Artif Intell Law (2009) 17:291–320
DOI 10.1007/s10506-009-9082-z

Meta-relation and ontology closure in Conceptual Structure Theory

Philip H. P. Nguyen · Ken Kaneiwa · Dan R. Corbett · Minh-Quang Nguyen

Published online: 13 November 2009

Abstract This paper presents an enhanced ontology formalization, combining previous work in Conceptual Structure Theory and Order-Sorted Logic. Most existing ontology formalisms place greater importance on concept types, but in this paper we focus on relation types, which are in essence predicates on concept types. We formalize the notion of 'predicate of predicates' as meta-relation type and introduce the new hierarchy of meta-relation types as part of the ontology definition. The new notion of closure of a relation or meta-relation type is presented as a means to complete that relation or meta-relation type by transferring extra arguments and properties from other related types. The end result is an expanded ontology, called the closure of the original ontology, on which automated inference could be more easily performed. Our proposal could be viewed as a novel and improved ontology formalization within Conceptual Structure Theory and a contribution to knowledge representation and formal reasoning (e.g., to build a query-answering system for legal knowledge).

P. H. P. Nguyen (✉)
Justice Technology Services, Department of Justice, Government of South Australia,
30, Wakefield Street, Adelaide, SA 5000, Australia
e-mail: philip.nguyen@sa.gov.au

K. Kaneiwa
National Institute of Information and Communications Technology, 3-5 Hikaridai, Seika, Soraku,
Kyoto 619-0289, Japan
e-mail: kaneiwa@nict.go.jp

D. R. Corbett
Schafer Corporation, 3811, N. Fairfax Drive, Arlington, VA, USA
e-mail: daniel.corbett.ctr@darpa.mil

M.-Q. Nguyen
Department of Computer Science, University of Quebec at Montreal, Montreal, QC, Canada
e-mail: nguyen.minh-quang@uqam.ca

Keywords Ontology formalization · Knowledge representation · Automated reasoning · Conceptual Structure Theory · Order-Sorted Logic · Type theory · Concept type · Relation type · Meta-relation type · Legal reasoning

1 Introduction

In a formalism based on Conceptual Structure Theory (Corbett 2003; Nguyen and Corbett 2006a, b, 2007), an ontology is essentially a mapping between a real world and an abstract conceptual world, and consists of a concept type hierarchy, a relation type hierarchy, and formal relationships between them. This formalization is similar to what is proposed by Web Ontology Language (OWL) in which an ontology is defined as a collection of a set of classes (unary predicates), a set of properties (binary predicates), and a set of declarations describing how classes and properties are related (W3C 2004). Ontology is usually considered different from database. Ontology represents shared and commonly-agreed-to knowledge while database stores specific knowledge for a particular application or set of related applications (Dillion et al. 2008). The two structures are complementary in problem solving. Ontologies could even be considered to be hard-coded in computer systems (Greiner et al. 2001) as they express factual knowledge not varied across applications. However, in our formalism, ontology is a formal definition of relationships between a real and an abstract worlds, and as such, it contains information in both worlds. This means that our definition of ontology encompasses both traditional definitions of ontology and database, and could be considered the same as that of knowledge base in its broadest meaning.

Independently from the above, a formalization of ontology based on Order-Sorted Logic has also been proposed (Cohn 1989; Kaneiwa 2004), and one of its applications to upper event ontology has been presented (Kaneiwa et al. 2007). In this logic, an ontology is represented by a "sort hierarchy" and a "predicate hierarchy". The former is a hierarchy of objects in the domain of discourse, structured according to a set of partially ordered sorts (simply called a "sort"). Order-Sorted Logic makes a distinction between classes and instances of those classes (Smith et al. 2005), e.g., "earthquake" is a class of events while "the 2004 earthquake in Indonesia" is an instance of that class. This is similar to the definitions of concept type and instance of concept type in Conceptual Structure Theory. In addition, classes and instances could be further described or qualified through their *n*-ary predicates. The relationships between these predicates form another hierarchy, called predicate hierarchy, which complements the sort hierarchy in the ontology. Predicates in Order-Sorted Logic are similar to relation types and instances of relation types in Conceptual Structure Theory. For example, a criminal justice ontology could consist of a hierarchy of individuals (offenders and victims) and a hierarchy of offences, which are predicates on those individuals. Individuals could be sorted by place of residence and by gender, to form the sort hierarchy, while offences could be classified according to their nature and their degree of severity, to form the predicate hierarchy. Hybrid inference systems that link taxonomical information in the sort hierarchy with assertional information in the assertional knowledge base have also been proposed (Beierle et al. 1992; Kaneiwa 2001). This is similar in Conceptual Structure Theory to establishing

relationships between the hierarchical structure of concept types and the instances of relation types linking those concept types.

This paper attempts to further enhance the conceptual structure ontology formalism by incorporating new ideas from the above, especially with regard to formalization of predicate on concepts and predicate on other predicates. The latter is a second-order relation between predicates and concepts, similar to meta-predicate of SICStus Prolog or other higher-order logics such as HiLog (Chen et al. 1993). However, the introduction of the *hierarchy of meta-relation types* and its formalization in an ontology are presented for the first time in this paper.

In our ontology formalism, taxonomical and assertional information is combined in a single and coherent structure in order to facilitate automated inference. We also attempt to identify semantic properties of our ontology formalism to ensure its completeness and soundness, e.g., our formal definition of *property* of an ontological object (i.e., a concept type, a concept, a relation type, a relation, a meta-relation type, or a meta-relation) bears some similarities with the ontological conceptual ideas proposed by Dillon et al. (Dillion et al. 2008) and is a special case of the OWL *ObjectProperty* construct (W3C 2004). The end result of our effort is the production of a more complete ontology, called the closure of the original ontology, in which missing arguments and properties in relation types, meta-relation types, and their instances are supplemented.

Our main motivation for this research is in the area of formal reasoning, of which one application is the development of systems that can answer queries on topics that do not explicitly exist in databases, through automated inference based on ontological relationships between database objects and their predicates. This motivation is similar to that described in (Kaneiwa 2004). For example, in the justice arena we may wish to have a system that can automatically answer questions like the following: "Knowing only that John's father is in jail, does John have a Police record and is he being monitored by a welfare agency?" We will see at the end of this paper how an ontology built according to our formalism could help answer these types of question.

This paper is organized as follows: Sect. 2 summarizes the ontology formalism previously proposed within Conceptual Structure Theory. Section 3 proposes an extension to the formalism with the introduction of a new meta-relation type hierarchy within the definition of an ontology. Section 4 describes the new notion of closure in relation and meta-relation types. Section 5 explores important properties of the new formalism, in particular to address the issue of missing arguments and missing properties in relation and meta-relation types. Section 6 concludes the paper together with some directions for future research. Note that Sects. 3, 4 and 5 expand on ideas briefly presented in (Nguyen et al. 2008).

2 Ontology formalization in Conceptual Structure Theory

This section summarizes previous work on ontology formalization within Conceptual Structure Theory (Corbett 2003; Nguyen and Corbett 2006a, b). In this approach, an *ontology* is defined as a semantically consistent subset of a *canon,*

which is in essence a mapping of a real world into an abstract conceptual world. To simplify, we consider the two concepts of ontology and canon identical in this paper.

Definition 1 (*Original Ontology*): An ontology $\boldsymbol{O}$ is a 5-tuple $\boldsymbol{O} = (\boldsymbol{T}, \boldsymbol{I}, <, conf, B)$ in which:

(1) $\boldsymbol{T}$ is the *set of types*, i.e., $\boldsymbol{T} = T_C \cup T_R$ where T_C is the set of *concept types* and T_R the set of *relation types*.
(2) $\boldsymbol{I}$ is the *set of individuals* or *instances* of concept types in T_C.
(3) The symbol "<"is the *subsumption relation* in $\boldsymbol{T}$, representing the semantic generalization or specialization relationship between two concept types or two relation types.
(4) *conf* is the *conformity* function that links each individual in $\boldsymbol{I}$ to the infimum (or greatest lower bound) of all concept types that could represent that individual.
(5) B is the *canonical basis* function that defines for each relation type in T_R the tuple of all concept types (called *relation type arguments*) that can be used in that relation type. For a relation type R, the number of elements in $B(R)$ is called the *arity* (or *valence*) of R or of $B(R)$.
(6) The function B must also satisfy the following association rule, called *B-rule*: *If a relation type subsumes another relation type, then they must have the same arity and their values through B (i.e., the two tuples of concept type arguments) must also be related through the subsumption relation "<"between their respective arguments.*

2.1 Notations

The following notations will be used in this paper unless otherwise stated:

- C: a concept type
- R: a relation type
- MR: a meta-relation type
- t: a type (t could be a C, R or MR)
- T_C: the set of all concept types
- T_R: the set of all relation types
- T_{MR}: the set of all meta-relation types
- $\boldsymbol{T}$: the set of all types (i.e., $\boldsymbol{T} = T_C \cup T_R \cup T_{MR}$)
- c: a concept (or instance of a concept type)
- r: a relation (or instance of a relation type)
- mr: a meta-relation (or instance of a meta-relation type)
- i: an individual or instance of a type (i could be a c, r or mr)
- I_C: the set of all concepts
- I_R: the set of all relations
- I_{MR}: the set of all meta-relations
- $\boldsymbol{I}$: the set of all individuals or instances of all types (i.e., $\boldsymbol{I} = I_C \cup I_R \cup I_{MR}$)
- Γ: the set of all tuples (in general)

- $\tau(S)$: the set of all tuples defined over the set S
- T: a tuple
- e: a component (or element) of a tuple (tuple components are written between angle brackets, e.g., $T = <e_1, \ldots, e_n>$)

As a convention, we also use nouns with the first character in upper case to label concept types and concepts (e.g., Man, Person, Family), verbs with the first character in lower case to label relation types and relations (e.g., isDaughterOf, monitors, hasAttribute), and non-verbs (usually nouns and prepositions) with the first character in lower case to label *properties* of a type or an instance (e.g., byAdoption, minimum JailTerm) (the formal definition of *property* will be given later).

2.2 Types and instances

In this paper, the words "class", "individual", "instance" have meanings similar to those defined in OWL (Bechhofer 2004). The notions of relation type and instance of relation type introduced here are similar to those of relation schema and relation instance in Relational Model Theory (Codd 1970).

In Statements (1) and (2) of Definition 1, a concept type is a class of entities that share some common properties. Its instances are simply called *concepts*. A relation type (also called conceptual relation type, concept relation type, and relational concept type in other work) is a class of relations over one or multiple concept types, with those relations sharing some common properties. Like concept type, a relation type also has instances, called *relations* (also called conceptual relations (ISO 2007), concept relations, and relational concepts in other work). For example, "Man" is a concept type, and "John" is an individual or an instance of that concept type (this relationship is stated as: "John is a man").

To ensure completeness and soundness of the T_C and T_R structures, we assume that they are semi-lattices, which means that there is always a unique infimum for any two types in each structure. This assumption is common in ontology formalization, such as in Formal Concept Analysis (Wille 1982; Stumme 2002) and in Order-Sorted Logic (Kaneiwa 2004). This unique infimum, if it does not exist already, could also be created using a technique proposed for ontology merging in (Nguyen and Corbett 2006b) as merging of existing ontologies may initially produce pairs of concept types, each with multiple infima. We also assume that the T_C and T_R lattices are *bounded*, that is, they include the *Top* and *Bottom* types (also called *Universal* and *Absurb* types, and usually represented by the symbols "$\top$" and "$\bot$").

It should be emphasized that in our formalism, types (or classes) and their instances are grouped in separate sets. The set of type instances $\boldsymbol{I}$ [also called the *universe of discourse* (ISO 2007)] is separate from the set of types $\boldsymbol{T}$.

2.3 Choice between concept type and relation type

While some ideas can be naturally classified as a concept type (e.g., Man, Animal, etc.) or a relation type (e.g., isRelatedTo, isMarriedTo, etc.), in some other cases,

the choice between the two types is arbitrary and usually depends on the domain of discourse and on the intended usage of the resulting ontology. Most relation types can also be converted to semantically equivalent concept types, e.g., stealing can be defined as a concept type, or as a relation type linking a thief and a victim.

Note that Conceptual Structure Theory is initially inspired by J. Sowa's conceptual graph theory (Sowa 1984; Sowa 2000; ISO 2007) in which all conceptual graphs are *bipartite*. This means that a concept can only connect to another concept through a relation, and a relation can only link to another relation through a concept (except when subsumption relations are represented).

2.4 Subsumption relation

In Statement (3) of Definition 1, a type is said to *subsume* another type when the former is a *semantic generalization* of the latter, or the latter is a *semantic entailment* or *specialization* of the former (Smith 2003). With regard to relation type subsumption, this could be translated as a logical implication relation between predicates or propositions within the framework of logic programming, i.e., a relation in the form of "$p(x) \rightarrow q(x)$ *in which the superordinate predicate q has a more abstract meaning than the subordinate predicate p*" (Kaneiwa 2001; in our formalism, we say that q subsumes p). For example, the subsumption relation $Man < Person$ means that the concept type *Person* subsumes the concept type *Man* as the former semantically generalizes the latter or the latter semantically specializes or entails the former. We can also express this idea as a subsumption relation between two relation types $isMan < isPerson$ or $isMan(x) \rightarrow isPerson(x)$, i.e., if "John is a man", then "John is a person". Another similar example is $isDaughterOf < isChildOf$. Other examples in the criminal justice ontology (Breuker et al. 2002) are $steals < offends$ (as stealing is a particular case of committing an offence) and $murder < manslaughter$ (as murder is a type of manslaughter with premeditation).

Subsumption is mainly used for inference, e.g., if a type is subsumed into another type, and if an instance of the first type exists then we can infer another instance of the second type. For example, if "Hurricane Galveston hit Texas in 1900", then we can infer that "There was a natural disaster in Texas in 1900", since the concept type *Hurricane* is subsumed into the concept type *NaturalDisaster*.

Another aspect of our formalism is that the subsumption relation applies to types only, and not to their instances, i.e., the notion of *instance subsumption* does not exist or is meaningless, although one instance may be *inferred* from another, such as in the above example.

It should be noted that the logical implication relation (e.g., $p \rightarrow q$) could also represent a *causal* or a *parthood* (or part-of) relation between p and q. Both of these relations are proper relation types in our formalism, and *not* subsumption or semantic entailment relations. This means that different types of predicate or relation could translate into the same first-order logic statement and could be equally used to infer the same new assertion (see also Sect. 3.4 on Translation to First-Order Logic).

Finally, note that the subsumption relation defined in our formalism is broad (as opposed to strict), that is, mathematically it is *reflexive* (i.e., $\forall t \in \boldsymbol{T}\ t < t$).

2.5 Type conformance

In Statement (4) of Definition 1, the *conf* function expresses the idea of an individual *conforming* to a particular type. For each individual, it defines the (unique) infimum (or greatest lower bound) of all concept types that that individual could represent, e.g., the individual "John" *conforms* to the concept type "Man", which is the infimum of all concept types that "John" could represent, such as "Man", "Person", "Mammal", "Living Entity", etc., and therefore "John" is an instance of those concept types, i.e., "John is a man, a person, a mammal and a living entity."

2.6 Relation usage pattern

In Statement (5) of Definition 1, the function B expresses the *usage pattern* (or *canonical basis*) of each relation type as it identifies all concept types that can be used in that relation type, i.e., B: $T_R \rightarrow \tau(T_C)$ where $\tau(T_C)$ denotes the set of all *tuples* over T_C, formally defined as $\tau(T_C) = \cup_{\{n>0\}} (T_C)^n$.

As per the mathematical definition of a tuple, the order in which its components (also called *arguments*) are listed, is significant, that is, they can not be swapped without altering the identity of that tuple. It should also be noted that mathematically a tuple is different from a *set* (in which duplicate members are not allowed) and also different from a *multiset* (in which duplicate members are allowed but the order in which members are listed is irrelevant). In particular, $\tau(T_C)$ is different from the set of all *subsets* of T_C, usually denoted as 2^{T_C}. It is also different from the set of all *multisets* over T_C.

As an example of relation usage pattern, let us consider the relation type *isDaughterOf*. Its value through the B function could be defined as the tuple *(Woman, Person)* in which the first argument is the daughter and the second argument is the parent. That relation type could have two instances *isDaughterOf (Woman:Mary, Person:Sue)* and *isDaughterOf (Woman:Sue, Person:Mary)* with quite opposite meanings. Thus in a relation type, the order in which its arguments are listed (through the B function) contributes to the definition of the *intensional meaning* of the relation type and its instances.

The B function is similar to the *ARG* function of a predicate introduced in (Kaneiwa and Tojo 1999). Both attempt to define a unique structure for the arguments of a relation type or a predicate.

2.7 Relation subsumption and argument subsumption

In Statement (6) of Definition 1, the B-rule is an attempt to link subsumption between two relation types and subsumption between their arguments. For example, let us consider the subsumption relation between two relation types "*isDaughterOf* $<$ *isChildOf*" with *B(isDaughterOf) = (Woman, Person)* and *B(isChildof)*

$=$ *(Person, Person)*. In this case, the *B*-rule imposes that the first argument in the first relation type (i.e., *Woman*) is subsumed into the first argument of the second relation type (i.e., *Person*), and likewise for their second arguments (i.e., *Person* $<$ *Person*).

2.8 Ontology construction

Through Definition 1, we see that ontology contains both static and generic information (such as types), and dynamic and particular information (such as individuals). The type hierarchies in the ontology are relatively static, often meant to be shared across different applications in the same domain of discourse. They usually represent common knowledge agreed to by experts of the domain, and usually are *not* built with a specific application in mind. The ontology also contains more dynamic and specific information (such as individuals), which is traditionally maintained in a separate database (e.g., the Customer or Billing database in a commercial company). In our ontology formalism, the *conf* and *B* functions are constantly re-evaluated as individuals are updated. A new piece of information concerning an individual may change the *conf* value for that individual, and/or may necessitate the creation of a new type. A new relation may introduce new arguments and hence may change the *B* value of the corresponding relation type.

3 Proposed new ontology formalism

This section expands ideas introduced in (Nguyen et al. 2008). We first introduce the new mathematical concepts of tuple membership, tuple extension and tuple subsumption, and then use them in our proposed new ontology formalism.

3.1 Tuple membership, extension and subsumption

Definition 2 (*Tuple Membership, Extension and Subsumption*):

1. Tuple Membership: A component e of a tuple T is written as $e \in T$.
2. Tuple Extension: Let $T_1 = <e_1, \ldots, e_n>$ be an n-tuple and $T_2 = <f_1, \ldots, f_m>$ be an m-tuple, T_1 is said to be an *extension* of T_2 (or T_1 is said to *extend* T_2, and we write $T_1 = \text{ext}(T_2)$) if all components of T_2 are also present in T_1 with their relative listing order respected, i.e., $T_1 = \text{ext}(T_2) \Leftrightarrow <e_1, \ldots, e_n> = \text{ext}(<f_1, \ldots, f_m>) \Leftrightarrow (m \leq n)$ and $(\forall k, l\ 1 \leq k \leq l \leq m\ \exists i, j$ with $1 \leq i \leq j \leq n$ and $e_i = f_k$ and $e_j = f_l)$
3. Tuple Subsumption: Let T_1 be an n-tuple and T_2 be an m-tuple with $m \leq n$, T_1 is said to *subsume* T_2 (and we write $T_2 < T_1$) if there exists an m-tuple T_2' such that:
   - $T_1 = \text{ext}(T_2')$ and
   - Each component of T_2 is subsumed into the corresponding component of T_2', i.e., if $T_2 = <f_1, \ldots, f_m>$ and $T_2' = <f_i', \ldots, f_m'>$ then $\forall i\ 1 \leq i \leq m$ $f_i < f_i'$

Notes:

1. In Definition 2(1), the notation "$e \in T$" is normally reserved for *set* membership, and means that e is a member of the *set* T (but here T is not a set but a tuple). We can thus write: $T = <e_1, \ldots, e_n> \Leftrightarrow \forall i\ 1 \leq i \leq n\ e_i \in T$
2. Definition 2(2) also implies that $\forall k\ 1 \leq k \leq m\ \exists i\ 1 \leq i \leq n\ e_i = f_k$
3. Definition 2(3) is an expansion of the definition of the subsumption relation "<"[introduced in Definition 1(3)] to tuples.

Example 1 (Tuple Subsumption): Let $T_1 = <Person, LivingEntity, Person>$ and $T_2 = <Woman, Animal>$.

We have the tuple subsumption relation: $T_2 < T_1$ because we can select $T_2' = <Person, LivingEntity>$ and fulfill the tuple subsumption conditions with:

- $T_1 = \text{ext}(T_2')$
- *Woman* $<$ *Person* (i.e., 1st argument of $T_2 <$ 1st argument of T_2')
- *Animal* $<$ *LivingEntity* (i.e., 2nd argument of $T_2 <$ 2nd argument of T_2')

Proposition 1 (Tuple Extension and Subsumption Properties): *The tuple extension relation is*: ($\forall T_1, T_2, T_3 \in \Gamma$)

(1) *reflexive, i.e.,* $T_1 = ext(T_1)$
(2) *anti-symmetrical, i.e.,* $T_1 = ext(T_2)$ *and* $T_2 = ext(T_1) \Rightarrow T_1 = T_2$
(3) *subsuming, i.e.,* $T_1 = ext(T_2) \Rightarrow T_2 < T_1$
(4) *transitive, i.e.,* $T_1 = ext(T_2)$ *and* $T_2 = ext(T_3) \Rightarrow T_1 = ext(T_3)$
(5) *transitive-2, i.e.,* $T_1 < T_2$ *and* $T_3 = ext(T_2) \Rightarrow T_1 < T_3$
(6) *transitive-3, i.e.,* $T_2 = ext(T_1)$ *and* $T_2 < T_3 \Rightarrow T_1 < T_3$

Proof This proposition can be easily proved with the definitions of tuple extension and tuple subsumption. In particular, Statement (3) means that tuple extension implies tuple subsumption, i.e., if T_1 extends T_2 then T_1 subsumes T_2. It is proved by selecting $T_2' = T_2$ in the definition of tuple subsumption [Definition 2(3)] and by using the reflexive property of tuple extension (Statement 1).

3.2 Meta-relation type

Definition 3 (*Meta-relationType*): A *meta-relation type* is a non-subsumption relation between at least one relation type and a number of concept types. An instance of a meta-relation type is called a *meta-relation*.

Notes:

1. A meta-relation type is a predicate on concept types and relation types with at least one relation type present. If a meta-relation type does not involve at least one relation type, it is simply a (simple) relation type. While a relation type represents a predicate on concept types, a meta-relation type is essentially a "predicate of predicates".

2. The main difference between relation type subsumption (i.e., subsumption between relation types) and meta-relation type is that subsumption is based on semantic generalization or specialization while meta-relation type is based on other types of semantic relationship.
3. In some cases, a meta-relation type can be semantically translated into a (simple) relation type (see Example 2).
4. We will use the phrase "relational object" to designate a relation type, a relation, a meta-relation type or a meta-relation in general.

Example 2 (Meta-relation Type): The expression *likelyCauses(Earthquake, Tsumani)* is a predicate on two concept types, expressing that an earthquake may cause a tsunami. It is a (simple) relation type in this case.

It could be generalized as *likelyCauses(Entity, Entity)*, in which *Entity* could be either a concept type or a relation type. In this case it could be considered a meta-relation type, in which the occurrence of the first entity could likely cause that of the second entity. For example, the assertion "a person in a dysfunctional family is likely to commit an offence" could be represented in the ontology if we construct the two relation types *isInDysfunctionalFamily(Person)* and *offends (Offender, Victim),* and define *likelyCauses(isInDysfunctionalFamily, offends)* as an instance of the meta-relation type *likelyCauses(Entity,Entity)*, in which each argument is a instance of the generic relation type *Entity*.

We could further generalize the predicate as a meta-relation type *likelyCauses (Antecedent, Consequent)* linking two *events* (or situations), an antecedent event and a consequent event, with an event being defined as a combination of a number of concept and relation types (i.e., an event is a subset of $T_C \cup T_R$). For example, if we wish to express that "Driving in bad weather may cause accident", we could consider the concept and relation types: *BadWeather, drives(Person), hasAccident (Person)*, and define the meta-relation between them as *likelyCauses({drives, BadWeather}, {hasAccident})* in which the antecedent is a combination of driving and bad weather and the consequent is the accident.

3.3 New ontology formalization

Definition 4 (*New Ontology with Meta-relation Type Hierarchy*): An ontology $\boldsymbol{O}$ is a 5-tuple $\boldsymbol{O} = (\boldsymbol{T}, \boldsymbol{I}, <, conf, B)$ as per Definition 1 with in addition the following features:

(1) The set of types $\boldsymbol{T}$ is extended to include the set of *meta-relation types* T_{MR}, i.e., $\boldsymbol{T} = T_C \cup T_R \cup T_{MR}$.
(2) The set of individuals or instances $\boldsymbol{I}$ is expanded to include the set of relations I_R and the set of meta-relations I_{MR}, i.e., $\boldsymbol{I} = I_C \cup I_R \cup I_{MR}$ with I_C being the set of all *concepts* (or instances of concept types), I_R the set of all *relations* (or instances of relation types) and I_{MR} the set of all *meta-relations* (or instances of meta-relation types).
(3) The subsumption relation "$<$", which represents semantic generalization or specialization between types, is extended to be also defined over the set of

meta-relation types, enabling the latter to be structured as a third hierarchy in the ontology, the *hierarchy of meta-relation types*.

(4) The function *conf* is extended to be defined over the combined set of I_C, I_R, and I_{MR}, i.e., *conf*: $\boldsymbol{I} \rightarrow \boldsymbol{T}$

$$\begin{array}{lll} \text{with :} & \forall c \in I_C & conf(c) \in T_C \\ & \forall r \in I_R & conf(r) \in T_R \\ & \forall mr \in I_{MR} & conf(mr) \in T_{MR} \end{array}$$

In the above, *conf(c)* is the infimum of all concept types that the instance c could represent, *conf(r)* is simply the relation type of which r is an instance, and *conf(mr)* is the meta-relation type representing the instance *mr*.

(5) The function B, which defines the usage pattern of a predicate, is extended to be defined over the combined set of all relation types and meta-relation types, i.e.,

$$\begin{array}{lll} & B : T_R \cup T_{MR} \rightarrow & \tau(T_C) \cup \tau(T_C \cup T_R) \\ \text{with:} & \forall R \in T_R & B(R) \in \tau(T_C) \\ & \forall MR \in T_{MR} & B(MR) \in \tau(T_C \cup T_R) \end{array}$$

(6) The B-rule is broadened as follows:
New *B*-rule (Relation and Meta-relation Type Extension): If a relation (or meta-relation) type R subsumes another relation (or meta-relation) type S (i.e., $S < R$), then there is a relation (or meta-relation) type, called an *extension of R with respect to S* and denoted as $R_S^{\wedge}$ (or simply $R^{\wedge}$ for short), such that the four following statements hold:

(a) $S < R^{\wedge}$
(b) $B(R^{\wedge}) = ext(B(R))$
(c) $B(S) < B(R^{\wedge})$
(d) $B(R) < B(R^{\wedge})$

Figure 1 graphically represents the main notions introduced above.

Notes:

1. Definition 4(1) introduces the new concept of meta-relation type to Conceptual Structure Theory, and together with Definition 1(3), enable the new meta-relation type hierarchy to be added to the ontology definition. This hierarchy is first introduced in this paper.

   It should be noted that in OWL, the OWL ObjectProperty construct could be used to express object predicates, in a way equivalent to our definition of relation type. However, there are no OWL constructs that are similar to our definition of meta-relation type to express "predicates of predicates of objects".
2. Definition 4(2) introduces the new concepts of instance of relation type and instance of meta-relation type to Conceptual Structure Theory.
3. Definition 4(5) means that the new B function now defines:

   - for a relation type, the tuple of concept types that can be used in that relation type, and

- for a meta-relation type, the tuple of concept types and relation types that can be used in that meta-relation type.

Thus, the new B function can now also express the structure of predicate on concept types, as well as predicate on a combination of other predicates and concept types.

By convention, when r is a relation of type R, we also denote by $B(r)$ the tuple of concepts (called *arguments* of r) that are used in r, and whose components are instances of the corresponding components of $B(R)$, although mathematically, B is a function defined over T_R and T_{MR} only, not I_R, nor I_{MR}.

Since $B(R)$ is the most important feature in a relation (or meta-relation) type R, we usually represent $B(R)$ together with R. For example, if $R = isDaughterOf$ is a relation type, then we often write (especially in Conceptual Graph Theory (Sowa 1984)) $R = isDaughterOf\ (Woman,\ Person)$, which means that we effectively write $R(B(R))$, and we say that *Woman* and *Person* are the two arguments of the relation type *isDaughterOf* while in reality, they are the two components of the 2-tuple $B(isDaughterOf)$. Similarly, when mr is a meta-relation of type MR, we also denote by $B(mr)$ the tuple of concepts and relations (called *arguments* of mr) that are used in mr and whose components are instances of the corresponding components of $B(MR)$.

4. In general, any object in the ontology, whether it is a concept type, a relation type, a concept or a relation, is defined by a label (or name) and the "context" of the object in the ontology. Together they represent the *intensional meaning* of the object. The clearer this context can be specified, the more accurate is the definition of the object. In our formalism, the context of the object is its relationships with other objects with respect to the three relations: $<$, *conf* and B. The main goal of any ontology formalism is to define that context as clearly and as accurately as possible for any ontological object. In a way, our emphasis in representing the intensional meaning of an object through its various relationships with other objects is similar to the approach used for object definition in the theory of Latent Semantic Analysis (LSA).
5. In Definition 4(6), the new B-rule states in essence that if a relation (or meta-relation) type R subsumes another relation (or meta-relation) type R', then there exists a new relation (or meta-relation) type $R\hat{}$ that extends the arguments of R such that each argument of R' is subsumed into a corresponding argument of $R\hat{}$. In other words, the arguments of R' (in fact their appropriate supertypes) are "merged" into the arguments of R to create the set of arguments for $R\hat{}$. The new B-rule can thus be summarized as: "*Arguments of a relation or meta-relation subtype can be merged into arguments of its supertype*".

   Finally, note that two subsuming relation (or meta-relation) types can have different arities, but the arguments of their *extensions* (as defined by the new B-rule) must be semantically related in a consistent manner.
6. The new B-rule is the first step to supplement missing arguments in relation and meta-relation types, similarly to the manipulation of predicate arguments in Order-Sorted Logic (Kaneiwa 2004; Nitta 1995).

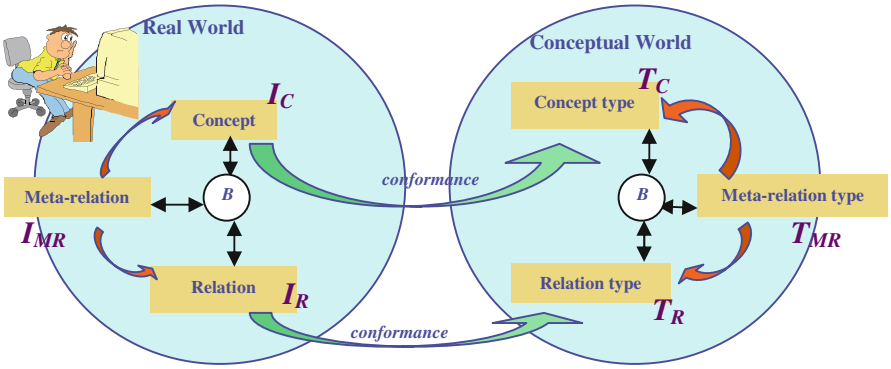

Fig. 1 Proposed ontology formalization

We will see later that the building of the *closure* of a relation (or meta-relation) type includes a recursive application of the *extension* of that relation (or meta-relation) type against all other relation (or meta-relation) types in the ontology. The completion of missing arguments in predicates facilitates automated inference on facts and assertions (Fig. 1).

Example 3 (Subsumption between Meta-relation Types): In a traffic accident ontology, to express the causal relation between an act performed under some atmospheric condition and its effect, such as "Driving into a hurricane will cause accident" and "Moving in bad weather may cause damage", we could define the following meta-relation types:

- *causes(drives(Person), Hurricane, Accident)*
- *likelyCauses(moves(Entity), BadWeather, Damage)*

In this case, we have a subsumption relation between those two meta-relation types "*causes* < *likelyCauses*" since the new *B*-rule could be satisfied with the following subsumption relations between their arguments:

- *drives* < *moves* (with also a subsumption relation between these relation types' own arguments: *Person* < *Entity*)
- *Hurricane* < *BadWeather*
- *Accident* < *Damage*

Example 4 (Extension of Relation Type): Suppose that we have in our ontology two relation types: *steals (Thief, TheftVictim)* and *offends(Offender).* Semantically, we have: *steals* < *offends* and we could construct the extended relation type: *offends^(Offender, OffenceVictim)* by adding a supertype of *TheftVictim,* which is *OffenceVictim,* to the tuple of arguments of the extended type.

3.4 Translation to first-order logic

To assist with clarification of the semantics of relations and meta-relations, their translation into first-order logic may sometimes be highly desirable, especially to

clarify the intensional meaning of each argument within the context of the whole relation or meta-relation. This translation happens at the *instance* level of the relation and meta-relation types.

For example, at the *type* level, the meta-relation type *likelyCauses (isInDysfunctionalFamily (Person), offends (Person, Person))* simply expresses a possible causal relationship between being in a dysfunctional family and being involved in an offence (with the two arguments of the relation type *offends* representing the offender and the victim, respectively).

Suppose now that we would like to represent the following assertions into the ontology:

- A person in a dysfunctional family is likely to offend.
- A person in a dysfunctional family is likely to be the victim of an offence.
- A person in a dysfunctional family is likely to self-harm.

These assertions could be represented by the three following instances of the meta-relation type *likelyCauses* as follows:

1. *likelyCauses (isInDysfunctionalFamily (Person:x), offends (Person:x, Person:y))*
2. *likelyCauses (isInDysfunctionalFamily (Person:x), offends (Person:y, Person:x))*
3. *likelyCauses (isInDysfunctionalFamily (Person:x), offends (Person:x, Person:x))*

(with self-harm defined in the legal sense as committing an offence against oneself)

The above can also be written in first-order logic notations as:

1. $\forall x \in \{\text{Person}\}$ *isInDysfunctionalFamily (x)* $\rightarrow \exists y \in \{\text{Person}\}$ *offends (x,y)*
2. $\forall x \in \{\text{Person}\}$ *isInDysfunctionalFamily (x)* $\rightarrow \exists y \in \{\text{Person}\}$ *offends (y,x)*
3. $\forall x \in \{\text{Person}\}$ *isInDysfunctionalFamily (x)* $\rightarrow$ *offends (x,x)*

(with the logical connective "→" loosely interpreted as "possibly implies"—see also Sect. 6)

The above highlights the importance of the order and meaning of each argument in a relation or meta-relation. They contribute to clarifying the intensional meaning of the overall relation or meta-relation. This is also the reason why arguments of a relation or meta-relation are mathematically defined as a tuple, rather than as a set or multiset (see the differences between them in Sect. 2.6).

4 Ontology closure

4.1 Type and instance properties

The main difference between a relation type and a relation is that the latter may include specific information that is pertinent to the particular context in which the relation is expressed. For example, *isDaughterOf* is a relation type, linking two concept types: *Woman*, *Person*. To express that "Mary is the daughter of John by adoption", we can use the relation type *isDaughterOf* but with a qualifier *byAdoption*. This means that the two instances of the concept types *Woman* and

Person (which are *Mary* and *John*) are linked through a particular instance of the relation type *isDaughterOf,* which contains the additional qualifier: *byAdoption* (this will be formally defined as a *property* of the relation). And we write: *isDaughterOf (Woman: Mary, Person: John, <byAdoption>)*. In general, a relation contains specific information that is not already contained in the concepts that it links. In the example, the qualifier *byAdoption* is not specific to the concept *Mary*, nor to the concept *John*, but is specific to a particular case (i.e., an instance) of the relation type *isDaughterOf*. If the specific information of the relation can be accommodated by other concept types (that are already in the concept type hierarchy of the ontology), then those concept types should be added to the corresponding relation type. For example, if we have *ChildParentRelationship* as a concept type in the ontology, then we can have a 3-ary relation *isDaughterOf (Woman: Mary, Person: John, ChildParentRelationship: Adoption)*, which is an instance of the new relation type *isDaughterOf(Woman, Person, ChildParentRelationship)*.

Another example is to represent the assertion "Sue is married to John for 5 years" into the ontology. In this case, we can define *isMarriedTo* as a relation type between two persons and *isMarriedTo(Person: Sue, Person: John, <duration, 5 years>)* is an instance of that relation type (with duration as a *property* of the relation). However, if Time (or Duration) is a concept type that is already included in the concept type hierarchy, then we should consider *isMarriedTo* as a relation type between three concept types: *Person*, *Person* and *Duration*, i.e., *isMarriedTo* (*Person*, *Person*, *Duration*). And the above assertion could be translated as *isMarriedTo* (*Person: Sue*, *Person: John*, *Duration: 5 years*). This is the case of the criminal justice ontology described in (Breuker et al. 2002), in which time and space, among other concept types such as person, role, action, process, procedure, time, space, document, information, intention, etc., are defined as part of the concept type hierarchy.

A relation or meta-relation type is an n-ary relation, and in principle, the larger its arity (the value of n), the better for deductive reasoning. However, this must be balanced against the cost of processing involved in the creation of a new relation or meta-relation type, which in turn may necessitate the creation of new concept types to fully express the intensional meaning of the new relation or meta-relation type. When a new ontological object is inserted into the ontology, a review of existing entries in the ontology is required to ensure the consistence between new and existing objects. For example, we may need to ask the following questions:

- Which existing concept types subsume, or are subsumed into, the newly introduced concept type, i.e., where to insert the new concept type to the concept type hierarchy?
- Could the new concept type be used as a new argument in existing relation or meta-relation types, in replacement for some existing arguments or properties?
- Could the *conf* values of some existing concepts, relations or meta-relations be changed to the new type?

To avoid proliferation of new concept types of minor significance, a new information item in a relation or meta-relation type could be defined as one of its *properties*, rather than as a new concept type argument. In general, such a decision

depends on the domain of discourse (e.g., concept types foreign to the domain may not need to be added) and on practical constraints on the computing environment of the ontology (e.g., a need to record new relations quickly into the ontology).

In our formalism, any extra piece of information that pertains to the intensional meaning of a type or an instance and that cannot be accommodated by existing types is called a property and retained as an attribute of that type or instance.

By convention, we write type and instance properties between angle brackets, such as <*byAdoption*> and <*duration, 5 years*> in the above examples.

The *ontology properties* defined in (Dillion et al. 2008) are what we classify in this paper as relation types, relations, type properties and instance properties, and what are termed *concept predicates* in Order-Sorted Logic (Kaneiwa 2004; Dillion et al. 2008) also considers that "*ontology properties are quite close to attributes in object-oriented modeling*" with which our formalism concurs. But our break-down of these ontological properties into different categories permits finer classification of these types of information.

4.2 Axioms on properties

Unless otherwise stated, for a relation or meta-relation type, we will use the words "type properties" to designate its proper properties (as defined above) as well as its arguments. The propagation of properties between types and instances are governed by the following axioms, which are essentially derived from the semantic relationships between the various ontological objects defined in Definition 4.

Axiom 1 *(Type Property Inheritance)*: For any type, its properties are inherited by all of its instances, and by all of its subtypes.

Notes:

1. For a relation or meta-relation type, its properties include its concept type arguments, their properties (i.e., properties of the concept type arguments), and properties that are proper to the relation or meta-relation type. The propagation of the arguments and properties of a supertype S to a subtype R transforms the latter to a richer type, called *inherited type* and denoted as R_S^{V}, or simply R^{V} (the superscript V expresses the *downward* propagation of arguments and properties). In essence, this is the inverse of the *extended type* $R_S\hat{}$ defined in Definition 4(6).
2. Axiom 1 is also a generalization of a statement in (Dillion et al. 2008) that "every ontology property of the superclass is a property of the subclasses as well".

Example 5 (Property Inheritance): Suppose that we have in an ontology:

- The relation type "*murder*" with one argument "*Person*" and one property of "*minimumNoParoleTerm: 2 years*".
- The relation type "*manslaughter*" with one argument "*Person*" and one property of "*minimumJailTerm: 3 years*".

As per Axiom 1 and other previous defined semantic rules, we can deduce that:

1. A subsumption relation between two types: *murder* < *manslaughter* (as murder is a type of manslaughter with premeditation).
2. All instances of *murder* carry a minimum no-parole period of 2 years (property inheritance by instance), e.g., if John is convicted of a murder charge, then John should have a minimum no-parole term of 2 years.
3. All instances of *manslaughter* carry a minimum jail term of 3 years (property inheritance by instance), e.g., if John is convicted of a manslaughter charge, then John should have a minimum jail term of 3 years.
4. The type *murder* carries a minimum jail term of 3 years (property inheritance by subtype).
5. All instances of *murder* carry a minimum jail term of 3 years (property inheritance by instance), e.g., if John is convicted of a murder charge, then John should have a minimum jail term of 3 years with a minimum no-parole term of 2 years.

Axiom 2 *(Instance Property Generalization)*: For any instance of a type and for any supertype of that type, there is another instance of that supertype such that the properties of the first instance also hold true for the second instance.

Note: In simple terms, Axioms 1 and 2 could be summarized as: "*Type properties go down and instantiate, while instance properties go up*".

Example 6 (Instance Property Generalization): In Example 5, if in addition we have the following instance:

- r_1 = *murder(Person: John, <jailTerm: 10 years>)* (i.e., "John is condemned to 10-year imprisonment for murder"),

Then we can infer the following instance of the type *manslaughter*, which is a supertype of *murder*:

- r_2 = *manslaughter(Person: John, <jailTerm: 10 years>)* (i.e., "John is condemned to 10-year imprisonment for manslaughter").

Combining with the previous result from Example 5, we can now say that "John is condemned to 10-year imprisonment with a minimum 2-year no-parole term".

As mentioned in Note 4 of Definition 1, the concept of *subsumption* applies to types only and not to their instances. In the above, we cannot say that the relation (instance) r_1 is *subsumed* into the relation r_2, but we can say that r_2 is *inferred* from r_1.

Example 7 (Instance Property Generalization): Suppose now that we have 2 relation types and 1 instance:

- *steals(Thief)*
- *offends(Offender, OffenceVictim)*
- r_1 = *steals(Thief:John)* (i.e., "John is a thief")

Since *steals* < *offends*, we can deduce the following:

- r_2 = *offends(Offender:John, OffenceVictim:⊥)*, i.e., "John commits an offence against some unknown person", as the instance "John" of the argument "Thief" in r_1 could "go up" to r_2.

Example 8 (Instance Property Generalization): Suppose that we have 2 relation types and 2 instances:

- *steals(Thief)*
- *offends(Offender, OffenceVictim)*
- r_1 = *steals(Thief:John)* (i.e., "John is a thief")
- r_2 = *offends(Offender: John, OffenceVictim:Mary)* (i.e., "John commits an offence against Mary")

First, since we have *steals* < *offends*, we can deduce the inherited type *steals (Thief, TheftVictim)* from Axiom 1. But we *cannot* deduce that the instance *steals (Thief: John, TheftVictim: Mary)* (i.e., "John steals from Mary") is valid, because the instance of the *OffenceVictim* argument of the relation *offends* (that is "Mary") does not "go down" (it can only "go up" as per Axiom 2).

Axiom 3 *(Relation and Meta-relation Type Closure)*: For any relation (or meta-relation) type R, there is another relation (or meta-relation) type R^*, called the closure of R, that satisfies the following conditions:

(1) R^* contains all the arguments of R, together with all the properties of R and all the properties of the arguments of R, if exist.
(2) R^* contains all the arguments of each supertype of R, with possibly additional properties for those arguments (i.e., properties that are specific to the semantics of R).
(3) For each subtype of R and for each argument of that subtype, R^* contains a supertype of that argument, together with all properties of that argument, if exist.
(4) R^* contains no semantically redundant arguments and properties.

Notes:

1. Statement (1) means that R^* is a semantic specialization of R as R^* contains the full semantics of R. Statement (2) means that R^* is a result of a recursive application of Axiom 1 (type inheritance) against all supertypes of R. Statement (3) means that R^* is a result of a recursive application of the new B-rule (type extension) against all subtypes of R. Statement (4) simply means that R^* is tidied up to remove redundant semantics. Therefore, Axiom 3 is essentially a combination of Axioms 1 and 2.
2. We denote by T^*_R and T^*_{MR} the sets T_R and T_{MR} to which their closures are added. The original subsumption relation with the same meaning (i.e., semantic generalization or specialization) is also extended to the new sets. Statement (1) therefore means R^* is subsumed into R, or $R^* < R$.

Example 9 (Relation Type Closure): Suppose that we have the following relation types in an ontology:

- *commitViolentAct(Offender, Victim, ViolenceMotive, ViolenceInstrument*) (e.g., an instance of it may be: "John threatens Mary with a knife because Mary annoys him").
- *rob(Robber, Victim, StolenObject*) (e.g., an instance of it may be: "John robs a pen from Mary").
- *robWithViolence(Robber*) (e.g., an instance of it may be: "John is a violent robber")

Then we have the following subsumption relations:

- $robWithViolence < rob$
- $robWithViolence < commitViolentAct$

And we can infer the following relation type closure:

- *robWithViolence*(Robber, Victim, StolenObject, ViolenceInstrument*) (e.g., an instance of it may be: "John commits a violent robbery with gun against a bank and steals a large sum of money"). This is because according to the above Axioms, we have:
- $Robber = \text{infimum}\ \{Robber, Offender\}$
- $StolenObject = \text{infimum}\ \{StolenObject, ViolenceMotive\}$

Example 10 (Relation Type Closure): Suppose that we have the following relation types in an ontology:

- *picksPocket (PettyLarcenist, PickpocketVictim, StolenAmount)*
- *steals (Thief)*
- *offends(Offender,OffenceVictim,OffenceAct,OffenceInstrument)*

Note that petty larceny is a minor theft, such as pick pocketing. Semantically, we have:

- $picksPocket < steals < offends$
- $PettyLarcenist < Thief < Offender$

As per Axiom 3, we could define the following relation type closures:

- *picksPocket*(PettyLarcenist, PickpocketVictim, OffenceAct: <pickPocketing>, OffenceInstrument: <byHands>, StolenAmount)*
- *steals*(Thief, TheftVictim, OffenceAct: <stealing>, OffenceInstrument, StolenObject)*
- *offends*(Offender, OffenceVictim, OffenceAct, OffenceInstrument, OffenceMotive)*

Axiom 4 *(Relation and Meta-Relation Closure)*: For any relation (or meta-relation) r of type R, there is another relation (or meta-relation) r^*, called the closure of r, such that r^* is an instance of the type closure R^*. In addition, r^* contains all the arguments of r, together with all the properties of r, and all the properties of the arguments of r, if exist.

Note: Axiom 4 simply states the following: $\forall r \in I_R \cup I_{MR}\ conf(r) = R \rightarrow \exists y \in I^*_R \cup I^*_{MR}\ conf(r^*) = R^*\ (= (conf(r))^*)$ in which I^*_R and I^*_{MR} are the sets I_R and

I_{MR} to which their closures are added. In simple terms, this Axiom states that "*the conformance of the closure is the closure of the conformance*".

Example 11 (Relation Closure): Suppose that we have the following types and instances in an ontology:

- *steals(Thief, TheftVictim)*
- *offends(Offender)*
- *steals(Thief:John, TheftVictim:Mary*) (i.e., "John steals from Mary")
- *offends(Offender:John*) (i.e., "John is an offender")

As per Axioms 3 and 4, we can infer the following relation closure:

- *offends*(Offender:John, OffenceVictim:Mary*) (i.e., "John commits an offence against Mary") as the argument *TheftVictim* of the relation type *steals* could be merged ("go up") into the arguments of the supertype *offends*.

Example 12 (Relation Closure): Suppose that we have the following types and instance in an ontology:

- *picksPocket(PettyLarcenist, PickpocketVictim, StolenAmount)*
- *steals(Thief)*
- *offends(Offender, OffenceVictim, OffenceAct, OffenceInstrument)*
- *picksPocket(PettyLarcenist: John, PickpocketVictim: Mary, StolenAmount: $5.00*) (i.e., "John picked $5.00 from Mary's pocket").

We have *picksPocket* < *steals* < *offends*, and as per Axioms 3 and 4, we can infer the following relation closures:

- *picksPocket*(PettyLarcenist: John, PickpocketVictim: Mary, OffenceAct: <pickPocketing>, OffenceInstrument: <byHands>, StolenAmount: $5.00)*, i.e., "John picked $5.00 from Mary's pocket".
- *steals*(Thief: John, TheftVictim: Mary, OffenceAct: <pickPocketing>, OffenceInstrument: <byHand>, StolenObject: $5.00)*, i.e., "John steals $5.00 from Mary by picking with his hand in Mary's pocket").
- *offends*(Offender: John, OffenceVictim: Mary, OffenceAct: <pickPocketing>, OffenceInstrument: <byHand>, OffenceMotive: $5.00)*, i.e., "John commits an offence against Mary by picking $5.00 with his hand from Mary's pocket".

Definition 5 (*Ontology Closure*): For an ontology $\boldsymbol{O}$, the ontology $\boldsymbol{O}^*$ obtained by adding all the type and instance closures built as per Axioms 3 and 4 is called the closure of the ontology $\boldsymbol{O}$.

Notes:

- The sets T^*_R, T^*_{MR} (in Axiom 3), I^*_R and I^*_{MR} (in Axiom 4) are part of the new ontology $\boldsymbol{O}^*$.
- For the rest of this paper, whenever we refer to the closure of a type or instance, we imply that the ontology in the background is the closure of the original ontology.

Proposition 2 (Soundness of Type Extension and Closure): *For any relation (or meta-relation) types R and S such that* $S < R$, R^* i*s an extension of R with respect to S*, i.e., $\forall R,S \in T_R \cup T_{MR}\ S < R \rightarrow R^* = R_S\wedge$

Proof This proposition can be easily proved with the definition of extension (in the sense of the new B-rule of Definition 4) and closure (Axiom 3).

Note: Proposition 2 reinforces the idea expressed in Axiom 3(3) that type closure is obtained by a process that includes a recursive extension of that type with respect to each of its subtypes.

Proposition 3 (Soundness of Type Closure): *Let R be a relation (or meta-relation) type. The following statements hold:*

(1) $(R^*)^* = R^*$
(2) $R^* < R$
(3) *Each argument of* R^* *is the infimum of all the semantically-related arguments of all supertypes of R and of an argument of R, if exists.*

Proof Statement (1) holds because there can only be one *unique* infimum for any set of concept types as per Definition 1(1). Statements (2) and (3) could easily be proven with the definition of the subsumption relation between relation or meta-relation types.

Note: Proposition 3 expresses that type closure is a semantic specialization of the original type, and incorporates the semantics of the part of the ontology in the background that relates to that type (i.e., its context or its relationship with the rest of the ontology). Therefore, we can say that in an ontology closure, the semantics of a type and its context are *condensed* into the type closure.

4.3 Ontology closure construction

The process of building the closure of a relation type could be formalized as a function *f*, called the *relation type closure function*, that associates each relation type with its closure, i.e., f: $T_R \rightarrow T^*_R$ with $\forall R \in T_R f(R) = R^*$.

In order to define *f*, let us first define the following:

- For a relation type S in T_R, let $f_S\wedge$ be a function from the set of all supertypes of S (denoted as $Sup(S) = \{R \in T_R \mid S < R\}$) to the set $T_R\wedge$ of all relation types and their *extensions* (as per the new *B*-rule). The function $f_S\wedge$ simply associates each supertype of S with its extension with respect to S, i.e., $\forall S \in T_R\ f_S\wedge$: $Sup(S) \rightarrow T\wedge_R$ with $\forall R \in Sup(S)\ f_S\wedge(R) = R_S\wedge$
- Similarly, for a relation type S in T_R, let $f_S^{\vee}$ be a function from the set of all subtypes of S (denoted as $Sub(S) = \{R \in T_R \mid R < S\}$) to the set $T_R^{\vee}$ of all relation types and their *inheritances* (as per Axiom 1). The function $f_S^{\vee}$ simply associates each subtype of S with its inheritance from S, i.e., $\forall S \in T_R\ f_S^{\vee}$: $Sub(S) \rightarrow T_R^{\vee}$ with $\forall R \in Sub(S)\ f_S^{\vee}(R) = R_S^{\vee}$
- In the above, note that $\forall S,R \in \boldsymbol{T}\ R \in Sup(S) \Leftrightarrow S \in Sub(R)$

The relation type closure function f could finally be defined as:

$$\forall R \in T_R \quad f(R) = R^* = \prod_{\{W \in Sup(R)\}} f_W^{\vee} \left(\prod_{\{S \in Sub(R)\}} f_{S^{\wedge}}(R) \right)$$

In essence, the closure of a relation type is obtained by performing a recursive application of relation type extension (the new *B*-rule) with respect to all of its subtypes (i.e., upward propagation of arguments and properties), followed by a recursive application of relation type inheritance (Axiom 1) from all of its supertypes (i.e., downward propagation of arguments and properties).

The construction of a *meta-relation type closure function* can be performed similarly to the above. The building of those two type closure functions shows how Axiom 3 is satisfied in practice.

With regard to instance closure, it can be inferred from type closure through the *conf* function. In essence, for a relation or meta-relation r, its closure r^* is built by adding any extra arguments and properties obtained from the closure of *conf(r)* (i.e., *(conf(r))**), as per Axiom 4.

Example 13 (Relation Type Closure Construction): Suppose that we have the following three relation types in the ontology:

- *W = offends (Offender, Witness)*
- *R = steals (Thief, Victim)*
- *S = picksPocket (PickPocket, StolenAmount)*
  We have: *picksPocket < steals < offends*

The closure of each of the above relation types is constructed as follows, e.g., to construct the closure of *steals*, we first define its extension with respect to its subtype *offends*, then apply type inheritance from its supertype *picksPocket*, i.e.,

- $f(R) = R^* = f_W^{\vee}$ (*steals(Thief, Victim)*)) $= f_W^{\vee}$ (*steals^(Thief, Victim, StolenObject)*) $=$ *steals*(Thief, Victim, Witness, StolenObject)* (with *StolenObject* as a supertype of *StolenAmount*)
- $f(S) = S^* = f_W^{\vee}$ ($f_R^{\vee}$ (*picksPocket(PickPocket, StolenAmount)*)) $= f_W^{\vee}$ (*picksPocket*^{V}*(PickPocket, Victim, StolenAmount)*) $=$ *picksPocket*(PickPocket, Victim, Witness, StolenAmount)*
- $f(W) = W^* = f_S{}^{\wedge}$ ($f_R{}^{\wedge}$ (*offends(Offender, Witness)*)) $= f_S{}^{\wedge}$ (*offends^(Offender, Victim, Witness)*) $=$ *offends*(Offender, Victim, Witness, OffenceMotive)* (with *OffenceMotive* as a supertype of *StolenAmount*)

In the above, note that the order of arguments in the relation types, their intensional meanings, and the creation of new supertypes *StolenObject* and *OffenceMotive* are determined by the ontology designer and domain expert.

4.4 Final notes on ontology closure

1. As mentioned in Note 4 of Definition 4, whenever we refer to a relational object r, we means "r within the context of the ontology $\boldsymbol{O}$" and that context helps provide

the missing arguments to r, as well as other properties that r inherits through its relationships with other objects. Closure is in essence an attempt to merge the contextual information of the object into the definition of the object itself.

2. Semantically, there is no new information introduced by the notion of closure but the addition of all possible arguments and properties that a relation/meta-relation type/instance could use facilitates inference and search on knowledge bases.
3. Closure is a way in Conceptual Structure Theory to complete missing arguments in a relation type or instance, similar to the way Order-Sorted Logic supplements missing arguments in concept predicates (Kaneiwa 2004; Nitta 1995) in order to improve inference and reasoning.

5 Properties of new ontology formalism

5.1 Line of identities

In a previous example on subsumption, we said that if "John is a man", then "John is a man, a person, and a living entity". The set of concept types {Man, Person, LivingEntity} constitutes the *line of identities* of the concept "John". This notion can be extended to relation and meta-relation types and their instances.

Definition 6 (*Line of Identities*): A line of identities *Id* is a function between the set of individuals and the set of all subsets of types, i.e., Id: $\boldsymbol{I} \rightarrow 2^{TC} \cup 2^{TR} \cup 2^{TMR}$ defined by $\forall i \in I\ Id(i) = \{t \in \boldsymbol{T} \mid conf(i) < t\}$.

Notes:

1. Definition 6 means that for each individual i, its line of identities is the set of all supertypes of $conf(i)$. Based on the definition of infimum and *conf*, Definition 6 is equivalent to: $\forall i \in I\ conf(i) = infimum(Id(i))$, and also equivalent to the combination of the following three statements:

   (1) $\forall i \in I_C\ \forall C \in Id(c)\ conf(c) < C$
   (2) $\forall r \in I_R\ \forall R \in Id(r)\ conf(r) < R$
   (3) $\forall mr \in I_{MR}\ \forall MR \in Id(mr)\ conf(mr) < MR$

2. Line of identities is a feature that can be used to distinguish between a taxonomy and an ontology. In a taxonomy, the line of identities of any individual is a totally-ordered set (ordered by the subsumption relation) while in an ontology there may be individuals whose line of identities is a partially-ordered set (e.g., when an individual belongs to a type which has two different immediate supertypes, i.e., two different suprema or least upper bounds). In simple parlance, when applied to a human, we say in this case that the person has multiple personalities or identities.

5.2 Semantic equivalence

Semantic equivalence is a definition that concerns types only, not instances.

Definition 7 (*Semantic Equivalence*): Two types t and t' are said to be semantically equivalent (or simply equivalent) and written as $t \equiv t'$ if and only if $t < t'$ and $t' < t$.

Notes:

1. In the case of concept type, two equivalent concept types can be used interchangeably in the ontology, and said to be synonymous in common languages (e.g., Car and Automobile).
2. In the case of relation type, two equivalent relation types may not be straightforwardly interchangeable since there may be a difference in the number and/or in the order of their arguments. However, in natural languages, the two relation types "*marries*" and "*isMarriedTo*" can be used interchangeably in any context (e.g., "John marries Sue" and "Sue is married to John"). However, for these two relation types to be semantically equivalent (as per our mathematical definition), the order of their concept arguments must be defined consistently in the ontology, e.g., the ontology should have *marries(Husband, Wife)* and *isMarriedTo(Husband, Wife)*, and not *marries(Husband, Wife)* and *isMarriedTo(Wife, Husband)*. In general, the order of arguments in a relation or meta-relation type is arbitrarily chosen by the ontology designer, but it must be *consistently* chosen for all semantically-related relation or meta-relation types in the ontology. This ensures that the ontology is semantically as well as syntactically consistent.

Proposition 4 (Soundness of Semantic Equivalence): *For any two relations or meta-relations r and r' such that conf(r) and conf(r') are not semantically equivalent, there exist i, a component of the tuple B(r) (written as $i \in B(r)$), and i', a component of the tuple B(r'), such that conf(i) $\notin$ Id(i')* or *conf(i') $\notin$ Id(i) holds true.*

Proof This Proposition is equivalent to: $\forall r,r' \in I_R \cup I_{MR}\ \forall i \in B(r)\ \forall i' \in B(r')$ if $conf(i) \in Id(i')$ and $conf(i') \in Id(i)$ then we must have: $conf(i) \equiv conf(i')$. This is true because the antecedent implies that $Id(i) = Id(i')$ or $B(r) = B(r')$ and this in turn implies that $conf(i) \equiv conf(i')$.

Proposition 5 (*Soundness of Semantic Equivalence and Closure*): *For any two semantically equivalent relation or meta-relation types R_1 and R_2, their closures R_1* and R_2* are semantically equivalent and have the same arity.*

Proof This Proposition can be easily proved with the definitions of closure and semantic equivalence.

5.3 Semantic disjunction

Semantic disjunction is a property that concerns types and instances.

Definition 8 (*Semantic Disjunction*):

(1) Two types are said to be semantically disjoint if one does not subsume the other.

(2) Two types are said to be semantically **strictly** disjoint if one does not subsume the other and if their infimum is the Bottom type.
(3) Two instances are said to be semantically disjoint if the line of identities of any one instance is not entirely included in the line of identities of the other, i.e., $\forall i,i' \in I\ Id(i) \not\subset Id(i')$ and $Id(i') \not\subset Id(i)$.

Proposition 6 (Soundness of Semantic Equivalence and Disjunction):

(1) *For any two semantically equivalent types R_1 and R_2, their closures R_1* and R_2* are semantically equivalent and have the same arity.*
(2) *Two types are semantically disjoint if and only if they are not semantically equivalent.*
(3) *Two instances are semantically disjoint if and only if their conformances (i.e., their values through the function c) are not semantically equivalent.*
(4) *If two types R and R′ are semantically disjoint, then we have either: $\exists c \in B(R^*)$ such that $c \notin B(R'^*)$ or $\exists c' \in B(R'^*)$ such that $c' \notin B(R^*)$.*
(5) If two instances i and i' are semantically disjoint, then we have:

- $\exists j \in Id(i)$ *such that* $j \notin Id(i')$
- $\exists j' \in Id(i')$ *such that* $j' \notin Id(i)$
- $\exists j \in Id(i)\ \exists j' \in Id(i')$ *such that j and j′ do not subsume each other.*

Proof This Proposition can be easily proved with the definitions of closure, semantic equivalence and semantic disjunction.

Example 14 (Semantic Disjunction): Suppose that we define the relation types *flies, walks* and *moves* as follows:

- *flies (Plane)* (i.e., we restrict the act of flying to planes only)
- *moves(Person)* (i.e., we restrict the act of moving to people only)
- *walks(Person)* (i.e., we restrict the act of walking to people only)

Then the following pairs of relation types are semantically disjoint because their arguments are disjoint (although it may seem natural to think that *flies* < *moves*): {*moves, flies*} and {*walks, flies*}.

However, if our ontology is broadened to redefine the relation type *moves* as: *moves(Entity)* (i.e., anything (or any entity) can move), then we can have the following subsumption relations: *flies* < *moves* and *walks* < *moves* (this is because the new *B*-rule is satisfied in this case with *Plane* < *Entity* and *Person* < *Entity*).

Example 15 (Justice System with Relation Types): This is an example with a governmental justice administration system. Let us suppose that we have the following information, derived from facts and common findings:

1. Any offender would have a record with Police.
2. Children in a dysfunctional family are more likely to offend.
3. Children in a family whose parents are often absent are monitored by a welfare agency for possible assistance.

And suppose that we also have in our knowledge database the only piece of information concerning an adolescent named "John", that is "John's parents are in jail". We would like the system to answer the following queries:

1. Is John being monitored by a welfare agency?
2. Does John have a Police record?

To systematically answer the above queries, we would first attempt to build an ontology $\boldsymbol{O}$ as follows:

- $\boldsymbol{O} = (T, I, <, conf, B)$
- $T = T_C \cup T_R$
- $T_C = \{\text{Person, WelfareAgency, DysfunctionalFamily, FamilyWithParentInJail, Offence, PoliceRecord}\}$
- $T_R = \{\text{monitors, likelyCauses, hasAttribute}\}$
- $I = I_C \cup I_R$
- $I_C = \{\text{John}\}$
- $I_R = \emptyset$
- The function *conf* is defined by: $conf(John) = Person$
- The function B is defined as represented by the single arrows in Fig. 2

From the initial facts and common findings, we could define two additional relation types: *hasParentInJail* and *isInDysfunctionalFamily,* with a subsumption relation between them: $hasParentInJail < isInDysfunctionalFamily$. However, in order to simplify the ontology by avoiding having to introduce a meta-relation type between those two relation types, we could turn them into equivalent concept types: *DysfunctionalFamily* and *FamilyWithParentInJail*, as the choice of defining a new notion as a concept type or a relation type is arbitrary to the ontology designer. We then have a subsumption relation between those two new concept types: $FamilyWithParentInJail < DysfunctionalFamily$.

The final ontology could be represented as per Fig. 2 (using graphical representation similar to that for Conceptual Graphs (Sowa 1984, 2000)), in which:

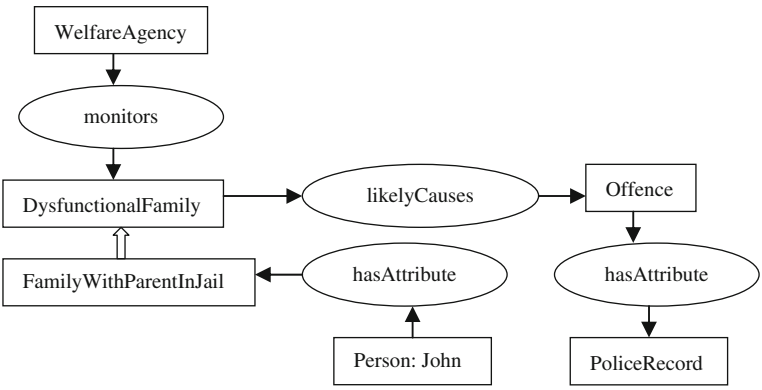

Fig. 2 Example of criminal justice ontology with relation types

- The relation type *likelyCauses* is a causal relation between the 2 new concept types: *likelyCauses (DysfunctionalFamily, Offence).*
- The relation type *hasAttribute* links the concept type *Offence* to the concept type *PoliceRecord* indicating that having a Police record is an attribute of committing an offence: *hasAttribute(Offence, PoliceRecord).*
- The relation type *monitors* links the concept type *WelfareAgency* to the concept type *DysfunctionalFamily* indicating that dysfunctional families are monitored by welfare agencies: *monitors(WelfareAgency, DysfunctionalFamily).*

Note that in all figures, rectangles represent concepts and concept types, ovals represent relation types, double ovals represent meta-relation types, dark arrows represent links expressed in the function *B,* and block arrows represent subsumption relations between types.

By navigating in the above ontology, we would find that the answer to question 1 is "yes" and that to question 2 is "likely".

Example 16 (Justice System with Meta-relation Types): The answer to the questions in Example 15 can also be achieved with an ontology formalization that includes meta-relation types. Such an ontology $\boldsymbol{O}$ is the same as in the above example except the following:

- $T = T_C \cup T_R \cup T_{MR}$
- $T_C = \{\text{Person, WelfareAgency, Offence, PoliceRecord}\}$
- $T_R = \{\text{hasParentInJail, isInDysfunctionalFamily, monitors, hasAttribute}\}$
- $T_{MR} = \{\text{likelyCauses, causes}\}$
- There are now two subsumption relations: hasParentInJail $<$ isInDysfunctionalFamily
- causes $<$ likelyCauses

In this ontology, note that:

- *likelyCauses* is now a meta-relation type, expressing a possible causal relation between a relation type and a concept type, i.e., *likelyCauses (isInDysfunctionalFamily, Offence).*
- *causes* is a new meta-relation type, expressing a definite causal relation between two relation types, i.e., *causes(isInDysfunctionalFamily, monitors).*
- The two new meta-relation types form the meta-relation type hierarchy of the ontology.
- *likelyCauses* is both a relation type (in Example 15) and a meta-relation type (in Example 16). The difference is in the types of their arguments.

By navigating in the ontology represented by Fig. 3, we would, of course, obtain the same answers as in Example 15.

The main difference between Figs. 2 and 3 is that reasoning in the latter follows more closely the semantics of the assertions, e.g., in Fig. 3, is expressed the assertion "being in a dysfunctional family likely causes an offence" while in Fig. 2, the assertion "a dysfunctional family likely causes an offence" is represented. This nuance may be of significance in some cases. For example, in a court of law, a defense statement such as "being in a disadvantaged family led to the offence" is

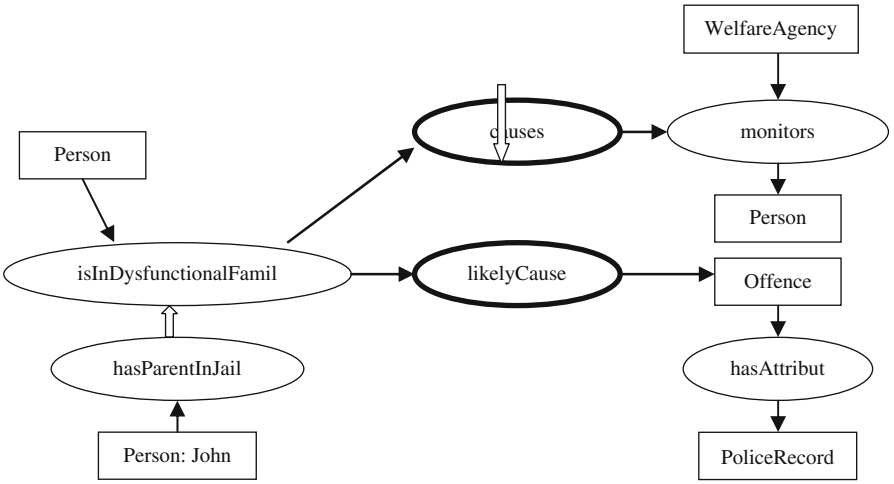

Fig. 3 Example of criminal justice ontology with meta-relation types

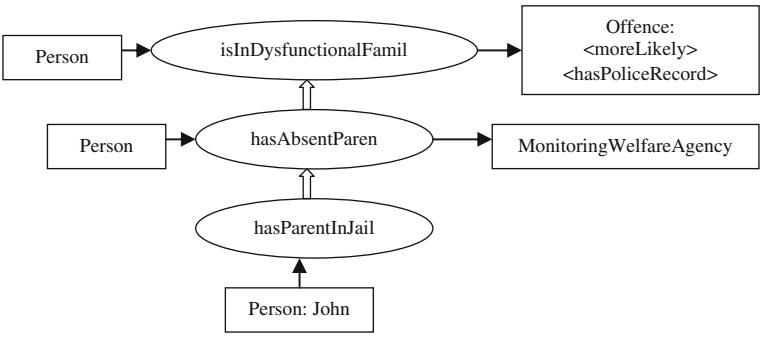

Fig. 4 Example of criminal justice ontology with relation closure

stronger than a statement such as "a disadvantaged family (of which the defendant is a member) led to the offence". The new formalism allows representation of both statements and is therefore more flexible.

Example 17 (Justice System with Relation Closure): This is another way to answer the same questions as in Example 15, by using relation type closure instead of meta-relation type. In this formalization, we first organize the general information into an ontology with the following 3 relation types and 1 concept type (Fig. 4):

- *hasParentInJail(Person)*
- *hasAbsentParent(Person, MonitoringWelfareAgency)*
- *isInDysfunctionalFamily(Person, Offence: <moreLikely>)*
- *Offence: <hasPoliceRecord>*

Note that *<moreLikely>* is now a *property* of the relation type *isInDysfunctionalFamily* and *<hasPoliceRecord>* is a *property* of the concept type *Offence*.

Semantically, we have the following subsumption relations between the above relation types: *hasParentInJail* < *hasAbsentParent* and *hasAbsentParent* < *isInDysfunctionalFamily*. From the information in the knowledge base, we also have the relation: *hasParentInJail (Person: John)*.

Based on Axioms 1, 3 and 4, we can deduce the following relation closure: *hasParentInJail*(Person: John, MonitoringWelfareAgency, Offence: <moreLikely> <hasPoliceRecord>)*. This new relation permits us to infer: "John is being monitored by a welfare agency, more likely to offend, and more likely to have a Police record". This answer is possible through the use of the ontology closure in our new formalism.

6 Conclusion and future work

This paper proposed an extension to the ontology formalization previously suggested for Conceptual Structure Theory, by integrating new ideas from Order-Sorted Logic and other logics. The enhanced formalism offers a flexible way to represent facts and assertions in the ontology.

Unlike OWL, our proposed ontology formalism contains multiple conceptual hierarchies enabling representation of complex relationships such as 'predicate of predicates' through *n*-ary relation type or meta-relation type. The expressive concepts are needed to deal with ontological relationships among any number of concepts and their predicates. In particular, the new notion of relation and meta-relation type closure enables completion of missing arguments in these types. The end result is the production of an 'ontology closure', which is sound for formal reasoning. Closure is in essence an attempt to merge contextual information of a relational object into the definition of the object itself, thus leading to a better and more precise expression of the semantics of the object. Based on such an ontology, we could answer queries concerning topics that are not explicitly present in the existing knowledge base.

Concerning future work, as mentioned in Sect. 3.4, the use of modal logic (expressing the notions of *necessity* and *possibility*) in conjunction with first-order logic, called *first-order modal logic*, could be explored to more accurately translate relations and meta-relations defined under our formalism into that new logic. Similarly, *first-order temporal logic* could also be explored to provide better translation of temporal constraints and temporal properties into first-order logic.

It would also be interesting to develop further detailed algorithms to automatically construct the closure of any type, and automatically determine the equivalence, disjointness, subsumption between any two concept, relation or meta-relation types.

Another research direction could be the application of the proposed ontology formalism to a particular domain of discourse, and compare its performance or usefulness against other formalisms. Further ontological properties may be derived from such a specific domain. A possible application could be in legal reasoning and criminal justice administration, in which an ontology and an inference engine could

be built on top of existing knowledge bases to profile individuals and/or to answer complex queries that currently could only be answered by human experts.

References

Bechhofer S et al (2004) OWL Web Ontology Language Reference Feb 2004, http://www.w3.org/TR/owl-ref/

Beierle C et al (1992) An order-sorted logic for knowledge representation systems. Artif Intell 55: 149–191

Breuker J, Elhag L, Petkov E, Winkels R (2002) Ontologies for legal information serving and knowledge management. In: Bench-Capon T et al (eds) Legal knowledge and information systems (Jurix 2002). IOS Press, Amsterdam, pp 73–82

Chen W, Kifer M, Warren D (1993) HiLog: a foundation for higher-order logic programming. J Log Program 15(3):187–230

Codd EF (1970) A relational model of data for large shared data banks. Commun ACM 13(6):377–387

Cohn AG (1989) Taxonomic reasoning with many sorted logics. Artif Intell Rev 3:89–128

Corbett D (2003) Reasoning and unification over conceptual graphs. Kluwer Academic Publishers, New York

Dillion T, Chang E, Hadzic M, Wongthongtham P (2008) Differentiating conceptual modelling from data modelling, knowledge modelling and ontology modelling and a notation for ontology modelling, 5th Asia-Pacific conference on conceptual modelling, Wollongong, Australia, Jan 2008

Greiner R, Darken C, Santoso N (2001) Efficient reasoning. ACM Comput Surv 33(1):1–30

ISO/IEC International Standard (2007) Information technology—common logic (CL): a framework for a family of logic-based languages, ISO/IEC 24707:2007(E), Oct 2007

Kaneiwa K (2001) An order-sorted logic with predicate hierarchy, eventuality and implicit negation, PhD thesis, Japan Advanced Inst. of Science and Technology, 2001

Kaneiwa K (2004) Order-sorted logic programming with predicate hierarchy. Artif Intell 158(2):155–188

Kaneiwa K, Tojo S (1999) Event, property and hierarchy in order-sorted logic. Proceedings of international conference on logic programming, Las Cruces, USA, pp 94–108

Kaneiwa K, Iwazume M, Fukuda K (2007) An upper ontology for event classifications and relations, 20th Australian joint conference on artificial intelligence, Dec 2007, Gold Coast, Australia, LNCS, Vol 4830, pp 394–403

Nguyen P, Corbett D (2006a) A basic mathematical framework for conceptual graphs. IEEE Trans Knowl Data Eng 18(2):261–271

Nguyen P, Corbett D (2006b) Building corporate knowledge through ontology integration. Pacific Rim Knowledge Acquisition Workshop (PKAW-06), 08-2006, Guilin, China, LNAI 4303, pp 223–229

Nguyen P, Corbett D (2007) A formalization of subjective and objective time ontologies. 3rd Australasian ontology workshop (AOW-07), Dec 2007, Gold Coast, Australia, CRPIT, Vol 85, pp 45–54

Nguyen P, Kaneiwa K, Corbett D, Nguyen MQ (2008) An ontology formalization of relation type hierarchy in conceptual structure theory. 21st Australasian conference on AI, Auckland, NZ, Dec 2008, LNAI 5360, pp 79–85

Nitta K et al (1995) New HELIC-II: A software tool for legal reasoning. 5th international conference on artificial intelligence and law, College Park, MD, ACM Press, pp 287–296

Smith P (2003) An introduction to formal logic. Cambridge University Press, Cambridge

Smith B et al (2005) Relations in biomedical ontologies. Genome Biol 6:46

Sowa J (1984) Conceptual structures—information processing in mind and machine. Addison-Wesley, Reading

Sowa J (2000) Knowledge representation: logical, philosophical, and computational foundations. Brooks Cole Publishing Co, Pacific Grove

Stumme G (2002) Using ontologies and formal concept analysis for organizing business knowledge. In: Becke J, Knackstedt R (eds) Wissensmanagement mit Referenzmodellen—Konzepte für die Anwendungssystem und Organisationsgestaltung. Physica, Heidelberg, pp 163–174

Wille R (1982) Restructuring lattice theory: an approach based on hierarchies of concepts. In: Rival I (ed) Ordered sets. Reidel, Dordrecht

World Wide Web Consortium (2004) OWL web ontology language—use cases and requirements, 2004, http://www.w3.org/TR/webont-req/