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

June 24, 2010, Québec, AMAST 2010

The Maude system

http://maude.cs.uiuc.edu

- Maude is a high-level language and high-performance system supporting both equational and rewriting logic computation.
- Maude modules correspond to specifications in rewriting logic.
- This logic is an extension of membership equational logic (MEL).
  - Sorts are grouped into equivalence classes called *kinds*.
  - Maude functional modules correspond to specifications in MEL.
  - They specify equations, that must be confluent and terminating.
  - In addition to equations, they allow the statement of membership axioms characterizing the elements of a sort.
- Rewriting logic extends MEL by adding rewrite rules.
  - Rules have to be coherent with equations, but they are not required to be either confluent or terminating.
  - Maude system modules correspond to specifications in rewriting logic.

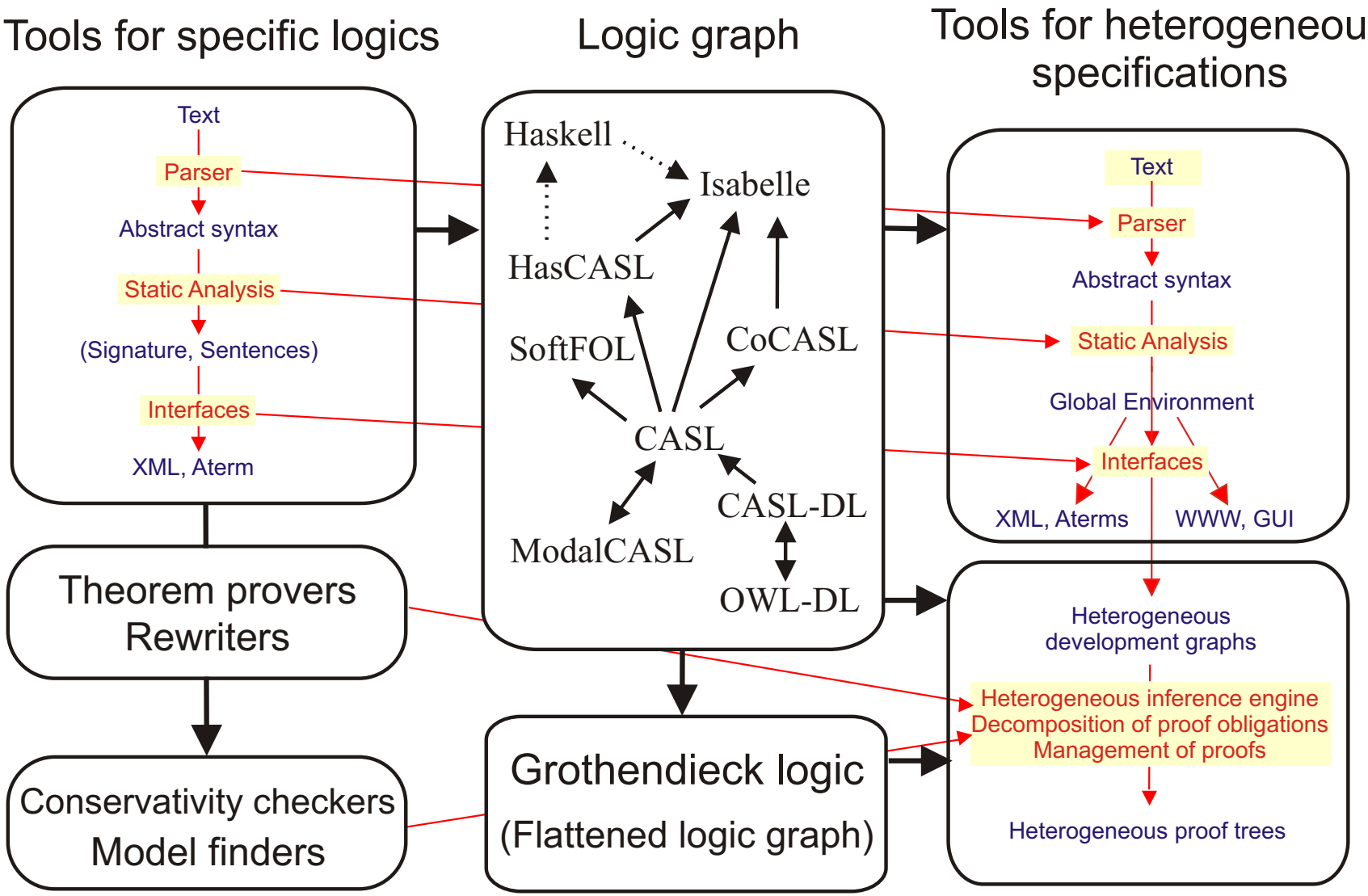
Motivation

- The key point is that there are three different uses of Maude modules:
  1. As programs, to implement some application.
  2. As formal executable specifications, that provide a rigorous mathematical model of an algorithm, a system, a language, or a formalism.
  3. As models that can be formally analyzed and verified with respect to different properties expressing various formal requirements.
- Although Maude can automatically perform analyses like model checking of temporal formulas, other formal analyses have to be done "by hand."
- HETS will be the suitable framework to prove these properties.

The Heterogeneous Tool Set (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—individual logics—can be plugged in.
- The benefit of plugging in Maude into HETS is the gained interoperability with the other logics and tools available in HETS.

Architecture of the heterogeneous tool set Hets



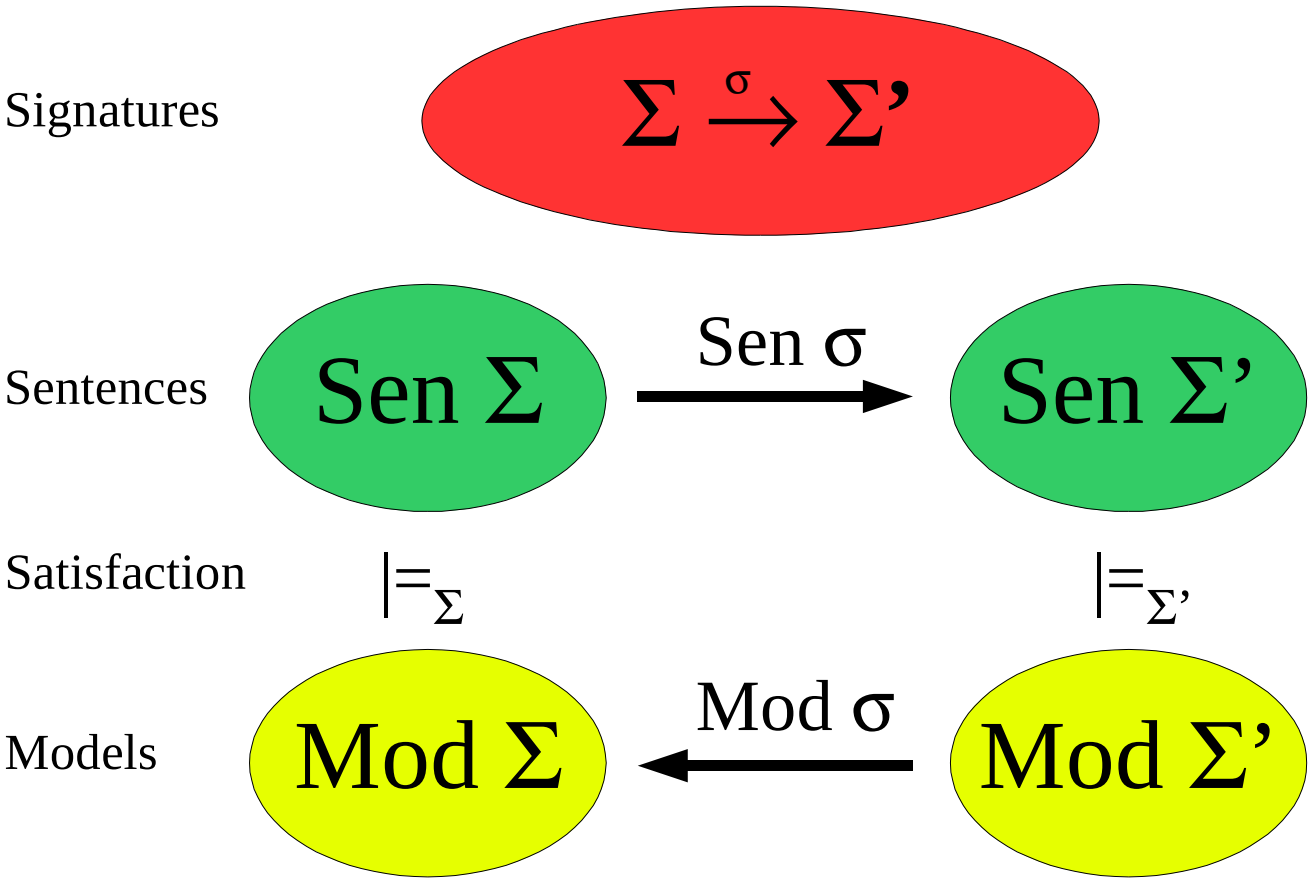
Logics currently supported by HETS

CASL	many-sorted first-order logic, partial functions, subsorting, datatypes (induction)
CoCASL	coalgebraic specification of reactive systems
ModalCASL	first-order modal logic
HasCASL	higher order logic, polymorphism, type classes
Haskell	pure functional programming language
CspCASL	combination of CASL with the process algebra CSP
OWL DL	description logic (DL) fragment of Web Ontology Language (OWL)
Maude	rewriting logic with preorder algebra semantics
VSE	a dynamic logic with Pascal-like programs
RelScheme	Relational schemes
Propositional	classical propositional logic
SoftFOL	softly typed first-order logic ($\Rightarrow$ TPTP)
Isabelle	Isabelle's higher-order logic

Plugging in Maude

- For such an integration is necessary to prepare both the Maude logic and tool so that they can act as an expansion card.
- On the side of the semantics, this means that the logic needs to be organized as an *institution*.

Institutions



Rewriting Logic and Maude

- 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.
- The data part is written in membership equational logic ($t = t'$ and $t : s$).
- The transition system is expressed in terms of a binary rewriting relation ($t \Rightarrow t'$).

```
mod LIST is
 sort Elt List OList .
 subsort Elt < OList < List .

 op nil : -> OList [ctor] .
 op __ : List List -> List [ctor assoc id: nil] .

 cmb E E' L : OList if E < E' /\ E' L : OList .
 op head : List ~> Elt .
 eq head(E L) = L .
 rl [tail] : E L => L .
endm
```

Rewriting Logic and Maude II

- How are the signature morphisms computed?

```
th T is
 sort S .
 ops a b : -> S .

 rl [my-rule] : a => b .
endth
```

```
mod M is
  sort M .
  ops c d e : -> M .
  rl [r1] : c => d .
  rl [r2] : c => e .
  rl [r3] : e => d .
endm
```

- Two corresponding logics have been introduced and studied in the literature: rewriting logic and preordered algebra.
- They essentially differ in the treatment of rewrites:
  - In rewriting logic, rewrites are named, and different rewrites between two given states (terms) can be distinguished.
  - In pre-ordered algebra, only the existence of a rewrite does matter.

Rewriting Logic and Maude III

- The logic underlying Maude differs from the rewriting logic. 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.

```
view V from T to M is
 sort S to M .
 op a to c .
 op b to d .
endv
```

- The most appropriate logic to use for Maude in HETS is preordered algebra.

The Maude institution

- We denote this institution as $Maude^{pre}$.
- Signatures of $Maude^{pre}$ are tuples $(K, F, kind : (S, \leq) \to K)$.
- Given two signatures $\Sigma_i = (K_i, F_i, kind_i)$, $i \in \{1, 2\}$, a signature morphism $\phi : \Sigma_1 \to \Sigma_2$ consists of a function $\phi^{kind} : K_1 \to K_2$ which preserves $\leq_1$, a function between the sorts $\phi^{sort} : S_1 \to S_2$ such that $\phi^{sort}; kind_2 = kind_1; \phi^{kind}$ and the subsorts are preserved, and a function $\phi^{op} : F_1 \to F_2$ which maps operation symbols compatibly with the types.
- Moreover, the overloading of symbol names must be preserved.
- 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 $t \Rightarrow t'$.

Heterogeneous Development Graphs

Heterogeneous structured specifications are mapped into heterogeneous development graphs:

- nodes correspond to individual specification modules
- definition links correspond to imports of modules
- theorem links express proof obligations

Development graphs

- are a tool for management and reuse of proofs.
- have already proved to scale to industrial applications.

Development graphs $\mathcal{S} = \langle \mathcal{N}, \mathcal{L} \rangle$

Nodes in $\mathcal{N}$: (Σ^N, Γ^N) with

- Σ^N signature,
- $\Gamma^N \subseteq \mathbf{Sen}(\Sigma^N)$ set of local axioms.

Links in $\mathcal{L}$:

- global $M \xrightarrow{\sigma} N$, where $\sigma : \Sigma^M \to \Sigma^N$,
- local $M \xrightarrow{\sigma} N$ where $\sigma : \Sigma^M \to \Sigma^N$,
- hiding $M \xrightarrow[h]{\sigma} N$ where $\sigma : \Sigma^N \to \Sigma^M$
  going against the direction of the link, or
- free $M \xrightarrow[free]{\sigma} N$, where $\sigma : \Sigma \to \Sigma^M$
  with Σ is a subsignature of Σ^M.

Theorem links

Theorem links come, like definition links, in different versions:

- global theorem links $M - - -^{\sigma}- - \succ N$, where $\sigma : \Sigma^M \longrightarrow \Sigma^N$,
- local theorem links $M - - -^{\sigma}- - \succ N$, where $\sigma : \Sigma^M \longrightarrow \Sigma^N$

Theorem

The proof calculus for heterogeneous development graphs is sound and complete.

From Maude to Development Graphs

Module importation. 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.

Renaming. The renaming expression allows to rename sorts, operators, 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.

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.

Example: Maude code

```
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
```

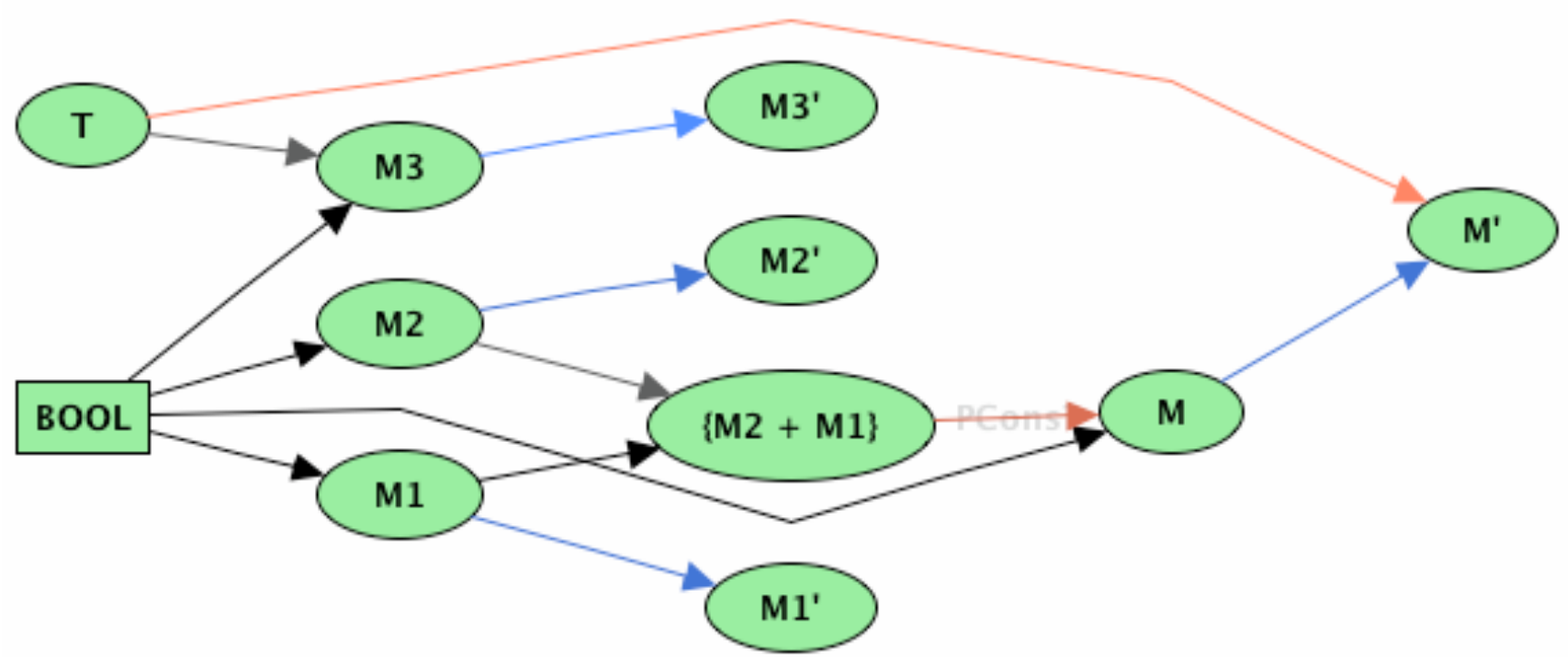
```
mod M is
 ex M1 + M2 * (sort S2 to S) .
endm
```

```
fmod M2 is
  sort S2 .
endfm
```

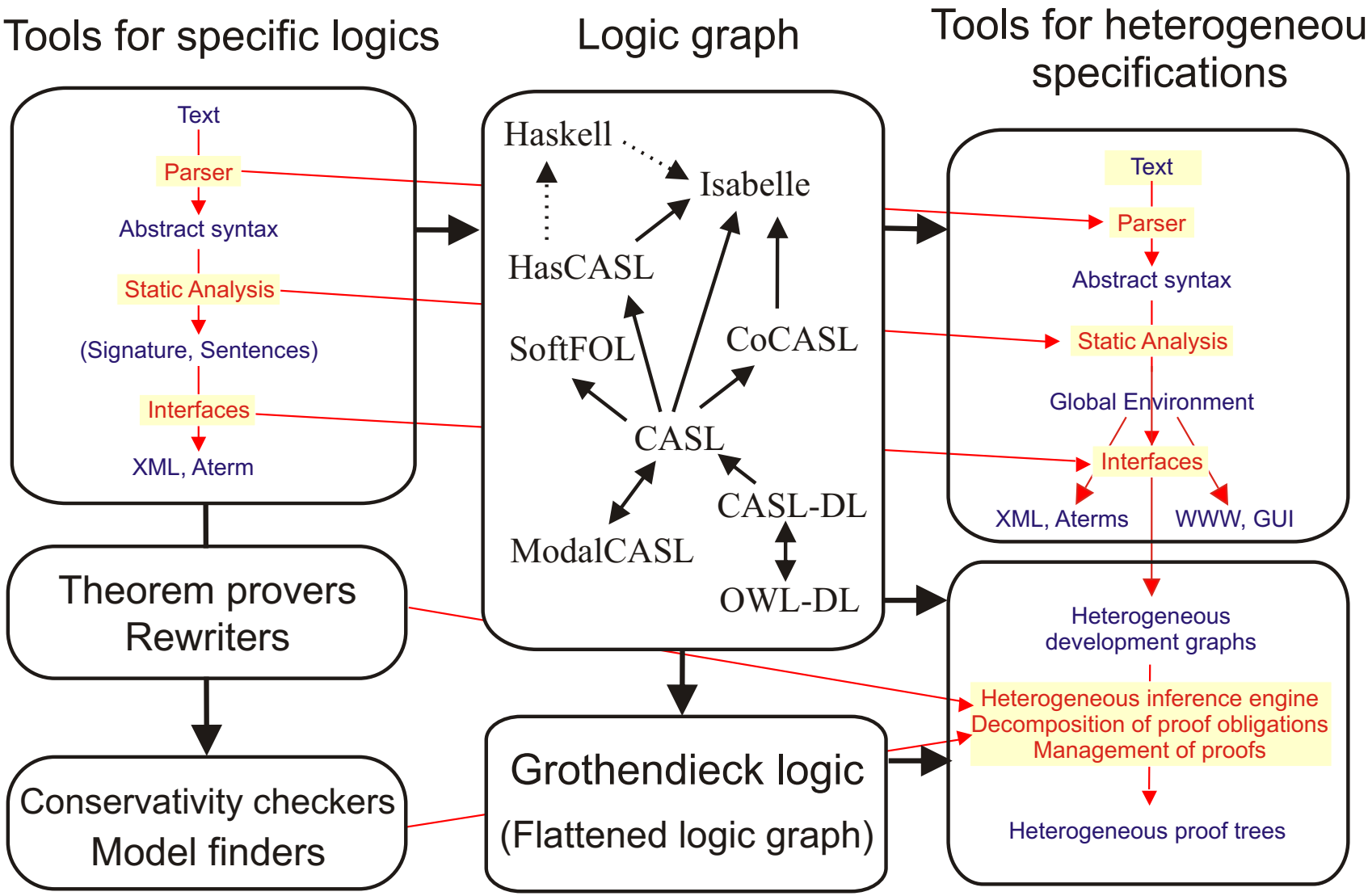
```
mod M3{X :: T} is
 sort S3 .
endm
```

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

Example: Development Graph



Architecture of the heterogeneous tool set Hets



Maude to CASL

- The underlying logic of CASL combines first-order logic and induction (the latter expressed by using sort generation constraints) with subsorts and partial functions.
- An encoding of Maude into CASL can be formalized as an institution comorphism.
- IDEA: We represent rewriting as a binary predicate and we axiomatize it as a preorder compatible with operations.
- Every Maude signature $(K, F, kind : (S, \leq) \rightarrow K)$ 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 rew is a preorder compatible with the operations (i.e., if $rew(x_i, y_i)$ holds, then $rew(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n))$ also holds).

Maude to CASL

- Let $\Sigma_i,\ i = 1, 2$ be two Maude signatures and let $\varphi : \Sigma_1 \rightarrow \Sigma_2$ be a Maude signature morphism. Then its translation $\Phi(\varphi) : \Phi(\Sigma_1) \rightarrow \Phi(\Sigma_2)$ 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:
  - 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.

Example: Comorphism

```
mod AMAST is
 sorts A B C .
 subsort A < B .
```

$S = \{A, B, C\}$
$K = \{[B], [C]\}$

$S' = \{A, B, C, [B], [C]\}$

$pred\ rew : A * A$
$pred\ rew : B * B$
$pred\ rew : [B] * [B]$
$\ldots$
$\forall\ V : A\ .\ rew(V, V)$
$\forall\ V1, V2, V3 : A\ .\ rew(V1, V2) \wedge rew(V1, V2)$
$\Rightarrow rew(V1, V3)$
$\ldots$
$A < [B]$
$B < [B]$
$C < [C]$

Example: Comorphism

```
op f : A B -> C .
op g : B B -> B [comm assoc] .
op b : -> B [ctor] .
```

$\forall\ V0, V1 : [B]\ .\ g(V0, V1) = g(V1, V0)$
$\forall\ V0, V1, V2, V3 : [B]\ .\ rew(V0, V2) \wedge rew(V1, V3) \Rightarrow rew(g(V0, V1), g(V2, V3))$

```
mb b : A .
```

$$b : A \quad b\ in\ A$$

```
vars X Y : A .
ceq g(X, Y) = X if X = Y .
```

$\forall\ X, Y : [B]\ .\ g(X, Y) = X\ if\ X = Y \wedge X : A \wedge Y : A$
$\forall\ X, Y : [B]\ .\ X = Y \wedge X\ in\ A \wedge Y\ in\ A \Rightarrow g(X, Y) = X$

Example: Comorphism

```
 op c : -> C [ctor] .
 rl f(X, Y) => c .
endm
```

$\forall\ X,\ Y : [B]\ .\ crl\ f(X,\ Y) => c\ if\ X : A \wedge Y : A$
$\forall\ X,\ Y : [B]\ .\ X\ in\ A \wedge Y\ in\ A \Rightarrow rew(f(X,\ Y),\ c)$

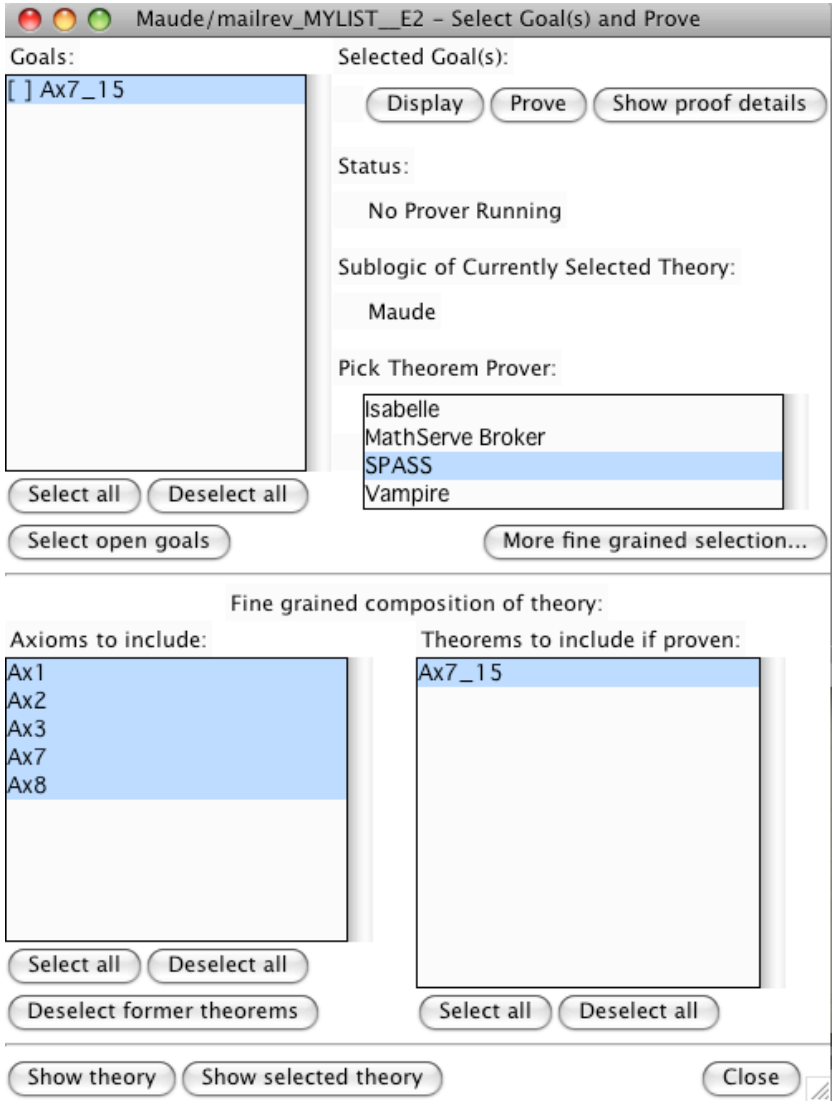
Proving properties

- What can we do now with this comorphism?
- We can prove properties.
- Automatic FOL provers as SPASS or Vampire can be used to prove properties from Maude views like e.g. "natural numbers are a total order."
- More complex properties can be proved with Isabelle HOL.

```
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 .
 eq reverse(nil) = nil .
 eq reverse(E L) = reverse(L) E .
endfm
fth REVIDEM  is
 pr MYLIST .
 op reverse : List -> 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 .*
}

Using theorem provers



Proving freeness constraints

- We implemented a method to prove freeness constraints for CASL.
- Maude uses initial and free semantics intensively.
- The semantics of freeness is different from the one used in CASL.
- Maude free extensions 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 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: non-persistent free links.
- In order not to break the development graph calculus, we need a way to normalize them.
- IDEA: to make a free extension persistent by duplicating parameter sorts appropriately, such that the parameter is always explicitly included in the free extension.

Conclusions

- We have presented how Maude has been integrated into HETS, a parsing, static analysis and proof management tool.
- 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, a Higher Order Logic prover.
- 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.

Ongoing work

- Since interactive proofs are often not easy to conduct, future work will make proving more efficient by adopting automated induction strategies.
- 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 and confluence checker, we map formulas to equations, whose execution in Maude is more efficient.
  - 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 for linear temporal logic.