Logical reasoning in Maude

Santiago Escobar
Universitat Politècnica de València (Spain)

Winter School on Theoretical Foundations of Computer Science
Tbilisi (Georgia)
February 4–9, 2019
Outline

1. Why logical features in rewriting logic?
2. What have we done
3. Rewriting logic in a nutshell
4. Unification
5. Variants in Maude
6. GLINTS
7. Variant-based Equational Unification
8. Narrowing
9. Applications
Outline

1. Why logical features in rewriting logic?
2. What have we done
3. Rewriting logic in a nutshell
4. Unification
5. Variants in Maude
6. GLINTS
7. Variant-based Equational Unification
8. Narrowing
9. Applications
Why rewriting logic?

1. Models and formal specification are easily written in Maude (simplicity, expressiveness, and performance)
2. Rewriting modulo associativity, commutativity and identity
3. Differentiation between concurrent and functional fragments of a model
4. Order-sorted and parameterized specifications
5. Infrastructure for formal analysis and verification (including search command, LTL model checker, theorem prover, etc.)
6. Reflection (meta-modeling, symbolic execution, building tools)
7. Application areas:
   - Models of computation ($\lambda$-calculi, $\pi$-calculus, petri nets, CCS),
   - Programming languages (C, Java, Haskell, Prolog),
   - Distributed algorithms and systems (security protocols, real-time, probabilistic),
   - Biological systems
Why adding logical features to Rewriting Logic?

1. Logical features were included in preliminary designs of the language (80’s) but never implemented in Maude
2. Automated reasoning capabilities by adding logical variables
3. Differentiation between concurrent and functional fragments of a model is lifted to differentiation between symbolic models and equational reasoning.
4. Unification and Narrowing modulo ACU
5. Infrastructure for formal analysis and verification lifted:
   - from equational reduction to equational unification,
   - from search to symbolic reachability,
   - from LTL model checker to logical LTL model checker,
Outline

1. Why logical features in rewriting logic?
2. What have we done
3. Rewriting logic in a nutshell
4. Unification
5. Variants in Maude
6. GLINTS
7. Variant-based Equational Unification
8. Narrowing
9. Applications
What have we done!!

- **Maude 2.4** (2009)
  - **Built-in Unification**: free or associative-commutative (AC)
  - **Narrowing-based search**: rules modulo axioms (no equations).
- **Maude 2.6** (2011)
  - **Built-in Unification**: free, C, AC, or **ACU** (AC + identity)
  - **Variant Unification**: Restricted equations modulo axioms.
  - **Narrowing-based search**: rules modulo equations and axioms.
- **Maude 2.7** (2015)
  - **Built-in Unification**: free, C, AC, or ACU, CU, U, UI, Ur
  - **Built-in Variant unification**: wide class of equational theories.
  - **Narrowing-based search**: rules modulo equations and axioms.
- **Maude 2.7.1** (2016)
  - **Built-in Unification**: previous cases + associativity
  - **Built-in Variant unification**: modulo all combinations
  - **Narrowing-based search**: modulo all combinations
- **Maude 2.8?**
  - **Built-in Narrowing-based search**: modulo all combinations
Outline

1. Why logical features in rewriting logic?
2. What have we done
3. Rewriting logic in a nutshell
4. Unification
5. Variants in Maude
6. GLINTS
7. Variant-based Equational Unification
8. Narrowing
9. Applications
A rewrite theory is
\[ \mathcal{R} = (\Sigma, Ax \uplus E, R), \] with:

1. \( (\Sigma, R) \) a set of rewrite rules of the form \( t \to s \) (i.e., system transitions)

2. \( (\Sigma, Ax \uplus E) \) a set of equational properties of the form \( t = s \) (i.e., \( E \) are equations and \( Ax \) are axioms such as \( ACU \))

Intuitively, \( \mathcal{R} \) specifies a concurrent system, whose states are elements of the initial algebra \( T_{\Sigma/(Ax \uplus E)} \) specified by \( (\Sigma, Ax \uplus E) \), and whose concurrent transitions are specified by the rules \( R \).
Rewriting logic in a nutshell

mod VENDING-MACHINE is
  sorts Coin Item Marking Money State .
  subsort Coin < Money .
  op empty : -> Money .
  subsort Money Item < Marking .
  op <> : Marking -> State .
  ops $ q : -> Coin .
  ops cookie cap : -> Item .
  var M : Marking .
  rl [add-q] : < M > => < M q > .
  rl [buy-a] : < M $ > => < M cookie q > .
  eq [change]: q q q q = $ [variant] .
endm
Rewriting logic in a nutshell

Maude> search $q q q =>! cookie cap St:State.
Solution 1 (state 3)
states: 6 rewrites: 5 in 0ms cpu (0ms real)
St:State --> null
No more solutions.
states: 6 rewrites: 5 in 0ms cpu (1ms real)
Maude> show path 3.
state 0, State: <$ q q q >
===[ rl St $ => St cookie q . ]===>
state 2, State: <$ cookie >
===[ rl St $ => St cap . ]===>
state 3, State: <$ cap cookie >
Rewriting modulo

Rewriting is

Given \((\Sigma, Ax \cup E, R)\), \(t \rightarrow_{R,(Ax\cup E)} s\) if there is

- a non-variable position \(p \in Pos(t)\);
- a rule \(l \rightarrow r\) in \(R\);
- a matching \(\sigma\) (\(E\)-normalized and modulo \(Ax\)) such that
  \(t|_p =_{(Ax\cup E)} \sigma(l)\), and \(s = t[\sigma(r)]|_p\).

Ex: \(< \$ q q q > \rightarrow < \$ \text{cookie} >\)
  using “\(rl < M \$ > \Rightarrow < M \text{cookie} \ q > \).”
  modulo AC of symbol “\_”

Ex: \(< q q q q > \rightarrow < \text{cap} >\)
  using “\(rl < M \$ > \Rightarrow < M \text{cap} > \).”
  modulo simplification with \(q q q q = \$\) and AC of symbol “\_”
Narrowing modulo

Narrowing is

Given \((\Sigma, \text{Ax} \uplus E, R)\), \(t \leadsto_{\sigma, R, (\text{Ax} \uplus E)} s\) if there is

- a non-variable position \(p \in \text{Pos}(t)\);
- a rule \(l \rightarrow r\) in \(R\);
- a unifier \(\sigma\) (\(E\)-normalized and modulo \(\text{Ax}\)) such that
  \[\sigma(t|_p) = (\text{Ax} \uplus E) \sigma(l),\]
  and \(s = \sigma(t[r]_p)\).

Ex: \(< X \ q \ q > \leadsto < $ \ \text{cookie} > \)
  using “rl \(< M \ $ > \Rightarrow < M \ \text{cookie} \ q > .”
  using substitution \(\{X \mapsto $ \ q\}\) modulo AC of symbol “\_”
Ex: \(< X \ q \ q > \leadsto < \text{cap} > \)
  using “rl \(< M \ $ > \Rightarrow < M \ \text{cap} > .”
  using substitution \(\{X \mapsto q \ q\}\)
  modulo simplification with \(q \ q \ q \ q = $\) and AC of symbol “\_”
Outline

1. Why logical features in rewriting logic?
2. What have we done
3. Rewriting logic in a nutshell
4. Unification
5. Variants in Maude
6. GLINTS
7. Variant-based Equational Unification
8. Narrowing
9. Applications
## Definition

Given equational theory \((\Sigma, Ax)\), an *Ax-unification problem* is

\[
t \overset{?}{=} t'
\]

An *Ax-unifier* is an order-sorted substitution \(\sigma\) s.t.

\[
\sigma(t) =_{Ax} \sigma(t')
\]

## Decidability

- at most one mgu (syntactic unification, i.e., empty theory)
- a finite number (associativity–commutativity)
- an infinite number (associativity)
Maude 2.7.1 provides order-sorted $Ax$-unification algorithm for all order-sorted theories $(\Sigma, E \cup Ax, R)$ s.t. $\Sigma$ is preregular modulo $Ax$ and axioms $Ax$ are:

1. arbitrary function symbols and constants with no attributes;
2. $\text{iter}$ equational attribute declared for some unary symbols;
3. $\text{comm}$, $\text{assoc}$, $\text{assoc comm}$, $\text{assoc comm id:}$, $\text{comm id:}$, $\text{id:}$, $\text{left id:}$, or $\text{right id:}$ attributes declared for some binary function symbols but no other equational attributes can be given for such symbols.
Unification Command in Maude

Maude provides a $Ax$-unification command of the form:

\[
\text{unify}\ [\ n\ ]\ \text{in}\ \langle\text{ModId}\rangle:\ \\
\langle\text{Term-1}\rangle =? \langle\text{Term'-1}\rangle \ \land \ldots \land \langle\text{Term-k}\rangle =? \langle\text{Term'-k}\rangle.
\]

- $\text{ModId}$ is the name of the module
- $n$ is a bound on the number of unifiers
- new variables are created as $\text{#n:Sort}$
- Implemented at the core level of Maude (C++)
AC-Unification in Maude

Maude> unify [100] in NAT :

Solution 1
X:Nat --> #1:Nat + #2:Nat + #3:Nat + #5:Nat + #6:Nat + #8:Nat
Y:Nat --> #4:Nat + #7:Nat + #9:Nat
A:Nat --> #1:Nat + #1:Nat + #2:Nat + #3:Nat + #4:Nat
B:Nat --> #2:Nat + #5:Nat + #5:Nat + #6:Nat + #7:Nat
C:Nat --> #3:Nat + #6:Nat + #8:Nat + #8:Nat + #9:Nat
...

Solution 100
X:Nat --> #1:Nat + #2:Nat + #3:Nat + #4:Nat
Y:Nat --> #5:Nat
A:Nat --> #1:Nat + #1:Nat + #2:Nat
B:Nat --> #2:Nat + #3:Nat
C:Nat --> #3:Nat + #4:Nat + #4:Nat + #5:Nat
ACU-Unification in Maude

Decision time: 0ms cpu (1ms real)

Solution 1
X:QidSet --> empty
Y:QidSet --> empty
A:QidSet --> empty
B:QidSet --> empty
C:QidSet --> empty

Solution 2
X:QidSet --> #1:QidSet
Y:QidSet --> empty
A:QidSet --> #1:QidSet, #1:QidSet
B:QidSet --> empty
C:QidSet --> empty
Identity Unification in Maude

mod LEFTID-UNIFICATION-EX is
  sorts Magma Elem . subsorts Elem < Magma .
  op _ : Magma Magma -> Magma [left id: e] .
  ops a b c d e : -> Elem .
endm

Maude> unify in LEFTID-UNIFICATION-EX : X:Magma a =? (Y:Magma a) a .
Solution 1
X:Magma --> a
Y:Magma --> e

Solution 2
X:Magma --> #1:Magma a
Y:Magma --> #1:Magma

Maude> unify in LEFTID-UNIFICATION-EX : a X:Magma =? (a a) Y:Magma .
No unifier.

mod COMM-ID-UNIFICATION-EX is
  sorts Magma Elem . subsorts Elem < Magma .
  op _ : Magma Magma -> Magma [comm id: e] .
  ops a b c d e : -> Elem .
endm

Maude> unify in COMM-ID-UNIFICATION-EX : X:Magma a =? (Y:Magma a) a .
Solution 1
X:Magma --> a
Y:Magma --> e
Solution 2
X:Magma --> a
Y:Magma --> e

Solution 3
X:Magma --> #1:Magma a
Y:Magma --> #1:Magma

Maude> unify in COMM-ID-UNIFICATION-EX : a X:Magma =? (a a) Y:Magma .
Solution 1
X:Magma --> a
Y:Magma --> e
Solution 2
X:Magma --> #1:Magma a
Y:Magma --> #1:Magma
Solution 3
X:Magma --> #1:Magma
Y:Magma --> #1:Magma
A-Unification in Maude


Solution 1
X:NList --> #1:NList : #2:NList
Y:NList --> #3:NList
Z:NList --> #4:NList
P:NList --> #1:NList
Q:NList --> #2:NList : #3:NList : #4:NList

Solution 2
X:NList --> #1:NList
Y:NList --> #2:NList : #3:NList
Z:NList --> #4:NList
P:NList --> #1:NList : #2:NList
Q:NList --> #3:NList : #4:NList

Solution 3
X:NList --> #1:NList
Y:NList --> #2:NList
Z:NList --> #3:NList : #4:NList
P:NList --> #1:NList : #2:NList : #3:NList
Q:NList --> #4:NList
Incomplete A-Unification in Maude

Possible warnings and situations:

- Associative unification using cycle detection.
- Associative unification algorithm detected an infinite family of unifiers.
- Associative unification using depth bound of 5.
- Associative unification algorithm hit depth bound.

Example:

Maude> unify in UNIFICATION-EX4 : 0 : X:NList =? X:NList : 0 .
Warning: Unification modulo the theory of operator _:_ has encountered an instance for which it may not be complete.

Solution 1
X:NList --> 0
Warning: Some unifiers may have been missed due to incomplete unification algorithm(s).
Outline

1 Why logical features in rewriting logic?
2 What have we done
3 Rewriting logic in a nutshell
4 Unification
5 Variants in Maude
6 GLINTS
7 Variant-based Equational Unification
8 Narrowing
9 Applications
Narrowing-based Equational Unification

**Definition**

Given an order-sorted equational theory \((\Sigma, Ax \cup E)\) and \(t \equiv t'\), an \((Ax \cup E)\)-unifier is an order-sorted subst. \(\sigma\) s.t. \(\sigma(t) =_{Ax\cup E} \sigma(t')\).

**When \(Ax = \emptyset\) and \(E\) convergent TRS**

Narrowing provides a complete (but semi-decidable) \(E\)-unification algorithm [Hullot80]. e.g. cancellation \(d(K, e(K, M)) = M\).

**When \(Ax \neq \emptyset\) and \(E\) convergent and coherent TRS modulo \(Ax\)**

Narrowing provides a complete (but semi-decidable) \(E\)-unification algorithm [Jouannaud-Kirchner-Kirchner-83] e.g. exclusive-or \(\text{eq } X \ast 0 = X, \text{eq } X \ast X = 0\) symbol \(*\) being AC
Narrowing-based Equational Unification

Decidable Classes of Equational Theories

Narrowing is very inefficient and may not terminate.
Narrowing strategies for classes of equational theories.

When $Ax = \emptyset$

Basic narrowing strategy [Hullot80] is complete for normalized substitutions.
Cases where basic narrowing terminates have been studied [Alpuente-Escobar-Iborra-TCS09].

When $Ax \neq \emptyset$

Folding variant-narrowing [Escobar-Meseguer-Sasse-JLAP12] is the optimal strategy for equational unification.
From equational reduction to variants (1/4)

**$E,Ax$-variant**

Given a term $t$ and an equational theory $Ax \cup E$, $(t', \theta)$ is an $E,Ax$-variant of $t$ if $\theta(t) \Downarrow_{E,Ax} = Ax t'$ [Comon-Delaune-RTA05]

**Exclusive Or**

\[
\begin{align*}
X \oplus 0 & \rightarrow X \\
X \oplus X & \rightarrow 0 \\
X \oplus X \oplus Y & \rightarrow Y
\end{align*}
\]

\[
\begin{align*}
X \oplus (Y \oplus Z) & = (X \oplus Y) \oplus Z \\
X \oplus Y & = Y \oplus X \\
(X \oplus Y) \oplus Z & = (X \oplus Y) \oplus Z
\end{align*}
\]

(axioms: $Ax$)

**Computed Variants**

For $X \oplus X$: $(0, id)$, $(0, \{X \mapsto a\})$, $(0, \{X \mapsto a \oplus b\})$, …
From equational reduction to variants (2/4)

Finite and complete set of $E,Ax$-variants

A preorder relation of generalization between variants provides a notion of most general variant.

Computed Variants

For $X \oplus Y$ there are 7 most general $E,Ax$-variants

1. $(X \oplus Y, id)$
2. $(0, \{X \mapsto U, Y \mapsto U\})$
3. $(Z, \{X \mapsto 0, Y \mapsto Z\})$
4. $(Z, \{X \mapsto Z \oplus U, Y \mapsto U\})$
5. $(Z, \{X \mapsto Z, Y \mapsto 0\})$
6. $(Z, \{X \mapsto U, Y \mapsto Z \oplus U\})$
### Finite Variant Property

Theory has FVP if **finite** number of most general variants for every term.

### Common

- **Cryptographic Security Protocols**: Public or shared encryption, Exclusive Or, Abelian groups, Diffie-Hellman
- **Satisfiability Modulo Theories** Natural Presburger Arithmetic, Integer Presburger Arithmetic, Lists, Sets

### Used in application areas

- Equational Unification, Logical Model Checking, Cyber-Physical systems, Partial evaluation, Confluence tools, Termination tools, Theorem provers
Test for FVP

Whether a theory has FVP is **undecidable** in general, though there are approximations techniques.

Computing most general variants

Given a theory that has FVP, it is possible to compute all the most general variants by using the **Folding Variant Narrowing Strategy** (Escobar et al. 2012)
Variant Command in Maude

Maude provides variant generation:

\[
\text{get variants } [n] \text{ in } \langle\text{ModId}\rangle : \langle\text{Term}\rangle.
\]

- ModId is the name of the module
- \( n \) is a bound on the number of variants
- new variables are created as \#n:Sort and %n:Sort
- Implemented at the core level of Maude (C++)
- Folding variant narrowing strategy is used internally
- Terminating if Finite Variant Property
- Incremental output if not Finite Variant Property
Exclusive-or Variants

fmod EXCLUSIVE-OR is
  sorts Nat NatSet .  subsort Nat < NatSet .
  op 0 : -> Nat .
  op s : Nat -> Nat .
  op mt : -> NatSet .
  vars X Z : [NatSet] .
endfm

Maude> get variants in EXCLUSIVE-OR : X * Y .
Variant 1
[NatSet]: #1:[NatSet] * #2:[NatSet] ........ Variant 7
[NatSet]: %1:[NatSet]
X --> #1:[NatSet] X --> %1:[NatSet]
Y --> #2:[NatSet] Y --> mt
Abelian Group Variants

fmod ABELIAN-GROUP is
sorts Elem.
  op _+_ : Elem Elem -> Elem [comm assoc].
  op -_ : Elem -> Elem.
  op 0 : -> Elem.
vars X Y Z : Elem.
eq X + 0 = X [variant].
eq X + (- X) = 0 [variant].
eq X + (- X) + Y = Y [variant].
eq -(- X) = X [variant].
eq - 0 = 0 [variant].
eq -(- X) + (- Y) = -(X + Y) [variant].
eq -(X + Y) + Y = - X [variant].
eq -(X + Y) = X + (- Y) [variant].
eq -(X + Y) + Z = -(X + Y) + Z [variant].
eq -(X + Y) + Y + Z = (- X) + Z [variant].
endfm

Maude> get variants in ABELIAN-GROUP : X + Y.
Variant 1
X --> #1:Elem X --> %4:Elem + %1:Elem + %2:Elem
Y --> #2:Elem Y --> %1:Elem + %3:Elem + %4:Elem

Another interesting feature is that variant generation is
incremental. In this
and the new

Note that the two forms have di
ference between unifiers returned by the

does not have the finite variant

The above output illustrates a di

Incremental Variant Generation

fmod NAT-VARIANT is
  sort Nat .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  op _+_ : Nat Nat -> Nat .
  vars X Y : Nat .
  eq [ind] : s(X) + Y = s(X + Y) [variant] .
endfm

Maude> get variants in NAT-VARIANT : s(0) + X .
Variant 1
Nat: s(#1:Nat)
X --> #1:Nat

Maude> get variants [10] in NAT-VARIANT : X + s(0) .
Variant 1
Nat: #1:Nat + s(0) .................................................. Nat: s(s(s(s(0))))
X --> #1:Nat .......................................................... X --> s(s(s(0)))

Infinite!!!
Outline

1. Why logical features in rewriting logic?
2. What have we done
3. Rewriting logic in a nutshell
4. Unification
5. Variants in Maude
6. GLINTS
7. Variant-based Equational Unification
8. Narrowing
9. Applications
Graphical Interactive Narrowing Tree Searcher (GLINTS)

1. An inspecting tool for variant computations in Maude
2. Shows whether a given theory satisfies the finite variant property
3. Allows graphical exploration of variant narrowing computations
4. Automatic checking of node embedding and closedness modulo axioms
5. Querying and inspecting selected parts of the variant trees
6. Beyond Maude by showing internal information: partially computed substitutions, Ax-matching and equational normalization steps

http://safe-tools.dsic.upv.es/glints
Outline

1. Why logical features in rewriting logic?
2. What have we done
3. Rewriting logic in a nutshell
4. Unification
5. Variants in Maude
6. GLINTS
7. Variant-based Equational Unification
8. Narrowing
9. Applications
Admissible Theories

Maude 2.7.1 provides order-sorted $Ax \cup E$-unification algorithm for all order-sorted theories $(\Sigma, Ax, \bar{E})$ s.t.

1. Maude has an $Ax$-unification algorithm,
2. $E$ equations specified with the eq and variant keywords.
3. $E$ is unconditional, convergent, sort-decreasing and coherent modulo $Ax$. The owise feature is not allowed.
Equational Unification Command in Maude

Maude provides a \((Ax \cup E)\)-unification command of the form:

```latex
\text{variant unify \{ n \} in \langle ModId \rangle :} \\
\langle \text{Term-1} \rangle \equiv \langle \text{Term'-1} \rangle \land \ldots \land \langle \text{Term-k} \rangle \equiv \langle \text{Term'-k} \rangle .
```

- \(\text{ModId}\) is the name of the module
- \(n\) is a bound on the number of unifiers
- New variables are created as \#n:Sort and \%n:Sort
- Implemented at the core level of Maude (C++)
- **Terminating** if Finite Variant Property
- **Incremental output** if not Finite Variant Property
Variant-based Unification Command in Maude

fmod NAT-VARIANT is
  sort Nat .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  op _+_ : Nat Nat -> Nat .
  vars X Y : Nat .
  eq [ind] : s(X) + Y = s(X + Y) [variant] .
endfm

Maude> variant unify in NAT-VARIANT : s(0) + X =? s(s(s(0))) .

Unifier #1
X --> s(s(0))

No more unifiers.

Maude> variant unify [1] in NAT-VARIANT : X + s(0) =? s(s(s(0))) .

Unifier #1
X --> s(s(0))

Infinite!!!
Incomplete Variant Unification (due to assoc)

Maude> variant unify in VARIANT-UNIFICATION-ASSOC :
   head(L) =? last(L) \prefix(L) =? tail(L) .

Warning: Unification modulo the theory of operator _:_ has encountered an instance for which it may not be complete.

Unifier #1
L --> %1:Nat : %1:Nat : %1:Nat

Unifier #2
L --> %1:Nat : %1:Nat

No more unifiers.
Warning: Some unifiers may have been missed due to incomplete unification algorithm(s).

Warning: Some unifiers may have been missed due to incomplete unification algorithm(s).

eq head(E : L) = E [variant] .
eq tail(E : L) = L [variant] .
eq prefix(L : E) = L [variant] .
eq last(L : E) = E [variant] .
Outline

1. Why logical features in rewriting logic?
2. What have we done
3. Rewriting logic in a nutshell
4. Unification
5. Variants in Maude
6. GLINTS
7. Variant-based Equational Unification
8. Narrowing
9. Applications
Symbolic reachability analysis in rewrite theories

- Given \((\Sigma, E \cup Ax, R)\) as a concurrent system, a symbolic reachability problem is:
  \[(\exists X) \ t \xrightarrow{\ast} t'\]

- Narrowing provides a sound and complete method for topmost theories.

- Narrowing with \(R\) modulo \(Ax \cup E\) requires \(Ax \cup E\)-unification at each narrowing step.

- Narrowing can be also used for logical model checking.
Narrowing in Maude 2.7.1

Narrowing generalizes term rewriting by allowing free variables in terms and by performing unification instead of matching in order to (non-deterministically) reduce a term.

1. Narrowing + simplification (for built-in operators and equational simplification)
2. Frozen arguments, similar to the context-sensitive narrowing
3. Extra variables in right hand sides of the rules for functional logic programming features (e.g. constraint programming and instantiation search).
Narrowing Search Command in Maude

Narrowing-based search command of the form:

\[(\text{search } [\ n, \ m \ ] \ \text{in} \ \langle \text{ModId} \rangle : \langle \text{Term-1} \rangle \langle \text{SearchArrow} \rangle \langle \text{Term-2} \rangle)\]

- \(n\) is the bound on the desired reachability solutions
- \(m\) is the maximum depth of the narrowing tree
- \(\text{Term-1}\) is not a variable but may contain variables
- \(\text{Term-2}\) is a pattern to be reached
- \(\text{SearchArrow}\) is either \(\Rightarrow 1\), \(\Rightarrow +\), \(\Rightarrow *\), \(\Rightarrow !\)
- \(\Rightarrow !\) denotes strongly irreducible terms or rigid normal forms.
- Implemented in Full Maude
- Already implemented as built-in in Maude 2.8
Variant-based unification in Narrowing Search Command

mod NARROWING-VENDING-MACHINE is
  sorts Coin Item Marking Money State .
  subsort Coin < Money .
  op empty : -> Money .
  subsort Money Item < Marking .
  op <_> : Marking -> State .
  ops $ q : -> Coin .
  ops a c : -> Item .
  var M : Marking .
  rl [buy-c] : < M $ > => < M c > .
  rl [buy-a] : < M $ > => < M a q > .
  eq [change] : q q q q = $ [variant] .
endm

Maude> (search [1] in NARROWING-VENDING-MACHINE :
    < M:Money > ~>* < a c > .)

Solution 1
M:Money --> $ q q q

No more solutions.
Variant-based unification in Narrowing Search Command

(mod AG-VENDING is
  sorts Item Items State Coin Money .
  subsort Item < Items . subsort Coin < Money .
  op _: Items Items -> Items [assoc comm id: mt] . op mt : -> Items .
  op <_|_> : Money Items -> State .
  ops a c : -> Item . ops q $ : -> Coin .
  eq $ = q + q + q + q [variant] . --- Property of the original vending machine example

  op _+_: Money Money -> Money [comm assoc] .
  op -_: Money -> Money .
  op 0 : -> Money .
  vars X Y Z : Money .
  ... (here come the variant equations shown before for Abelian Group)
endm)

Maude> (search [1] in AG-VENDING : < M:Money | mt > ^* < 0 | a c > .)
Solution 1
M:Money --> q + q + q + q + q + q + q
Assoc unification in Narrowing Search Command

(mod GRAMMAR is
    sorts Symbol NSymbol TSymbol String Production Grammar Conf .
    subsorts TSymbol NSymbol < Symbol < String .
    subsort Production < Grammar .
    ops 0 1 2 eps : -> TSymbol . ops S A B C : -> NSymbol .
    op _@_ : String Grammar -> Conf .
    op _->_ : String String -> Production .
    op mt : -> Grammar .
    vars L1 L2 U V : String . var G : Grammar . var N : NSymbol . var T : TSymbol .
    rl ( L1 U L2 @ (U -> V) ; G) => ( L1 V L2 @ (U -> V) ; G) .

e ndm)

Maude> (search [1] N @ (S -> eps) ; (S -> 0 S 1)
    ~>* 0 0 1 1 @ (S -> eps) ; (S -> 0 S 1) .)

Solution 1: N --> S

Maude> (search [1] S @ (N -> T) ; (S -> eps) ; (S -> 0 S 1)
    ~>* 0 0 1 @ (N -> T) ; (S -> eps) ; (S -> 0 S 1) .)

Solution 1: N --> S ; T --> 0

No warning is shown!!!
Outline

1. Why logical features in rewriting logic?
2. What have we done
3. Rewriting logic in a nutshell
4. Unification
5. Variants in Maude
6. GLINTS
7. Variant-based Equational Unification
8. Narrowing
9. Applications
Applications

- Variant-based unification itself
- Formal reasoning tools:
  - Relying on unification capabilities:
    - termination proofs
    - proofs of local confluence and coherence
  - Relying on narrowing capabilities:
    - narrowing-based theorem proving
    - testing
- Logical model checking (model checking with logical variables)
- Cryptographic protocol analysis:
  - the Maude-NPA tool (narrowing + unification in Maude)
  - the Tamarin tool also uses a variant-generation algorithm
- Program transformation: partial evaluation, slicing
- SMT based on narrowing or by variant generation.
Thank you!

More information in the Maude webpage.