WSCut	$[P_1(x,z_1) \wedge \ldots \wedge P_n(z_{n-1},y)] \supset P(x,y)$	Weak shortcut
FuncP	$[P(x,y) \wedge P(x,y')] \supset (y = y')$	Functional property
FuncIP	$[P(x,y) \wedge P(x',y)] \supset (x = x')$	Functional inverse property
DisC	$A(x) \supset \neg B(x)$	Class disjointness
AMetaP	$P.n(x,y,z) \supset \neg P.n(y,x,z)$	Asymmetric meta-property
WICut	$P(x,y) \supset (\exists z_1 \ldots z_{n-1})[[P_1(x,z_1) \wedge \ldots \wedge P_n(z_{n-1},y)]$	Weak Inverse shortcut
TotP	$A(x) \supset (\exists y)P(x,y)$	Total property
TotIP	$B(x) \supset (\exists y)P(y,x)$	Total inverse property
RefEq	$x = x$	Reflexivity of co-reference
SymEq	$(x = y) \supset (y = x)$	Simmetry of co-reference
TransEq	$[(x = y) \wedge (y = z)] \supset (x = z)$	Transitivity of co-reference
LLCl	$(x = y) \supset [C(x) \equiv C(y)]$	for every class predicate symbol C
LLPr	$[(x_1 = y_1) \wedge (x_2 = y_2)] \supset [P(\vec{x}) \equiv P(\vec{y})]$	for every property predicate symbol P
LLMP	$[(x_1 = y_1) \wedge (x_2 = y_2) \wedge (x_3 = y_3)] \supset [P.n(\vec{x}) \equiv P.n(\vec{y})]$	for every meta-property predicate symbol $P.n$

Table 3: Axiom schemas capturing the CRM constraints and co-reference

5 Knowledge Bases

Generally speaking, a CRM KB is a set of $\mathcal{L}_\mathcal{C}$ sentences that describe some slice of reality. We now turn to the task of defining exactly what kind of sentences we expect to find in a CRM KB.

First of all, a KB must include the axioms that capture the logical relationships between the terms of the $\mathcal{L}_\mathcal{C}$ vocabulary. These axioms are derived from the axiom schemas introduced in the previous Section and recapitulated in Table 3, by replacing the predicate symbols with actual predicate symbols in $\mathcal{L}_\mathcal{C}$, as described in Section 4. For instance, to capture that the CRM class E5 Event is a sub-class of E4 Period, we instantiate the SubC axiom scheme and obtain the $\mathcal{C}$ axiom:

$$E5(x) \supset E4(x) \tag{5}$$

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

$$P4(x, y) \wedge P4(x, y') \supset (y = y') \tag{6}$$

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.

In addition to the ontological knowledge, a KB must also contain sentences representing the state of the world in the domain of discourse. These sentences form *domain* knowledge. Based on the analysis carried out in Section 2, we envisage two kinds of domain knowledge:

Firstly, there are sentences representing the instantiation of classes and properties, as discussed in Section 2.1. In order to obtain a greater expressivity, we model also negative instantiation, asserting negative knowledge, such as that an individual is *not* an instance of a class. We call these sentences *instantiation literals,* as they are positive or negated ground atoms of $\mathcal{L}_\mathcal{C}$, in which both standard names and constants may occur, for the reasons spelled out in Section 2.2. It turns out that in the scholarly world, which is one of the domains addressed by the CRM, negative knowledge is as important as positive knowledge, if not more.

Secondly, there are sentences representing the referential relationships between the constants and the standard names, as discussed in tha last part of Section 2.2. We call these sentences *co-reference literals,* and they come in two sorts:

- Positive co-reference literals, falling into one of the following categories:
  - $(n = a)$ 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.

  We exclude senteces of the form $(n_1 = n_2)$ for obvious reasons: If n_1 and n_2 are different standard names, then the sentece is a clear inconsistency that no user would ever state. On the other hand, if n_1 anf n_2 are the same standard name, then the sentence brings no information and, again, no user could possibly be interested in stating it.
- Negative co-reference literals, falling into one of the following categories:
  - $\neg(n = a)$ asserting that the individual named a is not identified by n;
  - $\neg(a = b)$ asserting non-coreference between constants.

  We exclude senteces of the form $\neg(n_1 = n_2)$ for reasons similar to those given for the exclusion of senteces of the form $(n_1 = n_2)$.

We are now ready to define a CRM KB. Following a standard practice in knowledge representation, a CRM KB holds the ontological and the domain knowledge in separate sets of sentences, known as the TBox and the

ABox of the KB, respectively. While the TBox will be the same in every application, the ABox is expected to vary from application to application.

Technically, we define a $\mathcal{C}$ KB K as a pair $K = (T_{\mathcal{C}}, A)$, where:

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

A CRM world state w is a model of a KB K just in case $w \models \alpha$ for each sentence α in $(T_{\mathcal{C}} \cup A)$. A KB K is consistent in case it has a model. A sentence σ is a logical consequence of a KB K, $K \models \sigma$, iff σ is true in all the models of K, or equivalently iff $(T_{\mathcal{C}} \cup A \cup \{\neg\sigma\})$ is not consistent.

A *query* $q(\vec{x})$ is any open wff of $\mathcal{L}_{\mathcal{C}}$. The *answer* to a query $q(\vec{x})$ against a $\mathcal{C}$ KB K, $ans(q(\vec{x}), K)$ is the set of tuples of standard names that are instances of the query in every model of K :

$$ans(q(\vec{x}), K) = \{\vec{n} \mid K \models q(\vec{n})\}$$

The rest of the paper is devoted to design an implementation of a KB, relying on the basic machinery of datalog.

6 A datalog-based implementation of the CRM

The structure of the $\mathcal{C}$ axioms is very close to that of definite program clauses (DPCs, for short), which are sentences of the form [7]:

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

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, and in the rest of this Section we will show how this can be achieved.

6.1 Re-writing the CRM axioms as DPCs

In this Section, we re-write the $\mathcal{C}$ axioms in the TBox of a KB as equivalent DPCs, by removing negation and existential quantification, and by suitably re-writing the co-reference axioms; the last two transformations imply a loss in equivalence, but one can be afforded, as it will be argued.

6.1.1 Removing negation

Negation in a KB is used to state disjointness between classes (axiom schema DisC), asymmetry of meta-properties (axiom schema AMetaP) and negated instantiation or co-reference atoms. Since it does not appear in the body of any rule, negation can be handled without resorting to the techniques devised in datalog, such as stratification [1]. A much simpler approach is indeed possible, which consists in introducing a new set of predicate symbols, called *complements,* that are one-to-one with the predicate symbols in $\mathcal{L}_{\mathcal{C}}$, and that stand for the negation of the corresponding predicate symbols.

Technically, for every predicate symbol in $\mathcal{L}_{\mathcal{C}}$, we introduce a new predicate symbol called the complement of P. As customary, the complement of the co-reference symbol $=$ will be denoted as $\neq$, while the complement of any other predicate symbol P will be denoted as $\overline{P}$. We then modify the set of axioms $T_{\mathcal{C}}$ in the TBox of a KB as follows:

1. replace any instance of the DisC axiom schema $A(x) \supset \neg B(x)$ in the TBox of the KB by the corresponding instance of the schema:
$$A(x) \supset \overline{B}(x) \tag{7}$$

e.g., axiom $E2(x) \supset \neg E77(x)$ is replaced by

$$E2(x) \supset \overline{E77}(x) \tag{8}$$

2. replace any instance of the AMetaP axiom schema $P.n(x, y, z) \supset \neg P.n(y, x, z)$ in the TBox of the KB by the corresponding instance of the schema:

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

3. for each predicate symbol P in $\mathcal{L}_\mathcal{C}$ other than co-reference, add the axiom

$$P(\vec{x}) \wedge \overline{P}(\vec{x}) \supset (n_1 = n_2) \tag{9}$$

to the TBox of the KB, where n_1 and n_2 are any two distinct standard names, *e.g.*, $E77(x) \wedge \overline{E77}(x) \supset (n_1 = n_2)$

4. for co-reference, add the axiom

$$(x = y) \wedge (x \neq y) \supset (n_1 = n_2) \tag{10}$$

to the TBox of the KB, where n_1 and n_2 are any two distinct standard names.

By so doing, a new set of axioms is obrained from $T_\mathcal{C}$, which we denote as $T_\mathcal{C}^+$. Intuitively, $T_\mathcal{C}$ and $T_\mathcal{C}^+$ are equivalent sets of axioms, since they state the same constraints in different ways. Formally, this is proved by Propositions 1 and 2, given in Appendix.

Finally, we apply the above transformations to remove negation also from the ABox of a KB. To this end, it suffices to replace:

1. any negated class instantiation atom $\neg C(t)$ by $\overline{C}(t)$;
2. any negated property instantiation atom $\neg P(t_1, t_2)$ by $\overline{P}(t_1, t_2)$;
3. any negated meta-property instantiation atom $\neg P.n(t_1, t_2, t_3)$ by $\overline{P.n}(t_1, t_2, t_3)$; and
4. any negated co-reference atom $\neg(t_1 = t_2)$ by $(t_1 \neq t_2)$.

The ABox A of a KB resulting from these replacements contains only ground atoms.

6.1.2 Removing existential quantification

In order to remove existential quantification from $T_\mathcal{C}$, we resort to Skolemization, replacing each existential variable with a new constant, *i.e.*, a constant that does not occur in the KB. Technically, this amounts to introduce a set of new predicate symbols S_i, T_i, and V_i for holding new constants, and use these symbols for replacing the axiom schemas of the bottom group in Table 3 by the following schemas:

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, and one of the V_i for each instantiation of the ToTIP schema. In practice, these symbols play the role of generators of new tuples of constants. By so replacing the above axioms schemas, we obtain a new set of axioms that we denote as $T_\mathcal{C}^\star$. As it is well-known, skolemization preserves satisfiability, therefore $T_\mathcal{C}^\star$ is satisfiable iff $T_\mathcal{C}^+$ is.

6.1.3 Re-writing co-reference axioms

Finally, we deal with the co-reference axioms, showing in the fourth group in Table 3. The RefEq axiom is not a DPC, but it be dispensed with, since it states a mathematical property of co-reference that does not have any computational import: no user will ever be interested in the fact that every term co-refers with itself, nor this fact is going to be used for deriving new knowledge, as it can be checked from the rest of the axioms. SymEq and TransEq are clearly DPCs. Each one of the three Leibnitz Laws can be restated into an equivalent DPC. For brevity, we show how this can be done for LLCl, the other two can be trated in a similar way. LLCl is equivalent to the conjunction of the following DPCs:

$$\begin{array}{ll} \mathsf{LLCl1} & [(x = y) \wedge C(x)] \supset C(y) \\ \mathsf{LLCl2} & [(x = y) \wedge C(y)] \supset C(x) \end{array}$$

It is not difficult to see that LLCl2 can be derived by SymEq and LLCl1, so it can be dispensed with. We are therefore left with LLCl1. Based on these considerations, in the TBox $T^{\star}_{\mathcal{C}}$ we remove axiom RefEq and replace the axiom schemas LLCl, LLPr and LLMP, respectively, by:

$$\begin{array}{ll} \mathsf{LLCl1} & [(x = y) \wedge C(x)] \supset C(y) \\ \mathsf{LLPr1} & [(x_1 = y_1) \wedge (x_2 = y_2) \wedge P(\vec{x})] \supset P(\vec{y}) \\ \mathsf{LLMP1} & [(x_1 = y_1) \wedge (x_2 = y_2) \wedge (x_3 = y_3) \wedge P.n(\vec{x})] \supset P.n(\vec{y}) \end{array}$$

6.1.4 The program P

The axioms in the set $T^{\star}_{\mathcal{C}}$ resulting from the transformations seen so far, are DPCs that can be expressed as datalog rules forming a datalog program that we call $P_{\mathcal{C}}$. The middle column of Table 4 presents the rule schemas from which $P_{\mathcal{C}}$ is derived, in the same way the actual axioms of $\mathcal{C}$ are derived from the axioms schemas in Table 3. In order to highlight the correspondence between the rule schemas of Tables 3 and those in the second column of Tab 4, in the latter Table we have used the same rule schema names as in the former, showed in the left column. For instance, rule scheme SubC gives raise to the actual $P_{\mathcal{C}}$ rule:

$$E4(x) \leftarrow E5(x) \tag{11}$$

based on the $\mathcal{C}$ axiom (5). Likewise, rule scheme FuncP gives raise to the actual $P_{\mathcal{C}}$ rule:

$$(y = y') \leftarrow P4(x, y), P4(x, y') \tag{12}$$

For brevity, $P_{\mathcal{C}}$ is not reported.

However, the users of a CRM KB are also interested in the implicit *negative* knowledge, and it is not difficult to see that $P_{\mathcal{C}}$ is not sufficient to capture all such knowledge. To exemplify, let us consider a KB including rule (11) in its TBox, and the negated instantiation atom $\neg E4(n_2)$ in its ABox. These pieces of knowledge imply $\neg E5(n_2)$. A sound and complete first-order inference system, such as one based on resolution [7] 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 $P_{\mathcal{C}}$. This is not surprising, since datalog aims at deriving positive atoms that can be seen as elements of the interpretation of a program.

It is not difficult to see that we need to add more rules to those of $P_{\mathcal{C}}$ 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.

We will exemplify the last observation by considering a scholar who is developing a KB K establishing whether or not Dante Alighieri (whom he denotes by the 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. So, the domain of $P12$ is $E5$ and its range is $E77$. Further, we assume that the KB is powered by a sound and complete inference engine. Now, our scholar knows that D is an instance of class E21 Person, while b is an instance of class E63 Beginning of Existence, and he records this knowledge by entering the positive instantiation atoms $E21(D)$ and $E63(b)$ in the ABox of K. Next, our scholar finds enough evidence that Dante was *not* present at the birth of Francesco Petrarca and so he inserts into the K's ABox the negated atom $\neg P12(b, D)$ which reads "the birth of Petrarca did not occur in the presence of Dante", or alternatively, "Dante was not present at the birth of Petrarca". 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 $\neg P12(D, b)$. This sentence reads "Dante did not occur in the presence of the birth of Petrarca" and is hardly of any use, in fact it does not even read correctly. So the scholar wonders how came that such a useless, ungrammatical piece of knowledge ended up in his precious KB.

In search for an explation, he further inspects the KB and finds out that the atom $\neg E2(D)$ is also in the ABox. The scholar is able to explain this fact: since $E21$ is a subclass of $E77$ which is disjoint from $E2$, any instance of $E21$ is not an instance of $E2$, therefore $\neg E2(D)$ is implicit in the ABox, and the inference engine of the KB correctly derived it. But then, the scholar reckons, 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. Again, soundness and completeness of the underlying inference engine explain also the presence of this piece of knowledge in the KB. The scholar is now very satisfied to have found the explanation, but then he wonders whether the inference engine of his KB is really useful.

The doubts of the scholar are amply justified. 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 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.

Technically, we set a 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 third column of Table 4 presents the rule schemas required to derive implicit negated instantiation atoms from a KB following the relevance criteria outlined above. We call these schemas the *complementary* rule schemas. They are placed on the same rows of the rules for deriving the corresponding implicit positive atoms, and are the contrapositives of (and as such equivalent to) the original axiom schemes in Table 3, where negated atoms have been replaced by their corresponding complements. Notice that:

– 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;
- ClUn, ClBin, ClTer and ClCo are tautologies used solely for detecting inconsistencies and have therefore no corresponding complementary rule schema.

We denote as $\overline{P_\mathcal{C}}$ the set of rules obtained by instantiating the complementary rule schemas, and as P the datalog program obtained as the union of $P_\mathcal{C}$ and $\overline{P_\mathcal{C}}$, *i.e.*, $P = P_\mathcal{C} \cup \overline{P_\mathcal{C}}$.

We now turn to the last example, in order to show how the rules in P prevent the occurrence of undesired negative knowledge in the KB. Axiom RDom is instantiated on property $P12$ as:

$$\overline{P12}(x, y) \supset E5(x) \tag{13}$$

Atom $E21(D)$ implies $E77(D)$ (via SubC), which implies $\overline{E2}(D)$ (via DisC) which implies $\overline{E5}(D)$ (via the correspondent of SubC). On the other hand, atom $\overline{P12}(D, b)$ and (13) imply $E5(D)$ and so a contradiction is generated.

6.2 Computing implicit atoms

The ABox A of a KB can be viewed as an instance of the symbols in the datalog program $\overline{P_\mathcal{C}}$. By applying $\overline{P_\mathcal{C}}$ to A, the minimal model $A^\star$ of $\overline{P_\mathcal{C}}$ that includes A is obtained in an efficient manner, that is using limited space and time resources. More specifically, $A^\star$ includes the following types of atoms:

- The explicit instantiation atoms in A and those derived from A by applying the rules in $\overline{P_\mathcal{C}}$. For instance, assuming that $E5(1)$ is an instantion atom in A, the atom $E4(1)$ is in $A^\star$, due to rule (11). Likewise, assuming $E2(2)$ is in A, $\overline{E77}(2)$ is in $A^\star$ due to rule (8).
- The explicit co-reference atoms in A and those derived by applying the rules in $\overline{P_\mathcal{C}}$. For instance, assuming that $P4(1, 2)$ and $P4(1, a)$ are instantiaton atoms in A, due to rule 12, the atom $(2 = a)$ is in $A^\star$.
- Inconsistent atoms of the form $(n_1 = n_2)$ with n_1 different from n_2, resulting in $A^\star$ from two different derivation paths:
  - from the application of one of the rules having the sentence in their heads, *i.e.*, either rule ClEq or an instance of one of ClUn, ClBin, ClTer. In this case a common instance is shared by a predicate symbol and its complement, 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.

It suffices to inspect the heads of the rules in $P_\mathcal{C}$ to see that no atom can be in $A^\star$ other than the foregoing ones.

If an atom of the third kind above is in $A^\star$, then the application of $\overline{P_\mathcal{C}}$ to the KB reveals an inconsistency in the KB. As already noted, 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 $P_\mathcal{C}$ transforms a KB $K = (T_\mathcal{C}, A)$, into a new KB $K^\star = (T_\mathcal{C}, A^\star)$ that is an expansion of K, called the *closure* of K, including all implicit atoms in K.

The closure of a KB is a natural candidate to compute the answers to the queries stated against the KB, as it will be shown in the next Section.

SubC	$B(x) \leftarrow A(x)$	$\overline{A}(x) \leftarrow \overline{B}(x)$	
Dom	$D_P(x) \leftarrow P(x,y)$	$D_P(x) \leftarrow \overline{P}(x,y)$	RDom
Ran	$R_P(y) \leftarrow P(x,y)$	$R_P(y) \leftarrow \overline{P}(x,y)$	RRan
SubP	$Q(x,y) \leftarrow P(x,y)$	$\overline{P}(x,y) \leftarrow \overline{Q}(x,y)$	
SymP	$P(y,x) \leftarrow P(x,y)$	$\overline{P}(x,y) \leftarrow \overline{P}(y,x)$	
TransP	$P(x,z) \leftarrow P(x,y), P(y,z)$	$\overline{P}(x,y) \leftarrow \overline{P}(x,z), P(y,z)$	
		$\overline{P}(y,z) \leftarrow \overline{P}(x,z), P(x,y)$	
MetaP	$P(x,y) \leftarrow P.n(x,y,z)$	$\overline{P.n}(x,y,z) \leftarrow \overline{P}(x,y), E55(z)$	
	$E55(z) \leftarrow P.n(x,y,z)$	$\overline{P}(x,y) \leftarrow \overline{P.n}(x,y,z)$	RMetaP
WSCut	$P(x,y) \leftarrow P_1(x,z_1), \ldots, P_n(z_{n-1},y)$	$\overline{P_1}(x,z_1) \leftarrow \overline{P}(x,y), P_2(z_1,z_2), \ldots, P_n(z_{n-1},y)$	
		$\overline{P_2}(z_1,z_2) \leftarrow \overline{P}(x,y), P_1(x,z_1), \ldots, P_n(z_{n-1},y)$	
		$\ldots$	
		$\overline{P_n}(z_{n-1},y) \leftarrow \overline{P}(x,y), P_1(x,z_1), \ldots, P_{n-1}(z_{n-2},z_{n-1})$	
FuncP	$(y = y') \leftarrow P(x,y), P(x,y')$	$\overline{P}(x,y') \leftarrow P(x,y), (y \neq y')$	
		$\overline{P}(x,y) \leftarrow P(x,y'), (y \neq y')$	
FuncIP	$(x = x') \leftarrow P(x,y), P(x',y)$	$\overline{P}(x',y) \leftarrow P(x,y), (x \neq x')$	
		$\overline{P}(x,y) \leftarrow P(x',y), (x \neq x')$	
DisC	$\overline{B}(x) \leftarrow A(x)$	$\overline{A}(x) \leftarrow B(x)$	
AMetaP	$\overline{P.n}(y,x,z) \leftarrow P.n(x,y,z)$	$\overline{P.n}(x,y,z) \leftarrow P.n(y,x,z)$	
WICut	$P_1(x,h_1) \leftarrow P(x,y), S_i(h_1,\ldots,h_{n-1})$		
	$P_2(h_1,h_2) \leftarrow P(x,y), S_i(h_1,\ldots,h_{n-1})$		
	$\ldots$		
	$P_n(h_{n-1},y) \leftarrow P(x,y), S_i(h_1,\ldots,h_{n-1})$		
TotP	$P(x,h) \leftarrow A(x), T_i(h)$		
TotIP	$P(h,x) \leftarrow B(x), V_i(h)$		
SymEq	$(y = x) \leftarrow (x = y)$	$(x \neq y) \leftarrow (y \neq x)$	
TransEq	$(x = z) \leftarrow (x = y), (y = z)$	$(x \neq y) \leftarrow (x \neq z), (y = z)$	
		$(y \neq z) \leftarrow (x \neq z), (x = y)$	
LLCl1	$C(y) \leftarrow (x = y), C(x)$	$(x \neq y) \leftarrow C(x), \overline{C}(y)$	
		$\overline{C}(x) \leftarrow \overline{C}(y), (x = y)$	
LLPr1	$P(y_1,y_2) \leftarrow (x_1 = y_1),$	$(x_1 \neq y_1) \leftarrow (x_2 = y_2), P(x_1,x_2), \overline{P}(y_1,y_2)$	
	$(x_2 = y_2), P(x_1,x_2)$	$(x_2 \neq y_2) \leftarrow (x_1 = y_1), P(x_1,x_2), \overline{P}(y_1,y_2)$	
		$\overline{P}(x_1,x_2) \leftarrow (x_1 = y_1), (x_2 = y_2), \overline{P}(y_1,y_2)$	
LLMP1	$P.n(\vec{y}) \leftarrow (x_1 = y_1), (x_2 = y_2),$	$(x_1 \neq y_1) \leftarrow (x_2 = y_2), (x_3 = y_3), \overline{P.n}(\vec{y})$	
	$(x_3 = y_3), P.n(\vec{x})$	$(x_2 \neq y_2) \leftarrow (x_1 = y_1), (x_3 = y_3), \overline{P.n}(\vec{y})$	
		$(x_3 \neq y_3) \leftarrow (x_1 = y_1), (x_2 = y_2), \overline{P.n}(\vec{y})$	
		$\overline{P.n}(\vec{x}) \leftarrow (x_1 = y_1), (x_2 = y_2), (x_3 = y_3), \overline{P.n}(\vec{y})$	
ClUn	$(n_1 = n_2) \leftarrow C(x), \overline{C}(x)$		
ClBin	$(n_1 = n_2) \leftarrow P(x,y), \overline{P}(x,y)$		
ClTer	$(n_1 = n_2) \leftarrow P.n(x,y,z), \overline{P.n}(x,y,z)$		
ClCo	$(n_1 = n_2) \leftarrow (x = y), (x \neq y)$		
		$C(y) \leftarrow (x \neq y), C(x)$	RCo

Table 4: The rule schemas and the co-reference rules of $P_{\mathcal{C}}$

6.3 Query answering

The datalog program $P_{\mathcal{C}}$ cannot be used to compute answers to queries stated against a $\mathcal{L}_{\mathcal{C}}$ KB, due to the fact that query answering is defined in terms of logical implication and Skolemization does *not* preserve logical implication. On the other hand, we cannot reduce query answering to satisfiability since we would have existential quantifiers in the negation of universally quantifies queries.

A Property Quantification Tables

B Propositions

Proposition 1 *For each model w of $T_{\mathcal{C}}$ there exists a unique extension of w that is a model of $T'_{\mathcal{C}}$.*
Proof: Let w' be as follows:

1. *$w'(n) = w(n)$ for each standard name n;*
2. *$w'(a) = w(a)$ for each constant a;*
3. *$w'(\alpha) = w(\alpha)$ for each primitive atom α in $\mathcal{L}_{\mathcal{C}}$;*
4. *for each primitve atom $\overline{B}(n)$, $w'[\overline{B}(n)] = 0$ if $w[B(n)] = 1$ and $w'[\overline{B}(n)] = 1$ if $w[B(n)] = 0$;*
5. *for each primitve atom $\overline{P.n}(n_1, n_2, n_3)$, $w'[\overline{P.n}(n_1, n_2, n_3)] = 0$ if $w[P.n(n_1, n_2, n_3)] = 1$ and $w'[\overline{P.n}(n_1, n_2, n_3)] = 1$ if $w[P.n(n_1, n_2, n_3)] = 0$.*

w' is clearly unique, and it is an extension of w, since the two world states concide on the symbols in $\mathcal{L}_{\mathcal{C}}$. Now suppose w is a model of $T_{\mathcal{C}}$. Then it satisfies each DisC *axiom $A(x) \supset \neg B(x)$, therefore for each standard name n, if $w[A(n)] = 1$ then $w[B(n)] = 0$. This implies that $w'[A(n)] = 1$, $w'[B(n)] = 0$ and $w'[\overline{B}(n)] = 1$, therefore w' satisfies $A(x) \supset \overline{B}(x)$. In addition, w' does not satisfy $B(n) \wedge \overline{B}(n)$, hence no contradiction is generated by axiom $B(n) \wedge \overline{B}(n) \supset (n_1 = n_2)$. A very similar argument applies to the* AMetaP *axioms, leading to the conclusion that w' is a model of $T'_{\mathcal{C}}$.* □

Based on the last Proposition, we introduce a function $\cdot^+$ that maps every world state w of $\mathcal{L}_{\mathcal{C}}$ into its unique extension w^+ as defined by the previous Proposition.

Next, we prove that the new predicate symbols introduced for capturing negation are indeed doing so.

Given a $\mathcal{C}$ KB $K = (T_{\mathcal{C}}, A_I, A_C)$, let the *extension of* K, K^+, be the triple $K^+ = (T^+_{\mathcal{C}}, A_I, A_C)$, in which the set of axioms $T_{\mathcal{C}}$ is replaced by $T^+_{\mathcal{C}}$, while instantation and co-reference atoms are the same. This means that in an extension, the complementary predicate symbols are *not* used to make assertions.

Proposition 2 *For every $\mathcal{C}$ knowledge base K and terms t, t_1, t_2, and t_3 in $\mathcal{L}_{\mathcal{C}}$,*

1. *$K \models \neg C(t)$ iff $K^+ \models \overline{C}(t)$, for every class predicate symbol C.*
2. *$K \models \neg P.n(t_1, t_2, t_3)$ iff $K^+ \models \overline{P.n}(t_1, t_2, t_3)$, for every meta-property predicate symbols $P.n$.*

Proof: We carry out the proof only for the first part and for standard names, i.e., *$t = n$. The remaining cases are proved in a very similar way.*
($\rightarrow$) If K is inconsistent, then there is no world state w that satisfies it. But then, there is no extension w^+, therefore K^+ is inconsistent too, and the Proposition follows. Now suppose K is consistent, and let w be any model of it. By the previous Proposition, w^+ satisfies $T^+_{\mathcal{C}}$; moreover, w^+ is the same as w on the $\mathcal{L}_{\mathcal{C}}$ symbols, which are the only ones occurring in A_I and A_C. We conclude that w^+ satisfies K^+. By hypothesis, $w \models \neg C(n)$, that is $w[C(n)] = 0$. By definition, $w^+[\overline{C}(n) = 1,]$ therefore $w^+ \models \overline{C}(n)$. The Proposition follows.

many to many (0,n:0,n)	Unconstrained: An individual domain instance and range instance of this property can have zero, one or more instances of this property. In other words, this property is optional and repeatable for its domain and range.
one to many (0,n:0,1)	An individual domain instance of this property can have zero, one or more instances of this property, but an individual range instance cannot be referenced by more than one instance of this property. In other words, this property is optional for its domain and range, but repeatable for its domain only. In some contexts this situation is called a "fan-out".
many to one (0,1:0,n)	An individual domain instance of this property can have zero or one instance of this property, but an individual range instance can be referenced by zero, one or more instances of this property. In other words, this property is optional for its domain and range, but repeatable for its range only. In some contexts this situation is called a "fan-in".
many to many, necessary (1,n:0,n)	An individual domain instance of this property can have one or more instances of this property, but an individual range instance can have zero, one or more instances of this property. In other words, this property is necessary and repeatable for its domain, and optional and repeatable for its range.
one to many, necessary (1,n:0,1)	An individual domain instance of this property can have one or more instances of this property, but an individual range instance cannot be referenced by more than one instance of this property. In other words, this property is necessary and repeatable for its domain, and optional but not repeatable for its range. In some contexts this situation is called a "fan-out".
many to one, necessary (1,1:0,n)	An individual domain instance of this property must have exactly one instance of this property, but an individual range instance can be referenced by zero, one or more instances of this property. In other words, this property is necessary and not repeatable for its domain, and optional and repeatable for its range. In some contexts this situation is called a "fan-in".
one to many, dependent (0,n:1,1)	An individual domain instance of this property can have zero, one or more instances of this property, but an individual range instance must be referenced by exactly one instance of this property. In other words, this property is optional and repeatable for its domain, but necessary and not repeatable for its range. In some contexts this situation is called a "fan-out".
one to many, necessary, dependent (1,n:1,1)	An individual domain instance of this property can have one or more instances of this property, but an individual range instance must be referenced by exactly one instance of this property. In other words, this property is necessary and repeatable for its domain, and necessary but not repeatable for its range. In some contexts this situation is called a "fan-out".
many to one, necessary, dependent (1,1:1,n)	An individual domain instance of this property must have exactly one instance of this property, but an individual range instance can be referenced by one or more instances of this property. In other words, this property is necessary and not repeatable for its domain, and necessary and repeatable for its range. In some contexts this situation is called a "fan-in".
one to one (1,1:1,1)	An individual domain instance and range instance of this property must have exactly one instance of this property. In other words, this property is necessary and not repeatable for its domain and for its range.

Table 5: Property Quantifiers in the CRM specification

many to many (0,n:0,n)	Unconstrained: An individual domain instance and range instance of this property can have zero, one or more instances of this property. $\longrightarrow$ *No axiom is required* (*Nax* for short)
one to many (0,n:0,1)	An individual domain instance of this property can have zero, one or more instances of this property, $\longrightarrow$ *Nax* but an individual range instance cannot be referenced by more than one instance of this property. $\longrightarrow$ *Inverse functional*
many to one (0,1:0,n)	An individual domain instance of this property can have zero or one instance of this property, $\longrightarrow$ *Functional* but an individual range instance can be referenced by zero, one or more instances of this property. $\longrightarrow$ *Nax*
many to many, necessary (1,n:0,n)	An individual domain instance of this property can have one or more (*must have at least one?*) instances of this property, $\longrightarrow$ *Total* but an individual range instance can have zero, one or more instances of this property. $\longrightarrow$ *Nax*
one to many, necessary (1,n:0,1)	An individual domain instance of this property can have one or more (*must have at least one?*) instances of this property, $\longrightarrow$ *Total* but an individual range instance cannot be referenced by more than one instance of this property. $\longrightarrow$ *Inverse functional*
many to one, necessary (1,1:0,n)	An individual domain instance of this property must have exactly one instance of this property, $\longrightarrow$ *Total Functional* but an individual range instance can be referenced by zero, one or more instances of this property. $\longrightarrow$ *Nax*
one to many, dependent (0,n:1,1)	An individual domain instance of this property can have zero, one or more instances of this property, $\longrightarrow$ *Nax* but an individual range instance must be referenced by exactly one instance of this property. $\longrightarrow$ *Inverse Total Functional*
one to many, necessary, dependent (1,n:1,1)	An individual domain instance of this property can have one or more instances of this property, $\longrightarrow$ *Total* but an individual range instance must be referenced by exactly one instance of this property. $\longrightarrow$ *Inverse Total Functional*
many to one, necessary, dependent (1,1:1,n)	An individual domain instance of this property must have exactly one instance of this property, $\longrightarrow$ *Total Functional* but an individual range instance can be referenced by one or more instances of this property. $\longrightarrow$ *Inverse Total*
one to one (1,1:1,1)	An individual domain instance and range instance of this property must have exactly one instance of this property. $\longrightarrow$ *Total Functional* $\longrightarrow$ *Inverse Total Functional*

Table 6: Break-down of the Property Quantifiers definitions

(←) Suppose K^+ is inconsistent, then by the counterpositive of the previous Proposition, also K is inconsistent, and the Proposition follows. Now suppose K^+ is consistent, and let w^+ be any model of it. By construction w is a model of $T_{\mathcal{C}}$, therefore w^+ is also a model of K. By hypothesis, $w^+ \models \overline{C}(n)$, that is $w^+[\overline{C}(n)] = 1$. By definition, $w^+[C(n) = 0,]$ therefore $w[C(n) = 0]$ that is $w \models \neg C(n)$. The Proposition follows.

C Managing co-reference

This algorithm is useless because we compute all consequences of co-reference statements by datalog. Nevertheless, it may be used in case one decides to manage co-reference statements (supposedly many in any realistic KB) in an *ad-hoc* way. Whence the title.

Preparation In the preparation stage, the CCA computes the maximal sets of co-refering terms, based on the co-reference literals in the KB. Let the *co-reference graph* G_K of K be the directed di-graph having an arc (t_1, t_2) iff the atom $(t_1 = t_2)$ is in A_C. Let $G_1, \ldots, G_k$ be the components of G_K. Clearly, $k \geq 1$. By applying the Reflexivity, Symmetry and Transitivity axioms to G_K, a new graph $G_K^\star$ obtains that is the reflexive, symmetric and transitive closure of the original graph. It is not difficult to see that the components of $G_K^\star$ are the reflexive, symmetric and transitive closure of $G_1, \ldots, G_k$, $G_1^\star, \ldots, G_k^\star$, and that each $G_i^\star$ is a clique. For simplicity, we will equate each clique $G_i^\star$ with the set of its nodes. Then, each $G_i^\star$ is a maximal set of co-refering terms.

Checking In the checking stage, the CCA checks whether the KB is consistent by using the cliques produced in the preparation stage. To this end, for each clique $G_i^\star$:

- The algorithm checks whether $G_i^\star$ contains at least two standard names n_1 and n_2. If this is the case, the KB implies an atom of the form $(n_1 = n_2)$, where n_1 is different from n_2. This is a type-1 inconsistency, so the algorithm outputs "yes" and terminates. Otherwise,
- The algorithm checks whether $G_i^\star$ contains a pair of terms t_1 and t_2 such that $(t_1 \neq t_2)$ is in A_C. If this is the case, the KB contains a type-2 inconsistency, where the KB implies the positive side $(t_1 = t_2)$ and the co-reference ABox contains the negative side. Also in this case the algorithm outputs "yes" and terminates.

If all cliques are examined without detecting an inconsistency, the CCA outputs "no" and terminates.

We note that CCA does not use the Leibnitz Law axiom. This is due to the fact that this axiom derives new instantiation atoms and none of these atoms may produce an inconsistency, since negation can only occur in co-reference literals. To illustrate, consider a KB $K = (T_{\mathcal{C}}, A_I, A_C)$ such that $E4(a) \in A_I$ and $(1 = a) \in A_C$. By applying the Leibnitz Law axiom, we derive the new instantiation atom $E4(1)$, but this derivation may not lead to any inconsistency, because in no circumstance they KB may contain or imply a negated instantiation atom. However, we also note that $E4(1)$ brings more knowledge than $E4(a)$, because it contains a standard name and standard names identify individuals, whereas constants just describe them. In this sense, a KB with more atoms of the former kind is preferable to a logically equivalent KB with more atoms of the latter kind.

Based on this consideration, we conclude this Section by presenting an algorithm, which we call the *noise reduction* algorithm (NRA for short), for the transformation of a KB into an equivalent one with the same or a smaller number of constants. The NRA just applies the Leibnitz Law axiom to the input KB, whence its correctness.

C.1 Noise reduction

The NRA algorithm takes as input a consistent KB $K = (T_{\mathcal{C}}, A_I, A_C)$ and the set of cliques $G_1^\star, \ldots, G_k^\star$ as above; the algorithm works in two stages: the *re-writing* and the *reduction* stage.

Re-writing In the re-writing stage, the NRA checks whether a preferable (in the sense described above) KB can be derived from the given one. To this end, for each clique $G_i^\star$:

- The algorithm checks whether $G_i^\star$ contains one standard name n only. In this case, in every model of the KB the constants in the clique denote the same individual denoted by n, so we can make the KB more preferable by applying the Leibnitz Law axiom on the substitution of equals, and replace by n every occurrence of a constant c in the clique, in every sentence α of K. As a result of the replacement, an equivalent sentence α_n^c obtains that is preferable to α since it has a standard name in place of a constant.
- The algorithm checks whether $G_i^\star$ contains no standard name. In this case, we can pick any constant c in the clique and use it as a replacement of all the other constants in the clique, for the same reasons as in the previous case.

The result of the transformation phase is a KB equivalent to the input KB, with a number of constants that is no larger than the number of constants in the input KB.

Reduction In the reduction stage, the NRA reduces the set of co-reference literals in the co-reference ABox of the KB resulting from the previous stage. The basic consideration behind this reduction, is that a co-reference statement containing a constant that does not occur anywhere else in the KB is redundant, in that it brings no information to the KB. Formally, this can be proved by showing that the KB with such literals is equivalent to (*i.e.*, has the same models as) the KB without them. Which are then the redundant co-reference literals?

It is not difficult to see that after the execution of the re-writing stage, only one term for each clique $G_i^\star$ is left in the KB. Now, in every positive co-reference literal only terms from the same clique occur, therefore in every such statement at least one term has been replaced during the re-writing stage. So all positive co-reference literals are redundant.

On the contrary, in a consistent KB every negated co-reference literal relates terms from two different cliques, otherwise a type-2 inconsistency arises. Each of the terms in a negated co-reference statement may be the result of one or more replacements performed in the re-writing stage, yet the two terms belong to different cliques. As such, no negated co-reference literal is redundant.

Based on these considerations, the reduction stage of the NRA removes all positive co-reference literals from the co-reference ABox of the KB produced by the previous stage. After this removal, the NRA terminates.

References

[1] S. Abitebul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN: 0-201-53771-0.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2nd edition, 2003.

[3] Dan Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C Recommendation, WWW Consortium, February 2004. `http://www.w3.org/TR/rdf-schema/`.

[4] W3C OWL Working Group. Owl 2 web ontology language document overview (second edition). W3c recommendation, W3C, December 2012. `http://www.w3.org/TR/owl2-overview/`.

[5] Patrick Hayes. RDF Semantics. W3C Recommendation, WWW Consortium, February 2004. `http://www.w3.org/TR/rdf-mt/`.

[6] Hector J. Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases*. The MIT Press, 2001.

[7] John Wylie Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1987.

[8] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. OWL 2 Web Ontology Language direct semantics (second edition). Technical report, W3C Recommendation, 11 December 2012. `http://www.w3.org/TR/owl2-direct-semantics/`.

[9] Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[10] Raymond Reiter. Towards a logical reconstruction of relational database theory. In Michael L. Brodie, John Mylopoulos, and Joachim W. Schmidt, editors, *On Conceptual Modelling*, pages 191–233. Springer Verlag, New York, NY, 1984.

[11] ICOM/CIDOC CRM Special Interest Group. Definition of the CIDOC Conceptual Reference Model version 5.1.2. Technical report, International Council of Museums, October 2013.