Integrating Maude into Hets

Mihai Codescu,[1] Till Mossakowski,[1] Adrián Riesco,[2] and Christian Maeder[1]

[1] DFKI GmbH Bremen and University of Bremen, Germany
[2] Facultad de Informática, Universidad Complutense de Madrid, Spain

Abstract. Maude modules can be understood as models that can be formally analyzed and verified with respect to different properties expressing various formal requirements. However, Maude lacks the formal tools to perform some of these analyses and thus they can only be done by hand. The Heterogeneous Tool Set Hets is an institution-based combination of different logics and corresponding rewriting, model checking and proof tools. We present in this paper an integration of Maude into Hets that allows to use the logics and tools already integrated in Hets with Maude specifications. To achieve such integration we have defined an institution for Maude based on preordered algebras and a comorphism between Maude and Casl, the central logic in Hets.

Keywords: Heterogeneous specifications, rewriting logic, institution, Maude, Casl

1 Introduction

Maude [3] is a high-level language and high-performance system supporting both equational and rewriting logic computation for a wide range of applications. Maude modules correspond to specifications in rewriting logic, a simple and expressive logic which allows the representation of many models of concurrent and distributed systems.

The key point is that there are three different uses of Maude modules:

1. As programs, to implement some application. We may have chosen Maude because its features make the programming task easier and simpler than other languages.
2. As formal executable specifications, that provide a rigorous mathematical model of an algorithm, a system, a language, or a formalism. Because of the agreement between operational and mathematical semantics, this mathematical model is at the same time executable.
3. As models that can be formally analyzed and verified with respect to different properties expressing various formal requirements. For example, we may want to prove that our Maude module terminates; or that a given function, equationally defined in the module, satisfies some properties expressed as first-order formulas.

However, when we follow this last approach we find that, although Maude can automatically perform analyses like model checking of temporal formulas

or verification of invariants, other formal analyses have to be done "by hand," thus disconnecting the real Maude code from its logical meaning. Although some efforts, like the Inductive Theorem Prover [4], have been dedicated to palliate this problem, they are restricted to inductive proofs in Church-Rosser equational theories, and they lack the generality to deal with all the features of Maude. With our approach, we cover arbitrary first-order properties (also written in logics different from Maude), and open the door to automated induction strategies such as those of ISAplanner [7].

The Heterogeneous Tool Set, Hets [16] is an institution-based combination of different logics and corresponding rewriting, model checking and proof tools. Tools that have been integrated into Hets include the SAT solvers zChaff and MiniSat, the automated provers SPASS, Vampire and Darwin, and the interactive provers Isabelle and VSE.

In this paper, we describe an integration of Maude into Hets from which we expect several benefits: On the one hand, Maude will be the first dedicated rewriting engine that is integrated into Hets (so far, only the rewriting engine of Isabelle is integrated, which however is quite specialized towards higher-order proofs). On the other hand, certain features of the Maude module system like views lead to proof obligations that cannot be checked with Maude—Hets will be the suitable framework to prove them, using the above mentioned proof tools.

The rest of the paper is organized as follows: after briefly introducing Hets in Section 2 and Maude in Section 3, Section 4 describes the institution we have defined for Maude and the comorphism from this institution to Casl. Section 5 shows how development graphs for Maude specifications are built, and then how they are normalized to deal with freeness constraints. Section 6 illustrates the integration of Maude into Hets with the help of an example, while Section 7 concludes and outlines the future work.

2 Hets

The central idea of Hets is to provide a general logic integration and proof management framework. 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 benefit of plugging in a new logic and tool such as Maude into the Hets motherboard is the gained interoperability with the other logics and tools available in Hets.

The work that needs to be done for such an integration is to prepare both the Maude logic and tool so that it can act as an expansion card for Hets. On the side of the semantics, this means that the logic needs to be organized as an *institution* [12]. Institutions capture in a very abstract and flexible way the notion of a logical system, by leaving open the details of signatures, models, sentences (axioms) and satisfaction (of sentences in models). The only condition governing the behavior of institutions is the *satisfaction condition*, stating that *truth is invariant under change of notation* (or enlargement of context), which

is captured by the notion of signature morphism (which leads to translations of sentences and reductions of models), see [12] for formal details.

Indeed, HETS has interfaces for plugging in the different components of an institution: signatures, signature morphisms, sentences, and their translation along signature morphisms. Recently, even (some) models and model reducts have been covered, although this is not needed here. Note, however, that the model theory of an institution (including model reducts and the satisfaction condition) is essential when relating different logics via institution comorphisms. The logical correctness of their use in multi-logic proofs is ensured by model-theoretic means.

For proof management, HETS uses *development graphs* [15]. They can be defined over an arbitrary institution, and they are used to encode structured specifications in various phases of the development. Roughly speaking, each node of the graph represents a theory. The links of the graph define how theories can make use of other theories.

Definition 1. *A* development graph *is an acyclic, directed graph* $\mathcal{DG} = \langle \mathcal{N}, \mathcal{L} \rangle$.

$\mathcal{N}$ *is a set of nodes. Each node* $N \in \mathcal{N}$ *is a tuple* (Σ^N, Φ^N) *such that* Σ^N *is a signature and* $\Phi^N \subseteq Sen(\Sigma^N)$ *is the set of* **local axioms** *of* N.

$\mathcal{L}$ *is a set of directed links, so-called* **definition links**, *between elements of* $\mathcal{N}$. *Each definition link from a node* M *to a node* N *is either*

- **global** *(denoted* $M \overset{\sigma}{\Longrightarrow} N$*), annotated with a signature morphism* $\sigma : \Sigma^M \to \Sigma^N$, *or*
- **local** *(denoted* $M \overset{\sigma}{\longrightarrow} N$*), again annotated with a signature morphism* $\sigma : \Sigma^M \to \Sigma^N$, *or*
- **hiding** *(denoted* $M \overset{\sigma}{\underset{hide}{\Longrightarrow}} N$*), annotated with a signature morphism* $\sigma : \Sigma^N \to \Sigma^M$ going against the direction of the link, *or*
- **free** *(denoted* $M \overset{\sigma}{\underset{free}{\Longrightarrow}} N$*), annotated with a signature morphism* $\sigma : \Sigma \to \Sigma^M$ where Σ is a subsignature of Σ^M.

Definition 2. *Given a node* M *in a development graph* $\mathcal{DG}$, *its associated class* $\mathbf{Mod}_{\mathcal{DG}}(M)$ *of models (or* M*-models for short) is inductively defined to consist of those* Σ^M*-models* m *for which*

1. m *satisfies the local axioms* Φ^M,
2. *for each* $N \overset{\sigma}{\Longrightarrow} M \in \mathcal{DG}$, $m|_\sigma$ *is an* N*-model,*
3. *for each* $N \overset{\sigma}{\longrightarrow} M \in \mathcal{DG}$, $m|_\sigma$ *satisfies the local axioms* Φ^N,
4. *for each* $N \overset{\sigma}{\underset{hide}{\Longrightarrow}} M \in \mathcal{DG}$, m *has a* σ*-expansion* m' *(i.e.* $m'|_\sigma = m$*) that is an* N*-model, and*
5. *for each* $N \overset{\sigma}{\underset{free}{\Longrightarrow}} M \in \mathcal{DG}$, m *is an* N*-model that is persistently* σ*-free in* $\mathbf{Mod}(N)$. *The latter means that for each* N*-model* m' *and each model morphism* $h : m|_\sigma \to m'|_\sigma$, *there exists a unique model morphism* $h^\# : m \to m'$ *with* $h^\#|_\sigma = h$.

Complementary to definition links, which *define* the theories of related nodes, we introduce the notion of a *theorem link* with the help of which we are able to *postulate* relations between different theories. Theorem links are the central data structure to represent proof obligations arising in formal developments. Again, we distinguish between local and global theorem links (denoted by $N = \stackrel{\sigma}{=}\Rightarrow M$ and $N - \stackrel{\sigma}{-} \succ M$ respectively). We also need theorem links $N = \stackrel{\sigma}{\underset{hide}{=}}\Rightarrow_{\theta} M$ (where for some Σ, $\theta : \Sigma \to \Sigma^N$ and $\sigma : \Sigma \to \Sigma^M$) involving hiding. The semantics of global theorem links is given by the next definition; the others do not occur in our examples and we omit them.

Definition 3. *Let $\mathcal{DG}$ be a development graph and N, M nodes in $\mathcal{DG}$.*

$\mathcal{DG}$ **implies** *a global theorem link $N = \stackrel{\sigma}{=}\Rightarrow M$ (denoted $\mathcal{DG} \models N = \stackrel{\sigma}{=}\Rightarrow M$) iff for all $m \in Mod(M)$, $m|_\sigma \in Mod(N)$.*

3 Rewriting Logic and Maude

Maude is an efficient tool for equational reasoning and rewriting. Methodologically, Maude specifications are divided into a specification of the data objects and a specification of some concurrent transition system, the states of which are given by the data part. Indeed, at least in specifications with initial semantics, the states can be thought of as equivalence classes of terms. The data part is written in a variant of subsorted conditional equational logic. The transition system is expressed in terms of a binary rewriting relation, and also may be specified using conditional Horn axioms.

Two corresponding logics have been introduced and studied in the literature: rewriting logic and preordered algebra [13]. They essentially differ in the treatment of rewrites: whereas in rewriting logic, rewrites are named, and different rewrites between two given states (terms) can be distinguished (which corresponds to equipping each carrier set with a category of rewrites), in preordered algebra, only the existence of a rewrite does matter (which corresponds to equipping each carrier set with a preorder of rewritability).

Rewriting logic has been announced as the logic underlying Maude [3]. Maude modules lead to rewriting logic theories, which can be equipped with loose semantics (`fth`/`th` modules) or initial/free semantics (`fmod`/`mod` modules). Although rewriting logic is not given as an institution [6], a so-called specification frame (collapsing signatures and sentences into theories) would be sufficient for our purposes.

However, after a closer look at Maude and rewriting logic, we found out that de facto, the logic underlying Maude differs from the rewriting logic as defined in [13]. The reasons are:

1. In Maude, labels of rewrites cannot (and need not) be translated along signature morphisms. This means that *e.g. Maude views do not lead to theory morphisms in rewriting logic!*

2. Although labels of rewrites are used in traces of counterexamples, they play a subsidiary role, because they cannot be used in the linear temporal logic of the Maude model checker.

Specially the first reason completely rules out a rewriting logic-based integration of Maude into HETS: if a view between two modules is specified, HETS definitely needs a theory morphism underlying the view.[3] However, the Maude user does not need to provide the action of the signature morphism on labeled rewrites, and generally, there is more than one possibility to specify this action.

The conclusion is that the most appropriate logic to use for Maude is preordered algebra [10]. In this logic, rewrites are neither labeled nor distinguished, only their existence is important. This implies that Maude views lead to theory morphisms in the institution of preordered algebras. Moreover, this setting also is in accordance with the above observation that in Maude, rewrite labels are not first-class citizens, but are mere names of sentences that are convenient for decorating tool output (e.g. traces of the model checker). Labels of sentences play a similar role in HETS, which perfectly fits here.

Actually, the switch from rewriting logic to preordered algebras has effects on the consequence relation, contrary to what is said in [13]. Consider the following Maude theory:

```
th A is
 sorts S T .
 op a : -> S .
 eq X:S = a .
 ops h k : S -> T .
 rl [r] : a => a .
 rl [s] : h(a) => k(a) .
endfth
```

This logically implies $h(x) \Rightarrow k(x)$ in preordered algebra, but not in rewriting logic, since in the latter logic it is easy to construct models in which the naturality condition $r; k(r) = h(r); s$ fails to hold.

Before describing how to encode Maude into HETS we briefly outline the structuring mechanisms used in Maude specifications:

Module importation. In Maude, a module can be imported in three different modes, each of them stating different semantic constraints: Importing a module in `protecting` mode intuitively means that *no junk and no confusion* are added; importing a module in `extending` mode indicates that junk is allowed, but *confusion is forbidden*; finally, importing a module in `including` mode indicates that *no requirements* are assumed.

Module summation. The summation module operation creates a new module that includes all the information in its summands.

[3] If the Maude designers would let (and force) users to specify the action of signature morphisms on rewrite labels, it would not be difficult to switch the HETS integration of Maude to being based on rewriting logic.

Renaming. The renaming expression allows to rename sorts, operators (that can be distinguished by their profiles), and labels.

Theories. Theories are used to specify the requirements that the parameters used in parameterized modules must fulfill. Functional theories are membership equational specifications with *loose* semantics. Since the statements specified in theories are not expected to be executed in general, they do not need to satisfy the executability requirements.

Views. A view indicates how a particular module satisfies a theory, by mapping sorts and operations in the theory to those in the target module, in such a way that the induced translations on equations and membership axioms are provable in the module. Note that Maude does not provide a syntax for mapping rewrite rules; however, the existence of rewrites between terms must be preserved by views.

4 Relating the Maude and CASL Logics

In this section, we will relate Maude and CASL at the level of logical systems. The structuring level will be considered in the next section.

4.1 Maude

As already motivated in Section 3, we will work with preordered algebra semantics for Maude. We will define an institution, that we will denote $Maude^{pre}$, which can be, like in the case of Maude's logic, parametric over the underlying equational logic. Following the Maude implementation, we have used membership equational logic [14]. Notice that the resulting institution $Maude^{pre}$ is very similar to the one defined in the context of CafeOBJ [10,6] for preordered algebra (the differences are mainly given by the discussion about operation profiles below, but this is only a matter of representation). This allows us to make use of some results without giving detailed proofs.

Signatures of $Maude^{pre}$ are tuples $(K, F, kind : (S, \leq) \rightarrow K)$, where K is a set (of *kinds*), *kind* is a function assigning a kind to each *sort* in the poset $(S, \leq)$, and F is a set of function symbols of the form $F = \{F_{k_1 \ldots k_n \rightarrow x} \mid k_i, k \in K\} \cup \{F_{s_1 \ldots s_n \rightarrow s} \mid s_i, s \in S\}$ such that if $f \in F_{s_1 \ldots s_n \rightarrow s}$, there is a symbol $f \in F_{kind(s_1) \ldots kind(s_n) \rightarrow kind(s)}$. Notice that there is actually no essential difference between our putting operation profiles on sorts into the signatures and Meseguer's original formulation putting them into the sentences.

Given two signatures $\Sigma_i = (K_i, F_i, kind_i)$, $i \in \{1, 2\}$, a signature morphism $\phi : \Sigma_1 \rightarrow \Sigma_2$ consists of a function $\phi^{kind} : K_1 \rightarrow K_2$ which preserves $\leq_1$, a function between the sorts $\phi^{sort} : S_1 \rightarrow S_2$ such that $\phi^{sort}; kind_2 = kind_1; \phi^{kind}$ and the subsorts are preserved, and a function $\phi^{op} : F_1 \rightarrow F_2$ which maps operation symbols compatibly with the types. Moreover, the overloading of symbol names must be preserved, i.e. the name of $\phi^{op}(\sigma)$ must be the same both when mapping the operation symbol σ on sorts and on kinds. With composition defined component-wise, we get the category of signatures.

For a signature Σ, a model M interprets each kind k as a preorder $(M_k, \leq)$, each sort s as a subset M_s of $M_{kind(s)}$ that is equipped with the induced preorder, with M_s a subset of $M_{s'}$ if $s < s'$, and each operation symbol $f \in F_{k_1 \ldots k_n, k}$ as a function $M_f : M_{k_1} \times \ldots \times M_{k_n} \to M_k$ which has to be monotonic and such that for each function symbol f on sorts, its interpretation must be a restriction of the interpretation of the corresponding function on kinds. For two Σ-models A and B, a homomorphism of models is a family $\{h_k : A_k \to B_k\}_{k \in K}$ of preorder-preserving functions which is also an algebra homomorphism and such that $h_{kind(s)}(A_s) \subseteq B_s$ for each sort s.

The sentences of a signature Σ are Horn clauses built with three types of atoms: equational atoms $t = t'$, membership atoms $t : s$, and rewrite atoms $t \Rightarrow t'$, where t, t' are F-terms and s is a sort in S. Given a Σ-model M, an equational atom $t = t'$ holds in M if $M_t = M_{t'}$, a membership atom $t : s$ holds when M_t is an element of M_s, and a rewrite atom $t \Rightarrow t'$ holds when $M_t \leq M_{t'}$. Notice that the set of variables X used for quantification is K-sorted. The satisfaction of sentences extends the satisfaction of atoms in the obvious way.

4.2 CASL

CASL, the Common Algebraic Specification Language [1,5], has been designed by CoFI, the international *Common Framework Initiative for algebraic specification and development*. Its underlying logic combines first-order logic and induction (the latter is expressed using so-called sort generation constraints, which express term-generatedness of a part of a model; this is needed for the specification of the usual inductive datatypes) with subsorts and partial functions. The institution underlying CASL is introduced in two steps: first, many-sorted partial first-order logic with sort generation constraints and equality ($PCFOL^{=}$) is introduced, and then, subsorted partial first-order logic with sort generation constraints and equality ($SubPCFOL^{=}$) is described in terms of $PCFOL^{=}$. In contrast to Maude, CASL's subsort relations may be interpreted by arbitrary injections $inj_{s,t}$, not only by subsets. We refer to [5] for details. We will only need the Horn Clause fragment of first-order logic. For freeness (see Sect. 5.1), we will also need sort generation constraints, as well as the *second-order* extension of CASL with quantification over predicates.

4.3 Encoding Maude in CASL

We now present an encoding of Maude into CASL. It can be formalized as a so-called institution comorphism [11]. The idea of the encoding of $Maude^{pre}$ in CASL is that we represent rewriting as a binary predicate and we axiomatize it as a preorder compatible with operations.

Every Maude signature $(K, F, kind : (S, \leq) \to K)$ is translated to the CASL theory $((S', \leq', F, P), E)$, where S' is the disjoint union of K and S, $\leq'$ extends the relation $\leq$ on sorts with pairs $(s, kind(s))$, for each $s \in S$, $rew \in P_{s,s}$ for any $s \in S'$ is a binary predicate and E contains axioms stating that for any kind

k, $rew \in P_{k,k}$ is a preorder compatible with the operations. The latter means that for any $f \in F_{s_1..s_n,s}$ and any x_i, y_i of sort $s_i \in S'$, $i = 1, .., n$, if $rew(x_i, y_i)$ holds, then $rew(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n))$ also holds.

Let $\Sigma_i,\ i = 1, 2$ be two Maude signatures and let $\varphi : \Sigma_1 \to \Sigma_2$ be a Maude signature morphism. Then its translation $\Phi(\varphi) : \Phi(\Sigma_1) \to \Phi(\Sigma_2)$ denoted ϕ, is defined as follows:

- for each $s \in S$, $\phi(s) := \varphi^{sort}(s)$ and for each $k \in K$, $\phi(k) := \varphi^{kind}(k)$.
- the subsort preservation condition of ϕ follows from the similar condition for φ.
- for each operation symbol σ, $\phi(\sigma) := \varphi^{op}(\sigma)$.
- rew is mapped identically.

The sentence translation map for each signature is obtained in two steps. While the equational atoms are translated as themselves, membership atoms $t : s$ are translated to CASL memberships $t\ in\ s$ and rewrite atoms of form $t \Rightarrow t'$ are translated as $rew(t, t')$. Then, any sentence of Maude of the form $(\forall x_i : k_i)H \implies C$, where H is a conjunction of Maude atoms and C is an atom is translated as $(\forall x_i : k_i)H' \implies C'$, where H' and C' are obtained by mapping all the Maude atoms as described before.

Given a Maude signature Σ, a model M' of its translated theory (Σ', E) is reduced to a Σ-model denoted M where:

- for each kind k, define $M_k = M'_k$ and the preorder relation on M_k is rew;
- for each sort s, define M_s to be the image of M'_s under the injection $inj_{s,kind(s)}$ generated by the subsort relation;
- for each f on kinds, let $M_f(x_1, .., x_n) = M'_f(x_1, .., x_n)$ and for each f on sorts of result sort s, let $M_f(x_1, .., x_n) = inj_{s,kind(s)}(M'_f(x_1, .., x_n))$. M_f is monotone because axioms ensure that M'_f is compatible with rew.

The reduct of model homomorphisms is the expected one; the only thing worth noticing is that $h_{kind(s)}(M_s) \subseteq N_s$ for each sort s follows from the CASL model homomorphism condition of h.

Notice that the model reduct is an isomorphism of categories.

5 From Maude Modules to Development Graphs

We describe in this section how Maude structuring mechanisms described in Section 3 are translated into development graphs. Then, we explain how these development graphs are normalized to deal with freeness constraints.

Signature morphisms are produced in different ways; explicitly, renaming of module expressions and views lead to signature morphisms; however, implicitly we also find other morphisms: the sorts defined in the theories are qualified with the parameter in order to distinguish sorts with the same name that will be instantiated later by different ones; moreover, sorts defined (not imported) in parameterized modules can be parameterized as well, so when the theory is

instantiated with a view these sorts are also renamed (e.g. the sort `List{X}` for generic lists can become `List{Nat}`).

Each Maude module generates two nodes in the development graph. The first one contains the theory equipped with the usual loose semantics. The second one, linked to the first one with a free definition link (whose signature morphism is detailed below), contains the same signature but no local axioms and stands for the free models of the theory. Note that Maude theories only generate one node, since their initial semantics is not used by Maude specifications. When importing a module, we will select the node used depending on the chosen importation mode:

- The `protecting` mode generates a non-persistent free link between the current node and the node standing for the free semantics of the included one.
- The `extending` mode generates a global link with the annotation PCons?, that stands for proof-theoretic conservativity and that can be checked with a special conservativity checker that is integrated into HETS.
- The `including` mode generates a global definition link between the current node and the node standing for the loose semantics of the included one.

The summation module expression generates a new node that includes all the information in its summands. Note that this new node can also need a node with its free model if it is imported in protecting mode.

The model class of parameterized modules consists of free extensions of the models of their parameters, that are persistent on sorts, but not on kinds. This notion of freeness has been studied in [2] under assumptions like existence of top sorts for kinds and sorted variables in formulas; our results hold under similar hypotheses. Thus, we use the same non-persistent free links described for protecting importation to link these modules with their corresponding theories. Views do not generate nodes in the development graph but theorem links between the node corresponding to the source theory and the node with the free model of the target. However, Maude views provide a special kind of mapping between terms, that can in general map functions of different arity. When this mapping is used we generate a new inner node extending the signature of the target to include functions of the adequate arity.

We illustrate how to build the development graph with an example. Consider the following Maude specifications:

```
fmod M1 is
 sort S1 .
 op _+_ : S1 S1 -> S1 [comm] .
endfm

th T is
 sort S1 .
 op _._ : S1 S1 -> S1 .
 eq V1:S1 . V2:S1 = V2:S1 . V1:S1 [nonexec] .
endth
```

```
fmod M2 is
  sort S2 .
endfm

mod M3{X :: T} is
 sort S4 .
endm
```

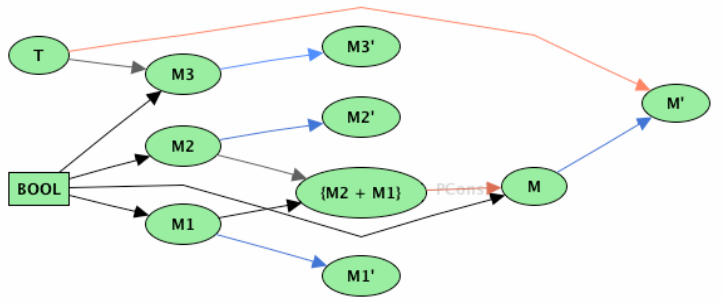

Fig. 1. Development Graph for Maude Specifications

```
mod M is
 ex M1 + M2 * (sort S2 to S) .
endm
```

```
view V from T to M is
 op _._ to _+_ .
endv
```

HETS builds the graph shown in Fig. 1, where the following steps take place:

- Each module has generated a node with its name and another primed one that contains the initial model, while both of them are linked with a non-persistent free link. Note that theory `T` did not generate this primed node.
- The summation expression has created a new node that includes the theories of `M1` and `M2`, importing the latter with a renaming; this new node, since it is imported in `extending` mode, uses a link with the PCons? annotation.
- There is a theorem link between `T` and the free (here: initial) model of `M`. This link is labeled with the mapping defined in the view `V`.
- The parameterized module `M3` includes the theory of its parameter with a renaming, that qualifies the sort. Note that these nodes are connected by means of a non-persistent freeness link.

It is straightforward to show:

Theorem 1. *The translation of Maude modules into development graphs is semantics-preserving.*

Once the development graph is built, we can apply the (logic independent) calculus rules that reduce global theorem links to local theorem links, which are in turn discharged by local theorem proving [15]. This can be used to prove Maude views, like e.g. "natural numbers are a total order." We show in the next section how we deal with the freeness constraints imposed by free definition links.

5.1 Normalization of free definition links

Maude uses initial and free semantics intensively. The semantics of freeness is, as mentioned, different from the one used in CASL in that the free extensions

of models are required to be persistent only on sorts and new error elements can be added on the interpretation of kinds. Attempts to design the translation to CASL in such a way that Maude free links would be translated to usual free definition links in CASL have been unsuccessful. We decided thus to introduce a special type of links to represent Maude's freeness in CASL. In order not to break the development graph calculus, we need a way to normalize them. The idea is to replace them with a semantically equivalent development graph in CASL. The main idea is to make a free extension persistent by duplicating parameter sorts appropriately, such that the parameter is always explicitly included in the free extension.

For any Maude signature Σ, let us define an extension $\Sigma^{\#} = (S^{\#}, \leq^{\#}, F^{\#}, P^{\#})$ of the translation $\Phi(\Sigma)$ of Σ to CASL as follows:

- $S^{\#}$ unites with the sorts of $\Phi(\Sigma)$ the set $\{[s] \mid s \in Sorts(\Sigma)\}$;
- $\leq^{\#}$ extends the subsort relation $\leq$ with pairs $(s, [s])$ for each sort s and $([s], [s'])$ for any sorts $s \leq s'$;
- $F^{\#}$ adds the function symbols $\{f : [w] \to [s]\}$ for all function symbols on sorts $f : w \to s$;[4]
- $P^{\#}$ adds the predicate symbol rew on all new sorts.

Now, we consider a Maude non-persistent free definition link and let $\sigma : \Sigma \to \Sigma'$ be the morphism labeling it.[5] We define a CASL signature morphism $\sigma^{\#} : \Phi(\Sigma) \to \Sigma'^{\#}$: on sorts, $\sigma^{\#}(s) := \sigma^{sort}(s)$ and $\sigma^{\#}([s]) := [\sigma^{sort}(s)]$; on operation symbols, we can define $\sigma^{\#}(f) := \sigma^{op}(f)$ and this is correct because the operation symbols were introduced in $\Sigma'^{\#}$; rew is mapped identically.

The normalization of Maude freeness is then illustrated in Fig.2. Given a free non-persistent definition link $M \xRightarrow[free]{\sigma} N$, with $\sigma : \Sigma \to \Sigma_N$, we first take the translation of the nodes to CASL (nodes M' and N') and then introduce a new node, K, labeled with $\Sigma_N^{\#}$, a global definition link from M' to M'' labeled with the inclusion ι_N of Σ_N in $\Sigma_N^{\#}$, a free definition link from M'' to K labeled with $\sigma^{\#}$ and a hiding definition link from K to N' labeled with the inclusion ι_N.[6]

Notice that the models of N are Maude reducts of CASL models of K, reduced along the inclusion ι_N.

The next step is to eliminate CASL free definition links. The idea is to use then a transformation specific to the second-order extension of CASL to normalize freeness. The intuition behind this construction is that it mimics the quotient term algebra construction, that is, the free model is specified as the homomorphic image of an absolutely free model (i.e. term model).

We are going to make use of the following known facts [18]:

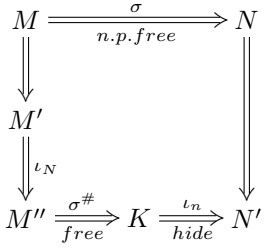

Fig. 2. Normalization of Maude free links

[4] $[x_1 \dots x_n]$ is defined to be $[x_1] \dots [x_n]$.

[5] In Maude, this would usually be an injective renaming.

[6] The arrows without labels in Fig.2 correspond to heterogeneous links from Maude to CASL.

Fact 1 *Extensions of theories in Horn form admit free extensions of models.*

Fact 2 *Extensions of theories in Horn form are monomorphic.*

Given a free definition link $M \xRightarrow[free]{\sigma} N$, with $\sigma : \Sigma \to \Sigma^N$ such that $Th(M)$ is in Horn form, replace it with $M \xRightarrow{incl} K \xRightarrow[hide]{incl} N'$, where N' has the same signature as N, *incl* denotes inclusions and the node K is constructed as follows.

The signature Σ^K consists of the signature Σ^M disjointly united with a copy of Σ^M, denoted $\iota(\Sigma_M)$ which makes all function symbols total (let us denote $\iota(x)$ the corresponding symbol in this copy for each symbol x from the signature Σ^M) and augmented with new operations $h : \iota(s) \to ?s$, for any sort s of Σ^M and $make_s : s \to \iota(s)$, for any sort s of the source signature Σ of the morphism σ labelling the free definition link.

The axioms ψ^K of the node K consist of:

- sentences imposing the bijectivity of *make*;
- axiomatization of the sorts in $\iota(\Sigma_M)$ as free types with all operations as constructors, including *make* for the sorts in $\iota(\Sigma)$;
- homomorphism conditions for h:

$$h(\iota(f)(x_1, \ldots, x_n)) = f(h(x_1), \ldots, h(x_n))$$

and

$$\iota(p)(t_1, \ldots, t_n) \Rightarrow p(h(t_1), \ldots, h(t_n))$$

- surjectivity of homomorphisms:

$$\forall y : s.\exists x : \iota(s).h(x) \stackrel{e}{=} y$$

- a second-order formula saying that the kernel of h is the least partial predicative congruence[7] satisfying $Th(M)$. This is done by quantifying over a predicate symbol for each sort for the binary relation and one predicate symbol for each relation symbol as follows:

$$\forall \{P_s : \iota(s), \iota(s)\}_{s \in Sorts(\Sigma_M)}, \{P_{p:w} : \iota(w)\}_{p:w \in \Sigma_M}$$
$$. \ symmetry \wedge transitivity \wedge congruence \wedge satThM \implies largerThenKerH$$

where *symmetry* stands for

$$\bigwedge_{s \in Sorts(\Sigma^M)} \forall x : \iota(s), y : \iota(s).P_s(x, y) \implies P_s(y, x),$$

[7] A partial predicative congruence consists of a symmetric and transitive binary relation for each sort and a relation of appropriate type for each predicate symbol.

transitivity stands for

$$\bigwedge_{s \in Sorts(\Sigma^M)} \forall x : \iota(s), y : \iota(s), z : \iota(s).P_s(x,y) \wedge P_s(y,z) \implies P_s(x,z),$$

congruence stands for

$$\bigwedge_{f_{w \to s} \in \Sigma^M} \forall x_1 \ldots x_n : \iota(w), y_1 \ldots y_n : \iota(w)\,.$$
$$D(\iota(f_{w,s})(\bar{x})) \wedge D(\iota(f_{w,s})(\bar{y})) \wedge P_w(\bar{x},\bar{y}) \implies P_s(\iota(f_{w,s})(\bar{x}), \iota(f_{w,s})(\bar{y}))$$

and

$$\bigwedge_{p_w \in \Sigma^M} \forall x_1 \ldots x_n : \iota(w), y_1 \ldots y_n : \iota(w)\,.$$
$$D(\iota(f_{w,s})(\bar{x})) \wedge D(\iota(f_{w,s})(\bar{y})) \wedge P_w(\bar{x},\bar{y}) \implies P_{p:w}(\bar{x}) \Leftrightarrow P_{p:w}(\bar{y})$$

where D indicates definedness. *satThM* stands for

$$Th(M)[\stackrel{e}{=} / P_s; p : w / P_{p:w}; D(t) / P_s(t,t); t = u / P_s(t,u) \vee (\neg P_s(t,t) \wedge \neg P_s(u,u))]$$

where, for a set of formulas Ψ, $\Psi[sy_1/sy'_1; \ldots; sy_n/sy'_n]$ denotes the simultaneous substitution of sy'_i for sy_i in all formulas of Ψ (while possibly instantiating the meta-variables t and u). Finally *largerThenKerH* stands for

$$\bigwedge_{s \in Sorts(\Sigma^M)} \forall x : \iota(s), y : \iota(s).h(x) \stackrel{e}{=} h(y) \implies P_s(x,y)$$
$$\bigwedge \wedge_{p_w \in \Sigma^M} \forall \bar{x} : \iota(w).\iota(p : w)(\bar{x}) \implies P_{p:w}(\bar{x})$$

Proposition 1. *The models of the nodes N and N' are the same.*

6 An example: reversing lists

The example we are going to present is a standard specification of lists with empty lists, composition and reversal. We want to prove that by reversing a list twice we obtain the original list. Since Maude syntax does not support marking sentences of a theory as theorems, in Maude we would normally write a view (*PROVEIDEM* in Fig. 3, left side) from a theory containing the theorem (*REVIDEM*) to the module with the axioms defining *reverse* (*LISTREV*).

The first advantage the integration of Maude in Hets brings in is that we can use heterogeneous Casl structuring mechanisms and the *%implies* annotation to obtain the same development graph in a shorter way – see the right side of Fig. 3. Notice that we made the convention in Hets to have non-persistent freeness for Maude specifications, modifying thus the usual institution-independent semantics of the freeness construct.

For our example, the development calculus rules are applied as follows. First, the library is translated to Casl; during this step, Maude non-persistent free links are normalized. The next step is to normalize Casl free links, using Freeness rule. We then apply the Normal-Form rule which introduces normal forms for the nodes with incoming hiding links (introduced at the previous step) and

```
fmod MYLIST is
 sorts Elt List .
 subsort Elt < List .
 op nil : -> List [ctor] .
 op __ : List List -> List
    [ctor assoc id: nil] .
endfm
fmod MYLISTREV is
 pr MYLIST .
 op reverse : List -> List .
 var L : List .
 var E : Elt .
 eq reverse(nil) = nil .
 eq reverse(E L) = reverse(L) E .
endfm
fth REVIDEM  is
 pr MYLIST .
 op reverse : List -> List .
 var L : List .
 eq reverse(reverse(L)) = L .
endfth
view PROVEIDEM
 from REVIDEM to MYLISTREV is
sort List to List .
op reverse to reverse .
endv
```

logic MAUDE
spec PROVEIDEM =
free
{*sorts Elt List* .
subsort Elt < *List* .
op nil : −> *List* [*ctor*] .
op __ : *List List* −> *List*
[*ctor assoc id*: *nil*] .
}
then {*op reverse* : *List* −> *List* .
var L : *List* . *var E* : *Elt* .
eq reverse(*nil*) = *nil* .
eq reverse(*E L*) = *reverse*(*L*) *E* .
} **then %implies**
{*var L* : *List* .
eq reverse(*reverse*(*L*)) = *L* .
}

Fig. 3. Lists with reverse, in Maude (left) and CASL (right) syntax.

then Theorem-Hide-Shift rule which moves the target of any theorem link targeting a node with incoming hiding links to the normal form of the latter. Calling then Proofs/Automatic, the proof obligation is delegated to the normal form node.

In this node, we now have a proof goal for a second-order theory. It can be discharged using the interactive theorem prover Isabelle/HOL [17]. We have set up a series of lemmas easing such proofs. First of all, normalization of freeness introduces sorts for the free model which are axiomatized to be the homomorphic image of a set of the absolutely free (i.e. term) model. A transfer lemma (that exploits surjectivity of the homomorphism) enables us to transfer any proof goal from the free model to the absolutely free model. Since the absolutely free model is term generated, we can use induction proofs here. For the case of datatypes with total constructors (like lists), we prove by induction that the homomorphism is total as well. Two further lemmas on lists are proved by induction: (1) associativity of concatenation and (2) the reverse of a concatenation is the concatenation (in reverse order) of the reversed lists. This infrastructure then allows us to demonstrate (again by induction) that $reverse(reverse(L)) = L$.

While proof goals in Horn clause form often can be proved with induction, other proof goals like the inequality of certain terms or extensionality of sets cannot. Here, we need to prove inequalities or equalities with more complex premises, and this calls for use of the special axiomatization of the kernel of the homomorphism. This axiomatization is rather complex, and we are currently setting up the infrastructure for easing such proofs in Isabelle/HOL.

7 Conclusions and Future Work

We have presented in this paper how Maude has been integrated into Hets, a parsing, static analysis and proof management tool that combines various tools for different specification languages. To achieve this integration, we consider preordered algebra semantics for Maude and define an institution comorphism from Maude to Casl. This integration allows to prove properties of Maude specifications like those expressed in Maude views. We have also implemented a normalization of the development graphs that allows us to prove freeness constraints. We have used this transformation to connect Maude to Isabelle [17], a Higher Order Logic prover, and have demonstrated a small example proof about reversal of lists. Moreover, this encoding is suited for proofs of e.g. extensionality of sets, which require first-order logic, going beyond the abilities of existing Maude provers like ITP.

Since interactive proofs are often not easy to conduct, future work will make proving more efficient by adopting automated induction strategies like rippling [7]. We also have the idea to use the automatic first-order prover SPASS for induction proofs by integrating special induction strategies directly into Hets.

We have also studied the possible comorphisms from Casl to Maude. We distinguish whether the formulas in the source theory are confluent and terminating or not. In the first case, that we plan to check with the Maude termination [8] and confluence checker [9], we map formulas to equations, whose execution in Maude is more efficient, while in the second case we map formulas to rules.

Finally, we also plan to relate Hets' Modal Logic and Maude models in order to use the Maude model checker [3, Chapter 13] for linear temporal logic.

Acknowledgments We wish to thank Francisco Durán for discussions regarding freeness in Maude, Martin Kühl for cooperation on the implementation of the theory presented here, and Maksym Bortin for help with the Isabelle proofs. This work has been supported by the German Federal Ministry of Education and Research (Project 01 IW 07002 FormalSafe), the German Research Council (DFG) under grant KO-2428/9-1 and the MICINN Spanish project *DESAFIOS10* (TIN2009-14599-C03-01).

References

1. M. Bidoit and P. D. Mosses. Casl *User Manual*. *LNCS* 2900 (IFIP Series). Springer, 2004.

2. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236(1-2):35–132, 2000.
3. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*, *LNCS* 4350. Springer, 2007.
4. M. Clavel, M. Palomino, and A. Riesco. Introducing the ITP tool: a tutorial. *Journal of Universal Computer Science*, 12(11):1618–1650, 2006. Programming and Languages. Special Issue with Extended Versions of Selected Papers from PROLE 2005: The Fifth Spanish Conference on Programming and Languages.
5. CoFI (The Common Framework Initiative). CASL *Reference Manual*, *LNCS* 2960. Springer, 2004.
6. R. Diaconescu. *Institution-independent Model Theory*. Birkhäuser Basel, 2008.
7. L. Dixon and J. D. Fleuriot. Higher order rippling in ISAplanner. In K. Slind, A. Bunker, and G. Gopalakrishnan, editors, *TPHOLs*, *LNCS* 3223, pages 83–98. Springer, 2004.
8. F. Durán, S. Lucas, and J. Meseguer. MTT: The Maude Termination Tool (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning, IJCAR 2008, Sydney, Australia, August 12-15*, *LNCS* 5195, pages 313–319. Springer, 2008.
9. F. Durán and J. Meseguer. A Church-Rosser checker tool for conditional order-sorted equational maude specifications. In *Proceedings of the 8th International Workshop on Rewriting Logic and its Applications (WRLA 2010)*, *LNCS*, 2010. To appear.
10. K. Futatsugi and R. Diaconescu. *CafeOBJ Report*. World Scientific, AMAST Series, 1998.
11. J. Goguen and G. Roşu. Institution morphisms. *Formal Aspects of Computing*, 13:274–307, 2002.
12. 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.
13. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
14. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97, Tarquinia, Italy, June 3–7, 1997, Selected Papers*, *LNCS* 1376, pages 18–61. Springer, 1998.
15. T. Mossakowski, S. Autexier, and D. Hutter. Development graphs - proof management for structured specifications. *Journal of Logic and Algebraic Programming, special issue on Algebraic Specification and Development Techniques*, 67(1-2):114–145, April 2006.
16. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editors, *TACAS 2007*, *LNCS* 4424, pages 519–522. Springer, 2007.
17. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, *LNCS* 2283. Springer, 2002.
18. H. Reichel. *Initial computability, algebraic specifications, and partial algebras*. Oxford University Press, Inc., New York, NY, USA, 1987.