Proof Support for Common Logic

Till Mossakowski¹, Mihai Codescu¹, Oliver Kutz¹, Christoph Lange², and Michael Gruninger³

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 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.

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

¹Published as “ISO/IEC 24707:2007—Information technology—Common Logic (CL): a framework for a family of logic-based languages” [12]
²For a detailed description of HETS, see the HETS user guide [26] (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.

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 ⊆ UR, the universe of discourse (the first-class objects) 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* = {⟨x₁, ..., xₙ⟩ | x₁, ..., xₙ ∈ UD, n ∈ 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 between 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 (relₘ(M))(sₘ) and a function application (t s), as (funₘ(M))(sₘ), while a name n occurring in a formula is interpreted as intₘ(M)(n) and a marker m as seqₘ(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](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/](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 predcations.

3 Institutions and the Heterogeneous Tool Set H$_{ETS}$

H$_{ETS}$ [26,27,22] is an open source software providing a general framework for formal methods integration and proof management. One can think of H$_{ETS}$ 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 H$_{ETS}$ 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.

H$_{ETS}$ 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 $\text{Sig}$), and for each signature $\Sigma$ in $\text{Sig}$, a set of sentences $\text{Sen}(\Sigma)$, a class of models $\text{Mod}(\Sigma)$ and a binary satisfaction relation $\models_\Sigma$ between models and sentences. Furthermore, given a signature morphism $\sigma: \Sigma_1 \rightarrow \Sigma_2$, an institution provides sentence translation along $\sigma$, written $\sigma(\varphi)$, and model reduct against $\sigma$, written $M\models_\sigma$, in a way that satisfaction remains invariant:

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

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

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, underlying the interchange language TPTP [36], see [http://www.tptp.org]. Hets offers several automated theorem proving (ATP) systems for TPTP, namely SPASS [47], 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 [11] 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 $\Phi$ of I-signatures to J-signatures (technically, a functor),
- for each signature $\Sigma$, a translation $\alpha_\Sigma$ of $\Sigma$-sentences in I to $\Phi(\Sigma)$-sentences in J, and
- for each signature $\Sigma$, a translation $\beta_\Sigma$ of $\Phi(\Sigma)$-models in J to $\Sigma$-models in I.

such that the following satisfaction condition holds:

$$\beta_\Sigma(M) I \models \varphi \iff M J \models \alpha_\Sigma(\varphi)$$

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

- faithful if logical consequence is preserved and reflected along the comorphism:
  $$\Gamma I \models \varphi \iff \alpha(\Gamma) J \models \alpha(\varphi)$$

- model-expansive if each $\beta_\Sigma$ is surjective;
- a subinstitution comorphism if $\Phi$ is an embedding, each $\alpha_\Sigma$ is injective and each $\beta_\Sigma$ is bijective;
- an inclusion comorphism if $\Phi$ and each $\alpha_\Sigma$ are inclusions, and each $\beta_\Sigma$ 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 $\Gamma I \models \varphi$ in I can be solved by a theorem prover for J by just feeding the theorem prover with $\alpha(\Gamma) J \models \alpha(\varphi)$. 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: $\Phi$ may map signatures to theories (where a theory $(\Sigma, \Gamma)$ is a signature $\Sigma$ equipped with a set $\Gamma$ of $\Sigma$-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 Herts 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* 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, $\alpha$ and $\beta$ 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.

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 substitutions are a bit more complex. Namely, the substitution comorphism from 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 \( \text{NDNames} = \text{Names} \setminus \text{DNames} \) as the set of non-discourse names of a CL signature \( \Sigma = (\text{Names}, \text{DNames}, \text{Markers}) \).

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 add function and predicate symbols of any arity for each name, taking also into account the difference between the interpretation of a name and its interpretation as a constant symbol (see below). This allows us to map the

---

\footnote{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 $\Sigma = (\text{Names}, \text{DNames}, \emptyset)$ is translated to a FOL theory $\Theta(\Sigma) = (\Sigma', \Gamma)$ where

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

$\Gamma$ consists of a sentence asserting injectivity of $\text{inj}$; for each non-discourse name $n$, the sentence $\forall x : \text{UD}.\text{inj}(x) \neq [n]$; for each pair of names $n_1, n_2$, the infinite set of sentences $[n_1] = [n_2] \implies \forall x_1, \ldots, x_k : \text{UD}.n_1(x_1, \ldots, x_k) = n_2(x_1, \ldots, x_k)$ where $k \in \mathbb{N}$; and $[n_1] = [n_2] \implies \forall x_1, \ldots, x_k : \text{UD}.n_1(x_1, \ldots, x_k) \iff n_2(x_1, \ldots, x_k)$ 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 mode.\(^{10}\)

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 $\alpha_\Sigma$ 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 predicate $(p s_1 \ldots s_k)$ 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 $\Sigma$ be a CL.Fol signature and let $N$ be a $\Theta(\Sigma)$-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 $N_{n^f}$ (and $N_{n^p}$), 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:

- $\text{UR}^M = N_{\text{UR}} \uplus \{\ast\}$,
- $\text{UD}^M = N_{\text{inj}}(\text{UD})$,
- for any $d \in \text{DNames}$, $\text{int}^M(d) = N_{\text{inj}}(N_{[d]})$ and for any $n \in \text{NDNames}$, $\text{int}^M(n) = N_{[n]}$,
- for any $x \in \text{UR}^M$ and any $s_1, \ldots, s_k \in \text{UD}^M$, $\text{fun}^M(x)(N_{\text{inj}}(s_1), \ldots, N_{\text{inj}}(s_k)) = N_{\text{inj}}(f_{n^f}(s_1, \ldots, s_k))$ if there exists a name $n$ such that $\text{int}^M(n) = \Gamma$ and $\text{fun}^M(x)(N_{\text{inj}}(s_1), \ldots, N_{\text{inj}}(s_k)) = \ast$ otherwise,
- for any $x \in \text{UR}^M$ and any $s_1, \ldots, s_k \in \text{UD}^M$, $(N_{\text{inj}}(s_1), \ldots, N_{\text{inj}}(s_k)) \in \text{rel}^M(x)$ iff $(s_1, \ldots, s_k) \in N_{n^p}$ and there exists a name $n$ such that $\text{int}^M(n) = x$.

4.2 CL.Imp to FOL

The main idea is that now we add in the translation of a signature function symbols $\text{fun}$ and predicate symbols $\text{rel}$ 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 interpretation as a constant, like we did for CL.Fol. The drawback is that now $\text{rel}$ and $\text{fun}$ appear in all translated sentences, but on their first argument any term can occur.

A signature $\Sigma = (\text{Names}, \text{DNames}, \emptyset)$ 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 $\text{UD}$ and $\text{UR}$ on the universe instead. While thus the injection $\text{inj}$ 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 $\Gamma$ always ensure that the particular choice of $n$ is irrelevant.
\begin{itemize}
  \item $\Sigma'$ has two sorts $UR$ and $UD$.
  \item there is a function symbol $inj : UD \to UR$.
  \item for each $d \in DNames$, we have a constant symbol $d : UD$ and for each $n \in NDNames$ we have a constant symbol $n : UR$.
  \item we have a family of functions $\{fun : UR \times UD^k \to UD\}_{k \in \mathbb{N}}$
  \item and a family of predicate symbols $\{rel : UR \times UD^k\}_{k \in \mathbb{N}}$
  \item $\Gamma$ has the sentence that asserts that $inj$ is injective and for each non-discourse name $n$, the sentence $\forall x : UD. inj(x) \neq n$.
\end{itemize}

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

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

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

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 $\alpha$ 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 $\Gamma$ 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 $(\Sigma, E)$ and $\alpha(\Sigma, E)$. It is however easy to notice that by applying the translation, the logical consequences of $(\Sigma, E)$ are preserved and reflected by $\alpha$: a $\Sigma$-sentence $e$ is a consequence of $E$ over the signature $\Sigma$ 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

\footnote{We made explicit that we use the symbols $fun$ and $rel$ of arity $k$.}
• Φ(Σ) has three sorts, UD, UR and list,
• there is a function symbol inj : UD → UR,
• on sort list we have nil : list and cons : UD × list → list and m : list for each m ∈ Markers,
• for each d ∈ DNames we have a function symbol d : UD, for each n ∈ NDNames we have a function symbol n : UR,
• for each n ∈ Names, there is a function symbol n : list → UD and a predicate symbol n : list,
• Σ 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 ∀x : UD.inj(x) ≠ n; and for each pair of names n₁, n₂, the infinite set of sentences n₁ = n₂ ⇒ ∀x₁,...,xₖ ∈ UD.n₁([x₁,...,xₖ]) = n₂([x₁,...,xₖ]) where k ∈ N and n₁ = n₂ ⇒ ∀x₁,...,xₖ ∈ UD.n₁([x₁,...,xₖ]) ⇐⇒ n₂([x₁,...,xₖ]) where k ∈ N and [x₁,...,xₖ] is a notation for the list with elements x₁,...,xₖ.

Sentences are mapped by induction on their structure, such that each predication or term (t s) is mapped as α₂((t s)) = t(α₂(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 Φ(Σ)-model N, its reduct M = β₂(N) is defined as follows, using again the notations Nᶠ and Nᵖ, to make explicit that we refer to the overloaded function and predicate symbol n.
• UDᴺ = N(UD) ⊕ {*},
• UDᴺ = N(inj(UD))
• for any d ∈ DNames, intᴺ(d) = N(inj(Nᵈ)) and for any n ∈ NDNames, intᴺ(n) = Nⁿ,
• for any x ∈ UDᴺ and any s ∈ (N(UD))*, funᴺ(x)(Nᵳ(s)) = Nᵳ(Nᴺᵳ(s)), if there exists a name n such that intᴺ(n) = x and funᴺ(x)(Nᵳ(s)) = * otherwise,
• for any x ∈ UDᴺ and any s ∈ (N(UD))*, Nᵳ(s) ∈ relᴺ(x) iff s ∈ Nᵖ and there exists a name n such that intᴺ(n) = x,
• for any m ∈ Markers, seqᴺ(m) = Nᵳ(Nᵐ), that is, Nᵳ is applied each element of the list Nᵐ to obtain a new list with elements in UDᴺ.

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 Σ = ⟨Names, DNames, Markers⟩ is translated to a CFOL theory Φ(Σ) = ⟨Σ’, Γ⟩ such that
• Σ’ has three sorts, UR, UD and list,
• on UD we have a constant symbol d for each discourse name d ∈ DNames,
• on UR we have a constant symbol n for each non-discourse name n ∈ NDNames,
• on sort list we have nil : list, cons : UD × list → list and m : list for each m ∈ Markers,
• there is a function symbol fun : UR × list → UD and a predicate symbol rel : UR × 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. ∀x : UD.inj(x) ≠ n.
Theorem 2. proceeds in a similar way as above.

Given a \((\Sigma',\Gamma)\)-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 \(\Gamma\) 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 \(CL.\text{Fol}\) to \(FOL\) and \(CL.\text{Seq}\) to \(CFOL\) the arguments are similar, so we only present the proof of faithfulness for the second one.

**Theorem 1.** The comorphism \(CL.\text{Seq}\) to \(CFOL\) is faithful.

**Proof.** It suffices to prove that for any \(CL\)-signature \(\Sigma\), if \(\alpha\Sigma(\Gamma) \models_{\Phi(\Sigma)} \alpha\Sigma(e)\), then \(\Gamma \models_{\Sigma} e\).

We define a mapping \(\delta\Sigma : Mod^{CL}(\Sigma) \rightarrow Mod^{CFOL}(\Phi(\Sigma))\) such that \(M \models e\) iff \(\delta\Sigma(M) \models \alpha(e)\) where \(M\) is a \(\Sigma\)-model and \(e\) is a \(\Sigma\)-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 \(\text{list}\) and of the operations \(\text{cons}, \text{nil}\) 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 \(\Sigma\)-sentence \(e\).

Assume \(\alpha\Sigma(\Gamma) \models_{\Phi(\Sigma)} \alpha\Sigma(e)\). In order to show \(\Gamma \models_{\Sigma} e\), let \(M\) be a \(\Sigma\)-model such that \(M \models_{\Sigma} \Gamma\). Then \(\delta\Sigma(M) \models_{\Phi(\Sigma)} \alpha\Sigma(\Gamma)\). Hence \(\delta\Sigma(M) \models_{\Phi(\Sigma)} \alpha\Sigma(e)\), and thus \(M \models_{\Sigma} e\).

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 \(CL.\text{Imp}\) to \(CFOL\) and \(CL.\text{Full}\) 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:

\[ \alpha_{\mathcal{E}}(P(t_1, \ldots, t_n)) = (P \alpha_{\mathcal{E}}(t_1) \ldots \alpha_{\mathcal{E}}(t_n)) \]

Within terms, function applications are translated similarly:

\[ \alpha_{\mathcal{E}}(f(t_1, \ldots, t_n)) = (f \alpha_{\mathcal{E}}(t_1) \ldots \alpha_{\mathcal{E}}(t_n)) \]

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 \( \text{rel}^\mathcal{M}(\text{int}^\mathcal{M}(P)) \) and restricting to the arity of \( P \); similarly for function symbols (using \( \text{fun}^\mathcal{M} \)). 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 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 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 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 SROIQ notions, the comorphism is theoroidal.

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 SROIQ, e.g.

```plaintext
(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)))  // self-restriction
(forall (c d) (iff ((OWLsubsumes c d) // class subsumption (subclass)
     (forall (x) (iff (c x) (d x))))))
(forall (r) (iff ((OWLirr r) // irreflexivity
     (forall (x y) (not (r x y))))))
(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.

- $\alpha(\neg C) = (\text{OWLnot} \hspace{0.1cm} \alpha(C))$
- $\alpha(\exists R. C) = (\text{OWLsome} \hspace{0.1cm} \alpha(R) \hspace{0.1cm} \alpha(C))$

Translators 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.
• $\alpha(\geq nR.C) = (\text{owlmin } \alpha(R) \alpha(C) (a\ldots a))$

Here, the sequence $(a\ldots a)$ 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 $(\text{same-length } c \ldots) \ldots a$ above tests whether $\ldots a$ has length $n+1$, where $n$ is encoded by $\ldots$. Note that the value of $c$ is irrelevant here, it is just used for increasing the length of the sequence $\ldots$ 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 substitutions.

5 Case Study: Verifying Interpretations in COLORE

We have used HERRS 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. 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 ...) )
% Namespaces of other COLORE ontologies used here:
bi: <http://colore.googlecode.com/.../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
```

---

13 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.

\[ \text{ontology int:owltime_linear = int:owltime_interval then ins:owltime_instant} \]

Here we use Common Logic's import construct and add one axiom:

\[ \forall (x\ y) (\text{forall } (x\ y) (\text{or } (\text{leq } x\ y) (\text{leq } y\ x) (= x\ y))) \]

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

ontology ord:linear_ordering =

%% Here we use Common Logic's import construct and add one axiom:

\[ \forall (x\ y) (\text{forall } (x\ y) (\text{or } (\text{leq } x\ y) (\text{leq } y\ x) (= x\ y))) \]

ontology ord:linear_ordering =

%% Here we use Common Logic's import construct and add one axiom:

\[ \forall (x\ y) (\text{forall } (x\ y) (\text{or } (\text{leq } x\ y) (\text{leq } y\ x) (= x\ y))) \]

%% (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:complete_graphical = bi:grapical_incidence

%% 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 =

\[ \forall (x) (\text{iff } (\text{Instant } x) (\text{point } x)) \]

// mapping time instants to geometrical points
\[ \forall (x) (\text{iff } (\text{Interval } x) (\text{line } x)) \]

// mapping time intervals to geometrical lines
\[ \forall (x\ y) (\text{iff } (\text{begins } x\ y) \ (\text{or } (\text{and } \text{point } x\ y) (\text{leq } x\ y))) \]

// x begins with y iff x is a point ... (and point x) (line y) (in x y) // ... that is \leq any other point incident ...
\[ \forall (x\ y) (\text{iff } (\text{and } \text{point z}) (\text{line } y) (\text{leq } x\ z))) \]

%% 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.

interpretation geometry_of_time %mcons :

%% Interpretation of linearly ordered time intervals that ...
\[ \text{int:owltime_le} \]

%% ... begin and end with an instant as lines that are incident with linearly ...
to \{ \text{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 \text{int:mappings/owltime_interval_reduction} \} = \text{ProperInterval} \rightarrow \text{Interval} \]

In the case of our example, some proof goals introduced when verifying correctness of the interpretation...
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

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