Ontologies for the Semantic Web in CASL

Klaus Lüttich, Till Mossakowski and Bernd Krieg-Brückner

BISS, FB3 – Dept. of Computer Science, Universität Bremen
{luettich,till,bkb}@tzi.de

Abstract. This paper describes a sublanguage of CASL, called CASL-DL, that corresponds to the Web Ontology Language (OWL) being used for the semantic web. OWL can thus benefit from CASL's strong typing discipline and powerful structuring concepts. Vice versa, the automatic decision procedures available for OWL DL (or more precisely, the underlying description logic $\mathcal{SHOIN}(\mathbf{D})$) become available for a sublanguage of CASL. This is achieved via translations between CASL-DL and $\mathcal{SHOIN}(\mathbf{D})$, formalized as so-called institution comorphisms.

1 Introduction

The internationally standardized Web Ontology Language (OWL) [10] is a major contribution to the upcoming Semantic Web [11, 5] that proposes a new form of web content meaningful to computers. One problem of the documents on the web is the restricted ability to search for certain topics without any knowledge how an author or organisation names the concept. Another problem results form multimedia files like audio or movie files which cannot be indexed by techniques available today; the meaning must be given by meta data. However, just giving a text describing a piece of multimedia yields only a very limited aid for searching.

Therefore, the W3C (World Wide Web Consortium) and Tim Berners Lee proposed the Semantic Web, where the meaning is given by shared and extended ontologies that provide organised knowledge about certain domains; thus the contents of the web is accessible by computers. Hence, it becomes possible e.g. to search for the least cost of a phone call from Singapore to Germany. The visitor from Europe does not need any knowledge of the foreign language, because the query is given in a semantic-based language that is also provided by the Singapore telephone company. Indeed, with Swoogle [1], a first search engine for OWL and RDF documents is available.

In this work, we interface OWL with the specification language CASL [6, 9]. CASL provides a strong typing discipline, which allows to find conceptual errors at an early phase. Moreover, powerful structuring constructs allow the modularization of large theories into manageable pieces. Both features are present in OWL in a very limited form only. We hence propose a sublanguage of CASL, called CASL-DL, which corresponds to OWL DL in expressive power, but which retains the above mentioned advantages. CASL-DL can also be used to interface CASL with efficient decision procedures that are available for description logics.

The paper is organised as follows: Section 2 recalls the underlying description logic $\mathcal{SHOIN}(\mathbf{D})$ of OWL DL. Section 3 describes the Web Ontology Language OWL DL. Section 4 introduces CASL and the sublanguage CASL-DL. Section 5 continues with translations between OWL DL and CASL-DL. Section 6 concludes the paper. Last but not least an appendix collects some tables showing the concrete translations between $\mathcal{SHOIN}(\mathbf{D})$ and CASL-DL constructs including semantics for the $\mathcal{SHOIN}(\mathbf{D})$ constructs.

2 $\mathcal{SHOIN}(\mathbf{D})$

$\mathcal{SHOIN}(\mathbf{D})$ is an expressive description logic [3, 19, 18]. Its main purpose is the definition of hierarchies of concepts and roles. In terms of logic, concepts are unary and roles are binary predicates. The general properties of concepts and roles are collected in a so-called TBox. By contrast, the ABox represents a particular database, i.e. defines individuals to belong to concepts and roles. It also defines concepts and roles involving predefined datatypes. See Fig. 1 for an example of a TBox describing the class definitions of a family.

$$
\begin{aligned}
Woman &\equiv Person \sqcap Female \\
Man &\equiv Person \sqcap \neg Woman \\
Mother &\equiv Woman \sqcap \exists hasChild.Person \\
Father &\equiv Man \sqcap \exists hasChild.Person \\
Parent &\equiv Mother \sqcup Father \\
Grandmother &\equiv Mother \sqcap \exists hasChild.Parent \\
MotherWithManyChildren &\equiv Mother \sqcap \geqslant 3\ hasChild \\
Wife &\equiv Woman \sqcap \exists hasHusband.Man
\end{aligned}
$$

Fig. 1. Example TBox: Family

The standard description logic that is the base of all description logics is called $\mathcal{ALC}$. $\mathcal{ALC}$ has a notation for the universal concept $\top$ and the bottom concept $\bot$ (representing the always true and the empty predicate). Moreover, new concepts can be built with unions, intersections and complements of concepts. Finally, concepts can be universally or existentially projected along roles (e.g. $\exists hasChild.Person$ means the concept that consists of all individuals having some person as their child).

The logic $\mathcal{ALC}_{R^+}$ adds the possibility to specify roles to be transitive. This logic is also abbreviated by $\mathcal{S}$, which is the first letter of the name $\mathcal{SHOIN}(\mathbf{D})$. Likewise, the other letters are used for various features of description logics [3, pp.494-495]. The letter $\mathcal{H}$ adds role hierarchies (i.e., the possibility to specify inclusions between roles) and the letter $\mathcal{I}$ adds inverse roles (i.e. the possibility to generate a new role by just swapping the arguments of a given role). Unqualified number restrictions and nominals are added by $\mathcal{N}$ and $\mathcal{O}$. The former allow

stating that any individual is related to at most (or at least) n individuals by a given role. This way, the functionality of relations can be specified. Nominals allow for explicitly enumerating the members of a concept. Datatypes are added by the suffix (**D**). The order of the letters does not really matter, so $\mathcal{SHOIN}(\mathbf{D})$ is sometimes called $\mathcal{SHION}(\mathbf{D})$. A $\mathcal{SHOIN}(\mathbf{D})$ axiom is either an equality or inclusion between concepts, or an inclusion between roles. A $\mathcal{SHOIN}(\mathbf{D})$ TBox consists of a set of $\mathcal{SHOIN}(\mathbf{D})$ axioms. The semantics of such a TBox is the set of all single-sorted first-order models interpreting concepts as unary and roles as binary relations, and satisfying the TBox axioms (see Figs. 12 and 13 for the $\mathcal{SHOIN}(\mathbf{D})$ syntax and Tables 1 to 7 for its semantics).

2.1 Tools for $\mathcal{SHOIN}(\mathbf{D})$

A crucial motivation for the use of description logics is that they usually correspond to decidable fragments of first-order logic. Indeed, there is a decision procedure for satisfiability and subsumption of $\mathcal{SHIN}(\mathbf{D})$ concepts ($\mathcal{SHIN}(\mathbf{D})$ is $\mathcal{SHOIN}(\mathbf{D})$ without nominals) that runs in non-deterministic exponential time, but performs much better for practical examples [24]. This has been the basis of efficient tools for OWL DL such as FaCT [17]. The tool Pellet [23] uses a combination of known algorithms for $\mathcal{SHIN}(\mathbf{D})$ and $\mathcal{SHON}(\mathbf{D})$ ($\mathcal{SHOIN}(\mathbf{D})$ without inverse properties). It is provably sound but incomplete with respect to the whole of $\mathcal{SHOIN}(\mathbf{D})$. The design of a decision procedure for the whole of $\mathcal{SHOIN}(\mathbf{D})$ is an open problem.

The Guarded Fragment of first-order logic has attracted much attention since it has shown to be decidable [2]. Note that many description logics (as well as propositional modal logic) can be translated into the guarded fragment. However, this does not hold for $\mathcal{SHOIN}(\mathbf{D})$ due to the presence of transitive roles and counting. Indeed, adding either of these features to the guarded fragment results in an undecidable logic.

3 OWL DL

OWL is not a single language, it has three sublanguages that can be ordered according to their expressiveness. The underlying idea of the W3C was to provide languages that were very expressive on the one hand but also useful in automated reasoning processes useful for the Semantic Web. The following species of OWL are available, starting with the most expressive language:

OWL Full provides unrestricted access to all OWL constructs. As RDF Schema, it does not enforce a strict separation of classes, properties, individuals and data values. Hence, there are e.g. no constraints on using a concept, called class in OWL, as an individual at the same time [10, Sect.8.1]. This corresponds to some untyped higher-order logic, leading to non-well-founded sets as semantics.

OWL DL restricts the constructs of OWL in such a way that they correspond to some fragment of first-order logic (actually, roughly to $\mathcal{SHOIN}(\mathbf{D})$). In particular, all axioms must form a tree-like structure. This means e.g. that every reference to a name in a "subclass of" axiom implies the presence of a declaration that this name refers to a class.

OWL Lite adds further constraints to OWL DL that lead to a straight-forward and easy implementation (whereas OWL DL tries to reach the very limits of description logics). For example, it disallows nominals (oneOf, hasValue), union and negation of class descriptions.

Note that neither OWL DL nor OWL Full provide full quantifier logic, but only some restricted forms of quantification corresponding to description logics. We will work with OWL DL here, since it comes closest to the typed first-order fragment of OWL, and we aim at a translation to the typed first-order language CASL. Moreover, apart from the need to involve non-well-founded sets, OWL Full has the additional drawback that no worked-out formal semantics exists, to our knowledge.

3.1 Classes (Concepts) in OWL DL

OWL DL can be understood as syntactic sugar on top of $\mathcal{SHOIN}(\mathbf{D})$, with a slightly new terminology. Concepts are called classes in OWL DL. The universal concept $\top$ is named class *Thing*, the empty concept $\bot$ is named class *Nothing*. New classes are introduced with either complete or partial descriptions. Complete descriptions are introduced by axioms stating equivalence of classes (=equality of concepts), while partial descriptions only specify a subclass (=subconcept) relation to a given class, which means that they are quite loose. Furthermore, the "one of" class axiom gives a complete definition by enumerating all individuals belonging to this class. Additionally, classes can be specified to be disjoint with other classes.

Classes can also be specified via restrictions. Cardinality restrictions specify a lower or upper bound or an exact number of properties that must be present. More precisely, this means that an individual belongs to such a class if and only if the number of individuals that it is related to (via some property, which is a binary relation) meets the specified bounds. "All values from" restrictions allow restricting the values that belong to a class or a role to come from another given class. It is also possible to demand a property to be present in relation to another class. A further restriction defines a class by having a property with (=relation to) a certain value. These restrictions are possible with object properties and datatype properties (see Sect. 3.2 and 3.3).

3.2 Properties (Roles) in OWL DL

Binary relations between individuals, called roles in $\mathcal{SHOIN}(\mathbf{D})$, are called properties in OWL DL. They are divided into two types: Datatype properties relate classes to datatypes (see Sect. 3.3 for the allowed datatypes), while object

properties relate classes with classes. The first type of properties can only have other datatype properties as super-properties. Both types of properties can be restricted to a certain domain and range. Without the definition of a domain and range, every class can be related with every other class or datatype by a given property. By giving a domain and range, the property looses this overloadingpossibility. Another way of restricting a property to a certain datatype or class is possible by giving an "all values from" class axiom.

Datatype properties can be defined as the sub-property of and as equivalent to another datatype property. Furthermore, it is possible to specify a datatype property to be functional.

Object properties can have the same axioms regarding subsumption, equivalence and functionality as datatype properties. Additionally, properties can be defined to be symmetric, transitive, functional, inverse functional, or the inverse of another property. OWL also has annotation properties, which have no semantic meaning given by the OWL recommendation.

3.3 XML Schema Datatypes in OWL DL

OWL DL treats the datatypes allowed in $\mathcal{SHOIN}(\mathbf{D})$ somewhat differently than $\mathcal{SHOIN}(\mathbf{D})$. Datatypes are restricted to some XML Schema Datatypes. These are numeric datatypes for integers and various subsets of the integers and for decimal, float and double numbers. Strings and various specialized versions of strings are allowed as well as base64 and hexadecimal encoded binary data. Finally, various time and date specific datatypes are allowed. There is no way of defining further datatypes than those listed in [22, Sect.2.1].

3.4 Facts

Axioms describing the membership of individuals in classes and describing relations between individuals are called facts in OWL DL. In description logics, a set of such axioms is often called an ABox (Fig. 2). It is possible to state that an individual belongs to several classes. Implicitly, every individual is of class *Thing*, because this is the implicit maximal superclass. For properties, it is possible to provide either datatype or object values that are related. Finally, it is possible to state that several names denote different individuals, or the same individual.

Mother(MARY)	Father(PETER)
hasChild(MARY,PETER)	hasChild(PETER,HARRY)
hasChild(MARY,PAUL)	

Fig. 2. Example ABox: Family individuals

4 CASL-DL

CASL, the *Common Algebraic Specification Language* [9], has been designed by CoFI, the *Common Framework Initiative* for algebraic specification and development. It has been designed by a large number of experts from different groups, and serves as a de-facto standard. The design of CASL has been approved by the IFIP WG 1.3 "Foundations of System Specification".

CASL consists of several major *levels*[1], which are quite independent and may be understood (and used) separately:

Basic specifications are written in many-sorted first-order logic. Subsorts (interpreted as injective embeddings) increase flexibility, while retaining a strong type system. Partial functions with possibly undefined values are distinguished from total functions. Finally, CASL basic specifications provide powerful and concise constructs for specifying datatypes, which, in the presence of subsorts, may also be used to specify disjoint, non-disjoint and exhaustive unions of sorts.

Structured specifications allow translation, reduction, union, and extension of specifications. A simple form of generic (parameterized) specifications is provided, allowing specifications to be re-used in different contexts.

Libraries allow the distributed storage and retrieval of (particular versions of) named specifications.

Major libraries of validated CASL specifications are freely available on the Internet, and the specifications can be reused simply by referring to their names. Tools are provided to support the practical use of CASL: checking the correctness of specifications, proving facts about them, etc.

While CASL has originally been designed for specifying requirements and design of software, we show here that CASL is also perfectly suited as a language for formalizing ontologies. In particular, we propose to use CASL sorts and subsorts for the development of a class hierarchy, and CASL binary predicates with axioms for the development of properties. This leads to a cleaner methodology for developing ontologies. With the aid of "strongly typed" ontologies it will be easier to avoid inconsistencies.

The sublanguage of CASL corresponding to $\mathcal{SHOIN}(\mathbf{D})$, called CASL-DL, is described as a syntactic restriction w.r.t. CASL. It contains sorts, subsorts, free and generated data types. The sorts *Thing* and *DATA* are assumed to be present in all signatures, and each sort must be a subsort of either of these. Figure 3 presents all declarations of sorts, predicates and functions possible in CASL-DL. Disjointness of concepts is defined easily with free type definitions with subsorts in CASL. Predicates are restricted to unary and binary predicates, where binary predicates relate *Thing* either with *Thing* or with *DATA* or relate subsorts of *Thing* with subsorts of *Thing* or *DATA*. Overloading is allowed, but the first position of the predicate must be a subsort of *Thing* and the second position must be filled either with a subsort of *Thing* or with one of the predefined

[1] We omit CASL architectural specifications, since these seem not so relevant here.

sorts	$S_1, ..., S_n < S$	(subsort declaration)
sorts	$S_1 = ... = S_n$	(sort equivalence)
sort	$S_1 = \{ x : S \bullet \phi(x)\}$	(subsort definition)
sort	$D_{dr} = \{ x : D \bullet x = v_1 \vee ... \vee x = v_n\}$	(data range subsort definition)
generated type	$S ::= o_1 \mid ... \mid o_n$	(generated sort)
free type	$S ::= o_1 \mid ... \mid o_n$	(enumerated free sort)
generated type	$S ::= sorts\ S_1, ..., S_n$	(generated sort)
free type	$S ::= sorts\ S_1, ..., S_n$	(free sort)
ops	$o_1, ..., o_n : Thing$	(function declaration)
ops	$o_1, ..., o_n : S$	(function declaration)
ops	$f_1, ..., f_n : S \rightarrow ?\ SD$	(partial function declaration)
preds	$P_1, ...P_n : Thing$	(predicate declaration)
preds	$R_1, ..., R_n : S \times SD$	(predicate declaration)

where S_i are subsorts of *Thing* and S is *Thing* or a subsort of it, and
where $o_1, ..., o_n$ are constant operations, and
where SD stands for either *Thing* or *DATA* or a subsort of these
where D stands for a subsort of *DATA*

Fig. 3. Declarations, sort and type definitions of CASL-DL

datatypes that are subsorts of *DATA*. Figure 5 shows some of the predefined data types in CASL-DL. Only partial functions with one argument of sort *Thing* (or a subsort of it) and with result of either type *Thing* or *DATA* (or a subsort of these) and 0-ary (constant) functions of type *Thing* (or typed with a subsort of it) are allowed. Formulas for binary predicates are restricted to those given by Fig. 4, with the further restriction that axioms for functional predicates cannot be combined with the transitivity axiom. For predicates and functions which relate to *DATA* only equivalence and implication axioms are allowed.

Descriptions are formulas that restrict the extension of a description, set-like combinations of descriptions or assertions of membership in a named concept as shown in Fig. 6. The formulas regarding the cardinality of predicates are obtained through the instantiation of the predefined parameterized specification GENCARDINALITY (s. Fig. 10). Special axioms restricting the sort *DATA* and subsorts of it are presented in Fig. 7. Implicit embedding to supersorts and explicit downcast to subsorts in terms are allowed. Also, terms with nested function symbols are supported. Figure 8 shows all axioms of CASL-DL built upon descriptions. Finally axioms relating constant operations are described in Fig. 9.

To illustrate the way in which we would like to write ontologies in CASL-DL, we give a short example in Fig. 11. Actually, it has been obtained as a result of a translation described in the next section; a direct specification in CASL-DL may be more concise at some places, e.g.

$$\textbf{sort}\ Mother = \{x : Woman \bullet \exists y : Person \bullet hasChild(x, y)\}.$$

The example shows the definition of the concept Child that has exactly two parents. In description logics, so-called cardinality constraints are often used to denote this. In CASL-DL, a (predefined) generic specification is used to introduce

$\forall x : S_1; y : S_3 \bullet R(x, y) \Rightarrow Q(x, y)$	(implication)
$\forall x : S_1; y : S_3 \bullet R(x, y) \Leftrightarrow Q(x, y)$	(equivalence)
$\forall x : S_1; y : S_2 \bullet R(x, y) \Leftrightarrow R(y, x)$	(symmetry)
$\forall x : S_1; y : S_2 \bullet R(x, y) \Leftrightarrow Q(y, x)$	(inverse)
$\forall x, y : S_1; z : S_2 \bullet R(x, z) \wedge R(y, z) \Rightarrow x = y$	(inverse functional)
$\forall x : S_1; y : S_2 \bullet R(x, y) \Rightarrow \phi(x) \wedge \psi(y)^a$	(argument restriction)
$\forall x : S_1; y : S_4 \bullet R(x, y) \Rightarrow \phi(x) \wedge \delta(y)^a$	(argument restriction)
$\forall x : S_1; y : S_3 \bullet f(x) = y \Rightarrow Q(x, y)$	(implication)
$\forall x : S_1; y : S_3 \bullet f(x) = y \Rightarrow g(x) = y$	(implication)
$\forall x : S_1; y : S_3 \bullet f(x) = y \Leftrightarrow Q(x, y)$	(equivalence)
$\forall x : S_1; y : S_3 \bullet f(x) = y \Leftrightarrow g(x) = y$	(equivalence)
$\forall x : S_1; y : S_2 \bullet f(x) = y \Leftrightarrow f(y) = x$	(symmetry)
$\forall x : S_1; y : S_2 \bullet f(x) = y \Leftrightarrow Q(y, x)$	(inverse)
$\forall x : S_1; y : S_2 \bullet f(x) = y \Leftrightarrow g(y) = x$	(inverse)
$\forall x, y : S_1; z : S_2 \bullet f(x) = z \wedge f(y) = z \Rightarrow x = y$	(inverse functional)
$\forall x, y : S_1; z : S_2 \bullet R(x, y) \wedge R(y, z) \Rightarrow R(x, z)$	(transitivity)
$\forall x : S_1; y : S_2 \bullet f(x) = y \Rightarrow \phi(x) \wedge \psi(y)^a$	(argument restriction)
$\forall x : S_1; y : S_4 \bullet f(x) = y \Rightarrow \phi(x) \wedge \delta(y)^a$	(argument restriction)

where S_1, S_2 is either *Thing* or a subsort of it, and
where S_3 is either *DATA* or *Thing* or a subsort of these
where S_4 is either *DATA* or a subsort of it

[a] one of the conjuncts may be omitted

Fig. 4. Predicate axioms in CASL-DL

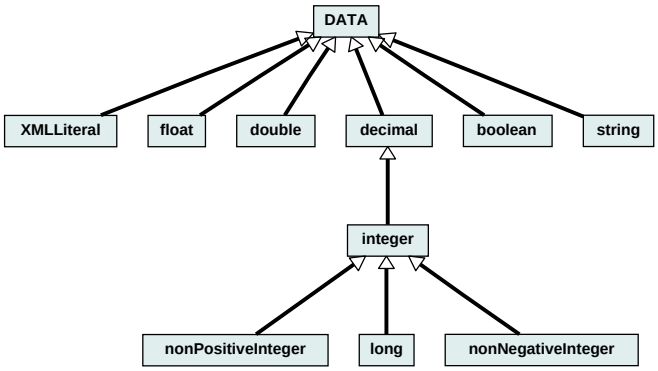

Fig. 5. Some of the predefined datatypes in CASL-DL and their subsort relations

$\phi(x), \psi(x) ::=$	$true \mid false \mid$	
	$x \in S \mid$	(sort membership)
	$P(x) \mid$	(concept membership)
	$\neg\phi(x) \mid$	(description negation)
	$\phi(x) \wedge \psi(x) \mid$	(description union)
	$\phi(x) \vee \psi(x) \mid$	(description intersection)
	$\exists y : S \bullet R(x,y)^a \mid$	(existential quantification)
	$\exists y : S \bullet R(x,y) \wedge \phi(y)^a \mid$	(existential quantification)
	$\forall y : S \bullet R(x,y) \Rightarrow \phi(y)^a \mid$	(value restriction)
	$R(x,o)^b \mid$	(has value restriction)
	$R(x,v)^c \mid$	(has value restriction)
	$x = o_1 \vee ... \vee x = {o_n}^b \mid$	(one of restriction)
	$minCardinality[P](x,n)^d \mid$	(cardinality restriction)
	$maxCardinality[P](x,n)^d \mid$	(cardinality restriction)
	$cardinality[P](x,n)^d \mid$	(cardinality restriction)
	$\exists y : D \bullet U(x,y)^e \mid$	(existential quantification)
	$\exists y : DATA \bullet U(x,y) \wedge \delta(y)^e \mid$	(existential quantification)
	$\forall y : DATA \bullet U(x,y) \Rightarrow \delta(y)^e \mid$	(value restriction)
	$\phi(f(x)) \mid$	(existential quantification)
	$def\, f(x) \Rightarrow \phi(f(x))$	(value restriction)

[a] where S is either *Thing* or a subsort of it
[b] where o, o_i is an individual, aka ground term, of sort *Thing*
[c] where v is a ground term of sort *DATA*
[d] where n is a ground term of sort *Nat*
[e] where D is either *DATA* or a subsort of it

Fig. 6. Formulas for Descriptions in CASL-DL

$\delta(x) ::= x \in D \mid$		(datatype membership)
	$x = v_1 \vee ... \vee x = v_2$	(one of)

where D is a subsort of *DATA* and
where $v_1 ... v_n$ are ground terms of *DATA*

Fig. 7. Formulas for *DATA* in CASL-DL

$\forall x : Thing \bullet \phi(x) \Rightarrow \psi(x)$	(partial definition)
$\forall x : Thing \bullet \phi(x) \Leftrightarrow \psi(x)$	(complete definition)
$\forall x : S \bullet \psi(x)$	(complete definition)

where S is a subsort of *Thing*

Fig. 8. Description axioms allowed in CASL-DL

$\bullet R(o_1, o_2)$ (predicate application)
$\bullet R(o, v)$ (predicate application)
$\bullet f(o_1) = o_2$ (function application)
$\bullet f(o) = v$ (function application)
$\bullet o \in S$ (membership in sort)
$\bullet \phi(o)$ (description of o)
$\bullet o_1 = o_2$ (same object)
$\bullet \neg\ o_1 = o_2$ (different object)

where o_i and o denote 0-ary constant operations of sort *Thing* or a subsort of it, and
where v is a ground term of sort *DATA*, and
where S is a subsort of *Thing*

Fig. 9. Axioms Relating Constants

cardinality predicates for a given predicate. So, the instantiation of GenCardinality with **pred** *hasChild* yields a new predicate *cardinality*[*hasChild*](p, n) that holds if and only if the Person p is exactly related to n Persons with the predicate *hasChild*.

spec GenCardinality [**sorts** *Subject*, *Object*
pred *predicate* : *Subject* × *Object*] =
{ Set [**sort** *Object*]
reveal *Set*[*Object*], ♯__, __ϵ__, *Nat*, *0*, *1*, *2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*, __@@__,
__≥__, __≤__
then op *toSet* : *Subject* → *Set*[*Object*]
∀ *x*: *Subject*; *y*: *Object* • *predicate*(*x*, *y*) ⇔ *y* ϵ *toSet*(*x*)
preds *minCardinality*[*predicate*](*s*: *Subject*; *n*: *Nat*) ⇔ ♯ *toSet*(*s*) ≥ *n*;
maxCardinality[*predicate*](*s*: *Subject*; *n*: *Nat*) ⇔ ♯ *toSet*(*s*) ≤ *n*;
cardinality[*predicate*](*s*: *Subject*; *n*: *Nat*) ⇔ ♯ *toSet*(*s*) = *n*
} **hide** *Pos*, *toSet*, *Set*[*Object*], ♯__, __ϵ__, __≤__, __≥__

spec PredefinedConcepts =
sort *Thing*
pred *Nothing* : *Thing*
∀ *x*: *Thing* • ¬ *Nothing*(*x*) %(empty_concept_Nothing)%

Fig. 10. Casl-DL Prelude

5 Translations Between Casl-DL and OWL DL

This paper provides a translation of OWL DL documents to Casl-DL and vice versa, in a way that an OWL DL ontology is optimised in the sense of

```
spec FAMILY =
     PREDEFINEDCONCEPTS
then sorts  Person, Female < Thing;
            Woman < Person;
            Woman < Female;
            Woman = {x: Thing • x ∈ Person ∧ x ∈ Female};
            Man < Person;
            Man = {x: Thing • x ∈ Person ∧ ¬ x ∈ Woman}
     pred   hasChild : Person × Person
     sorts  Mother < Woman;
            Mother = {x: Thing • x ∈ Woman ∧
                        ∃ y: Person • hasChild(x as Woman, y)};
            Father < Man;
            Father = {x: Thing • x ∈ Man ∧
                        ∃ y: Person • hasChild(x as Man, y)};
            Parent < Person %% smallest common supersort
     generated type Parent ::= sorts Mother, Father
     sorts  Grandmother < Mother;
            Grandmother = {x: Thing • x ∈ Mother ∧
                              ∃ y: Parent • hasChild(x as Mother, y)}
then GENCARDINALITY [sort Person < Thing
                     pred hasChild : Person × Thing]
then sorts  MotherWithManyChildren < Mother;
            MotherWithManyChildren =
                  {x: Thing • (x ∈ Mother) ∧
                    minCardinality[hasChild](x as Mother, 3)}
     preds  hasHusband : Woman × Man;
            Wife(w: Person) ⇔
            w ∈ Woman ∧ ∃ m: Man • hasHusband(w as Woman, m)
end
```

Fig. 11. Translation of the OWL-DL TBox from Fig.1 into CASL-DL

CASL's strong typing. By relying on the translations between OWL DL and $\mathcal{SHOIN}(\mathbf{D})$ from [18, 19], we just need to define the translation along the syntax of $\mathcal{SHOIN}(\mathbf{D})$. The remaining OWL DL specialties are dealt with as follows. An ontology imported via an import annotation is translated to a specification in CASL-DL and named with a part of the URI [4] of the OWL document. Then the "importing ontology" is an extension of the imported ontology. When defining the translation, we encountered the difficulty that in every OWL document other ontologies could be referenced directly as an URI pointing to a name defined in another OWL document. This problem is solved by localisation of the names as described in the semantics document of OWL [22, Sect.5.1]. The URIs are maintained via custom namespace annotations analogous to XML namespaces [7].

Figures 12 and 13 show the syntax production rules for the constructs of $\mathcal{SHOIN}(\mathbf{D})$. Tables 1 to 7 (see the appendix) describe in detail the $\mathcal{SHOIN}(\mathbf{D})$ constructs, their semantics (according to standard description logic semantics

$$
\begin{array}{rll}
C, D ::= & A \mid & \text{(atomic concept)} \\
& \top \mid & \text{(universal concept)} \\
& \bot \mid & \text{(bottom concept)} \\
& \neg A \mid & \text{(atomic negation)} \\
& C \sqcap D \mid & \text{(intersection)} \\
& C \sqcup D \mid & \text{(union)} \\
& \{o_1, ...\} \mid & \text{(nominals)} \\
& \forall R.C \mid & \text{(value restriction)} \\
& \exists R.C \mid & \text{(existential quantification)} \\
& R : o \mid & \text{(has value restriction)} \\
& \geqslant n\ R \mid & \text{(Unqualified} \\
& \leqslant n\ R \mid & \text{number} \\
& = n\ R & \text{restriction)}
\end{array}
$$

Fig. 12. Syntax for the Description of Concepts in $\mathcal{SHOIN}(\mathbf{D})$

$$
\begin{array}{rll}
R, S ::= & S \mid & \text{(atomic role)} \\
& R^- & \text{(inverse role)}
\end{array}
$$

Fig. 13. Syntax for the Description of Roles in $\mathcal{SHOIN}(\mathbf{D})$

as given by [19]), and their translation to CASL-DL. Here, $\Delta^{\mathcal{I}}$ is the domain of individuals disjoint from the domain of data values $\Delta^{\mathcal{I}}_D$. All the sorts are subsorts either of *DATA* (for datatypes), or of *Thing*. Class descriptions denoted as C or $C_1 ... C_n$ are translated to formulas with one free variable in CASL-DL. $[\![C]\!](x)$ is the translation of description C in CASL-DL with the free variable x.

In principle, there are two variants of axioms: (1) partial concept axioms and (2) complete concept axioms, presented in Table 1. Table 2 shows axioms for some special cases that can be translated more succinctly (but note that these translations are semantically equivalent to those that could be derived from Tables 1 and 3). For example, a class axiom that defines a named concept can be translated either to a formula of the form $x \in A \Leftrightarrow x \in C_1 \wedge ... \wedge x \in C_n$, or to a subsort definition (where A stands for the name that is given to the class in definition). All variables in the tables without an explicit typing are of type *Thing*.

The translation of OWL DL to CASL-DL is not entirely adequate in the sense that OWL DL admits empty classes, while subsorts in CASL-DL must be non-empty. However, note that *unnamed* classes are translated to formulas in CASL-DL, and the latter may well be unsatisfiable, i.e. denoting the empty class. Only *named* classes are translated to subsorts. Generally, we feel that named classes should be non-empty from a conceptual point of view. However, for the case that the user really wants to have a possibly empty class, we provide an annotation in OWL DL that leads to a translation of the class to a unary predicate in CASL-DL.

The translation from CASL-DL back to $\mathcal{SHOIN}(\mathbf{D})$ (OWL DL) results from the same tables given for the translation from $\mathcal{SHOIN}(\mathbf{D})$ to CASL-DL by reading them from right to left. Of course, this is not well-defined in cases where a certain CASL-DL construct is reached (via the left-to-right translation) either

from several $\mathcal{SHOIN}(\mathbf{D})$ constructs, or from none at all. In the first case, we just define the back translation to be the first $\mathcal{SHOIN}(\mathbf{D})$ construct in the table that fits. The second case can occur only for a limited number of constructs, for which we employ a special treatment: (1) Overloaded binary predicates are translated a bit differently from non-overloaded ones. The argument sorts yield two constructs of the form $S_1 \sqcup ... \sqcup S_n$ built from the sorts for the first and second argument. These constructs are then used in $\mathcal{SHOIN}(\mathbf{D})$ formulas for domain and range formulas (s. Tables 5 and 6). (2) The definitions for named classes resulting from unary predicates are marked with an OWL DL annotation. (3) Complete definitions of the form $\forall x : S \bullet \ \phi(x)$ are transformed to $\forall x : \mathit{Thing} \bullet \ x \in S \Leftrightarrow \phi(x)$ and this formula is then translated. (4) Subsort definitions of the form **sort** $S_1 = \{ \ x : S \bullet \phi(x)\}$ are transformed to **sort** $S_1 = \{ \ x : \mathit{Thing} \bullet \neg x \in S \vee \phi(x)\}$ before the translation.

5.1 Translations as Comorphisms

The translations can be shown to be institution comorphisms in the sense of [13]. First, we present briefly the involved signatures, sentences, models and satisfaction of $\mathcal{SHOIN}(\mathbf{D})$ and CASL-DL, giving rise to institutions in the sense of [14]. Then we show the signature, sentence and model mappings, giving rise to institution comorphisms.

The signatures in $\mathcal{SHOIN}(\mathbf{D})$ consist of several sets: (1) a set of concepts, (2) a subset of primary concepts, (3) a set of individual-valued roles, (4) a set of data-valued roles, (5) a set of individuals and (6) a set of axioms for subconcept relations, domain and range of roles, functional roles and concept membership. Productions of $\mathcal{SHOIN}(\mathbf{D})$ concepts are shown in Fig. 12 and 13. They are used to form sentences like $A \equiv C$ or $A \sqsubseteq C$, where A is a concept name and C is a concept. $\mathcal{SHOIN}(\mathbf{D})$ models consist of: (1) a set Δ^I of individuals and a set Δ^I_D of data values, (2) a subset $A^I \subseteq \Delta^I$ for each concept A in the signature, (3) an element of A^I for every individual, (4) a binary relation for every role: either (a) $R^I \subseteq A^I_1 \times A^I_2$ (individual valued roles) or (b) $R^I \subseteq A^I \times D^I$ with $D^I \subset \Delta^I_D$ (data-valued roles).The satisfaction relation between models and sentences is defined inductively in the obvious way.

Signatures of CASL-DL consist of: (1) a set of sorts, (2) a subsort hierarchy (which is just a pre-order) subsuming all sorts under the sort *Thing*, (3) a set of unary and binary typed relations, (4) a set of unary typed functions and (5) a set of typed constants. Productions of CASL-DL sentences are given in Fig. 4 to 9. CASL-DL models consist of non-empty carrier sets for each sort and an injective function for each subsort relation. Each predicate symbol is associated with a predicate declared on the appropriate carrier set. Each function symbol corresponds to a partial function between the appropriate carrier sets. Satisfaction is defined as standard in partial first-order logic.

Comorphism from $\mathcal{SHOIN}(\mathbf{D})$ to CASL-DL. Signature mappings: (1) Primary concepts are mapped to sorts and subconcept axioms of primary concepts

yield a subsort declaration. (2) All non-primary concepts are mapped to unary predicates on either *Thing* or their supersort according to the subconcept axioms. (3) Individual-valued roles with functionality axiom yield a unary operation and the others a binary predicate. According to the type of a role the operation or predicate is typed either with *Thing* or a subsort of it. If a formula is given as type instead of a primary concept (mapped to a sort) argument restriction axioms are generated. (4) Data-valued roles with functionality axiom yield a unary operation and the others a binary predicate. According to the type of a role the operation or predicate is typed either with *Thing* or a subsort of it in the (first) argument position and with *DATA* or or a subsort of it as result / in second argument position. (5) Individuals are mapped to operations either typed with *Thing* or with a subsort of it. (6) From subconcept axioms, which involve concepts mapped to unary predicates, implications are generated. The mappings of sentences are shown in Tab. 1 to 7. A subsorted CASL-DL model with injections can be translated to a subsorted model with inclusions by taking the colimit of the inclusion diagram, see [15] for details. From this, it is straightforward to construct a $\mathcal{SHOIN}(\mathbf{D})$ model.

Comorphism from CASL-DL to $\mathcal{SHOIN}(\mathbf{D})$. Signature mapping: (1) All subsorts of *Thing* and sort *Thing* itself are mapped to primary concepts. The subsort hierarchy yields subsumption axioms. (2) Unary relations are mapped concepts as well. Each type of a predicate gives a subsumption axiom. (3) Binary relations are mapped to either individual-valued properties or data-valued properties according to their type. (4) Unary functions are mapped to either individual-valued properties or data-valued properties according to their type and yield a functionality axiom. (5) Constant operations are mapped to individuals of those primary concepts where their sorts have been mapped to. Tables. 1 to 7 show the mapping of sentences by reading them from right to left. Further details are given above how to map the sentences. A $\mathcal{SHOIN}(\mathbf{D})$-model is mapped to CASL-DL by interpreting the subsorts of *Thing* with interpretations of the corresponding concepts in $\mathcal{SHOIN}(\mathbf{D})$, and similarly for the predicates and individuals. Partial functions are interpreted by taking the interpretation of the corresponding binary role in $\mathcal{SHOIN}(\mathbf{D})$; by the axiomatization, it is the graph of a partial function.

The model translations of both comorphisms are inverse to each other. As a consequence, the institution comorphisms can be used to borrow, in a sound and complete way [8], any proof system that works for either of the two logics also for use with the other one.

6 Conclusion and Future Work

We have defined a sublanguage CASL-DL of CASL that corresponds to the web ontology language OWL DL (where DL refers to the sublanguage corresponding to a description logic), and we have described a translation between OWL DL and CASL-DL in detail. We believe that the main benefit of CASL-DL is the

strong typing discipline, which may lead to detection of conceptual errors at a very early stage. By mapping OWL DL's classes and subclasses to CASL's sorts and subsorts, we inherit the flexibility of subsorting while retaining the strong type system. Moreover, CASL-DL offers the possibility to distinguish between a subsort and a unary predicate. Although this distinction is not relevant semantically, it has some conceptual importance [16]. We therefore propose CASL-DL as a language that can be used directly to specify ontologies. Moreover, via the correspondence to OWL DL, efficient tools (like FaCT [17] and Pellet [23]) providing decision procedures for (a large fragment of) OWL DL can also be used for CASL-DL.

A further advantage of CASL-DL is that it naturally comes with CASL's powerful structuring constructs. By contrast, OWL DL and other description logic languages only have quite limited ways to structure and combine ontologies. In fact, the only way that is recommended is a transitive import of other ontologies. To stay within OWL DL requires not to import the OWL Ontology, but to reference all names via URIs. Here CASL's structuring capabilities provide a much more elaborate and cleaner approach, by using hiding and translation of symbols. An interesting question is then the interaction of CASL's structuring concepts with the use of tools like FaCT and Pellet. We expect that the heterogeneous tool set HETS [21] will be a good starting point for answering this question. We also plan to implement the translations described in this paper within HETS.

From an ontological point of view, it would be better not to use CASL-DL for the development of ontologies but a more expressive language. Often KIF [12] (a first order language with Lisp-like syntax) is used for this purpose. But KIF lacks support for structuring ontologies in a nice way. Hence, it would be a good idea to use (full first-order) CASL instead of KIF to develop ontologies: as stated above, CASL provides strong typing and good structuring facilities. In this case, efficient tools like FaCT and Pellet are no longer applicable. Some ontology designers therefore provide their ontology both in a first-order version (e.g. in KIF) and in a description logic version (e.g. in OWL DL) [20]. However, the process of restricting first-order logic to description logic can hardly be automated, and keeping two different version of a document consistent manually is tedious and error-prone. We believe that it is more promising to use tools like the SPASS prover [25] that both support full first-order logic and, when applied to formulas from a suitable description logic fragment, such as CASL-DL, reach the efficiency of specialized tools like FaCT.

In general, we think that for mediating between different languages and tools, translations (formally realized as institution comorphisms) are extremely important. We have sketched two such comorphisms (between CASL-DL and $\mathcal{SHOIN}(\mathbf{D})$); future work will extend this to a graph of formalisms and translations.

References

1. Swoogle search engine. http://swoogle.umbc.edu.

2. H. Andréka, I. Németi, and J. van Benthem. Modal logic and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
4. T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic syntax. IETF Draft Standard (RFC 2396) `http://www.ietf.org/rfc/rfc2396.txt`, August 1998.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.
6. Michel Bidoit and Peter D. Mosses. CASL *User Manual*. LNCS 2900 (IFIP Series). Springer, 2004. With chapters by Till Mossakowski, Donald Sannella, and Andrzej Tarlecki.
7. T. Bray, D. Hollander, and A. Layman, editors. Namespaces in XML. W3C Recommendation `http://www.w3.org/TR/REC-xml-names/`, 14 January 1999.
8. M. Cerioli and J. Meseguer. May I borrow your logic? (transporting logical structures along maps). *Theoretical Computer Science*, 173:311–347, 1997.
9. CoFI (The Common Framework Initiative). CASL *Reference Manual*. LNCS 2960 (IFIP Series). Springer, 2004.
10. M. Dean and G. Schreiber, editors. OWL Web Ontology Language – Reference. W3C Recommendation `http://www.w3.org/TR/owl-ref/`, 10 February 2004.
11. D. Fensel, J. Hendler, H. Liebermann, and W. Wahlster, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. The MIT Press, Cambridge, MA, 2003.
12. M. R. Genesereth and R. E. Fikes. Knowlegde interchange format version 3.0 reference manual. Stanford Logic Group, Report Logic-92-1 `http://logic.stanford.edu/sharing/papers/kif.ps`, June 1992.
13. J. Goguen and G. Rosu. Institution morphisms. *Formal aspects of computing*, 13:274–307, 2002.
14. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992. Predecessor in: LNCS 164, 221–256, 1984.
15. J. A. Goguen and J. Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.
16. N. Guarino. Personal communication.
17. I. Horrocks. FaCT and iFaCT. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. F. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 133–135, 1999.
18. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17–29. Springer, 2003.
19. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
20. C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider. Wonderweb deliverable D17. The wonderweb library of foundational ontologies and the DOLCE ontology. November 29 2002. Preliminary Report (ver. 2.0, 15-08-2002).
21. T. Mossakowski. The heterogeneous tool set. Available at `www.tzi.de/cofi/hets`, University of Bremen.

22. P. F. Patel-Schneider, P. Hayes, and I. Horrocks, editors. OWL Web Ontology Language – Semantics and Abstract Syntax. W3C Recommendation `http://www.w3.org/TR/owl-semantics/`, 10 February 2004.
23. E. Sirin, M. Grove, B. Parsia, and R. Alford. Pellet OWL reasoner. `http://www.mindswap.org/2003/pellet/index.shtml`, May 2004.
24. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.
25. C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobalt, and D. Topic. SPASS version 2.0. In Andrei Voronkov, editor, *Automated Deduction – CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 275–279. Springer-Verlag, July 27-30 2002.

Appendix: Translation Tables

Table 1. Base Concept Axioms for the TBox

$\mathcal{SHOIN}(\mathbf{D})$	Semantics	CASL-DL
$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$	$\forall x : Thing \bullet [\![C_1]\!](x) \Rightarrow [\![C_2]\!](x)$
$C_1 \equiv C_2$	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$	$\forall x : Thing \bullet [\![C_1]\!](x) \Leftrightarrow [\![C_2]\!](x)$

Table 2. Derived Concept Axioms for the TBox

$\mathcal{SHOIN}(\mathbf{D})$	Semantics	CASL-DL
$\forall R.\neg A \sqcup C$	$(\forall R.\neg A \sqcup C)^{\mathcal{I}} = \{x \mid \forall y.\langle x,y\rangle \in R^{\mathcal{I}} \rightarrow y \in (\neg A^{\mathcal{I}} \cup C)^{\mathcal{I}}\}$	$\forall y : A \bullet R(x,y) \Rightarrow [\![C]\!](y)$
$\exists R.A \sqcap C$	$(\exists R.A \sqcap C)^{\mathcal{I}} = \{x \mid \exists y.\langle x,y\rangle \in R^{\mathcal{I}} \rightarrow y \in A^{\mathcal{I}} \cap C^{\mathcal{I}}\}$	$\exists y : A \bullet R(x,y) \wedge [\![C]\!](y)$

$\mathcal{SHOIN}(\mathbf{D})$	Semantics	CASL-DL
$A \sqsubseteq C_1 \sqcap ... \sqcap C_n$	$A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap ... \cap C_n^{\mathcal{I}}$	$\forall x : Thing \bullet x \in A \Rightarrow [\![C_1]\!](x) \wedge ... \wedge [\![C_n]\!](x)$
$A \equiv C_1 \sqcap ... \sqcap C_n$	$A^{\mathcal{I}} = C_1^{\mathcal{I}} \cap ... \cap C_n^{\mathcal{I}}$	**sort** $A = \{x : Thing \bullet [\![C_1]\!](x) \wedge ... \wedge [\![C_n]\!](x)\}$ [a]
$A_1 \sqsubseteq A_2$	$A_1^{\mathcal{I}} \subseteq A_2^{\mathcal{I}}$	**sorts** $A_1 < A_2$
$A \equiv A_1 \sqcup ... \sqcup A_n$	$A^{\mathcal{I}} = A_1^{\mathcal{I}} \cup ... \cup A_n^{\mathcal{I}}$	**generated type** $A ::= sorts\ A_1, ..., A_n$ [b]
$A_1 \equiv A_2$	$A_1^{\mathcal{I}} = A_2^{\mathcal{I}}$	**sorts** $A_1 = A_2$
$A \equiv \{o_1, ..., o_n\}$	$A^{\mathcal{I}} = \{o_1^{\mathcal{I}}, ..., o_n^{\mathcal{I}}\}$	**generated type** $A ::= o_1 \mid ... \mid o_n$ [c]
$C_1 \sqcap C_2 \equiv \bot$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \varnothing$	$\forall x : Thing \bullet [\![C_1]\!](x) \Rightarrow \neg [\![C_2]\!](x)$

[a] if C_i is an atomic concept, a subsort declaration for A is generated

[b] if pairwise disjointness axioms for $A_1, ..., A_n$ are present a **free type** definition is used; an axiom is generated that A is a subsort of the smallest common supersort of $A_1, ..., A_n$

[c] if axioms are present that $o_1, ..., o_n$ are all different individuals, a **free type** definition is used

Table 3. Descriptions of Concepts

$\mathcal{SHOIN}(\mathbf{D})$	Semantics	CASL-DL
Descriptions (C)		$[\![C]\!]$
A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	$x \in A$ for sorts $A(x)$ for predicates
$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$	true
$\bot$	$\bot^{\mathcal{I}} = \varnothing$	false
$C_1 \sqcap ... \sqcap C_n$	$C_1^{\mathcal{I}} \cap ... \cap C_n^{\mathcal{I}}$	$[\![C_1]\!](x) \wedge ... \wedge [\![C_n]\!](x)$
$C_1 \sqcup ... \sqcup C_n$	$C_1^{\mathcal{I}} \cup ... \cup C_n^{\mathcal{I}}$	$[\![C_1]\!](x) \vee ... \vee [\![C_n]\!](x)$
$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	$\neg [\![C]\!](x)$
$\{o_1, ..., o_n\}$	$(\{o_1, ..., o_n\})^{\mathcal{I}} = \{o_1^{\mathcal{I}}, ..., o_n^{\mathcal{I}}\}$	$x = o_1 \vee ... \vee x = o_n$
$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$	$\forall y : Thing \bullet R(x, y) \Rightarrow [\![C]\!](y)$ [a]
$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	$\exists y : Thing \bullet R(x, y) \wedge [\![C]\!](y)$ [b]
$\exists R.A$	$(\exists R.A)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in A^{\mathcal{I}}\}$	$\exists y : A \bullet R(x, y)$
$R : o$	$(R : o)^{\mathcal{I}} = \{x \mid \langle x, o^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}$	$R(x, o)$
$\geqslant n\ R$ $\leqslant n\ R$ $= n\ R$	$(\geqslant n\ R)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in R^{\mathcal{I}}\}) \geq n\}$ $(\leqslant n\ R)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in R^{\mathcal{I}}\}) \leq n\}$ $(= n\ R)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in R^{\mathcal{I}}\}) = n\}$	see discussion of GENCARDINALITY in Fig. 10 for the definition of number restrictions in CASL-DL
$\forall U.D$	$(\forall U.D)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in U^{\mathcal{I}} \rightarrow y \in D^{\mathcal{I}}\}$	$\forall y : DATA \bullet U(x, y) \Rightarrow [\![D]\!](y)$ [c]
$\exists U.D$	$(\exists U.D)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in U^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\}$	$\exists y : DATA \bullet U(x, y) \wedge [\![D]\!](y)$ [d]
$\exists U.D_0$	$(\exists U.D_0)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in U^{\mathcal{I}} \wedge y \in D_0^{\mathcal{I}}\}$	$\exists y : D_0 \bullet U(x, y)$ [e]
$U : v$	$(U : v)^{\mathcal{I}} = \{x \mid \langle x, v^{\mathcal{I}}\rangle \in U^{\mathcal{I}}\}$	$U(x, v)$
$\geqslant n\ U$ $\leqslant n\ U$ $= n\ U$	$(\geqslant n\ U)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in U^{\mathcal{I}}\}) \geq n\}$ $(\leqslant n\ U)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in U^{\mathcal{I}}\}) \leq n\}$ $(= n\ U)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in U^{\mathcal{I}}\}) = n\}$	see discussion of GENCARDINALITY and Fig. 10 for the definition of number restrictions in CASL-DL

[a] if R is declared as functional the formula is $def\, R(x) \Rightarrow [\![C]\!](R(x))$
[b] if R is declared as functional the formula is $[\![C]\!](R(x))$
[c] if U is declared as functional the formula is $def\, U(x) \Rightarrow [\![D]\!](U(x))$
[d] if U is declared as functional the formula is $[\![D]\!](U(x))$
[e] where D_0 is a data type name

Table 4. Declaration of Data Ranges, Roles, Individuals and Data Values

$\mathcal{SHOIN}(\mathbf{D})$	Semantics	CASL-DL
Data Ranges (D)		$[\![D]\!]$
D	$D^{\mathcal{I}} \subseteq \Delta_D^{\mathcal{I}}$	$x \in D$
$\{v_1, ..., v_n\}$	$\{v_1, ..., v_n\}^{\mathcal{I}} = \{v_1^{\mathcal{I}}, ..., v_n^{\mathcal{I}}\}$	$y = v_1 \vee ... \vee y = v_n$ [a]
Object Properties (R)		
R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	**pred** R : *Thing* $\times$ *Thing*
Datatype Properties (U)		
U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$	**pred** U : *Thing* $\times$ *DATA*
Individuals (o)		
o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$	**op** o : *Thing*
Data Values (v)		
v	$v^{\mathcal{I}} \in \Delta_D^{\mathcal{I}}$	some constant term of a predefined datatype

[a] for typing of binary predicates they are named subsorts of *DATA* (s. Fig. 3 (data range subsort definition))

Table 5. TBox Axioms for General Properties

$\mathcal{SHOIN}(\mathbf{D})$	Semantics	CASL-DL
$U \sqsubseteq U_i$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}$	$\forall y : DATA \bullet U(x, y) \Rightarrow U_i(x, y)$
$\geqslant 1\ U \sqsubseteq C_i$	$U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$	$\forall x$: *Thing* $\bullet$ *minCardinality*$[U](x, 1) \Rightarrow [\![C_i]\!](x)$ [a]
$\geqslant 1\ U \sqsubseteq A_i$	$U^{\mathcal{I}} \subseteq A_i^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$	**pred** $U : A_i \times DATA$ [b]
$\top \sqsubseteq \forall U.D_i$	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$	**pred** $U : Thing \times D$ [bc]
$U_1 = U_2$	$U_1^{\mathcal{I}} = U_2^{\mathcal{I}}$	$U_1(x, y) \Leftrightarrow U_2(x, y)$
$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$	$R(x, y) \Rightarrow R_i(x, y)$
$\geqslant 1\ R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	$\forall x$: *Thing* $\bullet$ *minCardinality*$[R](x, 1) \Rightarrow [\![C_i]\!](x)$ [a]
$\geqslant 1\ R \sqsubseteq A_i$	$R^{\mathcal{I}} \subseteq A_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	**pred** $R : A_i \times Thing$ [b]
$\top \sqsubseteq \forall R.C_j$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_j^{\mathcal{I}}$	$\forall x$: *Thing* $\bullet$ *true* $\Rightarrow \forall y$: *Thing* $\bullet\ R(x, y) \Rightarrow [\![C_j]\!](y)$ [a]
$\top \sqsubseteq \forall R.A_j$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times A_j^{\mathcal{I}}$	**pred** $R : Thing \times A_j$ [b]
$R = R_0^-$	$R^{\mathcal{I}} = (R_0^{\mathcal{I}})^-$	$R(x, y) \Leftrightarrow R_0(y, x)$
$R = R^-$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^-$	$R(x, y) \Leftrightarrow R(y, x)$
$\top \sqsubseteq \leqslant 1\ R^-$	$(R^{\mathcal{I}})^-$ is functional	$R(x, z) \wedge R(y, z) \Rightarrow x = y$
$Tr(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$	$R(x, y) \wedge R(y, z) \Rightarrow R(x, z)$
$R_1 = R_2$	$R_1^{\mathcal{I}} = R_2^{\mathcal{I}}$	$R_1(x, y) \Leftrightarrow R_2(x, y)$

[a] for C_i and/or C_j of form $A_1 \sqcup ... \sqcup A_n$ overloaded predicate profiles of all pairs are constructed; other description formulas for C_i and/or C_j yield argument restriction formulas in CASL-DL

[b] if both domain and range are specified U gets type $A_i \times D$

[c] D is either a datatype name or a named datatype range derived from $[\![D_i]\!]$

[d] if both domain and range are specified R gets type $A_i \times A_j$

Table 6. TBox Axioms for Functional Properties

$\mathcal{SHOIN}(\mathbf{D})$	Semantics	CASL-DL
$\top \sqsubseteq \leqslant 1\ U$	$U^{\mathcal{I}}$ is functional	**op** U : *Thing* $\rightarrow ?$ D[a]
$\top \sqsubseteq \leqslant 1\ R$	$R^{\mathcal{I}}$ is functional	**op** R : *Thing* $\rightarrow ?$ *Thing*
$U \sqsubseteq U_i$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}$	$\forall y : DATA \bullet U(x) = y \Rightarrow U_i(x,y)$[b]
$\geqslant 1\ U \sqsubseteq C_i$	$U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$	$\forall x$: *Thing* $\bullet$ *minCardinality*$[U](x,1) \Rightarrow [\![C_i]\!](x)$[cd]
$\geqslant 1\ U \sqsubseteq A_i$	$U^{\mathcal{I}} \subseteq A_i^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$	**op** $U : A_i \rightarrow ?$ $DATA$[e]
$\top \sqsubseteq \forall U.D_i$	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$	**op** U : *Thing* $\rightarrow ?$ D[ae]
$U = U_1$	$U^{\mathcal{I}} = U_1^{\mathcal{I}}$	$U(x) = y \Leftrightarrow U_1(x,y)$[f]
$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$	$R(x) = y \Rightarrow R_i(x,y)$[b]
$\geqslant 1\ R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	$\forall x$: *Thing* $\bullet$ *minCardinality*$[R](x,1) \Rightarrow [\![C_i]\!](x)$[cd]
$\geqslant 1\ R \sqsubseteq A_i$	$R^{\mathcal{I}} \subseteq A_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	**op** $R : A_i \rightarrow ?$ *Thing*[g]
$\top \sqsubseteq \forall R.C_j$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_j^{\mathcal{I}}$	$\forall x$: *Thing* $\bullet$ *true* $\Rightarrow$ *def* $R(x) \Rightarrow [\![C_j]\!](R(x))$[c]
$\top \sqsubseteq \forall R.A_j$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times A_j^{\mathcal{I}}$	**op** R : *Thing* $\rightarrow ?$ A_j[g]
$R = R^-$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^-$	$R(x) = y \Leftrightarrow R(y) = x$
$\top \sqsubseteq \leqslant 1\ R^-$	$(R^{\mathcal{I}})^-$ is functional	$R(x) = z \wedge R(y) = z \Rightarrow x = y$
$R = R_1$	$R^{\mathcal{I}} = R_1^{\mathcal{I}}$	$R(x) = y \Leftrightarrow R_1(x,y)$[b]

[a] D is either a datatype name or a named datatype range derived from $[\![D_i]\!]$

[b] if U_i or R_i is also declared as functional this formula is $\forall y : SD \bullet q(x) = y \Rightarrow q_i(x) = y$ where SD is either $DATA$ or *Thing* and q either U or R

[c] for C_i and/or C_j of form $A_1 \sqcup ... \sqcup A_n$ overloaded function profiles of all pairs are constructed

[d] where GENCARDINALITY is instantiated with **pred** $q(x : s_1; y : s_2) \Leftrightarrow q(x) = y$ where q is either R or U

[e] if both domain and range are specified U gets type $A_i \rightarrow ?$ D

[f] if U_1 or R_1 is also declared as functional the formula is $q(x) = y \Leftrightarrow q_1(x) = y$ where q is either U or R

[g] if both domain and range are specified R gets type $A_i \rightarrow ?$ A_j

Table 7. Axioms for the ABox

$\mathcal{SHOIN}(\mathbf{D})$	Semantics	CASL-DL
$o \in A$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$	**op** $o : A$
$o \in C_i$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$	$[\![C_i]\!](o)$
$\langle o, o_i \rangle \in R_i$	$\langle o^{\mathcal{I}}, o_i^{\mathcal{I}} \rangle \in R_i^{\mathcal{I}}$	$R_i(o, o_i)$[a]
$\langle o, v_i \rangle \in U_i$	$\langle o^{\mathcal{I}}, v_i^{\mathcal{I}} \rangle \in U_i^{\mathcal{I}}$	$U_i(o, v_i)$[a]
$o_1 = o_2$	$o_1^{\mathcal{I}} = o_2^{\mathcal{I}}$	$o_1 = o_2$
$o_1 \neq o_2$	$o_1^{\mathcal{I}} \neq o_2^{\mathcal{I}}$	$\neg o_1 = o_2$

[a] if R is declared as functional the translation result is $R_i(o) = o_i$

[b] if U is declared as functional the translation result is $U_i(o) = v_i$