

Reasoning Support for CASL with Automated Theorem Proving Systems*

Klaus Lüttich¹ and Till Mossakowski²

¹ SFB/TR8, University of Bremen

² University of Bremen and DFKI Lab Bremen
{luettich,till}@informatik.uni-bremen.de

Abstract. We connect the algebraic specification language CASL with a variety of automated first-order provers. The heart of this connection is an institution comorphism from CASL to *SoftFOL* (softly typed first-order logic); the latter is then translated to the provers' input syntaxes. We also describe a GUI integrating the translations and the provers into the Heterogeneous Tool Set. We report on experiences with provers, which led to fine-tuning of the translations. This framework can also be used for checking consistency of specifications.

1 Introduction

The Common Algebraic Specification Language (CASL) [3, 5] is a modern standard for axiomatic specification using first-order logic and datatypes. During the development of CASL specifications, it is crucial to have good proof support – be it for proving intended consequences of specifications, or checking their consistency.

So far, only interactive provers like Isabelle [12] have been connected to CASL. This paper describes reasoning support for CASL with automated first-order logic theorem proving (ATP) systems made available through the Heterogeneous Tool Set (HETS) [11]. Although ATP systems do not provide reasoning support for all features of CASL, they can take us surprisingly far.

The ATP reasoning support for CASL described in this paper is based on softly typed first-order logic (*SoftFOL*) which is presented as an institution. Furthermore, a coding of a sublogic of CASL into *SoftFOL* is formalized as an institution comorphism. Two different syntactical representations (DFG and TPTP) of *SoftFOL* allow the connection of SPASS [15] and MathServe [16] with HETS. While SPASS is an ATP system by itself, MathServe offers reasoning support for different ATP systems as web services and a broker service, which chooses a suitable ATP system after classification of the problem. Furthermore, the integration of reasoning support into HETS is presented together with a description of the graphical user interface (GUI) used to control the ATP. Finally, we summarize optimisations of the coding implemented in HETS.

* This paper has been supported by Deutsche Forschungsgemeinschaft in the SFB/TR8 “Spatial Cognition” and in the project MULTIPLE under grant KR 1191/5-2.

2 Institutions and Their Comorphisms

The use of automated first-order provers for CASL requires translations to be set up between the CASL logic and the provers' logics. We formalize these translations as *institution comorphisms*. Institutions were introduced by Goguen and Burstall [7] to capture the notion of logical system and abstract away from the details of signatures, sentences, models and satisfaction.

Let \mathcal{CAT} be the category of categories and functors.³

Definition 1. An institution $I = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ consists of

- a category \mathbf{Sign} of signatures,
- a functor $\mathbf{Sen}: \mathbf{Sign} \rightarrow \mathbf{Set}$ giving, for each signature Σ , the set of sentences $\mathbf{Sen}(\Sigma)$, and for each signature morphism $\sigma: \Sigma \rightarrow \Sigma'$, the sentence translation map $\mathbf{Sen}(\sigma): \mathbf{Sen}(\Sigma) \rightarrow \mathbf{Sen}(\Sigma')$, where often $\mathbf{Sen}(\sigma)(\varphi)$ is written as $\sigma(\varphi)$,
- a functor $\mathbf{Mod}: \mathbf{Sign}^{op} \rightarrow \mathcal{CAT}$ giving, for each signature Σ , the category of models $\mathbf{Mod}(\Sigma)$, and for each signature morphism $\sigma: \Sigma \rightarrow \Sigma'$, the reduct functor $\mathbf{Mod}(\sigma): \mathbf{Mod}(\Sigma') \rightarrow \mathbf{Mod}(\Sigma)$, where often $\mathbf{Mod}(\sigma)(M')$ is written as $M'|_\sigma$,
- a satisfaction relation $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ for each $\Sigma \in |\mathbf{Sign}|$,

such that for each $\sigma: \Sigma \rightarrow \Sigma'$ in \mathbf{Sign} the following satisfaction condition holds:

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

for each $M' \in |\mathbf{Mod}(\Sigma')|$ and $\varphi \in \mathbf{Sen}(\Sigma)$, expressing that truth is invariant under change of notation and enlargement of context. \square

A *theory* in an institution is a pair $T = (\Sigma, \Gamma)$ consisting of a signature $\text{Sign}(T) = \Sigma$ and a set of Σ -sentences $\text{Ax}(T) = \Gamma$, the axioms of the theory. *Theory morphisms* are signature morphisms that map axioms to logical consequences.

Institution comorphisms [6] allow the expression of the fact that one institution I is *included* or *encoded* into an institution J . Given institutions I and J , an *institution comorphism* $\rho = (\Phi, \alpha, \beta): I \rightarrow J$ consists of

- a functor $\Phi: \mathbf{Sign}^I \rightarrow \mathbf{Sign}^J$,
- a natural transformation $\alpha: \mathbf{Sen}^I \rightarrow \mathbf{Sen}^J \circ \Phi$,
- a natural transformation $\beta: \mathbf{Mod}^J \circ \Phi^{op} \rightarrow \mathbf{Mod}^I$

such that the following *satisfaction condition* is satisfied for all $\Sigma \in \mathbf{Sign}^I$, $M' \in \mathbf{Mod}^J(\Phi(\Sigma))$ and $\varphi \in \mathbf{Sen}^I(\Sigma)$:

$$M' \models_{\Phi(\Sigma)}^J \alpha_\Sigma(\varphi) \Leftrightarrow \beta_\Sigma(M') \models_\Sigma^I \varphi.$$

In more detail, this means that each signature $\Sigma \in \mathbf{Sign}^I$ is translated to a signature $\Phi(\Sigma) \in \mathbf{Sign}^J$, and each signature morphism $\sigma: \Sigma \rightarrow \Sigma' \in \mathbf{Sign}^I$ is

³ Strictly speaking, \mathcal{CAT} is not a category but only a so-called quasicategory, which is a category that lives in a higher set-theoretic universe.

translated to a signature morphism $\Phi(\sigma): \Phi(\Sigma) \longrightarrow \Phi(\Sigma') \in \mathbf{Sign}^J$. Moreover, for each signature $\Sigma \in \mathbf{Sign}^I$, we have a sentence translation map $\alpha_\Sigma: \mathbf{Sen}^I(\Sigma) \longrightarrow \mathbf{Sen}^J(\Phi(\Sigma))$ and a model translation functor $\beta_\Sigma: \mathbf{Mod}^J(\Phi(\Sigma)) \longrightarrow \mathbf{Mod}^I(\Sigma)$.

A *simple theoroidal comorphism* is like a comorphism, except that the signature translation functor Φ ends in the category of theories over the target institution.

3 The CASL logic

CASL, the Common Algebraic Specification Language, has been designed by COFI, the international *Common Framework Initiative for algebraic specification and development* [1], with the goal to subsume many previous algebraic specification languages and to provide a standard language for the specification and development of modular software systems. See the CASL user manual [3] and reference manual [5] for further information.

Here, we concentrate on *CASL basic specifications*, designed for writing single specification modules. CASL also provides constructs for structured and architectural specifications and specification libraries.

The logic of CASL basic specifications combines first-order logic and induction (the latter is expressed using so-called sort generation constraints, and is needed for the specification of the usual inductive datatypes) with subsorts and partial functions. The institution underlying CASL is introduced in two steps [5]: first, we introduce many-sorted partial first-order logic with sort generation constraints and equality ($PCFOL^\equiv$), and then, subsorted partial first-order logic with sort generation constraints and equality ($SubPCFOL^\equiv$) is described in terms of $PCFOL^\equiv$.

3.1 Partial First-Order Logic

We now sketch the institution $PCFOL^\equiv$ of many-sorted partial first-order logic with sort generation constraints and equality. Full details can be found in [10, 5].

A *many-sorted CASL signature* $\Sigma = (S, TF, PF, P)$ consists of a set S of sorts, two $S^* \times S$ -indexed⁴ sets $TF = (TF_{w,s})$ and $PF = (PF_{w,s})$ of total and partial operation symbols, and an S^* -indexed set $P = (P_w)$ of predicate symbols. Function and predicate symbols are written $f: \bar{s} \rightarrow t$ and $p: \bar{s}$, respectively, where t is a sort and \bar{s} is a list $s_1 \dots s_n$ of sorts, thus determining their *name* and *profile*. Symbols with identical names are said to be *overloaded*; they may be referred to by just their names in CASL specifications, but are always qualified by profiles in fully statically analysed sentences. Signature morphisms map the sorts and the function and predicate symbols in a compatible way, such that the totality of function symbols is preserved.

⁴ S^* is the set of strings over S .

Models are many-sorted partial first order structures, interpreting sorts as carrier sets, total (partial) function symbols as total (partial) functions and predicate symbols as relations. Homomorphisms between such models are so-called *weak homomorphisms*. That is, they are total as functions, and they preserve (but not necessarily reflect) the definedness of partial functions and the satisfaction of predicates.

Concerning *reducts*, if $\sigma: \Sigma_1 \longrightarrow \Sigma_2$ is a signature morphism and M is a Σ_2 -model, then $M|_\sigma$ is the Σ_1 -model which interprets a symbol by first translating it along σ and then taking M 's interpretation of the translated symbol. Reducts of homomorphisms are defined similarly.

Given a many-sorted signature $\Sigma = (S, TF, PF, P)$ and a pairwise disjoint S -indexed set of variables X , the set $T_\Sigma(X)$ of *terms* over Σ and X is defined inductively as usual, using variables and operation symbols. Concerning the semantic interpretation of terms in a model, variable assignments are total, but the value of a term w.r.t. a variable assignment may be undefined, due to the application of a partial function during the evaluation of the term. Undefinedness propagates from subterms to superterms.

Sentences are built from atomic sentences using the usual features of first order logic. Given Σ and X , the set $AF_\Sigma(X)$ of *many-sorted atomic Σ -formulas* with variables in X contains:

1. $p_w(t_1, \dots, t_n)$, for $t_i \in T_\Sigma(X)_{s_i}, p \in P_w, w = s_1 \dots s_n \in S^*$,
2. $t \stackrel{e}{=} t'$, for $t, t' \in T_\Sigma(X)_s, s \in S$ (existential equations),
3. $t = t'$, for $t, t' \in T_\Sigma(X)_s, s \in S$ (strong equations),
4. *def* t , for $t \in T_\Sigma(X)_s, s \in S$ (definedness assertions).

A definedness assertion holds w.r.t. a given valuation in a model if the term is defined under that valuation. A strong equation holds if its two sides are both defined or both undefined under the valuation, and in case of definedness, they are interpreted equally. An existence equation holds if both sides are defined and interpreted equally. A predicate application holds if all the terms are defined under the given valuation, and the resulting tuple of model elements is in the corresponding predicate. In this way, we retain the simplicity of a two-valued logic.

The satisfaction of compound formulas, built from atomic formulas using logical connectives and quantifiers, is defined as usual in first-order logic.

There is an additional type of sentence that goes beyond first-order logic: a *sort generation constraint* states that a given set of sorts is generated by a given set of functions, i.e. that all the values of the generated sorts are reachable by some term in the function symbols, possibly containing variables of other sorts.

Formally, a sort generation constraint over a signature Σ is a triple (S', F', θ) , where $\theta: \bar{\Sigma} \longrightarrow \Sigma$, $\bar{\Sigma} = (\bar{S}, \bar{TF}, \bar{PF}, \bar{P})$, $S' \subseteq \bar{S}$ and $F' \subseteq \bar{TF} \cup \bar{PF}$.

A Σ -constraint (S', F', θ) is satisfied in a Σ -model M if the carriers of $M|_\theta$ of the sorts in S' are generated by the function symbols in F' , i.e. for every sort $s \in S'$ and every value $a \in (M|_\theta)_s$, there is a $\bar{\Sigma}$ -term t containing only function symbols from F' and variables of sorts not in S' such that $\nu^\#(t) = a$ for some assignment ν into $M|_\theta$.

Translation of a sentence along a signature morphism just replaces all the symbols in the sentence according to the signature morphism; this (together with the reducts) fulfills the satisfaction condition [10]. Sort generation constraints cannot be translated in this way; instead, the extra signature morphism component is used: The translation of a constraint (S', F', θ) along σ is $(S', F', \sigma \circ \theta)$. This obviously leads to fulfillment of the satisfaction condition.

This completes the definition of the institution $PCFOL^=$.

CASL additionally has unique existential quantification and a conditional term construct. These are coded out; see Sect. 7 for details.

3.2 Subsorted Partial First-Order Logic

Subsorted partial first-order logic is defined in terms of partial first-order logic. The basic idea is to reduce subsorting to injections between sorts. While in the subsorted institution, these injections have to occur explicitly in the sentences, in the CASL language, they may be left implicit. Apart from the injections, one also has partial projection functions (one-sided inverses of the injections) and membership predicates.

The institution $SubPCFOL^=$ is defined as follows, extending the notion of order-sorted signatures as given by Goguen and Meseguer [8].

A *subsorted signature* $\Sigma = (S, TF, PF, P, \leq_S)$ consists of a many-sorted signature (S, TF, PF, P) together with a reflexive transitive *subsort relation* \leq_S on the set S of sorts.

For a subsorted signature, $\Sigma = (S, TF, PF, P, \leq_S)$, we define *overloading relations* (also called *monotonicity orderings*), \sim_F and \sim_P , for function and predicate symbols, respectively:

Let $f : w_1 \longrightarrow s_1, f : w_2 \longrightarrow s_2 \in TF \cup PF$, then

$$f : w_1 \longrightarrow s_1 \sim_F f : w_2 \longrightarrow s_2$$

iff there exist $w \in S^*$ with $w \leq w_1$ and $w \leq w_2$ and $s \in S$ with $s_1 \leq s$ and $s_2 \leq s$. Let $p : w_1, p : w_2 \in P$, then $p : w_1 \sim_P p : w_2$ iff there exists $w \in S^*$ with $w \leq w_1$ and $w \leq w_2$.

A *signature morphism* $\sigma : \Sigma \rightarrow \Sigma'$ is a many-sorted signature morphism that preserves the subsort relation and the overloading relations.

With each subsorted signature $\Sigma = (S, TF, PF, P, \leq_S)$ we associate a many-sorted signature $\hat{\Sigma}$, which is the extension of the underlying many-sorted signature (S, TF, PF, P) with new symbols,

- a total *injection* function symbol $\mathbf{inj} : s \rightarrow s'$, for each pair of sorts $s \leq_S s'$,
- a partial *projection* function symbol $\mathbf{pr} : s' \rightarrow? s$, for each pair of sorts $s \leq_S s'$, and
- a unary *membership* predicate symbol $\in^s : s'$, for each pair of sorts $s \leq_S s'$.

Subsorted Σ -models are ordinary many-sorted $\hat{\Sigma}$ -models satisfying the following properties (which can be formalized as a set of conditional axioms):

- Embedding operations are total and injective; projection operations are partial, and injective when defined.
- The embedding of a sort into itself is the identity function.
- All compositions of embedding operations between the same two sorts are equal functions.
- Embedding followed by projection is the identity function; projection followed by embedding is included in the identity function.
- Membership in a subsort holds just when the projection to the subsort is defined.
- Embedding is compatible with those operations and predicates that are in the overloading relations.

Signature morphisms, homomorphisms, reducts, sentences, sentence translation and satisfaction are simply inherited via the translation $\Sigma \mapsto \hat{\Sigma}$ from $PCFOL^=$.

This completes the definition of the institution $SubPCFOL^=$.

Every CASL basic specification SP generates, along with its $SubPCFOL^=$ -signature Σ , a set Γ of Σ -sentences; together, these determine the *theory* (Σ, Γ) generated by SP . Note that Γ contains not only explicitly stated sentences, but also sentences that are generated e.g. by CASL’s powerful datatype constructs (see below), such as the statement that selectors are one-sided inverses of their constructor. See [5] for further details.

3.3 Derived CASL sublogics

Let $SubCFOL^=$ be the restriction of $SubPCFOL^=$ to signatures without partial functions symbols, and $SulPCFOL^=$ be the restriction of $SubPCFOL^=$ to signatures with a locally filtered subsort relation. Furthermore, $SulCFOL^=$ is the similar restriction of $SubCFOL^=$. Recall that a pre-order is locally filtered, if each connected pair has an upper bound.

4 SPASS, MathServe and SoftFOL

This section introduces the ATP system SPASS and the MathServe-system. The latter provides web services for different FOL ATP systems, as well as a broker for FOL ATP web services. The following part of this section introduces the logic *SoftFOL* which provides softly typed FOL, while the last two parts introduce the input languages of SPASS and MathServe.

4.1 SPASS

SPASS [15] is a saturation based automated theorem prover and supports full sorted first-order logic with equality. It has been developed as an open source tool (GPL) since 1991 at the Max Planck Institut Informatik in Saarbrücken, Germany by Christoph Weidenbach, Thomas Hillenbrand, Dalibor Topić et al.

The reasoning methods utilized by SPASS include the following:

- superposition calculus,
- specific inference/reduction rules for sorts,
- splitting rules for explicit case analysis,
- sophisticated clause normal form (CNF) translation.

4.2 MathServe

The MathServe system has been developed by Jürgen Zimmer as open source software (GPL) and provides web services for ATP [16]. MathServe uses state-of-the-art Semantic Web technologies for describing the input, output and effects of the provided reasoning web services. It uses standardised protocols and formats to communicate with client software systems such as HTTP, SOAP and XML.

Table 1. ATP systems provided as web services by MathServe

ATP System	Version	Suitable Problem Classes ^a
DCTP	10.21p	effectively propositional
EP	0.91	effectively propositional; real first-order, no equality; real first-order, equality;
Otter	3.3	real first-order, no equality;
SPASS	2.2	effectively propositional; real first-order, no equality; real first-order, equality
Vampire	8.0	effectively propositional; pure equality, equality clauses contain non-unit equality clauses; real first-order, no equality, non-Horn;
Waldmeister	704	pure equality, equality clauses are unit equality clauses

^a The list of problem classes for each ATP system is not exhaustive, but only the most appropriate problem classes are named according to benchmark tests made with MathServe by Jürgen Zimmer.

The reasoners provided by MathServe (version 0.81) as web services are summarized in Table 1 (see [14] for further details – all listed ATP systems participated in the competition CASC-20). Additionally, a broker service is provided by MathServe which classifies a given reasoning problem given in FOL with equality and calls the most appropriate reasoning service.

4.3 SoftFOL

We now capture the logic underlying the SPASS theorem prover. The institution *SoftFOL*⁵ (softly typed first order logic) is a softly (i.e. semantically) typed variant of unsorted (i.e. single-sorted) first-order logic. In the *signatures*, it provides

⁵ *SoftFOL* is similar to membership equational logic [9], but provides only one (implicit) kind and has operation symbols as part of the signatures, instead of coding them as axioms.

sorts, predicates and total functions. Each predicate and function symbol may optionally have a type profile in terms of the sorts. Overloading of predicates and functions is only allowed if the kind of symbol (predicate or function) and the arities are the same. Subsorting is available as in CASL except that only a locally filtered subsort relation is allowed. *Signature morphisms* are similar to those of CASL; they have to preserve the typing, if present.

SoftFOL models have only one carrier set. Sorts are interpreted as subsets of the carrier. Subsorts must lead to subset inclusions. Each operation or predicate symbol is interpreted as one operation or predicate over the whole carrier. Each typing of an operation leads to the restriction that the operation takes arguments from the subsets as determined by the typing to results as determined by typing.

Sentences in *SoftFOL* are closed untyped FOL sentences, and hence applications of predicates and functions are not qualified with types and a type-correct usage of predicates and functions is not statically checked. (Only an invoked ATP may find incorrect applications to be inconsistent.) Variables may be typed (as operations may be), but again the typing information is not used for sentence formation. Sentences may also involve sort membership tests (as in CASL). *Satisfaction* is mostly as in untyped first-order logic – only the typing of variables leads to a restriction of their possible valuations. Sort generation constraints are available as sentences as in CASL, and their satisfaction is also inherited from CASL (note that this, unlike the case in the rest of *SoftFOL*, involves correctly strongly typed terms only!).

4.4 DFG Syntax

The input language of SPASS is called DFG and is a notation for *SoftFOL*. The DFG format distinguishes three relevant sections of a problem: (1) a list of symbols; (2) a list of declarations; and (3) two lists of formulas for axioms and conjectures. In the list of symbols the arities for functions and predicates are declared and the symbols for sorts are fixed. The list of declarations is a special form of axioms dealing with information about subsorting and (free) generatedness of sorts and the types of predicates and functions. Internally subsort and function type declarations are treated as axioms by SPASS and the other declarations are used for the Knuth-Bendix ordering of symbols. The lists of formulas allow for the typing of variables with sort predicates at the quantification level, but internally the typing of variables is treated as the antecedent of an implication where the quantified formula is the consequent. A symbol declared as sort, predicate or function cannot be used as a variable symbol. The DFG language has a built-in special predicate for equality.

4.5 TPTP Syntax

The TPTP (Thousands of Problems for Theorem Provers) language was invented as a uniform exchange language for ATP systems and is used for the TPTP library of logical problems [13]. This library forms the basis for the annual CADE ATP System Competition (CASC) [14] at the Conference on Automated

Deduction (CADE). It is also used as the input language for the MathServe web-services (Sect. 4.2). The TPTP format provides only untyped FOL with an equality predicate. So, there are no constructs for the declaration or typing of symbols. A TPTP problem consists simply of a list of labeled axioms and conjectures.

5 Coding of logics

The process of coding CASL into the input languages of SPASS and MathServe is performed in three steps, where the first step may be omitted if the CASL theory has no partial functions:

1. Coding of $SulPCFOL^=$ into $SulCFOL^=$, using a comorphism,
2. Coding of $SulCFOL^=$ into $SoftFOL$, using a comorphism,
3. Coding of $SoftFOL$ into DFG or TPTP, using a syntax translation.

The first translation has been described as translation (5a') in [10], modulo the – here inessential – *Sub* versus *Sul*. Note that this translation totalizes not only the user-declared partial function, but also uses of partial projection symbols (for the latter, total projection symbols are introduced). The second translation will be detailed below. The translations in the third item are not described as comorphisms, because the logic remains essentially the same and only the syntax changes, while usually a theory (with proof goals) is translated as a whole.

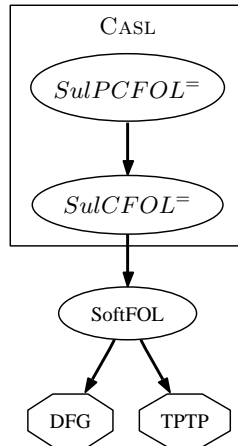


Fig. 1. Relation of the logics and translations

Figure 1 shows the codings as arrows and the logics and syntax formats used for automated theorem proving. The rest of this section covers the second and third step of the list shown above.

5.1 Coding CASL into *SoftFOL*

We now describe a simple theoroidal institution comorphism from $SulCFOL^=$ to *SoftFOL*.

Signatures. CASL signatures are mapped to *SoftFOL* signatures where each CASL symbol is qualified with its kind (sort, pred, op) and its arity, e.g. sort s becomes `sort_s` and operation $f : s * s \rightarrow r$ becomes `op_f_2`. The subsorting relation is kept, but for each subsorting relation $s < t$, an operation `op_inj: s → t` is added (for potential use in sort generation constraints), axiomatized to be the identity

$$\forall x:s. \text{op_inj}(x) = x$$

For each sort in the CASL signature an axiom is generated stating that the sort is not empty.

Models. *SoftFOL* models have only one carrier set. In order to obtain a CASL model, subsets of this carrier set (according to the interpretation of sorts in *SoftFOL*) are taken as carrier sets of a CASL model. Since CASL operations are translated to typed operations in *SoftFOL*, their restrictions are well-defined total operations on the subsets corresponding to the carriers of the CASL model. Subsort injections are interpreted as identities and membership as subset membership. This automatically ensures that the subsorting axioms (see Sect. 3.2) hold.

Sentences. Sentences are translated by erasing all subsorting injection operations; only in sort generation constraints are injections kept (therefore, we needed to introduce special injections into the signatures above).

Satisfaction. The satisfaction condition is clear from the fact that erasing all subsorting injection operations in the sentences corresponds to interpreting them as identities. Note that while *SoftFOL*-sentences may be ill-typed, all sentences in the image of the translation (and therefore all sentences relevant for the satisfaction condition) are well-typed.

Proposition 2. *For finite signatures, the model translation components of the comorphism from $SulCFOL^=$ to *SoftFOL* are surjective.*

Proof. Since we have restricted ourselves to $SulCFOL^=$, in a finite signature, each connected component of the subsort graph has a top element. A CASL model can be turned into a *SoftFOL* model as follows: take the disjoint union of all carriers of top sorts to be the carrier of the *SoftFOL* model. Predicates of same name and arity of the CASL model are united into a single predicate in the *SoftFOL* model. Similarly with operations, where operations need to be extended to the whole carrier of the *SoftFOL* model in an arbitrary way. \square

Corollary 3. *We can use the borrowing technique along the comorphism from CASL to *SoftFOL*. That is, CASL proof goals can be translated to *SoftFOL* proof goals in a sound and complete way.*

Proof. We assume that proof goals live over finite signatures and hence can apply Prop. 2. It is well known (see e.g. [4]) that surjectivity of the model translation leads to the property

$$\Gamma \models_{\Sigma} \varphi \text{ iff } Ax(\Phi(\Sigma)) \cup \alpha_{\Sigma}[\Gamma] \models_{Sign(\Phi(\Sigma))} \alpha_{\Sigma}(\varphi).$$

□

5.2 Generating DFG format from *SoftFOL*

A *SoftFOL* theory is transformed into a list of symbols, a list of declarations, and a list of axioms. The symbol list distinguishes sorts, predicates and functions and each predicate and function symbol is paired with its arity. Each subsort relation is transformed into a subsort declaration. The profiles of predicates and functions are also transformed into corresponding declarations. The sort generation axioms are turned into declarations of generated sorts (and to free generatedness, if the corresponding axioms are present). Note that declarations are treated by SPASS as special sentences.

5.3 Generating TPTP format from *SoftFOL*

Since the TPTP language has no notion of a signature, only type information given in the signature is taken into account. A *SoftFOL* theory is transformed into a list of sentences named *declaration_X* where *X* is a natural number. Each subsort relation and each function type is turned into a corresponding implication treating sorts as unary predicates. Typing information of predicates is not used. Concerning sentences, each variable list in a quantification that has type information is turned into an antecedent of an implication with the original quantified formula as consequent. The sorts are used like unary predicates. All other sentence constructs in *SoftFOL* have corresponding constructs in TPTP.

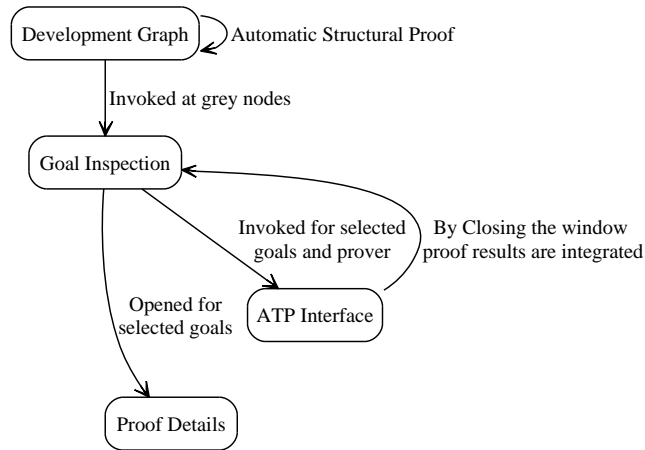


Fig. 2. Proof work-flow used for ATP proofs in HETS

6 Integration of ATP reasoning into HETS

The logics and comorphisms described in the previous sections are integrated into the Heterogeneous Tool Set HETS, which already offers static analysis of CASL specifications and libraries and an automatic proof system at the structuring level of CASL. The Graphical User Interface (GUI) initially presents an analyzed and structurally proved development graph and offers to open the Goal Inspection GUI for the selection of goals and theorem provers at nodes with proof obligations, which are colored grey (see Fig. 2 for the proof work-flow and see Fig. 3 for a development graph).

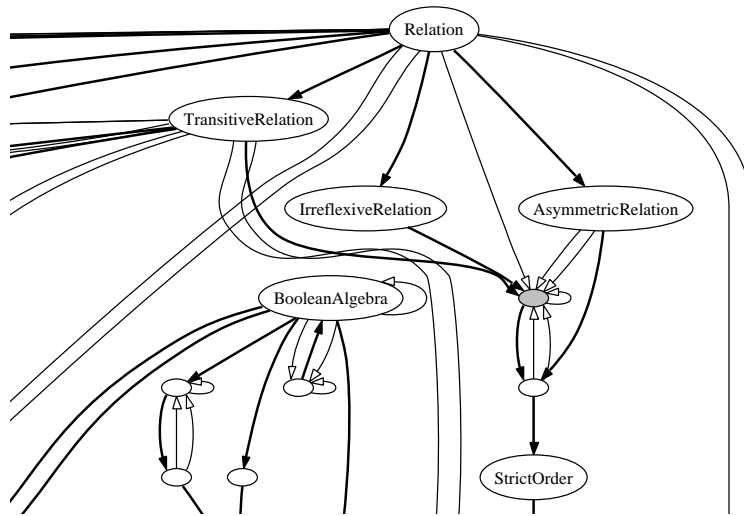


Fig. 3. Extract of the RELATIONSANDORDERS development graph

Figure 4 shows the interface for inspecting goals and discharging goals to different provers. The list on the left shows all goal names prefixed with the proof status in square brackets (see Tab. 2).

The two lists at the bottom of the window allow the detailed selection of axioms and proved theorems of this theory (see Sect. 7). By pressing the ‘Prove’ button the ATP interface of the selected prover is opened with the selected goals. The shortest (composed) comorphism is used for the translation into a prover supported logic. The button ‘More fine grained selection...’ allows the user to pick a (composed) comorphism in a separate window from where the prover interface is launched.

Table 2. Possible proof statuses

[+]	proved goal
[-]	disproved goal
[×]	proved goal revealing an inconsistent theory
[]	open goal

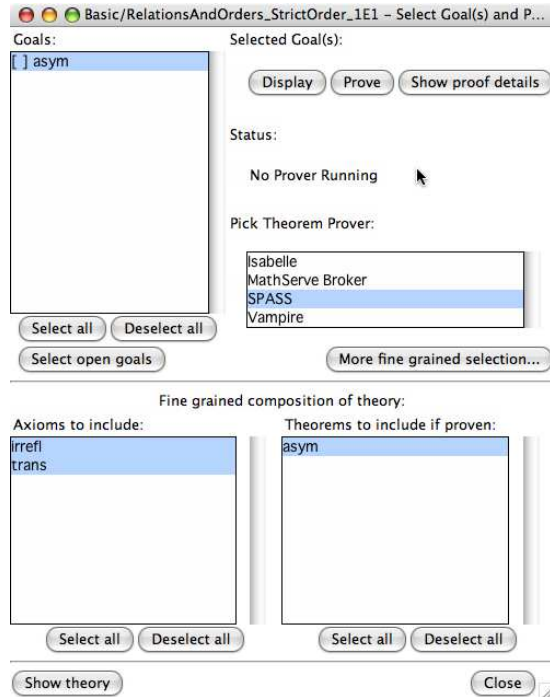


Fig. 4. HETS Goal Inspection Interface

Currently the ATP systems SPASS and Vampire are connected to HETS and the MathServe broker, which classifies the theory and chooses a suitable ATP system for the proof attempt. The ATP interface for *SoftFOL* based provers is implemented generically, such that each interface has the same layout. Figure 5 shows a screen shot of the ATP interface instantiated for SPASS. The Vampire interface looks the same except for the window title and the broker interface has no extra options fields.

The ATP interface offers functions to call and inspect the selected goal in the upper part and the indicators in the goal list are the same as in the goal inspection interface (see Tab. 2). The batch mode tries to prove each open goal without interaction. The details shown for each goal in the ATP interface are specific to the different provers, while the proof tree integrated into the development graph is always in the TSTP format which is the unified TPTP solution format.

After closing the ATP interface the goal inspection interface (see Fig. 4) gets the proof results and offers a unified view at the structural level. Figure 6 shows the proof details for some goal proved with SPASS where the tactic script and the proof tree are hidden initially and the underlined labels are used to toggle the information hiding. The (composed) comorphism used for the translation into *SoftFOL* is shown as well. This example is taken from the specification STRICTORDER in the library BASIC/RELATIONSANDORDERS.

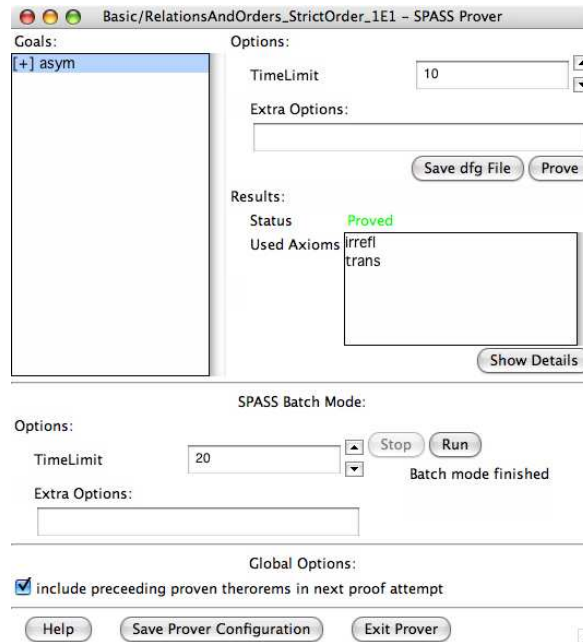


Fig. 5. ATP Interface of the SPASS prover

6.1 Connection of the ATPs

While SPASS is supposed to be installed and run locally with HETS on the same computer, MathServe is running on a dedicated central server which offers its reasoning services through HTTP and SOAP. The communication with SPASS is done via a standard Unix pipe and for MathServe all encoding, communication, and parsing of XML, SOAP, and HTTP is done through the HETS binary.

6.2 Consistency Checking with SPASS

Because SPASS can also disprove theorems, a consistency checker based on SPASS seemed appropriate. Because consistency checking of large libraries is time-

```

asym
Com: SuleCFOL2SoftFOL : CASL -> SoftFOL
Status: Proved
Used axioms: "irrefl", "trans"
Prover: SPASS
Tactic script
Time limit: 20
Extra options: ["-DocProof"]

Proof tree

```

Fig. 6. Proof details for goal asym

consuming and hence more suitable for batch processing, we did not integrate it into the GUI of HETS. Instead, consistency checking is invoked outside of HETS after generating DFG problem files for each CASL specification in a library with logical atom *false* as conjecture. Each of these files is run through SPASS with time limit t (initially, t is 500 seconds). If SPASS finds a proof for false from the current theory it is inconsistent. If false is disproved SPASS has found a completion for the current theory and it is logged as consistent. But, often the time limit is exceeded, because the search space for the given theory is too large. In this case, the theory is t -consistent.

We have taken CASL specifications from the repository available under www.cofi.info/Libraries and tested their 500-second-consistency. Actually, this already revealed a number of inconsistencies. Passing to 1500-second-consistency increased the number of inconsistencies, while at 5000 seconds, a kind of saturation could be observed. Inconsistencies have been caused by

- missing definedness conditions when using partial functions within axioms, that is, formulas of the shape

$$\forall x : s . f(x) = t,$$

which force f to be total. The correct form is

$$\forall x : s . \text{def } x \Rightarrow f(x) = t$$

- missing side-conditions ensuring the well-definedness of operations on non-freely generated datatypes (such as sets), that is, formulas like

$$\text{leftSummand}(x \cup y) = x$$

- erroneously declaring a partial function as total,
- oversight of CASL's non-empty carrier assumption when defining subsorts, that is, declarations of form

$$\text{sort } s = \{x : s' \bullet \varphi(x)\}$$

with the possibility that there is no x with $\varphi(x)$ in some circumstances.

It should be noted that consistency checking with SPASS only concerns the first-order fragment of CASL specifications. That is, even when SPASS proves a theory to be consistent, using saturation, there can still be inconsistencies due to the presence of sort generation constraints.

6.3 Induction

SPASS uses sort generation constraints only for obtaining efficient rewriting strategies, and the other provers do not use these at all. That is, all provers that we have considered in this work are pure first-order provers. Still, we have realised a strategy how to perform induction proofs using first order provers:

Along with the first-order goals sent to the prover, HETS takes the induction principles corresponding to sort generation constraints and instantiates them for the given proof goals. This already has been used for proving a number of theorems about inductive datatypes. Of course, with this method, one cannot expect the prover to find intermediate lemmas and prove them with induction – rather, the user has to provide the lemmas as proof goals along with the specification. Still, the method has turned out to be quite efficient: the fact that $reverse(reverse(L)) = L$ for any list L has been proved with just one lemma, which is much less than in the standard proof in Isabelle.

For induction proofs, it is crucial to carefully select the axioms that are fed into the first-order prover, otherwise it is very easy to run into a time-out (see also the next sections).

7 Used Optimisations

This section discusses some optimisations that we have implemented for the comorphism $SulCFOL =$ to $SoftFOL$ and the encodings into the DFG and TPTP syntaxes.

Most of the optimisations are needed to shorten the search space of the proof and have been discussed with the developers of SPASS: Christoph Weidenbach, Thomas Hillenbrand, and Dalibor Topić. The central feature shortening the search space is the selection of axioms and proved goals that are included in the theory sent to the ATP system (see Fig. 4).

The more specialized optimizations are the following ones. Single sorted theories are translated into untyped FOL theories, because the sort provides no additional information in a single sorted theory; if the single sort is generated, it is still kept in the theory. If a sort is generated by constant constructors only, a sentence stating the exhaustive generation is introduced. Binary predicates equivalent to the built-in equality predicate are removed from the signature; the definitions of such binary predicates are removed from the sentences and the application of such binary predicates is substituted with the built-in equality in the other sentences. Unique existential quantification is coded out into a conjunction of two formulas: (1) there exists an element that satisfies the quantified formula and (2) all elements fulfilling the formula are equal to the element of the first conjunct. The conditional term construct is coded out in a standard way as a conjunction of two implications [5].

Before the DFG and TPTP format generation those signature elements, which are not used in the axioms and conjecture sent to the ATP system, are removed, except for sorts. This removes the declaration sentences of unused symbols from the theories. Sentences related to the definedness encoding of partial functions are not considered for finding used symbols. All sort injection functions, which are not used as constructors for generated sorts, are removed from the $SoftFOL$ signature.

8 Conclusion and Future Work

We have connected SPASS and several other automated theorem provers to CASL. This has been done via an institution comorphism to an intermediate logic *SoftFOL* (softly typed first logic). Both the comorphism and the prover interfaces are integrated into the Heterogeneous Tool Set. The resulting tool has been used to verify a number of properties of CASL specifications, in particular for the complete verification of the composition table of the qualitative spatial calculi RCC5 and RCC8 based on Bennett’s first-order axiomatization of connectedness [2]. The latter verification goal involved 95 theorems and turned out to be much too tedious to be carried out with the interactive prover Isabelle [12], which previously was the only prover for CASL available in a stage beyond initial prototypes. With SPASS, now a higher degree of automation is available.

The possibility of using the *SoftFOL* coding for checking consistency of CASL specifications turned out to be extremely useful – several typical specification errors could be found in a number of specifications. It is therefore advisable to check specifications in this way before proceeding e.g. to further proofs or refinements.

Actually, the SPASS developers often have the opinion that certain specification styles are just wrong, because they lead to theories that are difficult to handle for SPASS. We have answered this by implementing several optimizations according to feedback from the SPASS developers about unsuccessful proof attempts. We think that it is more the responsibility of tools to transform specifications that are “bad” for provers to “good” ones, and not the responsibility of the specifier – especially since the latter has to consider other goals such as clarity and validity of specifications.

To obtain further optimization, we have considered analyzing the SPASS output of unsuccessful proof attempts in order to obtain hints which axioms to exclude from the next proof attempt because they lead the prover into infinite loops. However, we think that such an analysis should be done by SPASS itself, since it can lead to a more fined grained penalty system that only gradually removes axioms from the list of used input formulas. The SPASS developers are somewhat reluctant to follow this path, as they claim that it is difficult to design general strategies detecting loops, and they point to the fact that the general problem is undecidable. We nevertheless think that it is worth trying to obtain at least some partial information along these lines and to use it for loop avoidance.

Currently, proofs are only inspected to obtain the list of used axioms, information which is essential for efficient change management. A future extension of the tool described here will translate the proof trees returned by the various provers into a format that is more easily readable by the CASL specifier.

Acknowledgements

We thank Christoph Weidenbach, Thomas Hillenbrand, and Dalibor Topić for SPASS and fruitful discussions, Christian Maeder for weekly consistency checks, Jürgen Zimmer for MathServe and fruitful discussions, Stefan Wölfl for the

RCC8 case study, Hennes Maertins and Lutz Schröder for the *reverse* example, Rene Wagner and Rainer Grabbe for help with the implementation, and Erwin R. Catesbeina for pointing out quite a number of inconsistencies.

References

1. Common framework initiative for algebraic specification and development. www.cofi.info.
2. B. Bennett. *Logical Representations for Automated Reasoning about Spatial Relationships*. PhD thesis, 1997.
3. M. Bidoit and P. D. Mosses. *CASL User Manual*, volume 2900 of *LNCS*. Springer Verlag; Berlin, 2004. With chapters by Till Mossakowski, Donald Sannella, and Andrzej Tarlecki.
4. M. Cerioli and J. Meseguer. May I borrow your logic? (transporting logical structures along maps). *Theoretical Comput. Sci.*, 173:311–347, 1997.
5. CoFI (The Common Framework Initiative). *CASL Reference Manual*, volume 2960 of *LNCS*. Springer Verlag; Berlin, 2004.
6. J. Goguen and G. Rosu. Institution morphisms. *Formal aspects of computing*, 13:274–307, 2002.
7. 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.
8. J. A. Goguen and J. Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Comput. Sci.*, 105:217–273, 1992.
9. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi Presicce, editor, *Recent trends in algebraic development techniques. Proc. 12th International Workshop*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.
10. T. Mossakowski. Relating CASL with other specification languages: The institution level. *Theoretical Comput. Sci.*, 286:367–475, 2002.
11. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. Available at www.tzi.de/cofi/hets, University of Bremen.
12. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer Verlag; Berlin, 2002.
13. G. Sutcliffe and C. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
14. G. Sutcliffe and C. Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.
15. C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobalt, and D. Topic. SPASS version 2.0. In A. Voronkov, editor, *Automated Deduction — CADE-18*, volume 2392 of *LNCS*, pages 275–279. Springer Verlag; Berlin, 2002.
16. J. Zimmer and S. Autexier. The MathServe System for Semantic Web Reasoning Services. In U. Furbach and N. Shankar, editors, *Proceedings of the third International Joint Conference on Automated Reasoning*, volume 4130 of *LNCS*, pages 140–144. Springer Verlag; Berlin, 2006.