Proof Support for Common Logic

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

[1] DFKI GmbH, Bremen, Germany
[2] University of Bremen, Germany
[3] University of Birmingham, UK
[4] University of Toronto, Canada

Abstract We present 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 full theorem proving support for Common Logic, including the possibility of verifying meta-theoretical relationships between Common Logic theories.

1 Introduction

Common Logic (CL) is an ISO-standardised[5] language based on untyped first-order logic, but extending it in several ways that ease the formulation of complex first-order theories. [24] discusses in detail the motivations and philosophical background of CL, arguing that it not only is a natural formalism for knowledege 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 there are several (also large) CL theories around, surprisingly little tool support has been realised so far. In this work, we fill this gap using the Heterogeneous Tool Set (HETS [30,31,25]). 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, define a number of important sublogics, and discuss its use cases and hitherto existing tool support. Section 3 briefly outlines the Heterogeneous Tool Set HETS and describes in some detail the integration of CL into HETS, enabling proof support for CL and its sublogics via various logic translations to already supported logics and automated reasoners. Section 4 discusses meta-theoretical relations between CL theories, and Section 5 illustrates the achieved CL reasoning support for both theorem proving and establishing meta-theoretical relationships between CL theories.

[5] Published as "ISO/IEC 24707:2007 — Information technology — Common Logic (CL): a framework for a family of logic-based languages" [14]

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 directly, and in particular, provide a succinct way for axiomatising polyadic functions and predicates.[6]

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. A Σ-model consists of a set UR, the universe of reference, with a non-empty subset $UD \subseteq UR$, the universe of discourse[7], and four mappings:

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

A Σ-sentence is a first-order sentence, where predications and function applications are written in a higher-order like syntax as $(t\ s)$. Here, t is an arbitrary term, and s is a sequence term, which can be a sequence of terms $t_1 \ldots t_n$, or a sequence marker. However, a predication $(t\ s)$ is interpreted like the first-order formula $holds(t, s)$, and a function application $(t\ s)$ like the first-order term $app(t, s)$, where $holds$ and app are fictitious symbols denoting the semantic objects rel and fun. In this way, CL provides a first-order simulation of a higher-order language, while still keeping the option of representing ordinary FOL predicate and function symbols (namely via non-discourse names, which do not denote values in the universe of discourse).

Interpretation of terms and formulae is as in first-order logic, with the difference that the terms at predicate resp. function symbol positions are interpreted with rel resp. fun in order to obtain the predicate resp. function, as discussed above. 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. For details, see [14]. As an example, consider the formula of the upper ontology DOLCE $\forall\phi(\phi(x))$, corresponding to $\bigwedge_{\psi \in \Pi}(\psi(x))$, where predicate variables ϕ, ψ range over a finite set Π of explicitly introduced universals. In CL, this is written, using the Lisp-like syntax of the CL Interchange Format CLIF:

```
(forall (?phi) (if (pi ?phi) (?phi ?x)))
```

Sequence markers add even more flexibility. For example, it is possible to express that a list of items is mutually different as follows (using the sequence marker "..."):

```
(mutually-different)  //the empty sequence is mutually different
```

[6] Strictly speaking, only the sequence markers go beyond first-order logic.

[7] 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 Common Logic, and the role of datatypes is currently subject of the discussion of the revision of the CL standard.

```
(mutually-different x) //singleton sequences are mutually different
(iff (mutually-different x y ...)   //recursion for length > 1
     (and (not (= x y))
          (mutually-different x ...)
          (mutually-different y ...) ))
```

For the rationale and methodology of CL and the possibility to define dialects covering different first-order languages, see [14]. CL also includes modules as a syntactic category, with a semantics that restricts locally the universe of discourse (see [33] 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 Sublogics of Common Logic

We define four 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.

2.2 Uses of Common Logic

Common Logic has been used for a number of tasks, see Table 1 for a summary, showing the (rough) number of lines, the number of axiomatised concepts (names), and the sublogic of Common Logic used.

Collection	lines	concepts	sublogic
COLORE	22,500	400	mostly CL.Fol, also CL.Seq, CL.Imp
SUMO-KIF	150,000	32,000	CL.Full
SUMO-CL	12,000	1,700	CL.Full
fUML	2,000	200	CL.Fol
PSL	1,000	50	CL.Fol

Table 1. Summary of the largest Common Logic ontologies and repositories

COLORE (Common Logic Ontology Repository)[8] is an open repository of over 600 Common Logic ontologies. One of the primary applications of COLORE is to support the verification of ontologies for commonsense domains such as time, space, shape, and

[8] http://code.google.com/p/colore/

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 Common Logic, and, based on these, representation theorems for generic ontologies for the above-mentioned commonsense domains. COLORE is mostly written in CL.Fol, only 13 ontologies use CL.Seq, and only one uses CL.Imp.

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[9]), SUMO-CL is a CL variant of SUMO produced by Kojeware (see also [6] for a discussion of higher-order aspects in SUMO). In table 1, we have added SUMO-KIF with all the mid-level and domain ontologies that are shipped with SUMO.

fUML (Foundational UML) is a subset of the Unified Modelling Language (UML) version 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.

2.3 Tool support for Common Logic

Current software tool support for Common Logic 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 [20] proposed an environment for ontology development in Common Logic, named Macleod; although this work includes detailed design documents for the environment, there is as yet no available implementation. Similarly [37] proposed a system architecture for COLORE that supports services for manipulating Common Logic ontologies, once more merely with basic functionality such as parsing being implemented. There exist some ad-hoc syntax translations of Common Logic to first-order provers, but these seem not to be backed up with a semantic analysis of their correctness.

Our main motivation for the present work is to remedy this situation of little tool support for Common Logic. We thus have extended the Heterogeneous Tool Set (HETS) with several kinds of tool support for Common Logic:

- a **parser** for the Common Logic Interchange Format (CLIF) and the Knowledge Interchange Format (KIF);
- a **sublogic analysis** for CL;
- a **connection** of CL to well-known **first-order theorem provers** such as SPASS, darwin and Vampire, such that logical consequences of CL theories can be proved;
- a connection of CL to the **higher-order provers** Isabelle/HOL and Leo-II in order to perform induction proofs in theories involving sequence markers;

[9] Common Logic can be considered the ISO-standardised and revised successor of KIF.

– a connection to **first-order model finders** such as darwin that allow one to find models for CL theories;
– support for **proving interpretations** between CL theories to be correct;
– a translation that **eliminates** the use of CL **modules**. Since the semantics of CL modules is special to CL, this elimination of modules is necessary before sending CL theories to a standard first-order prover;
– a translation of the **Web Ontology Language OWL** to CL;
– a translation of **propositional logic** to CL.

All this is based upon a formal semantic background. For the full functionalities of HETS, see the HETS user guide [29] (and the special CL version of the guide).[10]

3 Common Logic and The Heterogeneous Tool Set

3.1 The Heterogeneous Tool Set HETS

The Heterogeneous Tool Set (HETS) [30,31,25] is an open source software providing a general framework for formal methods integration and proof management. One can think of HETS 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.

HETS supports a number of input languages directly, such as Common Logic and OWL 2. For expressing meta relations between logical theories, HETS supports the Distributed Ontology Language (DOL), which is currently being standardised as ISO 17347.[11] 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. A DOL file usually imports files written in specific logics such as Common Logic or OWL 2.

The semantic background of HETS is the theory of institutions [16], formalising the notion of a logic. An institution provides a notion of signature, and for each signature, a set of sentences, a class of models and a satisfaction relation between models and sentences. Furthermore, an institution provides a notion of signature morphism, such that sentences can be translated along signature morphisms, and models against signature morphisms, in a way that satisfaction is preserved. By equipping Common Logic as defined above with such signature morphisms, Common Logic (CL) can be formalised

[10] Available at `https://svn-agbkb.informatik.uni-bremen.de/Hets/trunk/doc/UserGuideCommonLogic.pdf`
[11] See `http://www.ontoiop.org`

as an institution, see [22]. HETS also allows for plugging in logic translations, formalised as so-called institution comorphisms (see Sect. 3.3).

HETS is currently available for Linux and Mac OS X from the HETS home page `http://www.dfki.de/cps/hets`, where you also find packages ready for Ubuntu Linux.

3.2 HETS' support for Common Logic and its Relatives

HETS supports a variety of different logics. The following ones are most important for use with Common Logic:

Common Logic The Common Logic Interchange Format (CLIF) provides a Lisp-like syntax for Common Logic. HETS currently supports parsing CLIF and KIF.

OWL 2 is the Web Ontology Language recommended by the World Wide Web Consortium (W3C, `http://www.w3.org`); see [2]. It is used for knowledge representation on the Semantic Web [8]. HETS supports OWL 2 DL and the provers Fact++ and Pellet.

FOL/TPTP is untyped first-order logic with equality[12], underlying the interchange language TPTP [39], see `http://www.tptp.org`. [23] offers several automated theorem proving (ATP) systems for SPASS [41], Vampire [36], Eprover [38], Darwin [4], E-KRHyper [35], and MathServe Broker[13] [42]. These together comprise some of the most advanced theorem provers for first-order logic.

CFOL is many sorted first-order logic with so-called sort generation constraints, expressing the generation of a sort from (constructor) functions. In particular, this allows the axiomatisation of lists and other datatypes. CFOL is a sublogic of the Common Algebraic Specification Language CASL, see [32,9]. Proof support for CFOL is available through a poor man's induction scheme in connection with automated first-order provers like SPASS [23], or via the comorphism to HOL.

HOL is typed higher-order logic [11]. HETS actually supports several variants of HOL, among them THF0 (the higher-order version of TPTP [7]), with automated provers Leo-II [5], Satallax [12] and an automated interface to Isabelle [34], as well as the logic of Isabelle, with an interactive interface to Isabelle.

Proof support for CL, for which there is no dedicated theorem prover available, can be obtained by using logic translations to a logic supported by some prover, as discussed next.

3.3 Logic translations supported by HETS

The logic translations in HETS are formalised as so-called institution comorphisms [15]. A comorphism from logic I to logic J consists of

- a translation Φ of I-signatures to J-signatures,
- 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,[14]

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

[13] which chooses an appropriate ATP upon a classification of the FOL problem

[14] Actually, α and β also have to commute with translations along signature morphisms.

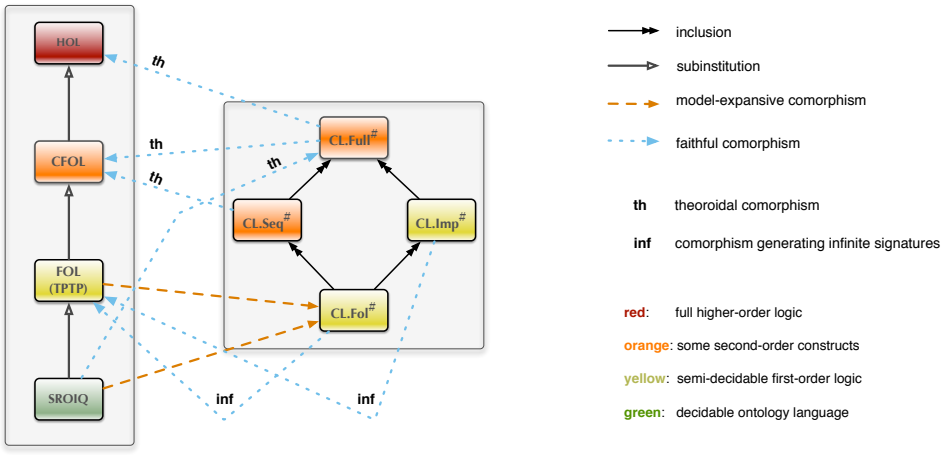

Figure 1. Graph of logics related to Common Logic that are currently supported by HETS.

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 crucial property of such comorphisms is *faithfulness*, which means that logical consequence is preserved and reflected along the comorphism:

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

In particular, this 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 $\alpha(\Gamma) \models^J \alpha(\varphi)$.

A comorphism is *model-expansive*, if each β_Σ is surjective. It is easy to see that each model-expansive comorphism is also faithful.

A comorphism is a *subinstitution comorphism*, if Φ is an embedding, each α_Σ is injective and each β_Σ is bijective[15]. Obviously, each subinstitution comorphism is model-expansive and hence also faithful. Moreover, subinstitution comorphism preserve the semantics of more advanced DOL structuring constructs such as renaming and hiding.

A subinstitution comorphism is an *inclusion comorphism*, if Φ and each α_Σ are inclusions, and each β_Σ is the identity.

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).

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,

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

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 SROIQ (the logic of OWL 2) into FOL is the standard translation [21], 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 one hand-side and Common Logic on the other hand-side, we cannot expect subinstitution comorphisms, but only model expansive and faithful comorphisms to run between the different parts.

In the direction from classical logic to Common Logic, the comorphisms in Fig. 1 are the following ones:

- a comorphism from classical FOL to CL.Fol. It maps constants to discourse names and function and predicate symbols to non-discourse names, with a straight-forward sentence and model translation;
- a comorphism from SROIQ to CL.Fol, corresponding to the standard translation [21], and
- a comorphism from SROIQ to CL.Full. Here, the higher-order like features of Common Logic are used to define all features of OWL 2, including Boolean operators on concepts, inverses of roles etc. directly, such that the translation of sentences becomes trivial. In particular, `mutually-different` plays a crucial role for the translation of number restrictions. Due to the theory needed for axiomatisation of the OWL 2 features, this comorphism is theoroidal.

In the direction from Common Logic to classical logic, the comorphisms in Fig. 1 are all quite similar. The idea behind these translations is to make explicit in the signature the functional and relational interpretations of the elements of the universe. The symbols introduced for this depend on the source sublogic. Namely, if arbitrary terms are allowed on function/predicate positions in the sentences, the interpretation is given by the signature symbols rel and fun; otherwise, the interpretation is given by signature symbols with the same name as the CL names. Furthermore, the arity of these symbols is determined by the presence of sequence markers: if the sublogic contains sequence markers, then in the translated signature we introduce the datatype of lists[16] which is then used as an argument; otherwise, the arguments are simply from the universe. The table on the next page gives an overview of the signature translation component of the four comorphisms in the table below. We denote by n an arbitrary name and by m an arbitrary marker in a CL signature.

CL.Fol $\rightarrow$ FOL	CL.Imp $\rightarrow$ FOL
CL.Seq $\rightarrow$ CFOL	CL.Full $\rightarrow$ CFOL

[16] Here, we need many-sortedness and sort generation constraints from CFOL.

op $n : k$ **pred** $n : k$ for each $k \in \mathbb{N}$	**op** $n : 1$ **op** $fun : k$ **pred** $rel : k$ for each $k \in \mathbb{N}$
op $n : ind$ **op** $m : list$ **op** $n : list \to ind$ **op** $++ : list \times list \to list$ **pred** $n : list$	**op** $n : ind$ **op** $m : list$ **op** $fun : ind \times list \to list$ **op** $++ : list \times list \to list$ **pred** $rel : ind \times list$

The sentence translation component of the comorphisms, denoted α_Σ, is defined inductively on the structure of sentences. The inductive base is given in the table below, where $t(s)$ denotes a term or a predicate in a CL sentence.

$t(\alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$	$rel^k(\alpha_\Sigma(t), \alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$
$t(\alpha_\Sigma(s))$	$rel(\alpha_\Sigma(t), \alpha_\Sigma(s))$

On the bottom line of the table, we denote by $\alpha_\Sigma(s)$ the translation of a sequence s to CFOL. Recall that a sequence s is either a sequence marker, or a juxtaposition of terms or a juxtaposition of sequences. The translation is defined inductively as taking each sequence marker to its corresponding constant of sort $list$, each juxtaposition of terms $t_1 \ldots t_n$ to $cons(\alpha_\Sigma(t_1), \ldots, cons(\alpha_\Sigma(t_n), nil))$ and each juxtaposition of sequences $s_1 \ldots s_n$ to $\alpha_\Sigma(s_1) + + \cdots + + \alpha_\Sigma(s_n)$.

Moreover, we have defined a comorphism $CL.Full \to HOL$ in a similar way as $CL.Full \to CFOL$, except that now HOL lists are used.

Note that for the comorphisms in the top line of our table we generate infinite signatures. Of course, a tool can only work with finite signatures. Therefore, after translating a CL theory (Σ, Γ) along one of the two translations, we further apply a syntactic transformation γ that removes from the signature of the translated theory the symbols that do not appear in the translation of Γ, denoted by Γ'. It is easy to notice that by doing this we have $\Gamma' \models_\Sigma e \iff \Gamma' \models_{\gamma(\Sigma)} e$ for each $\gamma(\Sigma)$-sentence e.

The proofs that each of these comorphisms are faithful are similar. The key idea is to define a function δ_Σ taking CL-models to (C)FOL-models such that $\beta_\Sigma(\delta_\Sigma(M))$ and M have the same Γ-consequences e where Γ is a Σ-theory, M is a Γ-model and e is a Σ-sentence that only uses terms on function/predicate positions with the same number of arguments as in Γ, and then the proof follows using the satisfaction condition of the comorphism.

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;

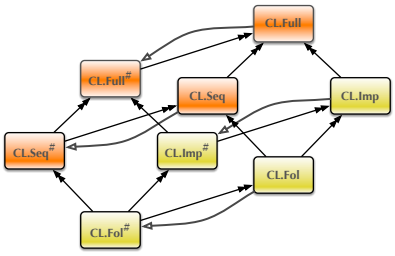

Figure 2. Elimination of the module construct in CL.

– while the restrictions are obviously inclusions, the reverse translations are obtained as subinstitutions.

4 Relations between Common Logic Texts

Common Logic itself does not support the specification of logical consequences, nor relative theory interpretations, nor other features that speak about structuring and comparing logical theories. Therefore, DOL must be used for these purposes, and consequently the Common Logic Repository COLORE [17] already contains several DOL files.

Relations between logical theories are expressed in the so-called development graph calculus [26]. Development graphs are a simple kernel formalism for (heterogeneous) structured theorem proving and proof management.

A development graph consists of a set of nodes (corresponding to whole structured specifications or parts thereof), and a set of arrows called *definition links*, indicating the dependency of each involved structured specification on its subparts. Each node is associated with a signature and some set of local axioms. The axioms of other nodes are inherited via definition links. Definition links are usually drawn as black solid arrows, denoting an import of another specification.

Complementary to definition links, which *define* the theories of related nodes, *theorem links* serve for *postulating* relations between different theories. Theorem links are the central data structure to represent proof obligations arising in formal developments.

HETS shows development graphs when started with the `-g` option.

We support the following relations between Common Logic texts:

Importation is defined in ISO/IEC 24707:2007 [14] as virtual copying of a resource. In HETS, a definition link to the imported theory is created. HETS also supports URIs for importing resources. The allowed URI schemes are `file:`, `http:` and `https:`.

```
(cl-imports file:///absolute/path/to/someFile.clif)
(cl-imports http://someDomain.com/path/to/someFile.clif)
(cl-imports https://someDomain.com/path/to/someFile.clif)
```

Relative interpretation is formally defined in [19]. Informally, one module relatively interprets those "modules whose theorems are preserved within the current module through [a] mapping. There exists a mapping between modules such that all theorems of the other module hold in the current module after the mapping is applied." [18] The formal semantics of a relative interpretation is that each model of the target theory, when reduced to the source theory (along the signature morphism induced by the symbol map), is a model of the source theory.

HETS represents relative interpretation by a theorem link (display as a red arrow) in the development graph.

The DOL syntax for relative interpretations is

interpretation *i* : *someCLText* **to** *someTargetCLText* **end**

or

interpretation *i* : *someCLText* **to** *someTargetCLText* = <*symbol map (see below)*> **end**

where a symbol map allows for renaming symbols, e.g.

name1Old $\mapsto$[17] *name1New*, *name2Old* $\mapsto$ *name2New*.

We provide an example interpretation (without symbol maps) below.

Just as with imports (see above), HETS supports different types of references to resources here, such as URIs.

Conservative extension A theory T_2 conservatively extends a theory T_1, if each model of T_1 can be expanded to a model of T_2. Conservative extensions are another form of proof obligation in HETS (annotated to definition links or theorem links), and they need to be discharged with specific tools. HETS interfaces several conservativity checkers for OWL 2, and features a built-in conservativity checker for CFOL.

Non-conservative extension is informally defined as follows: One module non-conservatively extends other modules, if its "axioms entail new facts for the shared lexicon [= signature in the terminology of this paper] of the [other] module(s). [That is, the] shared lexicon between the current module and a [non-conservatively extended] module are not logically equivalent with respect to their modules." [18].

HETS represents non-conservative extension by a definition link in the development graph.

Except for importation, one can specify an optional symbol map (name map) in a relation.[18] Names from the source theory are mapped to names from the target theory. The semantics is given by a signature morphism, which is used to decorate the corresponding (definition or theorem) link.

Relative Interpretation in COLORE We give an example for relative interpretation in COLORE. The COLORE [17] module `RegionBooleanContactAlgebra` relatively interprets the module `AtomlessBooleanLattice`. These two modules specify axioms about Booleans; thus, they have the same signature.

For use with HETS, we have made a dump of the COLORE contents available in `CommonLogic/colore` in the HETS library [1].

```
distributed-ontology COLORE-RelativeInterpretation

logic CommonLogic

ontology AtomlessBooleanLattice =
  http://colore.googlecode.com/svn/trunk/ontologies/complex/lattices/boolean_lattice.clif
then
. (forall (x) (exists (y) (and (not (= y 0)) (leq y x))))
end

ontology RegionBooleanContactAlgebra =
  http://colore.googlecode.com/svn/trunk/ontologies/core/contact_algebras/boolean_contact_algebra.
clif
then
. (forall (x)
          (if (and (not (= x 0)) (not (= x 1)))
              (exists (y) (and (complement x y) (C x y)))))
end

interpretation i : AtomlessBooleanLattice to RegionBooleanContactAlgebra
```

[17] alternative ASCII syntax: |->

[18] While the "copy" semantics of Common Logic importations does not permit renamings, DOL's extension mechanism offers an alternative possibility to reuse ontologies and rename some of their symbols, using the "*importedSpec* **with** *name1Old* $\mapsto$ *name1New*, *name2Old* $\mapsto$ *name2New* **then** *importingSpec*" syntax.

5 Proof Support for Common Logic in HETS

The proof calculus for development graphs [26] reduces global theorem links to local proof goals. This reduction can be done in the HETS development graph window via the Edit/Proofs/Auto-DG-Prover menu item. Local proof goals (indicated by red nodes in the development graph) can eventually be discharged using a theorem prover, i.e. by using the "Prove" menu of a red node.

The graphical user interface (GUI) for calling a prover is shown in Fig. 3 — we call it "Proof Management GUI". The top list on the left shows all goal names prefixed with their proof status in square brackets. A proved goal is indicated by a '+', a '–' indicates a disproved goal, a space denotes an open goal, and a '×' denotes an inconsistent theory (think of a fallen '+').

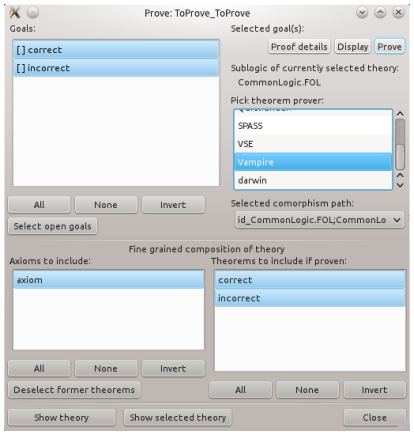

Figure 3. HETS Goal and Prover Interface

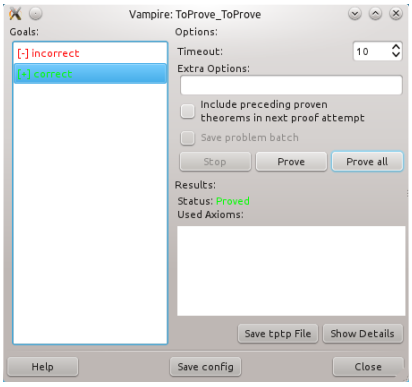

Figure 4. Interface of Vampire Prover

When opening this GUI when processing the goals of one node for the first time, it will show all goals as open. Within this list one can select those goals that should be inspected or proved. The GUI elements are the following:

- The button 'Display' shows the selected goals in the ASCII syntax of this theory's logic in a separate window.
- The 'Proof details' button opens a window that shows for each proved goal the used axioms, its proof script, and its proof — the level of detail depends on the used theorem prover.
- The list 'Pick Theorem Prover:' lets you choose one of the connected provers. 'Prove' launches the selected prover; the theory along with the selected goals is translated via the shortest possible path of comorphisms into the prover's logic.
- The pop-up choice box below 'Selected comorphism path:' lets you pick a (composed) comorphism to be used for the chosen prover. In HETS, the comorphisms CL.Fol → FOL and CL.Imp → FOL have been united into `CommonLogic2CASLCompact`,

while CL.Seq → CFOL and CL.Full → CFOL have been united into `CommonLogic2CASLFol`.
- Since the amount and kind of sentences sent to an ATP system is a major factor for the performance of the ATP system, it is possible to fine tune the lists of the axioms and proven theorems that will comprise the theory of the next proof attempt.
- When pressing the bottom-right 'Close' button the window is closed and the status of the goals' list is integrated into the development graph. If all goals have been proved, the selected node turns from red into green.
- All other buttons control selecting list entries.

5.1 Consistency Checker and Disproving

Since proofs are void if theories are inconsistent, their consistency should be checked by the HETS Consistency checker interface, which provides access to a list of model finders. As with proving, suitable comorphisms can be used to bridge logics such as FOL/TPTP that come with model finders. For example, when checking consistency of a CL.Imp theory, using the comorphism CL.Imp → FOL, the FOL model finder darwin can be used for model finding. Darwin will output the found model as a theory in FOL syntax. With the model translation component β of the comorphism, the found model can be translated back to a model in CL.Imp.

Each development graph node that is red (i.e. has open proof obligations) also features a "Disprove" button. HETS then adds then negation of the proof goal to the theory of the node and calls model finders. If a model of the thus extended theory has been found, it is a countermodel to the provability of the goal.

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 and the structuring capabilities of the DOL language. As CL is a popular language within ontology communities interested in greater expressive power than provided by the decidable OWL DL language, this is a substantial step towards supporting more ambitious ontology engineering efforts.

We have used HETS for the verification of a number of consequences and interpretations of COLORE theories, as well as for the check of their consistency. During this process, numerous errors in COLORE have been found and corrected. 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.

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`.

References

1. Hets library. http://www.cofi.info/Libraries.
2. OWL 2 web ontology language: Document overview. W3C recommendation, World Wide Web Consortium (W3C), October 2009.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
4. Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Implementing the Model Evolution Calculus. In Stephan Schulz, Geoff Sutcliffe, and Tanel Tammet, editors, *Special Issue of the International Journal of Artificial Intelligence Tools (IJAIT)*, volume 15 of *International Journal of Artificial Intelligence Tools*, 2005. Preprint.
5. Christoph Benzmüller, Lawrence C. Paulson, Frank Theiss, and Arnaud 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 *Lecture Notes in Computer Science*, pages 162–170. Springer, 2008.
6. 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. DOI 10.1016/j.websem.2011.11.008.
7. 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.
8. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.
9. M. Bidoit and P. D. Mosses. CASL *User Manual*, volume 2900 of *LNCS*. Springer, 2004. Free online version available at http://www.cofi.info.
10. Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.
11. Tomasz Borzyszkowski. Higher-order logic and theorem proving for structured specifications. In Didier Bert, Christine Choppy, and Peter D. Mosses, editors, *WADT*, volume 1827 of *Lecture Notes in Computer Science*, pages 401–418. Springer, 1999.
12. Chad E. Brown. Satallax: An automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 111–117. Springer, 2012.
13. CoFI (The Common Framework Initiative). CASL *Reference Manual*. LNCS Vol. 2960 (IFIP Series). Springer, 2004.
14. 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.
15. J. Goguen and G. Roşu. Institution morphisms. *Formal aspects of computing*, 13:274–307, 2002.
16. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992. Predecessor in: LNCS 164, 221–256, 1984.
17. M. Gruninger. COLORE – Common Logic Repository. Available at http://stl.mie.utoronto.ca/colore/.
18. M. Gruninger et al. COLORE – metadata definitions. Available at http://stl.mie.utoronto.ca/colore/metadata.html.
19. Michael Gruninger, Torsten Hahmann, Ali Hashemi, and Darren Ong. Ontology verification with repositories. In Antony Galton and Riichiro Mizoguchi, editors, *FOIS 2010*, number 209 in Frontiers in Artificial Intelligence and Applications, pages 317–330. IOS Press, 2010.
20. 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.

21. Yevgeny Kazakov. Riq and sroiq are harder than shoiq. In Gerhard Brewka and Jérôme Lang, editors, *KR*, pages 274–284. AAAI Press, 2008.
22. 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. Special Issue on 'Is Logic Universal?'.
23. 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.
24. Christopher Menzel. Knowledge representation, the World Wide Web, and the evolution of logic. *Synthese*, 182:269–295, 2011.
25. T. Mossakowski. Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen, 2005.
26. T. Mossakowski, S. Autexier, and D. Hutter. Development graphs – proof management for structured specifications. *Journal of Logic and Algebraic Programming*, 67(1-2):114–145, 2006.
27. Till Mossakowski. Relating CASL with other specification languages: the institution level. *Theoretical Computer Science*, 286:367–475, 2002.
28. Till Mossakowski and Oliver Kutz. The Onto-Logical Translation Graph. In *Modular Ontologies—Proceedings of the Fifth International Workshop (WoMO 2011)*, volume 230 of *Frontiers in Artificial Intelligence and Applications*, pages 94–109. IOS Press, 2011.
29. Till Mossakowski, Christian Maeder, and Mihai Codescu. Hets user guide, 2011. `http://www.dfki.de/cps/hets`.
30. Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set. In Orna Grumberg and Michael Huth, editors, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer-Verlag Heidelberg, 2007.
31. Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set. In Bernhard Beckert, editor, *VERIFY 2007*, volume 259 of *CEUR Workshop Proceedings*. 2007.
32. 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`.
33. Fabian Neuhaus and Pat Hayes. Common logic and the Horatio problem. accepted by Applied Ontology, 2011.
34. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer Verlag, 2002.
35. 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.
36. Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2-3):91–110, 2002.
37. C. Ross. COLORE system architecture. Available at `http://ontolog.cim3.net/cgi-bin/wiki.pl?OpenOntologyRepository_Architecture/From_COLORE`.
38. S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.
39. Geoff Sutcliffe. The TPTP world – infrastructure for automated reasoning. In Edmund M. Clarke and Andrei Voronkov, editors, *LPAR (Dakar)*, volume 6355 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2010.
40. Geoff Sutcliffe and Christian B. Suttner. Evaluating general purpose automated theorem proving systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.
41. C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobalt, and D. Topic. SPASS version 2.0. In Andrei Voronkov, editor, *Automated Deduction – CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 275–279. Springer-Verlag, July 27-30 2002.
42. 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.

Optional Appendix (not for inclusion in the final version)

A Institutions for Common Logic and its Neighbours

We now cast Common Logic and its most important neighbours in the logic graph as institutions, following [22,28], but adding a more fine-grained analysis of CL's subinstitutions and their connections.

First, we recall how specification frameworks in general may be formalized in terms of so-called institutions [16].

An *institution* $I = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models)$ consists of

- a category $\mathbf{Sig}$ of *signatures*,
- a functor $\mathbf{Sen}\colon \mathbf{Sig} \to \mathbf{Set}$ giving, for each signature Σ, a set of *sentences* $\mathbf{Sen}(\Sigma)$, and for each signature morphism $\sigma\colon \Sigma \to \Sigma'$, a *sentence translation map* $\mathbf{Sen}(\sigma)\colon \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$, where $\mathbf{Sen}(\sigma)(\varphi)$ is often written $\sigma(\varphi)$,
- a functor $\mathbf{Mod}\colon \mathbf{Sig}^{op} \to \mathbf{Cat}$ [19] giving, for each signature Σ, a category of *models* $\mathbf{Mod}(\Sigma)$, and for each signature morphism $\sigma\colon \Sigma \to \Sigma'$, a *reduct functor* $\mathbf{Mod}(\sigma)\colon \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$, where $\mathbf{Mod}(\sigma)(M')$ is often written $M'|_\sigma$, and
- a satisfaction relation $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ for each $\Sigma \in \mathbf{Sig}$,

such that for each $\sigma\colon \Sigma \to \Sigma'$ in $\mathbf{Sig}$ the following *satisfaction condition* holds:

$$M' \models_{\Sigma'} \sigma(\varphi) \qquad \iff \qquad M'|_\sigma \models_\Sigma \varphi$$

for each $M' \in \mathbf{Mod}(\Sigma')$ and $\varphi \in \mathbf{Sen}(\Sigma)$.

Definition 1 (Untyped First-order Logic (FOL)). *In the institution* $\mathsf{FOL}^{=}$ *of untyped first-order logic with equality, signatures are first-order signatures, consisting of a set of function symbols with arities, and a set of predicate symbols with arities. Signature morphisms map symbols such that arities are preserved. Models are first-order structures, and sentences are first-order formulas. Sentence translation means replacement of the translated symbols. Model reduct means reassembling the model's components according to the signature morphism. Satisfaction is the usual satisfaction of a first-order sentence in a first-order structure.*

Definition 2 (Many-sorted First-order Logic with sort generation constraints (CFOL) [9,13]). *The institution* $\mathsf{FOL}^{ms=}$ *of many-sorted first-order logic with equality is similar to* $\mathsf{FOL}^{=}$*, the main difference being that signatures are many-sorted first-order signatures, consisting of sorts and typed function and predicate symbols, and that formulas need to be well-typed. For details, see [16]. A* sort generation constraint *states that a given set of sorts is generated by a given set of functions. Technically, sort generation constraints also contain a signature morphism component; this allows them to be translated along signature morphisms without sacrificing the satisfaction condition. Formally, a sort generation*

[19] Here, $\mathbf{Cat}$ is the quasi-category of all categories. As metatheory, we use $ZFCU$, i.e. ZF with axiom of choice and a set-theoretic universe U. This allows for the construction of quasi-categories, i.e. categories with more than a class of objects.

constraint over a signature Σ is a triple $(\widetilde{S}, \widetilde{F}, \theta)$, where $\theta\colon \overline{\Sigma} \to \Sigma$, $\overline{\Sigma} = (\overline{S}, \overline{TF}, \overline{PF}, \overline{P})$, $\widetilde{S} \subseteq \overline{S}$ and $\widetilde{F} \subseteq \overline{TF} \cup \overline{PF}$.

A Σ-constraint $(\widetilde{S}, \widetilde{F}, \theta)$ is satisfied in a Σ-model M iff the carriers of $M|_\theta$ of sorts in $\widetilde{S}$ are generated by the function symbols in $\widetilde{F}$, i.e. for every sort $s \in \widetilde{S}$ and every value $a \in (M|_\theta)_s$, there is a $\overline{\Sigma}$-term t containing only function symbols from $\widetilde{F}$ and variables of sorts not in $\widetilde{S}$ such that $\nu^\#(t) = a$ for some valuation ν into $M|_\theta$. Here, $\nu^\#$ is the usual extension of the valuation ν from variables to terms. □

Although not strictly more expressive than untyped FOL$^=$, introducing a sort structure allows a cleaner and more principled design of first-order ontologies. Moreover, axioms involving different sorts can be stated more succinctly, and static type checking gives more control over correct modelling.

Definition 3 (Common Logic - CL). *Common Logic (CL) has first been formalised as an institution in [22]. We here only provide the missing details that have not been presented in Section 2. A signature morphism consists of two maps between these sets, such that the property of being a discourse name is preserved and reflected.*[20] *Model reducts leave UR, UD, rel and fun untouched, while int and seq are composed with the appropriate signature morphism component. Sentence translation along a signature morphism is just replacement of names.*

Definition 4 (Description Logics: OWL and its profiles EL, QL, RL). *Signatures of the description logic $\mathcal{ALC}$ consist of a set $\mathcal{A}$ of atomic concepts, a set $\mathcal{R}$ of roles and a set $\mathcal{I}$ of individual constants, while signature morphisms provide respective mappings. Models are single-sorted first-order structures that interpret concepts as unary and roles as binary predicates. Sentences are subsumption relations $C_1 \sqsubseteq C_2$ between concepts, where concepts follow the grammar*

$$C ::= \mathcal{A} \mid \top \mid \bot \mid C_1 \sqcup C_2 \mid C_1 \sqcap C_2 \mid \neg C \mid \forall R.C \mid \exists R.C$$

These kind of sentences are also called TBox sentences. Sentences can also be ABox sentences, which are membership assertions of individuals in concepts (written $a : C$ for $a \in \mathcal{I}$) or pairs of individuals in roles (written $R(a, b)$ for $a, b \in \mathcal{I}, R \in \mathcal{R}$). Sentence translation and reduct is defined similarly as in FOL$^=$. Satisfaction is the standard satisfaction of description logics.

The logic $\mathcal{SROIQ}$, which is the logical core of the Web Ontology Language OWL 2 DL[21] *extends $\mathcal{ALC}$ with the following constructs: (i) complex role boxes (denoted by $\mathcal{SR}$): these can contain: complex role inclusions such as $R \circ S \sqsubseteq S$ as well as simple role hierarchies such as $R \sqsubseteq S$, assertions for symmetric, transitive, reflexive, asymmetric and disjoint roles (called RBox sentences), as well as the construct $\exists R.$Self (collecting the set of 'R-reflexive points'); (ii) nominals (denoted by $\mathcal{O}$); (iii) inverse roles (denoted by $\mathcal{I}$); qualified and unqualified number restrictions ($\mathcal{Q}$). $\mathcal{SROIQ}$ can be straightforwardly rendered as an institutions following the previous examples.*

*The OWL 2 specification contains three further DL fragments of $\mathcal{SROIQ}$, called **profiles**, namely EL, QL, and RL.*[22] *These are obtained by imposing syntactic restrictions on*

[20] That is, a name is a discourse name if and only if its image under the signature morphism is.

[21] See also `http://www.w3.org/TR/owl2-overview/`

[22] See `http://www.w3.org/TR/owl2-profiles/` for details of the specifications.

the language constructs and their usage, with the motivation that these fragments are of lower expressivity and support specific computational tasks. For instance, RL *is designed to make it possible to implement reasoning systems using rule-based reasoning engines,* QL *to support query answering over large amounts of data, and* EL *is a sub-Boolean fragment sufficiently expressive e.g. for dealing with very large biomedical ontologies such as the NCI thesaurus. To sketch one of these profiles in some more detail, the (sub-Boolean) description logic $\mathcal{EL}$ underlying* EL *has the same sentences as $\mathcal{ALC}$ but restricts the concept language of $\mathcal{ALC}$ as follows:*

$$C ::= B \mid \top \mid C_1 \sqcap C_2 \mid \exists R.C$$

Given that EL*,* QL*, and* RL *are obtained via syntactic restrictions but leaving the overall $\mathcal{SROIQ}$ semantics intact, it is obvious that they are subinstitutions of $\mathcal{SROIQ}$.* □

Apart from some exceptions[23], description logics can be seen as fragments of first-order logic via the standard translation [3] that translates both the syntax and semantics of various DLs into untyped first-order logic.

Definition 5 (HOL). *[11] presents an institution for a higher-order logic extending Church's type theory with polymorphism; this is basically the higher-order logic used in modern interactive theorem provers like Isabelle/HOL [34] (one additional feature of Isabelle are type classes).*

B Institution Comorphisms

We can define a lattice of sublogics for Common Logic, based on its features: sequence markers and occurrence of arbitrary terms on function/predicate positions of a theory. We thus obtain:

CL.Full : both sequence markers and arbitrary terms on function/predicate positions are allowed. This is the full Common Logic.

CL.Imp : there are no sequence markers in the signature, arbitrary terms on function/-predicate positions are allowed.

CL.Seq : the signature contains sequence markers, but only names are permitted on function/predicate positions.

CL.FOL : no sequence markers or other terms than names are used.

Definition 6. *Given two institutions I_1, I_2 with $I_i = (\mathbf{Sig}_i, \mathbf{Sen}_i, \mathbf{Mod}_i, \models^i)$, an institution comorphism from I_1 to I_2 consists of a functor $\Phi : \mathbf{Sig}_1 \to \mathbf{Sig}_2$ and natural transformations $\beta : \Phi;\ \mathbf{Mod}_2 \Rightarrow \mathbf{Mod}_1$ and $\alpha : \mathbf{Sen}_1 \Rightarrow \Phi;\ \mathbf{Sen}_2$, such that the following satisfaction condition holds:*

$$M' \models^2_{\Phi(\Sigma)} \alpha_\Sigma(e) \iff \beta_\Sigma(M') \models^1_\Sigma e,$$

where Σ is an I_1-signature, e is a Σ-sentence in I_1 and M' is a $\Phi(\Sigma)$-model in I_2.

[23] For instance, adding transitive closure of roles or fixpoints to DLs makes them decidable fragments of second-order logic [10].

B.1 CL.FOL to CFOL.

A signature $\Sigma = (Names, DNames, \emptyset)$ is translated to a CFOL signature $\Phi(\Sigma)$ with a single sort, that we denote by $individual$, and which has for each name n and each natural number k

- a k-ary function symbol $n : individual^k \to individual$ and
- a k-ary predicate symbol $n : individual^k$.

Sentences are mapped inductively on their structure and based on the translation of terms. Since we are in the FOL sublogic of Common Logic, we know that we have only names on function/predicate positions, and therefore we can define the translation α_Σ of a term $f(s)$ as $\alpha_\Sigma(f(s)) = f(\alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$ and the translation of a predication $p(s)$ as $\alpha_\Sigma(p(s)) = p(\alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_k))$, if s is a sequence of length k.

For the model reduction component of the comorphism, let $\Sigma = (Names, DNames, \emptyset)$ be a Common Logic.FOL signature and let N be a $\Phi(\Sigma)$-model. Then $M = \beta_\Sigma(N)$ is defined as follows:

- $UR^M = N_{individual} \uplus Names$,
- $UD^M = N_{individual} \uplus DNames$,
- for any $n \in Names$, $int^M(n) = n$
- for any $x \in UR^M$, $fun^M(x) : (UD^M)^* \to UD$ is defined as $fun^M(x)(s) = N_x(s_1, \ldots, s_k)$ if s is a sequence of length k of elements of $N_{individual}$ and x is in $Names$, [24], or $fun^M(x)(s) = choose(Names)$, where $choose(Names)$ is a function returning an element of $Names$ otherwise (that is, either s contains an element from $Names$ or x is an element of $N_{individual}$).
- for any $x \in UR^M$, $rel^M(x)(s)$ is defined as either $N_x(s_1, \ldots, s_k)$, if s is a sequence of length k of elements of $N_{individual}$ and x is in $Names$, or $False$, otherwise.

B.2 CL.Imp to CFOL.

A signature $\Sigma = (Names, DNames, \emptyset)$ is translated to a CFOL signature $\Phi(\Sigma)$ with a single sort, that we denote by $individual$, and which has

- a constant $n : individual$ for each domain name $n \in Names$
- a family of function symbols $\{fun : individual^{k+1} \to individual\}_{k \in \mathbb{N}}$
- a family of predicate symbols $\{rel : individual^{k+1}\}_{k \in \mathbb{N}}$

Sentences are translated inductively on their structure, with predications $t(s)$ translated to $\alpha_\Sigma(t(s)) = rel(\alpha_\Sigma(t), \alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_n))$ and terms $t(s)$ translated to $\alpha_\Sigma(t(s)) = fun(\alpha_\Sigma(t), \alpha_\Sigma(s_1), \ldots, \alpha_\Sigma(s_n))$, where n is the length of s. Since there are no sequence markers, the length of s is always known.

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

- $UR^M = N_{individual} \uplus Names$,
- $UD^M = N_{individual} \uplus DNames$,
- for any $n \in Names$, $int^M(n) = n$,

[24] Notice that since N is a $\Phi(\Sigma)$-model, it must provide a function N_x of any arity.

- for any $x \in UR^M$, $fun^M(x)(s) = N_{fun^k}(n, s_1, ..., s_k)$ if x is a name and s is a sequence of elements in $N_{individual}$ of length k,[25] or $fun^M(x)(s) = choose(Names)$ otherwise
- for any $x \in UR^M$, $rel^M(x)(s_1, ..., s_k) = N_{rel^k}(x, s_1, ..., s_k)$ if x is a name and s is a sequence of elements in $N_{individual}$ of length k (with the same notational convention as above), or $rel^M(x)(s) = False$ otherwise.

B.3 Getting finite signatures.

The signatures $\Phi(\Sigma)$ obtained for the two comorphisms are infinite. Of course, a tool works with finite signatures. 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 theory Γ all symbols that do not occur in Γ. For the first comorphism, this means that for each name $n \in DNames$, we only introduce a function/predicate symbol of arity k if the sentences in Γ 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 Γ. Since this transformation is only syntactical, we do not need to define a corresponding translation between the class of models of (Σ, Γ) and $(\Phi(\Sigma), \Gamma)$. It is however easy to notice that for any Σ-sentence e we have that $\Gamma \models_\Sigma e \Leftrightarrow \Gamma \models_{\Phi(\Sigma)} e$.

B.4 CL.Seq to CFOL.

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

- $\Phi(\Sigma)$ has two sorts, $individual$ and $list$
- on sort $list$ we have $nil : list$ and $cons : individual \times list \to list$ and $m : list$ for each $m \in Markers$
- for each $n \in Names$ we have a function symbol $n : individual$, a function symbol $n : list \to individual$ and a predicate symbol $n : list$
- Γ consists of the sort generation constraint that asserts that $list$ is a free type over its constructors nil and $cons$

Sentences are mapped inductively on their structure, such that each predicate or term $t(s)$ is mapped to $\alpha_\Sigma(t(s)) = t(\alpha_\Sigma(s))$.

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

- $UR^M = N_{individual} \uplus Names$,
- $UD^M = N_{individual} \uplus DNames$,
- for any $n \in Names$, $int^M(n) = n$,
- for any $x \in UR^M$, $fun^M(x)(s) = N_x(s)$ if x is a name and s is a sequence of elements in $N_{individual}$, or $fun^M(x)(s) = choose(Names)$ otherwise
- for any $x \in UR^M$, $rel^M(x)(s_1, ..., s_k) = N_x(s)$ if x is a name and s is a sequence of elements in $N_{individual}$, or $rel^M(x)(s) = False$ otherwise.
- for any $m \in Markers$, $seq^M(m) = N_m$.

[25] Note that we made explicit that we use the interpretation of the function symbol $fun : individual^{k+1} \to individual$.

B.5 CL.Full to CFOL.

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

- $\Phi(\Sigma)$ has two sorts, $individual$ and $list$
- on sort $list$ we have $nil : list$ and $cons : individual \times list \to list$ and $m : list$ for each $m \in Markers$
- $\Phi(\Sigma)$ has a function symbol $fun : individual \times list \to individual$, predicate symbol $rel : individual \times list$ and constant symbols $n : individual$ for each $n \in Names$
- Γ consists of the sort generation constraint that asserts that $list$ is a free type over its constructors nil and $cons$

Sentences are translated inductively on their structure, with predications $t(s)$ translated to $rel(t, s)$ and terms $t(s)$ translated to $fun(\alpha_\Sigma(t), \alpha_\Sigma(s))$.

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

- $UR^M = N_{individual} \uplus Names$,
- $UD^M = N_{individual} \uplus DNames$,
- for any $n \in Names$, $int^M(n) = n$,
- for any $x \in UR^M$, $fun^M(x)(s) = N_{fun}(x, s)$ if s is a sequence of elements in $N_{individual}$, or $fun^M(x)(s) = choose(Names)$ otherwise
- for any $x \in UR^M$, $rel^M(x)(s_1, ..., s_k) = N_{rel}(x, s)$ if s is a sequence of elements in $N_{individual}$, or $rel^M(x)(s) = False$ otherwise.
- for any $m \in Markers$, $seq^M(m) = N_m$.

Note that for the last two comorphisms we make use of the translation of a sequence to CFOL defined on page 9.

B.6 Faithful comorphisms

The proofs that these four comorphisms are faithful are similar, therefore we only present it for the translation CL.Seq to CFOL.

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 $\Gamma \models^{CL}_{\Sigma} e$.

We define a mapping $\delta_\Sigma : \mathbf{Mod}^{CL}(\Sigma) \to \mathbf{Mod}^{CFOL}(\Phi(\Sigma))$ such that $\beta_\Sigma(\delta_\Sigma(M))$ and M have the same Γ-consequences e, where Γ is a Σ-theory, M is a Γ-model and e is a Σ-sentence that only uses terms on function/predicate positions with the same number of arguments as in Γ.

We denote $N = \delta_\Sigma(M)$ and $NDNames = Names \setminus DNames$. N is defined as follows:

- $N_{individual} = UD^M \uplus \{int^M(n) | n \in NDNames\}$;
- 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 $Names$, $N_x(s) = fun^M(int^M(x))(s)$ if s has only elements in UD^M, or $N_x(s) = choose(NDNames)$, otherwise
– for each x in $Names$, $N_x(s) = rel^M(int^M(x))(s)$ if s has only elements in UD^M, or $N_x(s) = False$, otherwise

It is easy to see that $\beta_\Sigma(\delta_\Sigma(M))$ and M have the same Γ-consequences.

Let M be a Σ-model such that $M \models^{CL}_\Sigma \Gamma$ and assume that e does not hold in M. Then e also does not hold in $\beta_\Sigma(\delta_\Sigma(M))$. By the satisfaction condition of the comorphism we get that $\alpha_\Sigma(e)$ does not hold in $\delta_\Sigma(M)$. But it is easy to see that $M \models \Gamma$ implies $\delta_\Sigma(M) \models \alpha_\Sigma(\Gamma)$ and thus we get a contradiction with the fact that $\alpha_\Sigma(\Gamma) \models^{CFOL}_{\Phi(\Sigma)} \alpha_\Sigma(e)$.

□

B.7 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 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). The satisfaction condition is straightforward.

B.8 FOL to CFOL

A 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.

B.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 respetive higher-order type. Translation of sentences and models is then straightforward, except for sort generation constraints.

For a sort generation constraint

$$(\dot{S}, \dot{F}, \theta\colon \bar{\Sigma} \to \Sigma)$$

we assume without loss of generality that all the result sorts of function symbols in $\dot{F}$ occur in $\dot{S}$. Let

$$\dot{S} = \{s_1;\ \ldots;\ s_n\}, \quad \dot{F} = \{f_1 \colon s_1^1 \ldots s_{m_1}^1 \to s^1;\ \ldots;\ f_1 \colon s_1^k \ldots s_{m_k}^k \to s^k\}$$

The sort generation constraint is now translated to the second-order sentence

$$\forall P_{s_1} : pred(\theta(s_1)) \ldots \forall P_{s_n} : pred(\theta(s_n)) \bullet (\varphi_1 \wedge \cdots \wedge \varphi_k) \Rightarrow \bigwedge_{j=1,\ldots,n} \forall x : \theta(s_j) \bullet P_{s_j}(x)$$

where

$$\varphi_j = \forall x_1 : \theta(s_1^j), \ldots, x_{m_j} : \theta(s_{m_j}^j) \bullet \left(\bigwedge_{i=1,\ldots,m_j;\ s_i^j \in \dot{S}} P_{s_i^j}(x_i) \right) \Rightarrow P_{s^j}\left(\theta(f_j)(x_1, \ldots, x_{m_j})\right)$$

For a proof of the satisfaction condition, see [27].

B.10 SROIQ to FOL

Translation of Signatures $\Phi((\mathbf{C}, \mathbf{R}, \mathbf{I})) = (F, P)$ with

- function symbols: $F = \{a^{(1)} | a \in \mathbf{I}\}$
- predicate symbols $P = \{A^{(1)} | A \in \mathbf{C}\} \cup \{R^{(2)} | R \in \mathbf{R}\}$

Translation of Sentences Concepts are translated as follows:

- $\alpha_x(A) = A(x)$
- $\alpha_x(\neg C) = \neg \alpha_x(C)$
- $\alpha_x(C \sqcap D) = \alpha_x(C) \wedge \alpha_x(D)$
- $\alpha_x(C \sqcup D) = \alpha_x(C) \vee \alpha_x(D)$
- $\alpha_x(\exists R.C) = \exists y.(R(x, y) \wedge \alpha_y(C))$
- $\alpha_x(\exists U.C) = \exists y.\alpha_y(C)$
- $\alpha_x(\forall R.C) = \forall y.(R(x, y) \to \alpha_y(C))$
- $\alpha_x(\forall U.C) = \forall y.\alpha_y(C)$
- $\alpha_x(\exists R.\mathsf{Self}) = R(x, x)$
- $\alpha_x(\leq nR.C) = \forall y_1, \ldots, y_{n+1} \cdot \bigwedge_{i=1,\ldots,n+1}(R(x, y_i) \wedge \alpha_{y_i}(C)) \to \bigvee_{1 \leq i < j \leq n+1} y_i = y_j$
- $\alpha_x(\geq nR.C) = \exists y_1, \ldots, y_n \cdot \bigwedge_{i=1,\ldots,n}(R(x, y_i) \wedge \alpha_{y_i}(C)) \wedge \bigwedge_{1 \leq i < j \leq n} y_i \neq y_j$
- $\alpha_x(\{a_1, \ldots a_n\}) = (x = a_1 \vee \ldots \vee x = a_n)$

For inverse roles R^-, $R^-(x, y)$ has to be replaced by $R(y, x)$, e.g.

$$\alpha_x(\exists R^-.C) = \exists y.(R(y, x) \wedge \alpha_y(C))$$

This rule also applies below.

Sentences are translated as follows:

- $\alpha_\Sigma(C \sqsubseteq D) = \forall x.\,(\alpha_x(C) \to \alpha_x(D))$

- $\alpha_\Sigma(a : C) = \alpha_x(C)[a/x]$[26]
- $\alpha_\Sigma(R(a,b)) = R(a,b)$
- $\alpha_\Sigma(R \sqsubseteq S) = \forall x, y.R(x,y) \rightarrow S(x,y)$
- $\alpha_\Sigma(R_1; \ldots; R_n \sqsubseteq R) =$
  $\forall x, y.(\exists z_1, \ldots z_{n-1}.R_1(x,z_1) \wedge R_2(z_1,z_2) \wedge \ldots \wedge R_n(z_{n-1},y)) \rightarrow R(x,y)$
- $\alpha_\Sigma(\mathsf{Dis}(R_1,R_2)) = \neg\exists x, y.R_1(x,y) \wedge R_2(x,y)$
- $\alpha_\Sigma(\mathsf{Ref}(R)) = \forall x.R(x,x)$
- $\alpha_\Sigma(\mathsf{Irr}(R)) = \forall x.\neg R(x,x)$
- $\alpha_\Sigma(\mathsf{Asy}(R)) = \forall x, y.R(x,y) \rightarrow \neg R(y,x)$
- $\alpha_\Sigma(\mathsf{Tra}(R)) = \forall x, y, z.R(x,y) \wedge R(y,z) \rightarrow R(x,z)$

Translation of Models

- For $M' \in \mathrm{Mod}^{FOL}(\phi\Sigma)$ define $\beta_\Sigma(M') := (\Delta, \cdot^I)$ with $\Delta = |M'|$ and $A^I = M'_A, a^I = M'_a, R^I = M'_R$.

Proposition 1. $C^{\mathcal{I}} = \left\{ m \in M'_{Thing} | M' + \{x \mapsto m\} \models \alpha_x(C) \right\}$

Proof. By Induction over the structure of C.

- $A^{\mathcal{I}} = M'_A = \left\{ m \in M'_{Thing} | M' + \{x \mapsto m\} \models A(x) \right\}$
- $(\neg C)^{\mathcal{I}} = \Delta \setminus C^{\mathcal{I}} =^{I.H.} \Delta \setminus \{m \in M'_{Thing} | M' + \{x \mapsto m\} \models \alpha_x(C)\} = \{m \in M'_{Thing} | M' + \{x \mapsto m\} \models \neg\alpha_x(C)\}$

  The satisfaction condition holds as well.

B.11 SROIQ to CL.Fol

This comorphism can be obtained as the composition SROIQ → FOL → CL.Fol.

B.12 SROIQ to CL.Full

Translation of Signatures A signature is translated by mapping all individuals, concepts and roles to discourse names, and augmenting this with the following CL.Imp theory:

```
(forall (c x) (iff ((OWLnot c) x) (not (c x))))
(forall (c x) (iff ((OWLand c d) x) (and (c x) (d x))))
(forall (c x) (iff ((OWLor c d) x) (or (c x) (d x))))
(forall (r c x) (iff ((OWLsome r c) x)
                     (exists (y) (and (r x y) (c y)))))
(forall (r c x) (iff ((OWLall r c) x)
                     (forall (y) (implies (r x y) (c y)))))
(forall (x y) (OWLU x y))
(forall (r x) (iff ((OWLself r) x) (r x x)))
(forall (r x y) (iff ((OWLinv r) x y) (r x y)))
(forall (c d) (iff ((OWLsubsumes c d)
```

[26] Replace x by a.

```
                  (forall (x) (if (c x) (d x))))))
(forall (r s) (iff ((OWLsubsumesRole r s)
                  (forall (x y) (if (r x y) (s x y))))))
(forall (r s) (iff ((OWLdisjoint r s)
                  (forall (x y) (not (and (r x y) (s x y)))))))
(forall (r) (iff ((OWLref r)
                 (forall (x y) (r x x)))))
(forall (r) (iff ((OWLirr r)
                 (forall (x y) (not (r x x))))))
(forall (r) (iff ((OWLasy r)
                 (forall (x y) (if (r x x) (not (r y x)))))))
(forall (r) (iff ((OWLtra r)
                 (forall (x y) (if (and (r x y) (r y z)) (r x z))))))
(forall (r c) (iff ((OWLmax r c) x)
                  (forall (y) (not (and (r x y) (c y))))))

(mutually-different)
(mutually-different x)
(iff (mutually-different x y ...)
     (and (not (= x y))
          (mutually-different x ...)
          (mutually-different y ...) ))

(forall (p) (holds-all p))
(forall (p x ...) (iff (holds-all p x ...)
                       ((and (p x) (holds-all p ...)))))
(forall (p) (not (holds-some p)))
(forall (p x ...) (iff (holds-some p x ...)
                       ((or (p x) (holds-some 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 ((OWLmax r c ...) x)
    (forall (...a) (if (and ((same-length c ...) ...a)
                            (holds-all (restrict r c x) ...a))
                       (not ((mutually-different ...a)))))))
(forall (r c ...) (iff ((OWLmin r c ...) x)
    (exists (...a) (and ((same-length ...) ...a)
                        (holds-all (restrict r c x) ...a)
                        ((mutually-different ...a))))))
```

```
(forall x (not ((OWLnominal) x)))
(forall ... x y (iff ((OWLnominal y ...) x)
                     (or (= x y) ((OWLnominal ...) x))))

(forall (r x y) (iff ((OWLcomp r) x y) (r x y)))
(forall (r ... x y)
   (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 Sentences Concepts are translated as follows:

- $\alpha(A) = A$
- $\alpha(\neg C) = (\texttt{OWLnot}\ \alpha(C))$
- $\alpha(C \sqcap D) = (\texttt{OWLand}\ \alpha(C)\ \alpha(D))$
- $\alpha(C \sqcup D) = (\texttt{OWLor}\ \alpha(C)\ \alpha(D))$
- $\alpha(\exists R.C) = (\texttt{OWLsome}\ \alpha(R)\ \alpha(C))$
- $\alpha(\forall R.C) = (\texttt{OWLall}\ \alpha(R)\ \alpha(C))$
- $\alpha(\exists R.\mathsf{Self}) = (\texttt{OWLself}\ R)$
- $\alpha(\leq nR.C) = (\texttt{OWLmax}\ \alpha(R)\ \alpha(C)\ (a \ldots a))$
- $\alpha(\geq nR.C) = (\texttt{OWLmin}\ \alpha(R)\ \alpha(C)\ (a \ldots a))$
- $\alpha(\{a_1, \ldots a_n\}) = (\texttt{OWLnominal}\ a_1 \ldots a_n)$

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.

Roles are translated by: $\alpha(R^-) = (\texttt{OWLinv}\ \alpha R)$.

Sentences are translated as follows:

- $\alpha_\Sigma(C \sqsubseteq D) = (\texttt{OWLsubsumes}\ \alpha(C)\ \alpha(D))$
- $\alpha_\Sigma(a : C) = (\alpha(C)\ a)$
- $\alpha_\Sigma(R(a, b)) = (\alpha(R)\ a\ b)$
- $\alpha_\Sigma(R \sqsubseteq S) = (\texttt{OWLsubsumesRole}\ \alpha(R)\ \alpha(S))$
- $\alpha_\Sigma(R_1;\ \ldots; R_n \sqsubseteq R) = (\texttt{OWLsubsumesRole}\ (\texttt{OWLcomp}\ \alpha(R_1)\ \ldots \alpha(R_n))\ \alpha(R))$
- $\alpha_\Sigma(\mathsf{Dis}(R_1, R_2)) = (\texttt{OWLdisjoint}\ \alpha(R)\ \alpha(S))$
- $\alpha_\Sigma(\mathsf{Ref}(R)) = (\texttt{OWLref}\ \alpha(R))$
- $\alpha_\Sigma(\mathsf{Irr}(R)) = (\texttt{OWLirr}\ \alpha(R))$
- $\alpha_\Sigma(\mathsf{Asy}(R)) = (\texttt{OWLasy}\ \alpha(R))$
- $\alpha_\Sigma(\mathsf{Tra}(R)) = (\texttt{OWLtra}\ \alpha(R))$

Translation of Models

– For $M' \in \mathrm{Mod}^{CL.Imp}(\Phi\Sigma)$ define $\beta_\Sigma(M') := (\Delta, \cdot^I)$ with Δ being the universe of discourse of M', and the interpretation of individuals, concepts and roles given by the interpretation of the respective names.

C Further Examples

Relative Interpretation (Standalone Example) This example defines a partial order twice: once as an extension of a strict partial order, and once directly. Then, we connect both definitions by a view that expresses the relative interpretation.

```
logic CommonLogic

ontology Strict_Partial_Order =
%% strict
. (forall (x)
          (not (lt x x)))
%% asymmetric
. (forall (x y)
          (if (lt x y)
              (not (lt y x))))
%% transitive
. (forall (x y z)
          (if (and (lt x y)
                   (lt y z))
              (lt x z)))
end

ontology Partial_Order_From_Strict_Partial_Order =
    Strict_Partial_Order
then
%% define "less or equal" in terms of "less than"'
. (forall (x y)
          (iff (le x y)
               (or (lt x y)
                   (= x y))))
end

ontology Partial_Order =
%% reflexive
. (forall (x)
          (le x x))
%% antisymmetric
. (forall (x y)
          (if (and (le x y)
                   (le y x))
              (= x y)))
%% transitive
. (forall (x y z)
          (if (and (le x y)
                   (le y z))
              (le x z))))
end

interpretation v : Partial_Order to Partial_Order_From_Strict_Partial_Order
```

Heterogeneous Views from OWL to Common Logic An interpretation from one ontology to another ontology in the same logic has been shown in Sect. 4, but it is also possible to have interpretations across logics, as long as there is a translation between these logics that is known to HETS (cf. Sect. 3.3).

The following example establishes an interpretation between the OWL Time ontology and its reimplementation in Common Logic, using the "OWL22CommonLogic" translation:

```
logic OWL
ontology TimeOWL =
  Class: TemporalEntity
  ObjectProperty: before
    Domain: TemporalEntity
    Range: TemporalEntity
    Characteristics: Transitive
end

logic CommonLogic
ontology TimeCL =
%% CommonLogic equivalent of Domain and Range above
. (forall (t1 t2)
          (if (before t1 t2)
              (and (TemporalEntity t1)
                   (TemporalEntity t2))))
%% CommonLogic equivalent of Transitive above
. (forall (t1 t2 t3)
          (if (and (before t1 t2)
                   (before t2 t3))
              (before t1 t3)))
%% A new axiom that cannot be expressed in OWL
. (forall (t1 t2)
          (or (before t1 t2)
              (before t2 t1)
              (= t1 t2)))
end

interpretation TimeOWLtoCL : { TimeOWL with logic OWL22CommonLogic } to TimeCL
%% As OWL22CommonLogic is the default translation,
%% it is optional to specify it.
```

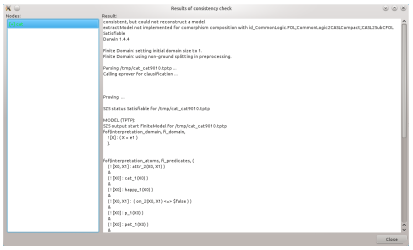

Figure 5. Consistency checker results

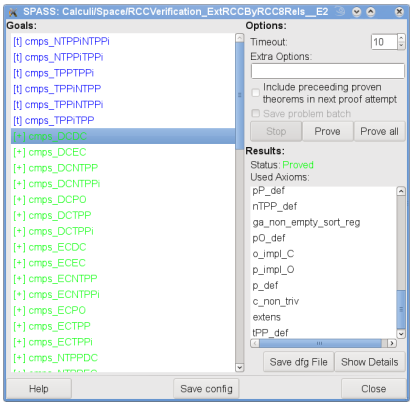

Figure 6. Interface of the SPASS prover

Automated Theorem Proving Systems Interface All ATPs integrated into HETS share the same GUI, with only a slight modification for the MathServe Broker: the input field

for extra options is inactive. Fig. 6 shows the instantiation for SPASS, where in the top right part of the window the batch mode can be controlled. The left side shows the list of goals (with status indicators). If goals are timed out (indicated by 't') it may help to activate the check box 'Include preceding proven theorems in next proof attempt' and pressing 'Prove all' again.

On the bottom right the result of the last proof attempt is displayed. The 'Status:' indicates 'Open', 'Proved', 'Disproved', 'Open (Time is up!)', or 'Proved (Theory inconsistent!)'. The list of 'Used Axioms:' is filled by SPASS. The button 'Show Details' shows the whole output of the ATP system. The 'Save' buttons allow you to save the input and configuration of each proof for documentation. By 'Close' the results for all goals are transferred back to the Proof Management GUI.

The MathServe system [42] developed by Jürgen Zimmer provides a unified interface to a range of different ATP systems. Their capabilities are derived from the *Specialist Problem Classes* (SPCs) defined upon the basis of logical, language and syntactical properties by Sutcliffe and Suttner [40]. Only two of the Web services provided by the MathServe system are used by HETS currently: Vampire and the brokering system. The ATP systems are offered as Web Services using standardised protocols and formats such as SOAP, HTTP and XML. Currently, the ATP system Vampire may be accessed from HETS via MathServe; the other systems are only reached after brokering.

For details on the ATPs supported, see the HETS user guide [29].

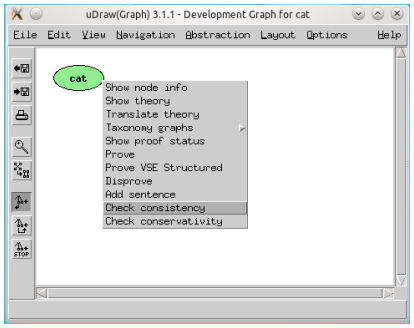

Figure 7. Selection of consistency checker

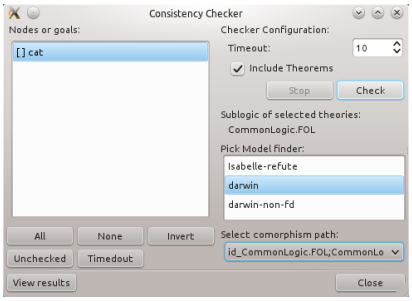

Figure 8. HETS Consistency Checker Interface

Consistency Checker Interface The consistency checker interface is shown in Fig. 8. This GUI is invoked from the 'Edit' menu as it operates on all nodes.

The list on the left shows all node names prefixed with a consistency status in square brackets that is initially empty. A consistent node is indicated by a '+', a '–' indicates an inconsistent node, a 't' denotes a timeout of the last checking attempt.

For some selection of development graph nodes having Common Logic theories, a model finder should be selectable from the 'Pick Model finder:' list. When pressing 'Check', possibly after 'Select comorphism path:', all selected nodes will be checked, spending at most the number of seconds given under 'Timeout:' on each node. Pressing

‘Stop’ allows to terminate this process if too many nodes have been chosen. Either by ‘View results’ or automatically the ‘Results of consistency check’ (Fig. 9) will pop up and allow you to inspect the models for nodes, if they could be constructed.

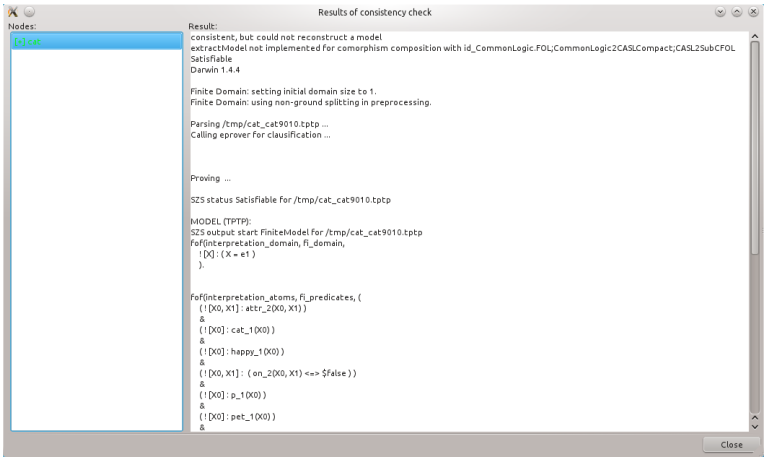

Figure 9. Consistency checker results