Proof Support for Common Logic

Till Mossakowski[1], Mihai Codescu[1], Oliver Kutz[1],
Christoph Lange[2], and Michael Gruninger[3]

[1] Institute of Knowledge and Language Engineering, Otto-von-Guericke University of Magdeburg, Germany
[2] University of Bonn and Fraunhofer IAIS, Germany
[3] University of Toronto, Canada

Abstract

We present the theoretical background for an extension of the Heterogeneous Tool Set HETS that enables proof support for Common Logic. This is achieved via logic translations that relate Common Logic and some of its sublogics to already supported logics and automated theorem proving systems. We thus provide the first theorem proving support for Common Logic covering the full language, including the possibility of verifying meta-theoretical relationships between Common Logic theories.

1 Introduction

Common Logic (CL) is an ISO-standardised[1] language based on untyped first-order logic, but extending it in several ways that ease the formulation of complex first-order theories. [21] discusses in detail the motivations and philosophical background of CL, arguing that it not only is a natural formalism for knowledge representation in the context of the Web, but that it also constitutes a natural evolution from the canonical textbook notation and semantics of first-order logic (FOL for short), dispensing with some deeply entrenched views that are reflected in FOL's syntax (and that partly go back to, e.g., Frege's metaphysical views), in particular the segregation of objects, functions, and predicates, fixed arities and signatures, and strict syntactic typing.

Although a substantial number of CL theories have been developed (see Section 2), surprisingly little tool support has been realised so far. In this work, we fill this gap using the Heterogeneous Tool Set HETS [26, 27, 22]). HETS is a general purpose analysis and proof management tool for logical theories. We show that the HETS architecture eases the implementation of a comprehensive tool set for CL, including parsing, theorem proving, checking for consistency, and more.[2]

This paper is organised as follows. In Section 2, we recall CL, and discuss its use cases and hitherto existing tool support. In Section 3, we recall HETS and its theoretical grounding in institution theory. Section 4 contains the core contribution of the paper, namely the integration of CL into HETS, enabling, for the first time, full proof support for CL and its sublogics via various logic translations to already supported logics and automated reasoners. Section 5 presents an example illustrating the achieved CL reasoning support in the context of the COLORE repository. We close in Section 6 with a discussion of the achieved proof support and an outlook to future work.

2 Common Logic

CL is based on untyped first-order logic, but extends first-order logic in two ways: (1) any term can be used as function or predicate, and (2) sequence markers allow for talking about sequences of individuals

[1] Published as "ISO/IEC 24707:2007—Information technology—Common Logic (CL): a framework for a family of logic-based languages" [12]

[2] For a detailed description of HETS, see the HETS user guide [25] (and the special CL version of the guide). HETS is currently available for Linux and Mac OS X from the HETS home page `http://hets.eu`, where you also find the user guides and packages ready for Ubuntu Linux.

directly, and in particular, provide a succinct way for axiomatising polyadic functions and predicates.[3]

A CL signature Σ (called vocabulary in CL terminology) consists of a set of names, with a subset called the set of discourse names, and a set of sequence markers. Discourse names denote first-class objects, which may be individuals, predicates and functions (by Common Logic's "wild west" syntax, any individual can be used in functional and predicate position). Non-discourse names denote second-class objects which can be predicates or functions, but not individuals. Sequence markers denote sequences of (first-class) objects. A Σ-model consists of a set *UR*, the universe of reference (consisting of all first-class and second-class objects), with a non-empty subset $UD \subseteq UR$, the universe of discourse (the first-class objects),[4] and four mappings:

- *int* from names in Σ to *UR*, such that $int(n)$ is in *UD* if and only if *n* is a discourse name;
- *rel* from *UR* to subsets of $UD^* = \{\langle x_1, \ldots, x_n \rangle \mid x_1, \ldots, x_n \in UD, n \in \mathbb{N}\}$ (i.e., the set of finite sequences of elements of *UD*);
- *fun* from *UR* to total functions from UD^* into *UD*;
- *seq* from sequence markers in Σ to UD^*.

Σ-sentences are first-order-like sentences formed with the help of Boolean connectives and quantification over names and sequence markers from atoms, which are either equations betwen two terms or predications which consist of a term, called the predicate, and a term sequence, called the argument sequence. A term is either a name or a functional term, where the latter consists of a term, called the operator, and a term sequence. A term sequence is a finite list of terms or sequence markers. It is important that CL distinguishes between a name *n* and a functional term (n) where *n* is a 0-ary operator, i.e. it takes no arguments. We will write functional terms and predications in a higher-order like syntax as $(t\ s)$. As an example, consider the formula `(instance instance BinaryPredicate)`, stating that the predicate `instance` is an instance of a binary predicate.

Interpretation of terms and formulae is as in first-order logic, with the difference that a predication $(t\ s)$ is interpreted in a model *M* as $(rel^M(t^M))(s^M)$ and a function application $(t\ s)$, as $(fun^M(t^M))(s^M)$, while a name *n* occuring in a formula is interpreted as $int^M(n)$ and a marker *m* as $seq^M(m)$. A further difference to first-order logic is the presence of sequence terms (namely sequence markers and juxtapositions of terms), which denote sequences in UD^*, with term juxtaposition interpreted by sequence concatenation (or details, see [12]). For example, it is possible to express that the items of a list are distinct as follows (using the sequence marker "..."):

```
(distinct)        // the empty sequence is distinct
(distinct x)    // singleton sequences are distinct
(iff (distinct x y ...) // recursion for length > 1
     (and (not (= x y))       // the first two elements must be different
          (distinct x ...)    // and each of them distinct
          (distinct y ...) )) // to the rest
```

For the rationale and methodology of CL and the possibility to define dialects covering different first-order languages, see [12]. CL also includes modules as a syntactic category, with a semantics that restricts locally the universe of discourse.[5]

[3] Strictly speaking, only the sequence markers go beyond first-order logic, as the other higher-order features of CL can be simulated in FOL.

[4] The CLIF dialect of CL also requires that the natural numbers and the strings over unicode characters are part of the universe of discourse. Even if we use CLIF input syntax, we ignore this requirement here, as it is not a general requirement for CL, and the role of datatypes is currently under discussion for the next revision of the CL standard.

[5] See [29] for technical details of the revised semantics for CL modules, which is also considered in the current revision process of ISO/IEC 24707.

2.1 Uses of Common Logic

COLORE (Common Logic Ontology Repository, available at `http://colore.oor.net`) is an open repository of over 600 CL ontologies. One of the primary applications of COLORE is to support the verification of ontologies for common-sense domains such as time, space, shape, and processes. Verification consists in proving that the ontology is equivalent to a set of core ontologies for mathematical domains such as orderings, incidence structures, graphs, and algebraic structures. COLORE comprises core ontologies that formalise algebraic structures (such as groups, fields, and vector spaces), orderings (such as partial orderings, lattices, betweenness), graphs, and incidence structures in CL, and, based on these, representation theorems for generic ontologies for the above-mentioned common-sense domains.

SUMO (Suggested Upper Merged Ontology) is a large upper ontology, covering a wide range of concepts. It is one candidate for the "standard upper ontology" of IEEE working group 1600.1. While SUMO has originally been formulated in the Knowledge Interchange Format (KIF, a precursor of CL), SUMO-CL is a CL partial variant of SUMO produced by Kojeware (see also [4] for a discussion of higher-order aspects in SUMO). The SUMO-KIF version with all the mid-level and domain ontologies that are shipped with SUMO consists of roughly 32,000 concepts in 150,000 lines of specification.

fUML (Foundational UML) is a subset of the Unified Modelling Language (UML 2) defined by the Object Management Group (OMG). The OMG has specified a "foundational execution semantics" for fUML using CL (see `http://www.omg.org/spec/FUML/`).

PSL (Process Specification Language, ISO standard 18629), developed at the National Institute of Standards and Technology (NIST), is an ontology of processes, their components and their relationships. It is also part of COLORE.

2.2 Tool support for Common Logic

Current software tool support for CL is still ad hoc, and is primarily restricted to parsers and translators between CLIF and the TPTP exchange syntax for first-order logic. The work in [17] proposed an environment for ontology development in CL, named Macleod; although this work includes detailed design documents for the environment, there is as yet no available implementation. Similarly, [34] proposed a system architecture for COLORE that supports services for manipulating CL ontologies, once more merely with basic functionality such as parsing being implemented. There exist some ad-hoc syntax translations of CL to first-order provers, but these seem not to be backed up with a semantic analysis of their correctness and they only support the classical first-order fragment of CL. This means that one cannot reason about a CL theory that uses sequence markers or complex terms t, possibly involving quantified variables, as operators in functional terms or predicates in predications.

3 Institutions and the Heterogeneous Tool Set HETS

HETS [26, 27, 22] is an open source software providing a general framework for formal methods integration and proof management. One can think of HETS as acting like a motherboard where different expansion cards can be plugged in, the expansion cards here being individual logics (with their analysis and proof tools) as well as logic translations. The HETS motherboard already has plugged in a number of expansion cards (e.g., theorem provers like SPASS, Vampire, LEO-II, Isabelle and more, as well as model finders). Hence, a variety of tools is available, without the need to hard-wire each tool to the logic at hand.

HETS consists of logic-specific tools for the parsing and static analysis of basic logical theories written in the different involved logics, as well as a logic-independent parsing and static analysis tool for structured theories and theory relations. The latter of course needs to call the logic-specific tools whenever a basic logical theory is encountered.

3.1 Institutions

The semantic background of HETS is the theory of institutions [14], formalising the notion of a logic. An institution provides notions of signature and signature morphism (formally, this is given by a category **Sig**), and for each signature Σ in **Sig**, a set of sentences $Sen(\Sigma)$, a class of models $Mod(\Sigma)$ and a binary satisfaction relation $\models_\Sigma$ between models and sentences. Furthermore, given a signature morphism $\sigma\colon \Sigma_1 \to \Sigma_2$, an institution provides sentence translation along σ, written $\sigma(\varphi)$, and model reduct against σ, written $M|_\sigma$, in a way that satisfaction remains invariant:

$$M'|_\sigma \models_{\Sigma_1} \varphi \text{ iff } M' \models_{\Sigma_2} \sigma(\varphi)$$

for each $\varphi \in Sen(\Sigma_1)$ and $M' \in Mod(\Sigma_2)$.

3.2 Logics supported by Hets

Based on this foundation, HETS supports a variety of different logics. The following ones are most important for use with CL:

OWL 2 is the Web Ontology Language recommended by the World Wide Web Consortium (W3C, `http://www.w3.org`); see [31]. It is used for knowledge representation on the Semantic Web [6]. HETS supports OWL 2 DL and the provers Fact++ and Pellet.

FOL/TPTP is untyped first-order logic with equality,[6] underlying the interchange language TPTP [36], see `http://www.tptp.org`. HETS offers several automated theorem proving (ATP) systems for TPTP, namely SPASS [37], Vampire [33], Eprover [35], Darwin [2], E-KRHyper [32], and MathServe Broker (which chooses an appropriate ATP upon a classification of the FOL problem) [38].

CFOL is many-sorted first-order logic with so-called sort generation constraints, expressing that each value of a given sort is the interpretation of some term involving certain functions (called constructors). This is equivalent to an induction principle and allows the axiomatisation of lists and other datatypes, using the usual Peano-style axioms (such an axiomatisation is called a *free type*). CFOL is a sublogic of the Common Algebraic Specification Language CASL, see [28, 7]. Proof support for CFOL is available through a simple induction scheme in connection with automated first-order provers like SPASS [20], or via a comorphism to HOL. A connection to the induction prover KIV [1] is under development.

HOL is typed higher-order logic [9]. HETS actually supports several variants of HOL, among them THF0 (the higher-order version of TPTP [5]), with automated provers LEO-II [3], Satallax [10] and an automated interface to Isabelle [30], as well as the logic of Isabelle, with an interactive interface.

HETS supports the input languages of these logics directly. Adding a new logic can be done by writing a number of Haskell data types and functions, providing abstract syntax, parser, static analysis and prover interfaces for the logic. It is also possible to integrate logics (as described in [11]) by specifying them in a logical framework like LF [15].

For expressing meta relations between logical theories, HETS supports the Distributed Ontology Language (DOL). DOL can express relations of theories such as logical consequences, relative interpretations of theories and conservative extensions. DOL is also capable of expressing such relations across theories written in different logics, as well as translations of theories along logic translations [24]. DOL is going to be submitted as response to the Object Management Group's (OMG) Ontology, Model and Specification Integration and Interoperability (OntoIOp) Request For Proposal, see `http://www.ontoiop.org`.

[6] FOL/TPTP is called SoftFOL in the HETS implementation. SoftFOL extends first-order logic with equality with a softly typed logic used by SPASS; however, in this paper we will only use the sublanguage corresponding to FOL.

3.3 Institution Comorphisms

HETS' logic translations are formalised as so-called institution comorphisms [13]. A comorphism from institution I to institution J consists of

- a translation Φ of I-signatures to J-signatures (technically, a functor),
- for each signature Σ, a translation α_Σ of Σ-sentences in I to $\Phi(\Sigma)$-sentences in J, and
- for each signature Σ, a translation β_Σ of $\Phi(\Sigma)$-models in J to Σ-models in I,[7]

such that the following satisfaction condition holds:

$$\beta_\Sigma(M)^I \models \varphi \text{ iff } M \models^J \alpha_\Sigma(\varphi)$$

for each signature Σ, Σ-sentence φ and $\Phi(\Sigma)$-model M. A comorphism is:

- *faithful* if logical consequence is preserved and reflected along the comorphism:

$$\Gamma \models^I \varphi \text{ iff } \alpha(\Gamma) \models^J \alpha(\varphi)$$

- *model-expansive* if each β_Σ is surjective;
- a *subinstitution comorphism* if Φ is an embedding, each α_Σ is injective and each β_Σ is bijective[8];
- an *inclusion comorphism* if Φ and each α_Σ are inclusions, and each β_Σ is the identity.

It is known that each subinstitution comorphism is model-expansive and each model-expansive comorphism is also faithful. Faithfulness means that a proof goal $\Gamma \models^I \varphi$ in I can be solved by a theorem prover for J by just feeding the theorem prover with $\alpha(\Gamma) \models^J \alpha(\varphi)$. Subinstitution comorphism preserve the semantics of more advanced DOL structuring constructs such as renaming and hiding. The notion of *theoroidal* comorphisms provides a generalisation of the notion of a comorphism: Φ may map signatures to theories (where a theory (Σ, Γ) is a signature Σ equipped with a set Γ of Σ-sentences). We need theoroidal comorphisms to axiomatize some properties of CL models in first-order logic.

Our main motivation for the present work is to address the lack of tool support for CL (see Sect. 2.2). We have thus extended HETS with parsers for CLIF and KIF. The Common Logic Interchange Format (CLIF) provides a Lisp-like syntax for Common Logic, while KIF is an older precursor of Common Logic. Moreover, we have implemented a sublogic analysis and proof support for CL, discussed in detail below.

3.4 Sublogics of Common Logic

By equipping Common Logic (CL) as defined in Sect. 2 with signature morphisms, it can be formalised as an institution, see [19]. We define four subinstitutions (sublogics) of CL, all defined through restrictions on the sentences. Moreover, given a sublogic CL.X, we also define a logic CL.X$^\sharp$ which results from CL.X by eliminating the use of the module construct.

CL**.Full:** full Common Logic,

CL**.Seq:** Common Logic with sequence markers, but without impredicative higher-order like features. That is, in each predication and function application $(t\ s)$, t must be a name.

CL**.Imp:** Common Logic with impredicative features, but without sequence markers.

CL**.Fol:** Common Logic without impredicative features and without sequence markers.

COLORE is mostly written in CL.Fol, but some ontologies use CL.Seq or CL.Imp.

[7] Actually, α and β also have to commute with translations along signature morphisms. Technically, they are natural transformations.

[8] An isomorphism if model morphisms are taken into account.

4 Logic translations involving Common Logic

For CL, no dedicated theorem prover is available. Proof support for CL is obtained in HETS using logic translations to a logic supported by some prover, as discussed next.[9]

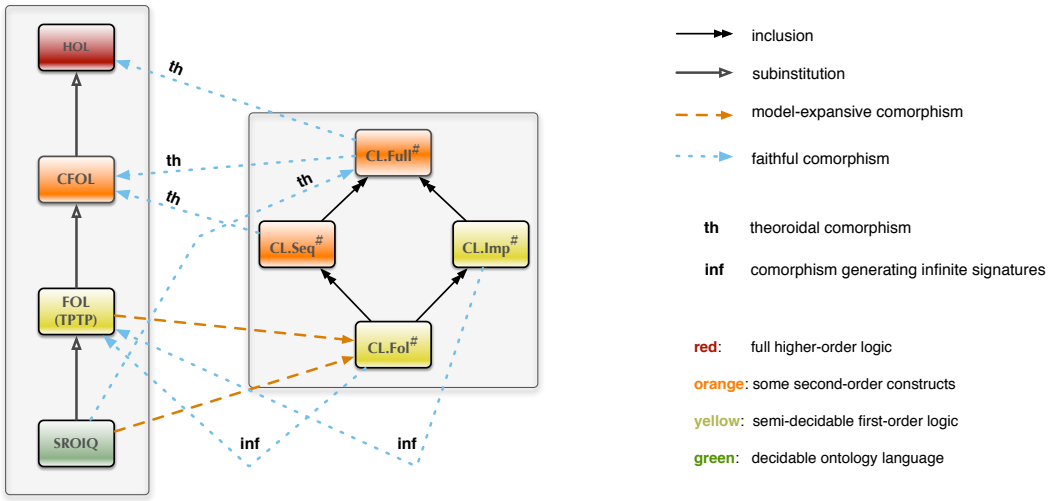

Figure 1: Graph of logics related to Common Logic that are currently supported by HETS

The logic graph in Fig. 1 is naturally divided into two parts: on the left hand-side, we find classical first- and higher-order logics, on the right hand-side the sublogics of Common Logic. *Within* both parts, we have subinstitution relations: for Common Logic, these are obvious (they are even inclusions), while on the left hand-side, the subinstitutions are a bit more complex. Namely, the subinstitution comorphism from $\mathcal{SROIQ}$ (the logic of OWL 2 [16]) into FOL is the standard translation [18], FOL is an obvious sublogic of CFOL, and the translation from CFOL to HOL expands the generation axioms to explicit Peano-style higher-order induction axioms.

Between the parts of the logic graph, the relations are less tight. Due to the different model theories of classical first- and higher-order logics on the left side and Common Logic on the right, we cannot expect subinstitution comorphisms, but only model-expansive and faithful comorphisms to connect the different parts. We now spell out the comorphisms in more detail. In the following subsections, we define $NDNames = Names \setminus DNames$ as the set of non-discourse names of a CL signature $\Sigma = (Names, DNames, Markers)$.

4.1 CL.Fol to FOL

We give a translation of the classical first-order fragment of CL to FOL such that the translated sentences are as similar as possible to the original ones. The intuition behind the translation is to make the two universes (of reference and of discourse) explicit in the translated signature as new sorts, such that the universe of discourse is injectively embedded into the universe of reference. Moreover, we we add function and predicate symbols of any arity for each name, taking also into account the difference between the interpretaton of a name and its interpretation as a constant symbol (see below). This allows us to map the

[9] We also have realised a translation that eliminates the use of CL modules. Since the semantics of CL modules is specific to CL, this elimination of modules is necessary before sending CL theories to a standard first-order prover.

sentences almost identically and to make use of the FOL interpretations of the two sorts and of the function and predicate symbols when defining the CL reduct of a FOL-model of a translated CL signature.

A CL signature $\Sigma = (Names, DNames, \emptyset)$ is translated to a FOL theory $\Phi(\Sigma) = (\Sigma', \Gamma)$ where

- Σ' has two sorts, *UR* and *UD*,
- there is a function symbol $inj : UD \to UR$,
- for each discourse name d there is a constant symbol $[d] : UD$ and for each non-discourse name n there is a constant symbol $[n] : UR$,
- for each $k \in \mathbb{N}$, we have a k-ary function symbol $n : UD^k \to UD$, capturing the functional interpretation of a name, and a k-ary predicate symbol $n : UD^k$, capturing the relational interpretation of a name.

Γ consists of a sentence asserting injectivity of *inj*; for each non-discourse name n, the sentence $\forall x : UD.inj(x) \neq [n]$; for each pair of names n_1, n_2, the infinite set of sentences $[n_1] = [n_2] \implies \forall x_1, \ldots, x_k : UD.n_1(x_1, \ldots, x_k) = n_2(x_1, \ldots, x_k)$ where $k \in \mathbb{N}$; and $[n_1] = [n_2] \implies \forall x_1, \ldots, x_k : UD.n_1(x_1, \ldots, x_k) \iff n_2(x_1, \ldots, x_k)$ where $k \in \mathbb{N}$. We have to introduce for each name n both a symbol $[n]$ for its interpretation as a name n and a symbol n for its interpretation as application (n) of a 0-ary function symbol because they can be different in a CL model[10].

Sentences are mapped by induction on their structure and based on the translation of terms. Since we are in the FOL sublogic of Common Logic, we know that on function and predicate positions, only names occur. Therefore, we can define the translation α_Σ of a functional term $(f\ s_1 \ldots s_k)$ as $\alpha_\Sigma((f\ s_1 \ldots s_k)) = f(\alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$. Likewise, the translation of a predication $(p\ s_1 \ldots s_k)$ is defined as $\alpha_\Sigma((p\ s_1 \ldots s_k)) = p(\alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$. A name n is translated to the constant $[n]$.

For the model reduction component of the comorphism, let Σ be a CL.Fol signature and let N be a $\Phi(\Sigma)$-model. Slightly deviating from our notation for CL, in FOL we will use N_X to denote the interpretation of X in N, where X may be a signature symbol or a term. We will also use the notations $N_{n_f^k}$ (and $N_{n_p^k}$), to make explicit that we refer to the overloaded function (and predicate) symbol n with arity k, for some $k \in \mathbb{N}$. Then $M = \beta_\Sigma(N)$ is defined as follows:

- $UR^M = N_{UR} \uplus \{*\}$,
- $UD^M = N_{inj}(N_{UD})$,
- for any $d \in DNames$, $int^M(d) = N_{inj}(N_{[d]})$ and for any $n \in NDNames$, $int^M(n) = N_{[n]}$.
- for any $x \in UR^M$ and any $s_1, \ldots, s_k \in N_{UD}$, $fun^M(x)(N_{inj}(s_1), \ldots, N_{inj}(s_k)) = N_{inj}(N_{n_f^k}(s_1, \ldots, s_k))$ if there exists a name n such that $int^M(n) = x$[11], and $fun^M(x)(N_{inj}(s_1), \ldots, N_{inj}(s_k)) = *$ otherwise,
- for any $x \in UR^M$ and any $s_1, \ldots, s_k \in N_{UD}$, $(N_{inj}(s_1), \ldots, N_{inj}(s_k)) \in rel^M(x)$ iff $(s_1, \ldots, s_k) \in N_{n_p^k}$ and there exists a name n such that $int^M(n) = x$.

4.2 CL.Imp to FOL

The main idea is that now we add in the translation of a signature function symbols *fun* and predicate symbols *rel* taking $k+1$ arguments for every natural number k. There is no need thus to introduce two constants to distinguish between the interpretation of a name and functional intepretation as a constant, like we did for CL.Fol. The drawback is that now *rel* and *fun* appear in all translated sentences, but on their first argument any term can occur.

A signature $\Sigma = (Names, DNames, \emptyset)$ is translated to a theory $\Phi(\Sigma) = (\Sigma', \Gamma)$ such that

[10] We could also encode CL.Fol to single-sorted first-order logic and have two predicates *UD* and *UR* on the universe instead. While thus the injection *inj* would not be needed anymore, sentences get more complex, and the definition of the model reduct is not substantially simplified, so we choose not to follow this alternative.

[11] It can happen that n is not unique, but the axioms in Γ always ensure that the particular choice of n is irrelevant.

- Σ' has two sorts *UR* and *UD*,
- there is a function symbol $inj : UD \to UR$,
- for each $d \in DNames$, we have a constant symbol $d : UD$ and for each $n \in NDNames$ we have a constant symbol $n : UR$,
- we have a family of functions $\{fun : UR \times UD^k \to UD\}_{k\in\mathbb{N}}$
- and a family of predicate symbols $\{rel : UR \times UD^k\}_{k\in\mathbb{N}}$
- Γ has the sentence that asserts that *inj* is injective and for each non-discourse name *n*, the sentence $\forall x : UD.inj(x) \neq n$.

Sentences are translated by induction on their structure, with predications $(t\ s)$ translated as $\alpha_\Sigma((t\ s)) = rel^k(\alpha_\Sigma(t), \alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$ and terms $(t\ s)$ translated as $\alpha_\Sigma((t\ s)) = fun^k(\alpha_\Sigma(t), \alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$, where s consists of the terms $s_1, \ldots, s_k$[12]. Since there are no sequence markers, the length k of s is always known. Names *n* are translated identically.

Given a signature Σ, a $\Phi(\Sigma)$-model N reduces to $M = \beta_\Sigma(N)$ as follows:

- $UR^M = N_{UR}$,
- $UD^M = N_{inj}(N_{UD})$,
- for any $d \in DNames$, $int^M(d) = N_{inj}(N_d)$ and for any $n \in NDNames$, $int^M(n) = N_n$,
- for any $x \in UR^M$ and any $s_1, \ldots, s_k \in N_{UD}$, $fun^M(x)(N_{inj}(s_1), \ldots, N_{inj}(s_k)) = N_{inj}(N_{fun^k}(x, s_1, ..., s_k))$
- for any $x \in UR^M$ and any $s_1, \ldots, s_k \in N_{UD}$, $(N_{inj}(s_1), \ldots, N_{inj}(s_k)) \in rel^M(x)$ iff $(x, s_1, ..., s_k) \in N_{rel^k}$.

4.3 Getting finite signatures

We obtain infinite objects by translating CL signatures along the two comorphisms introduced above: infinite theories in the first case and infinite signatures in the second. Of course, a software tool will work only with finite objects. A feature of Common Logic is that signatures are implicitly defined by the symbols used in the sentences of a theory, and reasoning in that theory makes use only of those symbols. We can therefore apply a syntactic transformation α that removes from the signature of a CL theory E all symbols that do not occur in E. For the first comorphism, this means that for each name $n \in Names$, we only introduce a function/predicate symbol of arity k and its corresponding sentences in Γ if the sentences in E contain a term/predication where n takes k arguments. Similarly, for the second comorphism, we keep in the signature only those function symbols fun and predicate symbols *rel* whose arity is given by the terms/predications in E. Since this transformation is only syntactical, we do not need to define a corresponding translation between the class of models of (Σ, E) and $\alpha(\Sigma, E)$. It is however easy to notice that by applying the translation, the logical consequences of (Σ, E) are preserved and reflected by α: a Σ-sentence e is a consequence of E over the signature Σ if and only if it is a consequence of e over the signature $\Phi(\Sigma)$ of $\alpha(\Sigma, E)$ (which also implies that the sentence e itself only uses symbols in $\Phi(\Sigma)$).

4.4 CL.Seq to CFOL

In the presence of markers, we make explicit in the translation of a CL signature a sort of lists of elements of the universe of discourse, axiomatized as a free type to ensure that the interpretation of lists is the expected one. This allows us to distinguish between individuals and sequences. Similarly to the case of the translation of CL.Fol to FOL, we introduce for each name both a constant that gives its interpretation as a name and a function/predicate symbol taking as argument a list. Thus we can distinguish between the name n and its functional application with the empty list as argument.

A signature $\Sigma = (Names, DNames, Markers)$ is translated to a CFOL theory $\Phi(\Sigma) = (\Sigma', \Gamma)$ such that

[12] We made explicit that we use the symbols *fun* and *rel* of arity *k*.

- $\Phi(\Sigma)$ has three sorts, *UD*, *UR* and *list*,
- there is a function symbol $inj : UD \to UR$,
- on sort *list* we have $nil : list$ and $cons : UD \times list \to list$ and $m : list$ for each $m \in Markers$,
- for each $d \in DNames$ we have a function symbol $d : UD$, for each $n \in NDNames$ we have a function symbol $n : UR$,
- for each $n \in Names$, there is a function symbol $n : list \to UD$ and a predicate symbol $n : list$,
- Γ contains the Peano-style axioms asserting that *list* is a free type over its constructors *nil* and *cons*; moreover, a sentence asserting that *inj* is an injection; for each non-discourse name n, the sentence $\forall x : UD.inj(x) \neq n$; and for each pair of names n_1, n_2, the infinite set of sentences $n_1 = n_2 \implies \forall x_1, \ldots, x_k \in UD.n_1([x_1, \ldots, x_k]) = n_2([x_1, \ldots, x_k])$ where $k \in \mathbb{N}$ and $n_1 = n_2 \implies \forall x_1, \ldots, x_k \in UD.n_1([x_1, \ldots, x_k]) \iff n_2([x_1, \ldots, x_k])$ where $k \in \mathbb{N}$ and $[x_1, \ldots, x_k]$ is a notation for the list with elements $x_1, \ldots, x_k$.

Sentences are mapped by induction on their structure, such that each predication or term $(t\ s)$ is mapped as $\alpha_\Sigma((t\ s)) = t(\alpha_\Sigma(s))$ and each name n is mapped to the constant n. This works because all operators of functional terms and all predicates of predications are names.

Given a $\Phi(\Sigma)$-model N, its reduct $M = \beta_\Sigma(N)$ is defined as follows, using again the notations N_{n_f} and N_{n_p}, to make explicit that we refer to the overloaded function and predicate symbol n.

- $UR^M = N_{UR} \uplus \{*\}$,
- $UD^M = N_{inj}(N_{UD})$,
- for any $d \in DNames$, $int^M(d) = N_{inj}(N_d)$ and for any $n \in NDNames$, $int^M(n) = N_n$,
- for any $x \in UR^M$ and any $s \in (N_{UD})^*$, $fun^M(x)(N_{inj}(s)) = N_{inj}(N_{n_f}(s))$, if there exists a name n such that $int^M(n) = x$ and $fun^M(x)(N_{inj}(s)) = *$ otherwise,
- for any $x \in UR^M$ and any $s \in (N_{UD})^*$, $N_{inj}(s) \in rel^M(x)$ iff $s \in N_{n_p}$ and there exists a name n such that $int^M(n) = x$,
- for any $m \in Markers$, $seq^M(m) = N_{inj}(N_m)$, that is, N_{inj} is applied each element of the list N_m to obtain a new list with elements in UD^M.

4.5 CL.Full to CFOL

Finally, the translation of the full CL to CFOL combines the translations CL.Seq to CFOL and CL.Imp to FOL.

A signature $\Sigma = (Names, DNames, Markers)$ is translated to a CFOL theory $\Phi(\Sigma) = (\Sigma', \Gamma)$ such that

- Σ' has three sorts, *UR*, *UD* and *list*,
- there is a function $inj : UD \to UR$,
- on *UD* we have a constant symbol d for each discourse name $d \in DNames$,
- on *UR* we have a constant symbol n for each non-discourse name $n \in NDNames$,
- on sort *list* we have $nil : list$, $cons : UD \times list \to list$ and $m : list$ for each $m \in Markers$,
- there is a function symbol $fun : UR \times list \to UD$ and a predicate symbol $rel : UR \times list$,
- Γ consists of the Peano-style axioms asserting that *list* is a free type over its constructors *nil* and *cons*, the sentence asserting that *inj* is injective and the sentences asserting for each non-discourse name n that n is not in the image of *UD* along *inj*, i.e. $\forall x : UD.inj(x) \neq n$.

Sentences are translated by induction on their structure, with predications $(t\ s)$ translated to predications $rel(\alpha_\Sigma(t), \alpha_\Sigma(s))$, terms $(t\ s)$ translated to terms $fun(\alpha_\Sigma(t), \alpha_\Sigma(s))$ and names n translated to the corresponding constants n in Σ'.

Given a (Σ', Γ)-model N, its reduct $M = \beta_\Sigma(N)$ is defined as follows:

- $UR^M = N_{UR}$,
- $UD^M = N_{inj}(N_{UD})$, that is, the image of N_{UD} along the injection N_{inj},
- for any $d \in DNames$, $int^M(n) = inj(N_d)$, and for any $n \in NDNames$, $int^M(n) = N_n$. The sentences in Γ ensure that *int* maps a name x to an element of UD^M iff x is a discourse name,
- for any $x \in UR^M$ and any $s \in (N_{UD})^*$, $fun^M(x)(N_{inj}(s)) = N_{inj}(N_{fun}(x,s))$,
- for any $x \in UR^M$ and any $s \in (N_{UD})^*$, $N_{inj}(s) \in rel^M(x)$ iff $(x,s) \in N_{rel}$,
- for any $m \in Markers$, $seq^M(m) = N_{inj}(N_m)$.

4.6 Faithful comorphisms

We now come to proving faithfulness of the comorphisms, which ensures we can borrow proofs. For CL.Fol to FOL and CL.*Seq* to CFOL the arguments are similar, so we only present the proof of faithfulness for the second one.

Theorem 1. *The comorphism* CL.*Seq to CFOL is faithful.*

Proof. It suffices to prove that for any CL-signature Σ, if $\alpha_\Sigma(\Gamma) \models^{CFOL}_{\Phi(\Sigma)} \alpha_\Sigma(e)$, then $\Gamma \models^{\mathsf{CL}}_\Sigma e$.

We define a mapping $\delta_\Sigma : \mathbf{Mod}^{\mathsf{CL}}(\Sigma) \to \mathbf{Mod}^{CFOL}(\Phi(\Sigma))$ such that $M \models e$ iff $\delta_\Sigma(M) \models \alpha(e)$ where M is a Σ-model and e is a Σ-sentence.

We denote $N = \delta_\Sigma(M)$. N is defined as follows:

- $N_{UR} = UR^M$,
- $N_{UD} = UD^M$,
- N_{inj} is the inclusion of UD^M in UR^M,
- the interpretation of the sort *list* and of the operations *cons*, *nil* and $++$ is the expected one,
- $N_m = seq^M(m)$ for each $m \in Markers$,
- $N_n = int^M(n)$ for each $n \in Names$
- for each x in *Names* and any $s \in UD^*$, $N_x(s) = fun^M(int^M(x))(s)$,
- for each x in *Names* and any $s \in UD^*$, $s \in N_x$ iff $s \in rel^M(int^M(x))$.

It is easy to see that N satisfies the sentences in $\Phi(\Sigma)$ and $M \models e$ iff $N \models \alpha(e)$ for each Σ-sentence e.

Assume $\alpha_\Sigma(\Gamma) \models^{CFOL}_{\Phi(\Sigma)} \alpha_\Sigma(e)$. In order to show $\Gamma \models^{\mathsf{CL}}_\Sigma e$, let M be a Σ-model such that $M \models^{\mathsf{CL}}_\Sigma \Gamma$. Then $\delta_\Sigma(M) \models^{CFOL}_{\Phi(\Sigma)} \alpha_\Sigma(\Gamma)$. Hence $\delta_\Sigma(M) \models^{CFOL}_{\Phi(\Sigma)} \alpha_\Sigma(e)$, and thus $M \models^{\mathsf{CL}}_\Sigma e$. □

For the other two comorphisms we can prove a stronger result, namely model-expansiveness. The proof proceeds in a similar way as above.

Theorem 2. *The comorphisms* CL.*Imp to CFOL and* CL.*Full to CFOL are model-expansive.*

4.7 FOL to CL.Fol

This comorphism maps classical FOL to CL.Fol.

A FOL signature is translated to CL.Fol by turning all constants into discourse names, and all other function symbols and all predicate symbols into non-discourse names. A FOL sentence is translated to CL.Fol by a straightforward recursion, the base being translations of predications:

$$\alpha_\Sigma(P(t_1,\ldots,t_n)) = (P\ \alpha_\Sigma(t_1)\ \ldots\ \alpha_\Sigma(t_n))$$

Within terms, function applications are translated similarly:

$$\alpha_\Sigma(f(t_1,\ldots,t_n)) = (f\ \alpha_\Sigma(t_1)\ \ldots\ \alpha_\Sigma(t_n))$$

A CL.Fol model is translated to a FOL model by using the universe of discourse as FOL universe. The interpretation of constants is directly given by the interpretation of the corresponding names in CL.Fol. The interpretation of a predicate symbol P is given by using $rel^M(int^M(P))$ and restricting to the arity of P; similarly for function symbols (using fun^M). Both the satisfaction condition and model-expansiveness of the comorphism are straightforward.

4.8 FOL to CFOL

A (single-sorted) FOL signature is mapped to (many-sorted) CFOL by introducing a sort s, and letting all function and predicate symbols be typed by a list of s's, the length of the list corresponding to the arity. The rest is straightforward.

4.9 CFOL to HOL

A CFOL signature is translated to a HOL signature by mapping all sorts to type constants, and all function and predicate symbols to constants of the respective higher-order type. Translation of sentences and models is then straightforward, except for sort generation constraints. The latter can be translated to second-order induction axioms, see [23].

4.10 SROIQ to FOL

This comorphism extends the well-known standard translation for description logics [18, 8] to the additional expressive means of $\mathcal{SROIQ}$. Individuals are translated to constants, concepts to unary predicates and roles to binary predicates. Concepts are translated to formulas with one free variable. FOL-models are translated to $\mathcal{SROIQ}$-models by keeping the universe of discourse, and using the interpretations of constants, unary and binary predicates for interpreting individuals, concepts and roles, respectively.

4.11 SROIQ to CL.Fol

This comorphism can be obtained as the composition SROIQ $\rightarrow$ FOL $\rightarrow$ CL.Fol.

4.12 SROIQ to CL.Full

This comorphism internalises all notions (like Boolean operators on concepts, inverses of roles, etc.) used in $\mathcal{SROIQ}$ using the higher-order like features of CL. Then the translation of sentences becomes trivial.

Due to the theory needed for the axiomatisation of these $\mathcal{SROIQ}$ notions, the comorphism is theoroidal.[13]

A signature is translated by mapping all individuals, concepts and roles to discourse names, and augmenting this with a CL.Imp theory that internalises all operations on concepts that are possible in $\mathcal{SROIQ}$, e.g.

```
(forall (c x) (iff ((OWLnot c) x) (not (c x))))                    // complement
(forall (r c x) (iff ((OWLsome r c) x)             // existential quantification
                     (exists (y) (and (r x y) (c y)))))
(forall (r x) (iff ((OWLself r) x) (r x x)))                 // self-restriction
(forall (c d) (iff ((OWLsubsumes c d)            // class subsumption (subclass)
                    (forall (x) (if (c x) (d x))))))
(forall (r) (iff ((OWLirr r)                                    // irreflexivity
                  (forall (x y) (not (r x x))))))

(forall (p) (holds-all p))
(forall (p x ...) (iff (holds-all p x ...)
                       ((and (p x) (holds-all p ...)))))
((same-length))
(forall (x ...) (not ((same-length) x ...)))
(forall (x ...) (not ((same-length x ...))))
(forall (x y ...a ...b) (iff ((same-length x ...a) y ...b)
                             ((same-length ...a) ...b)))

(forall (r c x y) (iff ((restrict r c x) y)
                       (and (r x y) (c y))))
(forall (r c ...) (iff ((OWLmin r c ...) x)              // minimum cardinality
    (exists (...a) (and ((same-length ...) ...a)
                        (holds-all (restrict r c x) ...a)
                        ((distinct ...a))))))

(forall x (not ((OWLnominal) x)))      //  enumeration of individuals (nominals)
(forall ... x y (iff ((OWLnominal y ...) x)
                     (or (= x y) ((OWLnominal ...) x))))

(forall (r x y) (iff ((OWLcomp r) x y) (r x y)))              // property chain
(forall (r ... x y)                                          // (role composition)
   (iff ((OWLcomp r ...) x y)
        (exists (z) (and (r x z) ((OWLcomp ...) z y)))))
```

Note the use of sequence markers for handling lists of variable length. Functions cannot take two sequence markers as argument unless they are written in curried form (otherwise, the two sequences will be concatenated into one argument). Therefore, functions like `same-length` use a curried form; that is, two separate function applications are used for the two arguments, as in `((f x) y)`.

Translation of concepts sentences is then straightforward, e.g.

- $\alpha(\neg C) = (\texttt{OWLnot}\ \alpha(C))$
- $\alpha(\exists R.C) = (\texttt{OWLsome}\ \alpha(R)\ \alpha(C))$

[13] Translations of SROIQ to CL.Full have been introduced in `http://philebus.tamu.edu/cmenzel/Papers/AxiomaticSemantics.pdf` and `http://www.ihmc.us/users/phayes/CL/SW2SCL.html`. Our translation is simpler as in the second reference due to the use of a rich background theory, as in the first reference. Comparing to the latter, we use standard lists, as provided by CL sequences, in contrast to a weak axiomatization of lists in CL that admits non-standard lists as models as well.

- $\alpha(\geq nR.C) = (\texttt{OWLmin}\ \alpha(R)\ \alpha(C)\ (a \ldots a))$

Here, the sequence `(a ...a)` repeats an arbitrary name a n times. This is used as a coding of the natural number n. The function `same-length` above is then used for quantifying over all sequences of length n. The term `((same-length c ...) ...a)` above tests whether `...a` has length $n+1$, where n is encoded by `...`. Note that the value of `c` is irrelevant here, it is just used for increasing the length of the sequence `...` by one.

Models are translated by keeping the universe of discourse, and the interpretation of individuals, concepts and roles are given by the interpretation of the respective names.

4.13 Module elimination

Finally, Fig. 2 shows the square of CL.Full and its sublogics CL.Imp, CL.Seq, and CL.Fol and the square of logics obtained by eliminating the module construct from the languages (denoted by CL.Full$^{\#}$ CL.Imp$^{\#}$, CL.Seq$^{\#}$, and CL.Fol$^{\#}$):

- the obtained cube relates CL (and its sublogics) with the respective restrictions;
- logics without a module construct are obtained (essentially) as a re-writing using quantifier restrictions;
- while the restrictions are obviously inclusions, the reverse translations are obtained as subinstitutions.

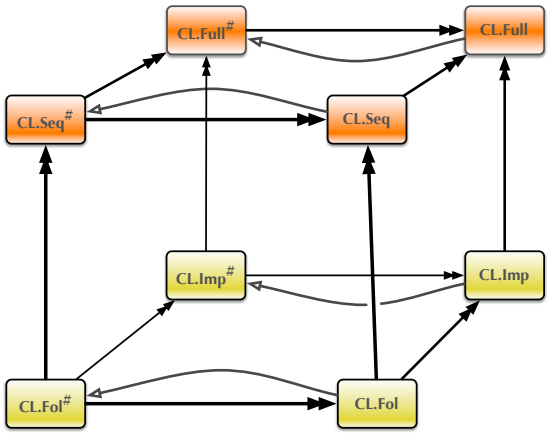

Figure 2: Elimination of the module construct in CL

5 Case Study: Verifying Interpretations in COLORE

We have used HETS for verifying a number of consequences and interpretations of COLORE theories, and for checking their consistency.

The listing below shows a typical example of a COLORE interpretation.[14] Linearly ordered time intervals that begin and end with an instant (`ins:owltime_le`) are interpreted as lines that are incident with linearly ordered points in a special geometry (union of `ord:linear_ordering` and `bi:complete_graphical`), given an appropriate mapping, which is defined as another ontology (`owltime_interval_reduction`). We state that the source ontology can be interpreted in terms of the union of the target ontologies and the mapping ontology in a model-theoretically conservative way, and that the mapping ontology merely extends the target ontologies with definitions.

```
%prefix( %% declaration of prefixes for abbreviating long network identifiers
 %% this distributed ontology (further technical stuff abbreviated with ...)
 :    <http://colore.googlecode.com/.../complex/owltime/owltime_interval/mappings/owltime_le.dol#>
 %% Namespaces of other COLORE ontologies used here:
 bi:  <http://colore.../core/bipartite_incidence/>   ins: <http://.../complex/owltime/owltime_instant/>
 int: <http://.../complex/owltime/owltime_interval/> ord: <http://.../complex/orderings/> )%

%% The following ontologies are in the logic Common Logic
logic CommonLogic

%% The ontology of linearly ordered time intervals that begin and end with an instant extends ...
%% ... linearly ordered time intervals with intervals that begin and end with an instant.
ontology int:owltime_le = int:owltime_linear then int:owltime_e
```

[14] This is an excerpt from `https://colore.googlecode.com/svn/trunk/ontologies/complex/owltime/owltime_interval/mappings/owltime_le.dol` in the COLORE repository. The individual ontologies are actually stored in separate files, but this listing includes them for conciseness, thus demonstrating DOL's ability to maintain different ontologies within one file.

```
%% The ontology of linearly ordered time intervals extends intervals with linearly ordered instants.
ontology int:owltime_linear = int:owltime_interval then ins:owltime_instant_l

%% (int:owltime_e, int:owltime_interval, ins:owltime_instant_l and their imports not shown here)

ontology ord:linear_ordering =        %% Here we use Common Logic's import construct and add one axiom:
  (cl-imports ord:partial_ordering) (forall (x y) (or (leq x y) (leq y x) (= x y)))

ontology ord:linear_ordering =
  (cl-imports ord:partial_ordering) (forall (x y) (or (leq x y) (leq y x) (= x y))))
  %% (ord:partial_ordering and its imports not shown here)

ontology bi:complete_graphical = bi:grapical_incidence                                  %% see below
  then bi:line_existence                          %% "There is a line through any two distinct points."

ontology bi:graphical_incidence = bi:partial_bipartite    %% "Every line has a point incident with it."
  then bi:double_points           %% a geometry in which at most two points can be incident with a line
  %% The incidence relation (further imports not shown here) is reflexive and symmetric.

ontology int:mappings/owltime_interval_reduction =
  (forall (x) (iff (Instant x) (point x)))               // mapping time instants to geometrical points
  (forall (x) (iff (Interval x) (line x)))               // mapping time intervals to geometrical lines
  // skipping some axioms
  (forall (x y) (iff (begins x y)                                  // x begins with y iff x is a point ...
                     (or (and (point x) (line y) (in x y) // ... that is ≤ any other point incident ...
                              (forall (z) (if (and (point z) (in z y)) (leq x z))))
                         (and (point x) (= x y)))))// ... to the line y, or x and y are the same point.

interpretation geometry_of_time %mcons :  %% Interpretation of linearly ordered time intervals that ...
  int:owltime_le      %% ... begin and end with an instant as lines that are incident with linearly ...
  to { ord:linear_ordering and bi:complete_graphical   %% ... ordered points in a special geometry, ...
     %% ... given an appropriate mapping ontology (defined above), plus the following explicit mapping:
       and %def int:mappings/owltime_interval_reduction } = ProperInterval |-> Interval end
```

In the case of our example, some proof goals introduced when verifying correctness of the interpretation `geometry_of_time` actually end up *disproved*. In the concrete case of the goal `ins:owltime_instants` (corresponding to the theory of partially ordered time instants), Hets finds a countermodel where the predicate `before` is neither irreflexive nor asymmetric, thus violating the strict order axioms one would expect them to satisfy. In the source theory, `before` is axiomatised as intended, whereas the target theory does not do anything besides defining it – via the mapping ontology – equivalent to a predicate `lt`, about which nothing is stated at all. This suggests that an axiomatisation of `lt` is missing, and in fact the problem can be fixed by adding to the target of the interpretation the ontology `ord:orderings_def`, which defines the "less than" operator.

6 Discussion and Outlook

We have established the first full theorem proving support for Common Logic as well as the possibility of verifying meta-theoretical relationships between Common Logic theories via an integration into the HETS system, primarily exploiting the power of logic translation. Using the translations in Sect. 4, we provide proof support for the various sublanguages of Common Logic. Some sublanguages can be translated into classical first-order logic, such that automated theorem provers like SPASS, Vampire or Darwin can be used. Via faithfulness of the translations, soundness and completeness of this proof support for Common Logic is inherited from that for the provers w.r.t. FOL. Other sublanguages need provers with induction capabilities, or higher-order provers like Leo-II or Isabelle. This is still sound, but only complete w.r.t. Henkin semantics for HOL, i.e. a semantics allowing for models with non-standard sequences. By Gödel's incompleteness theorem, we cannot expect more.

All these translations and provers are integrated within HETS. As CL is a popular language within ontology communities interested in greater expressive power than provided by the decidable OWL DL language, this tool support for CL is a substantial step towards supporting more ambitious ontology engineering efforts. As a first evaluation of the provided CL reasoning support, we have used HETS for verifying consequences of and interpretations between COLORE theories, and for checking their consistency. During

this process, numerous errors in COLORE have been found and corrected (including parsing errors, wrong symbol mappings, missing Boolean operators, and inconsistencies). The sublogic analysis for Common Logic provided by HETS was of particular importance here, because automation and efficiency of proofs greatly varies among the sublogics. Most proof goals in the CL.Fol theories of COLORE could be proved using SPASS, while for COLORE's graph theories involving recursive use of sequence markers, the interactive theorem prover Isabelle needed to be used.

Related work includes the bridges of provers like Isabelle and Ωmega to automated theorem proving (ATP) systems like SPASS. HETS also provides bridges from CL (and other logics) to ATP systems. However, while other systems have the bridges built-in in a hard-coded way, HETS realises bridges as institution comorphisms and supports them as first-class citizens. Hence, HETS offers a greater flexibility by letting the user chose among different bridges (or even develop new ones). We have exploited this flexibility by providing different bridges, depending on the sublogic of CL in which a given theory is formulated.

Future work should analyse non-recursive uses of sequence markers (as they occur in theories that are generic over the arity of certain predicates and functions) more carefully and provide automated first-order proof support for these. We also plan to integrate our work into the web ontology repository engine `ontohub.org` and exploit more explicitly the structuring mechanisms of DOL in connection with CL.

Acknowledgements. Work on this paper was supported by the DFG-funded Research Center on Spatial Cognition (SFB/TR 8). We thank Fabian Neuhaus for his detailed comments on several versions of this paper, and to Christian Maeder, Sören Schulze, Eugen Kuksa for contributing to the implementation of support of CL in HETS. We gratefully acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open Grant number: 611553, project COINVENT.

References

[1] Michael Balser, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. Kiv 3.0 for provably correct systems. In Dieter Hutter, Werner Stephan, Paolo Traverso, and Markus Ullmann, editors, *FM-Trends*, volume 1641 of *Lecture Notes in Computer Science*, pages 330–337. Springer, 1998.

[2] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the Model Evolution Calculus. *Special Issue of the International Journal of Artificial Intelligence Tools (IJAIT)*, 15(1), 2005.

[3] C. Benzmüller, L. C. Paulson, F. Theiss, and A. Fietzke. LEO-II - A Cooperative Automatic Theorem Prover for Classical Higher-Order Logic (System Description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 162–170. Springer, 2008.

[4] Christoph Benzmüller and Adam Pease. Higher-order aspects and context in SUMO. *Journal of Web Semantics (Special Issue on Reasoning with context in the Semantic Web)*, 12-13:104–117, 2012.

[5] Christoph Benzmüller, Florian Rabe, and Geoff Sutcliffe. THF0 – the core of the TPTP language for higher-order logic. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2008.

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[7] M. Bidoit and P. D. Mosses. CASL *User Manual*, volume 2900 of *LNCS*. Springer, 2004. `www.cofi.info`.

[8] A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82:353–367, 1996. Research Note.

[9] T. Borzyszkowski. Higher-order logic and theorem proving for structured specifications. In D. Bert, C. Choppy, and P. D. Mosses, editors, *WADT*, volume 1827 of *LNCS*, pages 401–418. Springer, 1999.

[10] Chad E. Brown. Satallax: An automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *LNCS*, pages 111–117. Springer, 2012.

[11] Mihai Codescu, Fulya Horozal, Michael Kohlhase, Till Mossakowski, Florian Rabe, and Kristina Sojakova. Towards logical frameworks in the heterogeneous tool set hets. In Till Mossakowski and Hans-Jörg Kreowski, editors, *WADT 2010*, volume 7137 of *Lecture Notes in Computer Science*, pages 139–159. Springer, 2012.

[12] Information technology — Common Logic (CL): a framework for a family of logic-based languages. Technical Report 24707:2007, ISO/IEC, 2007. `http://iso-commonlogic.org`.

[13] J. Goguen and G. Roşu. 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.

[15] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[16] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006)*, pages 57–67. AAAI Press, June 2006.

[17] Megan Katsumi. A methodology for the development and verification of expressive ontologies. M. Sc. Thesis, Department of Mechanical and Industrial Engineering, University of Toronto, 2011.

[18] Yevgeny Kazakov. RIQ and SROIQ Are Harder than SHOIQ. In Gerhard Brewka and Jérôme Lang, editors, *KR*, pages 274–284. AAAI Press, 2008.

[19] O. Kutz, T. Mossakowski, and D. Lücke. Carnap, Goguen, and the Hyperontologies: Logical Pluralism and Heterogeneous Structuring in Ontology Design. *Logica Universalis*, 4(2):255–333, 2010.

[20] Klaus Lüttich and Till Mossakowski. Reasoning Support for CASL with Automated Theorem Proving Systems. In J. Fiadeiro, editor, *WADT 2006*, number 4409 in LNCS, pages 74–91. Springer, 2007.

[21] Christopher Menzel. Knowledge representation, the World Wide Web, and the evolution of logic. *Synthese*, 182:269–295, 2011.

[22] T. Mossakowski. Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen, 2005.

[23] Till Mossakowski. Relating CASL with other specification languages: the institution level. *Theoretical Computer Science*, 286:367–475, 2002.

[24] Till Mossakowski, Christoph Lange, and Oliver Kutz. Three Semantics for the Core of the Distributed Ontology Language. In Maureen Donnelly and Giancarlo Guizzardi, editors, *FOIS 2012*, volume 239 of *Frontiers in Artificial Intelligence and Applications*, pages 337–352. IOS Press, 2012. FOIS Best Paper Award.

[25] Till Mossakowski, Christian Maeder, and Mihai Codescu. Hets user guide, 2011. See `hets.eu`.

[26] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set. In Orna Grumberg and Michael Huth, editors, *TACAS 2007*, volume 4424 of *LNCS*, pages 519–522. Springer, 2007.

[27] Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set. In Bernhard Beckert, editor, *VERIFY 2007*, volume 259 of *CEUR Workshop Proceedings*. 2007.

[28] Peter D. Mosses, editor. *CASL Reference Manual*, volume 2960 of *Lecture Notes in Computer Science*. Springer, 2004. Free online version available at `http://www.cofi.info`.

[29] Fabian Neuhaus and Pat Hayes. Common logic and the Horatio problem. *Applied Ontology*, 7(2):211–231, 2012.

[30] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer Verlag, 2002.

[31] OWL Working Group. OWL 2 web ontology language: Document overview. W3C recommendation, World Wide Web Consortium (W3C), October 2009.

[32] Björn Pelzer and Christoph Wernhard. System description: E-KRHyper. In Frank Pfenning, editor, *CADE*, volume 4603 of *Lecture Notes in Computer Science*, pages 508–513. Springer, 2007.

[33] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2-3):91–110, 2002.

[34] C. Ross. COLORE system architecture. Available at `http://ontolog.cim3.net/cgi-bin/wiki.pl?OpenOntologyRepository_Architecture/From_COLORE`.

[35] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[36] Geoff Sutcliffe. The TPTP world – infrastructure for automated reasoning. In Edmund M. Clarke and Andrei Voronkov, editors, *LPAR (Dakar)*, volume 6355 of *LNCS*, pages 1–12. Springer, 2010.

[37] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobalt, and D. Topic. SPASS version 2.0. In A. Voronkov, editor, *Automated Deduction – CADE-18*, LNCS 2392, pages 275–279, 2002.

[38] Jürgen Zimmer and Serge Autexier. The MathServe System for Semantic Web Reasoning Services. In U. Furbach and N. Shankar, editors, *3rd IJCAR*, LNCS 4130. Springer, 2006.