Unification and Narrowing in Security Applications

Santiago Escobar

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
sescobar@dsic.upv.es
Outline

1. Why unification and narrowing?
2. Rewriting logic in a nutshell
   Narrowing versus Rewriting
3. Unification
   $Ax$-Unification in Maude
4. Variant-based Equational Unification
   Variants
   $Ax \uplus E$-Unification in Maude
   Combination methods
5. Narrowing
   Model Checking
   BAKERY example
   Narrowing in Maude
   The Bounded Logical LTL Model Checker Tool
6. Narrowing with irreducibility constraints
   Contextual Symbolic Reachability
   Asymmetric Unification
Outline

1. Why unification and narrowing?

2. Rewriting logic in a nutshell
   Narrowing versus Rewriting

3. Unification
   $Ax$-Unification in Maude

4. Variant-based Equational Unification
   Variants
   $Ax \uplus E$-Unification in Maude
   Combination methods

5. Narrowing
   Model Checking
   BAKERY example
   Narrowing in Maude
   The Bounded Logical LTL Model Checker Tool

6. Narrowing with irreducibility constraints
   Contextual Symbolic Reachability
   Asymmetric Unification
Why rewriting logic (Maude)?

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. Models of computation (\(\lambda\)-calculi, \(\pi\)-calculus, petri nets, CCS), Programming languages (Java, Haskell, Prolog), Distributed algorithms and systems (real-time, probabilistic), Biological systems in Maude
8. Similar arguments for other rewriting-based programming languages or paradigms
Why unification and narrowing?

1. **Automated reasoning** about models and formal specification are easily written in Maude by using **logical variables** and its ground instances.

2. Unification and narrowing modulo **associativity, commutativity** and **identity**.

3. Differentiation between **concurrent** and **functional** fragments of a model is **lifted** to differentiation between **symbolic models** and **equational reasoning**.

4. Infrastructure for formal analysis and verification could be lifted to symbolic versions (**search** to **symbolic reachability**, **LTL model checker** to **logical LTL model checker**, **theorem prover** to **advanced theorem prover**, etc.)

5. **Reflection** (new tools based on unification or narrowing, for instance, confluence checker, coherence checker, termination tools).

6. Similar arguments for other rewriting-based programming languages or paradigms: **Functional-Logic Programming Area**.
Current applications

- **Maude-NPA.** Cryptographic protocol analysis tool based on symbolic reachability analysis using narrowing and variants.
- **Tamarin.** ETH Zurich Cryptographic protocol analysis tool using variant-based unification.
- **CRC&ChC.** Maude Coherence and Confluence checker that computes critical pairs using narrowing and variant generation.
- **MTT:** Maude Termination Tool that computes approximations of termination dependency graphs using variant generation.
- **BPM.** Business process modeling using symbolic reachability.
- **Testing.** Narrowing-based test case generation.
- **Theorem proving.** Narrowing-based Theorem Proving
- **Decision procedures.** Superposition-Based Decision Procedures based on narrowing in Maude.
- **Fold/unfold.** Fold/unfold transformations using narrowing.
- **Maude Invariant Analyzer tool.** Safety properties
- **Unification:** homomorphism, xor, variant-based unification
Outline

1. Why unification and narrowing?
2. Rewriting logic in a nutshell
   Narrowing versus Rewriting
3. Unification
   $Ax$-Unification in Maude
4. Variant-based Equational Unification
   Variants
   $Ax \cup E$-Unification in Maude
   Combination methods
5. Narrowing
   Model Checking
   BAKERY example
   Narrowing in Maude
   The Bounded Logical LTL Model Checker Tool
6. Narrowing with irreducibility constraints
   Contextual Symbolic Reachability
   Asymmetric Unification
A rewrite theory is

\[ \mathcal{R} = (\Sigma, Ax \uplus E, R), \] with:

1. \((\Sigma, R)\) a set of rewrite rules of the form \(t \rightarrow s\) (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 [1] < $ q q q > =>* < cookie cap St:Marking > .
Solution 1 (state 12) states: 13 rewrites: 22
St:Marking --> empty

Maude> show path 3 .
state 0, State: < $ q q q >
===[ rl < $ M > => < M q cookie > [label buy-a ] . ]====>
state 4, State: < $ cookie >
===[ rl < $ M > => < cap M > [label buy-c ] . ]====>
state 12, State: < cookie cap >

Maude> reduce q q q q q q .
rewrites: 1 in 0ms cpu (0ms real) (1000000 rewrites/second)
result Money: $ q q q
Outline

2 Rewriting logic in a nutshell
   Narrowing versus Rewriting
Rewriting modulo

Rewriting is

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

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

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

Narrowing is

Given \((\Sigma, Ax \sqcup E, R)\), \(t \rightsquigarrow_{\sigma, R, (Ax \sqcup E)} s\) if there is

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

Ex: \(< X \quad q \quad q > \rightsquigarrow < $ \quad cookie >\)
  
  using “rl \(< M \quad $ > \Rightarrow < M \quad cookie \quad q > .”
  
  using substitution \(\{X \mapsto $ \quad q\}\) modulo AC of symbol “_”

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

1. Why unification and narrowing?
2. Rewriting logic in a nutshell
   Narrowing versus Rewriting
3. Unification
   \(Ax\)-Unification in Maude
4. Variant-based Equational Unification
   Variants
   \(Ax \oplus E\)-Unification in Maude
   Combination methods
5. Narrowing
   Model Checking
   BAKERY example
   Narrowing in Maude
   The Bounded Logical LTL Model Checker Tool
6. Narrowing with irreducibility constraints
   Contextual Symbolic Reachability
   Asymmetric Unification
Unification in Maude 2.6

Unification is

- Fundamental **deductive mechanism** used in automated tools
- Helps **reasoning** about **equational** theories and **rewrite** theories.

**Definition**

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

\[
t \equiv t'
\]

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

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

Most general unifiers

The set of most general $Ax$-unifiers of $s \equiv t$ is the set $\Gamma$ such that any $Ax$-unifier is an instance (or renaming) of a $Ax$-unifier in $\Gamma$.

Decidability

Problem in general undecidable, so different algorithms devised for different theories

- at most one mgu (syntactic unification, i.e., empty theory)
- a finite number (associativity–commutativity)
- an infinite number (associativity)
Ax-Unification

Compared to syntactic unification:

- \( f(a, X) = f(Y, b) \) has solution \( X \mapsto b, Y \mapsto a \)
- \( f(a, X) = f(b, Y) \) has no solution
- \( f(a, X) =_{AC} f(b, Y) \) has solution \( X \mapsto b, Y \mapsto a \)
- \( X + 0 = X \) has no solution
- \( X + 0 =_{ACU} X \), where 0 is the identity, has solution \( id \)
Outline

3 Unification

Ax-Unification in Maude
Admissible Theories

Maude 2.6 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. $\texttt{iter}$ equational attribute declared for some unary symbols;
3. $\texttt{comm}$, $\texttt{assoc comm}$, or $\texttt{assoc comm id}$: attributes declared for some binary function symbols but no other equational attributes can be given for such symbols.
Admissible Theories

Maude 2.6 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. iter equational attribute declared for some unary symbols;
3. comm, assoc comm, or assoc comm id: attributes declared for some binary function symbols but no other equational attributes can be given for such symbols.

Explicitly excluded are theories having

- binary function symbols having any identity attributes without assoc and comm;
- the assoc attribute without the comm one
Unification Command in Maude

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

\[
\text{unify } [\ n\ ] \ \text{in } \langle\text{ModId}\rangle:\ 
\langle\text{Term}-1\rangle =? \langle\text{Term}’-1\rangle \ \text{\&} \cdots \ \text{\&} \langle\text{Term}-k\rangle =? \langle\text{Term}’-k\rangle.
\]

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

1. Why unification and narrowing?
2. Rewriting logic in a nutshell
   Narrowing versus Rewriting
3. Unification
   $Ax$-Unification in Maude
4. Variant-based Equational Unification
   Variants
   $Ax \cup E$-Unification in Maude
   Combination methods
5. Narrowing
   Model Checking
   BAKERY example
   Narrowing in Maude
   The Bounded Logical LTL Model Checker Tool
6. Narrowing with irreducibility constraints
   Contextual Symbolic Reachability
   Asymmetric Unification
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. \(\sigma(t) =_{Ax \cup E} \sigma(t')\).

Compared to syntactic unification:

- \(f(a, X) = f(Y, b)\) has solution \(X \mapsto b, Y \mapsto a\)
- \(f(a, X) =_{AC} f(b, Y)\) has solution \(X \mapsto b, Y \mapsto a\)
- \(X + 0 =_{ACU} X\), where 0 is the identity, has solution \(id\)
- \(X + a + b =_{XOR} a\) has solution \(X \mapsto b\)
- \(X + a + b =_{AG} a\) has solution \(X \mapsto inv(b)\)
Equational Unification

Many methodologies:

- **Semantic unification**: use algebraic methods of the underlying equational theory:
  AC, ACU, ACUI, AG

- **Inference rules**: dedicated algorithm for one equational theory based on inference rules:
  homomorphism, xor, modular exponentiation

- **R/Ax-Narrowing**: use narrowing with oriented equations $R$ modulo a sub theory $Ax$:
  basic narrowing, completeness results for narrowing modulo $Ax$
Challenging Example: XOR + Cancellation of encryption & decryption

\[
\begin{align*}
X \oplus 0 & \rightarrow X \\
X \oplus X & \rightarrow 0 \\
X \oplus X \oplus Y & \rightarrow Y \\
\text{(cancellation rules: } E) \\
X \oplus (Y \oplus Z) & = (X \oplus Y) \oplus Z \\
X \oplus Y & = Y \oplus X \\
\text{(axioms: } Ax) \\
e(K, d(K, M)) & \rightarrow M \\
d(K, e(K, M)) & \rightarrow M \\
\text{(cancellation rules: } E) \\
\text{Issues of coherence modulo assoc & comm: add } X \oplus X \oplus Y & \rightarrow Y
\end{align*}
\]
Equational Unification (History - Completeness)

When $Ax = \emptyset$ and $E$ convergent TRS

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

When $Ax \neq \emptyset$ and $E$ convergent and coherent TRS modulo $Ax$

Narrowing provides a complete (but semi-decidable) $E$-unification procedure [Jouannaud-Kirchner-Kirchner-83] e.g. exclusive-or

\[
\begin{align*}
X \ast 0 & \rightarrow X \\
(X \ast Y) \ast Z & = X \ast (Y \ast Z) \\
X \ast X & \rightarrow 0 \\
X \ast Y & = Y \ast X
\end{align*}
\]
Equational Unification (History - Decidable)

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 $Ax \neq \emptyset$

- $E \uplus Ax$ and $E$ convergent and coherent modulo $Ax$, no strategies have been studied,
- basic narrowing modulo $Ax$ incomplete, and AC-narrowing is highly non-terminating.
- Folding variant-narrowing [Escobar-Meseguer-Sasse-JLAP12] is the most promising strategy for equational unification.
Outline

4 Variant-based Equational Unification

Variants

$Ax \oplus E$-Unification in Maude

Combination methods
**E,Ax-variants**

**E,Ax-variant**

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

1. $t = M \oplus d(K, e(K, M))$ and only variant is $(0, \text{id})$ since $t \downarrow_{E,Ax} = 0$.
2. $s = X \oplus d(K, e(K, Y))$ and two possible variants $(0, \{X/Z, Y/Z\}), (X \oplus Y, \text{id})$
Complete set of $E,Ax$-variants

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

\[ \forall \sigma \text{ s.t. } \sigma(t) \downarrow_{E,Ax} = t', \exists (t'', \theta) \in V_{E,Ax}(t) \text{ s.t.} \]

1. $t''$ is in $\rightarrow_{E,Ax}$-normal form
2. $t'$ and $t''$ ($\sigma \downarrow_{E,Ax}$ and $\theta$) are just renamings modulo $Ax$.

- For $X \oplus X$ only $E,Ax$-variant is: $(0, id)$
- 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. $(Z, \{X \mapsto U, Y \mapsto Z \oplus U\})$
  7. $(Z_1 \oplus Z_2, \{X \mapsto U \oplus Z_1, Y \mapsto U \oplus Z_2\})$
Finite Variant Property

Theory has FVP if there is a finite number of most general $E,Ax$-variants for every term.

- Slightly different notions of variants and finite variant property in literature (only substitution, both substitution and normal form, only normal form).
- If finite number of variants from $t$, $E,Ax$-narrowing must compute them, though infinite redundant $E,Ax$-narrowing sequences may exist.
- [Comon-Delaune-RTA05] An equational theory has the finite variant property if there is a bound $n$ in the number of steps for each term

$$\forall t, \exists n, \forall \sigma \text{ s.t. } (\sigma \downarrow_{E,Ax})(t) \xrightarrow{\leq n}_{E,Ax} \sigma(t) \downarrow_{E,Ax}$$
Finite Variant Property

1. [Comon-Delaune-RTA05] Exclusive Or (max. bound 1)
2. [Comon-Delaune-RTA05] Abelian group (max. bound 2)
3. [Comon-Delaune-RTA05] Diffie-Hellman (max. bound 4)
4. [Comon-Delaune-RTA05] Homomorphism (NOT)
5. [Escobar-Meseguer-Sasse-RTA08-JLAP12] XOR + cancellation, AG + cancellation, XOR + AG for different symbols, etc (Sufficient & necessary conditions for FVP using dependency pair framework)
Left-distributivity

Encryption homomorphic over concatenation

\[ e(K, X; Y) = e(K, X); e(K, Y) \]

Infinite number of substitutions and/or normal forms for \( e(K, X) \)

\[
\sigma_1 = \{ X \mapsto X_1; X_2 \} \quad \Rightarrow \quad e(K, X_1); e(K, X_2) \\
\sigma_2 = \{ X \mapsto X_1; (X_2; X_3) \} \quad \Rightarrow \quad e(K, X_1); (e(K, X_2); e(K, X_3)) \\
\vdots \quad \Rightarrow
\]
Outline

4 Variant-based Equational Unification
   Variants
   $Ax \sqcup E$-Unification in Maude
   Combination methods
Admissible Theories

- Full Maude 2.6 provides order-sorted $Ax \cup E$-unification algorithm for all order-sorted theories $(\Sigma, E \cup Ax, R)$ s.t.
  1. Maude has an $Ax$-unification algorithm,
  2. $E$ is a set of equations specified with the eq keyword.
  3. Equations in $E$ are unconditional, convergent, sort-decreasing and coherent modulo $Ax$. The owise feature is not allowed.
  4. For $LHS \rightarrow RHS$ in $E$, $LHS$ cannot be a variable
  5. $RHS$ must be a strongly irreducible term (made of constructor symbols and variables).
  6. $LHS$ must be variant-preserving (to ensure eager generation of variants).
Admissible Theories

- Full Maude 2.6 provides order-sorted $Ax \cup E$-unification algorithm for all order-sorted theories $(\Sigma, E \cup Ax, R)$ s.t.
  1. Maude has an $Ax$-unification algorithm,
  2. $E$ is a set of equations specified with the eq keyword.
  3. Equations in $E$ are unconditional, convergent, sort-decreasing and coherent modulo $Ax$. The owise feature is not allowed.
  4. For $LHS \rightarrow RHS$ in $E$, $LHS$ cannot be a variable
  5. $RHS$ must be a strongly irreducible term (made of constructor symbols and variables).
  6. $LHS$ must be variant-preserving (to ensure eager generation of variants).

- A prototype for full variant-based unification (without strongly irreducible rhs’s) is under implementation in Core Maude (Maude 2.7 to appear)
Maude provides a \((Ax \cup E)\)-unification command of the form:

\[
\text{variant unify } [\text{ in } \langle \text{ModId}\rangle : ] \langle \text{Term1}\rangle =? \langle \text{Term2}\rangle .
\]

- ModId is the name of the module
- All unifiers are returned.
- Metalevel version has bound for number of unifiers.
- Folding variant narrowing is used internally
(\textit{Ax} \uplus \textit{E})\text{-Unification Command in Maude}

Maude> (variant unify in NARROWING-VENDING-MACHINE :
  \textless \textit{q q X:Marking} \textgreater =? \textless \textit{$ Y:Marking} \textgreater .)

Solution 1
\textit{X:Marking} \rightarrow \textit{q q Y:Marking}

Solution 2
\textit{X:Marking} \rightarrow \textit{$ \#12:Marking} ; \textit{Y:Marking} \rightarrow \textit{q q \#12:Marking}
Outline

4 Variant-based Equational Unification

Variants

Ax ⊔ E-Unification in Maude

Combination methods
Combination methods in Maude

\( E \cup Ax \)-unification framework in Maude

1. Built-in Maude algorithms \( Ax \)-unification
2. Dedicated algorithms \( Ax \)-unification
3. Variant-based algorithm \( Ax \)-narrowing with rules \( E \)
4. Hybrid approach: combinations of three previous approaches

Dedicated algorithms tested in Maude-NPA: homomorphism & xor
Outline

1 Why unification and narrowing?
2 Rewriting logic in a nutshell
   Narrowing versus Rewriting
3 Unification
   $Ax$-Unification in Maude
4 Variant-based Equational Unification
   Variants
   $Ax \cup E$-Unification in Maude
   Combination methods
5 Narrowing
   Model Checking
   BAKERY example
   Narrowing in Maude
   The Bounded Logical LTL Model Checker Tool
6 Narrowing with irreducibility constraints
   Contextual Symbolic Reachability
   Asymmetric Unification
Symbolic reachability analysis in rewrite theories

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

- 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 symbolic model checking and the temporal logic of rewriting.
Outline

6 Narrowing
   Model Checking
   BAKERY example
   Narrowing in Maude
   The Bounded Logical LTL Model Checker Tool
Model Checking of Finite State Systems

- Model checking techniques effective in verification of concurrent systems with a finite number of states.
Model Checking of Finite State Systems

- Model checking techniques effective in verification of concurrent systems with a finite number of states.

- A concurrent system can be naturally modelled as a rewrite theory \( R = (\Sigma, E, R) \), order-sorted signature \( \Sigma \), equational theory \( E \), and rewrite rules \( R \)
  - states are elements of the initial algebra \( T_{\Sigma/E} \)
  - concurrent transitions are axiomatized by the rewrite rules \( R \)
  - order-sorted parameterised specifications
  - simplificity and expresiveness
VENDING-MACHINE in Rewriting Logic

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 [buy-a] : < M $ > => < M cookie q > .
    eq [change]: q q q q = $ [variant] .
endm
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 [buy-a] : < M $ > => < M cookie q > .
eq [change]: q q q q = $ [variant] .
endm

(one initial state - finite space)
Model Checking of Infinite State Systems

- Various model checking techniques for infinite-state systems exist, but they are less developed, though important advances have been made in recent years:
  - regular languages [Abdulla et. al.-CAV02]
  - string/multiset grammars [Boujjani&Esparza-RTA06, Vardhan et. al.-TACAS05]
  - tree automata [Genet&Tong-LPAR01, Ohsaki&Seki&Takai-RTA03]
  - constraint logic programming [Delzanno&Podelski-STTT01]
  - Presburger arithmetic [Bultan&Gerber&Pugh-CAV97]
  - program specialization [Pettorossi&Proietti-LOPSTR10]
  - abstraction methods [Boujjani&Touili-STTT12, Burkart et. al.-01, Genet&Rusu-JSC10]
Model Checking of Infinite State Systems

- Various **model checking** techniques for **infinite-state systems** exist, but they are less developed, though important advances have been made in recent years:
  - regular languages [Abdulla et. al.-CAV02]
  - string/multiset grammars [Boujjani&Esparza-RTA06, Vardhan et. al.-TACAS05]
  - tree automata [Genet&Tong-LPAR01, Ohsaki&Seki&Takai-RTA03]
  - constraint logic programming [Delzanno&Podelski-STTT01]
  - Presburger arithmetic [Bultan&Gerber&Pugh-CAV97]
  - program specialization [Pettorossi&Proietti-LOPSTR10]
  - abstraction methods [Boujjani&Touili-STTT12, Burkart et. al.-01, Genet&Rusu-JSC10]

- An **infinite-state system** modelled again as a **rewrite theory** \( \mathcal{R} = (\Sigma, E, R) \). Finite state system obtained by:
  - abstraction methods
  - ad-hoc partial orders
mod VENDING-MACHINE-INF 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-\$] : < M > => < M $ > .
    rl [add-\q] : < M > => < M q > .
    rl [buy-a] : < M $ > => < M cookie q > .
    eq [change]: q q q q = $ [variant] .

endm
mod VENDING-MACHINE-INF 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

< $ $ $ > \rightarrow < $ $ $ $ > \rightarrow < $ $ $ $ $ > \rightarrow \infty

< $ $ q > \rightarrow < $ $ q q > \rightarrow \infty

(one initial state - infinite space)
Alternative Solution to Infinite State Systems: Logical Model Checking [RTA07]

- **Logical Model Checking** is a challenging possibility where an infinite set of terms is represented by a logical variable. Term \(< X \ q \ q >\) denotes an infinite set of possible instances.

\[
\begin{align*}
< \text{Money} > & \quad \rightarrow \quad < \text{cap Money'} > \\
\text{Money} \rightarrow $ Money' & \quad \rightarrow \quad \text{Money'} \rightarrow $ Money'' \\
< \text{cap Money'} > & \quad \rightarrow \quad < \text{cap cap Money''} > \\
\infty &
\end{align*}
\]

*(multiple initial states - infinite space)*
Alternative Solution to Infinite State Systems: Logical Model Checking [RTA07]

- **Logical Model Checking** is a challenging possibility where an infinite set of terms is represented by a logical variable. Term $\langle X \ q \ q \rangle$ denotes an infinite set of possible instances.

$\langle \text{Money} \rangle \xrightarrow{\text{Money} \rightarrow \$ \ \text{Money}'} \langle \text{cap Money'} \rangle \xrightarrow{\text{Money'} \rightarrow \$ \ \text{Money''}} \langle \text{cap cap Money''} \rangle \xrightarrow{\infty} \langle \text{cap Money'} \rangle \xrightarrow{\text{cap cap Money''}} \langle \infty \rangle$

(multiple initial states - infinite space)

- Based on **narrowing** with rules $R$ modulo the equational theory $E$ instead of **rewriting** with rules $R$ modulo $E$.
Alternative Solution to Infinite State Systems: 
Logical Model Checking [RTA07]

- **Logical Model Checking** is a challenging possibility where an infinite set of terms is represented by a logical variable. 
  Term $< X \ q \ q >$ denotes an infinite set of possible instances.

  
  
  $< \text{Money} > \rightarrow < \text{cap Money’} > \rightarrow < \text{cap cap Money’’} >$

  $\text{Money} \rightarrow$ Money’

  $\text{Money’} \rightarrow$ Money’’

  $\text{Money’’} \rightarrow$ Money’’’

  $\text{cap Money’} \rightarrow \text{cap Money’’}$

  $\text{cap Money’’} \rightarrow \text{cap Money’’’}$

  (multiple initial states - infinite space)

- Based on **narrowing** with rules $R$ modulo the equational theory $E$ instead of **rewriting** with rules $R$ modulo $E$

- Originally **symbolic reachability analysis** [Meseguer&Thati-HOSC07] but then **LTL logical model checking** [Escobar&Meseguer-RTA07]
Infinite State into Finite State

- **Logical Model Checking** can handle an infinite set of states by using logical variables
- **Finite state systems** are obtained by over-approximation of state transitions (rather than data approximation) using folding relations
Infinite State into Finite State

- **Logical Model Checking** can handle an infinite set of states by using logical variables.
- Finite state systems are obtained by over-approximation of state transitions (rather than data approximation) using folding relations.

\[
\begin{array}{c}
\text{Narrowing + folding relation} \Rightarrow (\text{multiple initial states - finite space}) \\
\text{(equality $=_{E}$)} \\
\text{(renaming $\approx_{E}$)} \\
\text{(instantiation $\preceq_{E}$)}
\end{array}
\]
Outline

5 Narrowing

Model Checking

BAKERY example

Narrowing in Maude

The Bounded Logical LTL Model Checker Tool
Bakery algorithm: Transition System

Token to give ; Token serving ; Set of Processes
Nat Nat \{ idle, wait(Nat), crit(Nat) \}

\( r \| N ; M ; [idle] PS \Rightarrow (s N) ; M ; [wait(N)] PS \cdot \)
\( r \| N ; M ; [wait(M)] PS \Rightarrow N ; M ; [crit(M)] PS \cdot \)
\( r \| N ; M ; [crit(M)] PS \Rightarrow N ; (s M) ; [idle] PS \cdot \)
Bakery algorithm: Transition System

Token to give ; Token serving ; Set of Processes
Nat Nat \[
\{ \text{idle, wait(Nat), crit(Nat)} \} \]

\[ r \mid N ; M ; [\text{idle}] \text{ PS} \Rightarrow (s N) ; M ; [\text{wait}(N)] \text{ PS} . \]
\[ r \mid N ; M ; [\text{wait}(M)] \text{ PS} \Rightarrow N ; M ; [\text{crit}(M)] \text{ PS} . \]
\[ r \mid N ; M ; [\text{crit}(M)] \text{ PS} \Rightarrow N ; (s M) ; [\text{idle}] \text{ PS} . \]

\[
\begin{align*}
0 ; 0 ; [\text{idle}] & \quad s ; s ; [\text{idle}] & \quad ss ; ss ; [\text{idle}] \\
\downarrow & \quad \downarrow & \quad \downarrow \\
\downarrow & \quad s ; 0 ; [\text{wait}(0)] & \quad s ; s ; [\text{wait}(s)] & \quad sss ; ss ; [\text{wait}(ss)] \\
\downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow \\
\downarrow & \quad s ; 0 ; [\text{crit}(s)] & \quad ss ; s ; [\text{crit}(s)] & \quad \infty
\end{align*}
\]

(Transition System: one initial state - infinite space)
Bakery algorithm: Logical Transition System

\[
\begin{align*}
&0; 0; \text{[idle]} \quad s; s; \text{[idle]} \quad ss; ss; \text{[idle]} \\
&\quad \downarrow \quad \downarrow \quad \downarrow \\
&s; 0; \text{[wait(0)]} \quad s; s; \text{[wait(s)]} \quad sss; ss; \text{[wait(ss)]} \\
&\quad \downarrow \quad \downarrow \quad \downarrow \\
&s; 0; \text{[crit(s)]} \quad ss; s; \text{[crit(s)]} \\
&\quad \downarrow \quad \downarrow \quad \downarrow \\
&s; s; [\text{crit(s)}] \quad ss; s; [\text{crit(s)}] \\
&\quad \downarrow \quad \downarrow \\
&\quad \infty
\end{align*}
\]

(Transition System: one initial state - infinite space)
Bakery algorithm: Logical Transition System

\[ 0 ; 0 ; [\text{idle}] \quad s ; s ; [\text{idle}] \quad ss ; ss ; [\text{idle}] \]
\[ \downarrow \quad \downarrow \quad \downarrow \]
\[ s ; 0 ; [\text{wait(0)}] \quad s ; s ; [\text{wait(s)}] \quad sss ; ss ; [\text{wait(ss)}] \]
\[ \downarrow \quad \downarrow \quad \downarrow \]
\[ s ; 0 ; [\text{crit(s)}] \quad ss ; s ; [\text{crit(s)}] \quad \infty \]

(Transition System: one initial state - infinite space)

\[ N ; N ; [\text{idle}] \quad sN ; sN ; [\text{idle}] \quad ssN ; ssN ; [\text{idle}] \]
\[ \downarrow \quad \downarrow \quad \downarrow \]
\[ sN ; N ; [\text{wait(N)}] \quad ssN ; sN ; [\text{wait(sN)}] \quad sssN ; ssN ; [\text{wait(ssN)}] \]
\[ \downarrow \quad \downarrow \quad \downarrow \]
\[ sN ; N ; [\text{crit(N)}] \quad ssN ; sN ; [\text{crit(sN)}] \quad \cdots \]

(Logical Transition System: infinite initial state - infinite space)
Bakery algorithm: Folding Logical Transition System

\[
\begin{align*}
N; N; \text{[idle]} & \Rightarrow E \Rightarrow E \\
& \downarrow \downarrow \\
& sN; N; \text{[wait(N)]} \Rightarrow E \\
& \downarrow \\
sN; N; \text{[crit(N)]} & \Rightarrow E \\
& \downarrow \\
sN; N; \text{[idle]} & \Rightarrow E \\
& \downarrow \\
sN; N; \text{[wait(N)]} & \Rightarrow E \\
& \downarrow \\
sN; N; \text{[crit(N)]} & \Rightarrow E \\
\end{align*}
\]

(Folding Logical Transition System: infinite initial state - finite space)
Bakery algorithm: Folding Logical Transition System

(Folding Logical Transition System: infinite initial state - finite space)

- If $\mathcal{K}_2$ simulates $\mathcal{K}_1$, any LTL formula satisfied in $\mathcal{K}_2$ is satisfied in $\mathcal{K}_1$
Bakery algorithm: Folding Logical Transition System

\[
\begin{align*}
N;N;[idle][idle] & \Rightarrow_E sN;N;[wait(N)][idle] \\
                & \Rightarrow_E sN;N;[crit(N)][idle] \\
                & \Rightarrow_E s(sN);N;[wait(N)][wait(sN)] \\
                & \Rightarrow_E s(sN);N;[crit(N)][wait(sN)] \\
\end{align*}
\]

(Folding Logical Transition System: infinite initial state - finite space)

- If $\mathcal{K}_2$ simulates $\mathcal{K}_1$, any LTL formula satisfied in $\mathcal{K}_2$ is satisfied in $\mathcal{K}_1$
- $\mathcal{K}_2$ is a faithful abstraction of $\mathcal{K}_1$ if there is no spurious counterexample
Bakery algorithm: Folding Logical Transition System

\[
\begin{align*}
N; N; [idle] & \xrightarrow{E} N; N; [idle] \\
\downarrow & \downarrow & \downarrow \\
N; N; [wait(N)] & \xrightarrow{E} N; N; [idle] & \xrightarrow{E} N; N; [idle] [wait(N)] \\
\downarrow & \downarrow & \downarrow \\
N; N; [crit(N)] & \xrightarrow{E} N; N; [idle] [crit(N)] & \xrightarrow{E} N; N; [idle] [crit(N)] \\
\downarrow & \downarrow & \downarrow \\
s(N); N; [wait(N)] & \xrightarrow{E} s(N); N; [crit(N)] [wait(sN)] & \xrightarrow{E} s(N); N; [crit(N)] [wait(sN)] \\
\downarrow & \downarrow & \downarrow \\
s(N); N; [wait(sN)] [crit(N)] & \xrightarrow{E} s(N); N; [wait(sN)] [crit(N)] & \xrightarrow{E} s(N); N; [wait(sN)] [crit(N)] \\
\end{align*}
\]

(Folding Logical Transition System: infinite initial state - finite space)

- If $K_2$ simulates $K_1$, any LTL formula satisfied in $K_2$ is satisfied in $K_1$
- $K_2$ is a faithful abstraction of $K_1$ if there is no spurious counterexample
- Any folding abstraction (e.g. $\equiv_E$, $\approx_E$, $\preceq_E$) are faithful abstractions for safety LTL formulas
Bakery algorithm: However, Infinite-State System

(Infinite Folding Logical Transition System: infinite initial state - infinite space)
Bakery algorithm: However, Infinite-State System

\[ \text{Narrowing} \quad \text{BAKERY example} \]

(Narrowing Logical Transition System: infinite initial state - infinite space)

- Many verification problems for infinite-state systems are due to unbounded number of processes
- All approaches use a symbolic finite representation of an infinite number of processes
- Bisimulation proofs written by hand or hard to reuse
Bakery algorithm: Bisimilar Equational abstractions

- Easy conditions to be checked [Bae, Escobar, Meseguer RTA2013]
- Necessary/sufficient conditions are provided
Bakery algorithm: Bisimilar Equational abstractions

- Easy conditions to be checked [Bae, Escobar, Meseguer RTA2013]
- Necessary/sufficient conditions are provided
- Example:
  \[
  \text{eq} \ (ss sLM) ; M ; PS_0 \ [\text{wait}(sLM)] \ [\text{wait}(ssLM)] \\
  = (ssLM) ; M ; PS_0 \ [\text{wait}(sLM)].
  \]

(\text{Abstract Bisimilar Folding Logical Transition System})
Bakery algorithm: Bisimilar Equational abstractions

- Easy conditions to be checked [Bae, Escobar, Meseguer RTA2013]
- Necessary/sufficient conditions are provided
- Example:
  
  \[
  \text{eq} (s \cdot s \cdot s \cdot L \cdot M) ; M ; PS_0 \ [\text{wait}(s \cdot L \cdot M)] \ [\text{wait}(s \cdot s \cdot L \cdot M)] \\
  = (s \cdot s \cdot L \cdot M) ; M ; PS_0 \ [\text{wait}(s \cdot L \cdot M)] .
  \]

(N; N; IS) \[\text{IS/IS}_1[\text{idle}]\] \rightarrow (sN; N; IS_1 [\text{wait}(N)]) \[\text{IS}_1/\text{IS}_2[\text{idle}]\] \rightarrow (sN; N; IS_1 [\text{crit}(N)])

(IS_2/IS_3[\text{idle}] \bigcirc) \text{sN}; N; IS_2 [\text{wait}(N)] [\text{wait}(sN)] \rightarrow (\text{sN}; N; IS_2 [\text{crit}(N)] [\text{wait}(sN)]

(Abstract Bisimilar Folding Logical Transition System)

- Bisimilar equational abstractions are obvious faithful abstractions for any LTL formula
Outline

5 Narrowing
Model Checking
BAKERY example
Narrowing in Maude
The Bounded Logical LTL Model Checker Tool
Narrowing in Maude 2.6

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 of S. Lucas.

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

Full Maude provides a 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 \(\sim>1\), \(\sim>+\), \(\sim>*\), \(\sim>!\)
- \(\sim>!\) denotes strongly irreducible terms or rigid normal forms.
- Implemented in Full Maude (no Core Maude yet).
Narrowing Search Command in Maude

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

Maude> (search [1,4] in VENDING-MACHINE : < M:Money > ~>* < a c > .)
Solution 1
M:Money --> $ q q q
Outline

5 Narrowing
   Model Checking
   BAKERY example
   Narrowing in Maude

The Bounded Logical LTL Model Checker Tool
The Bounded Logical LTL Model Checker Tool

- Extended with a $k$-step under-approximation of the abstract folding logical Kripke structure. Better than standard Bounded Model Checking!!!!
- Implemented in Logical LTL Model Checker
  http://formal.cs.illinois.edu/kbae/lmc/
Output 1/3: Logical Model Checking without Folding

```
logical model check in BAKERY-SATISFACTION :
   N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result:
   no counterexample found within bound 10
```
Output 2/3: Folding Logical Model Checking

logical folding model check in BAKERY-SATISFACTION :
N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result:
no counterexample found within bound 50
Output 3/3: Bisimilar Abstract Folding Logical Model Checking

Maude> (lfmc N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex? .)
logical folding model check in BAKERY-SATISFACTION-ABS :
N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result: true
Outline

1. Why unification and narrowing?
2. Rewriting logic in a nutshell
   Narrowing versus Rewriting
3. Unification
   $Ax$-Unification in Maude
4. Variant-based Equational Unification
   Variants
   $Ax \sqcup E$-Unification in Maude
   Combination methods
5. Narrowing
   Model Checking
   BAKERY example
   Narrowing in Maude
   The Bounded Logical LTL Model Checker Tool
6. Narrowing with irreducibility constraints
   Contextual Symbolic Reachability
   Asymmetric Unification
Narrowing with irreducibility constraints

- Some applications require **extra features or adaptations** of the standard model checking approach
- In **security**, some terms may have extra properties (such as frozen arguments or the new irreducibility constraints).
- **Maude-NPA, ProVerif, OFMC, ...** many protocol verification tools include implicit irreducibility constraints
Example with Irreducibility constraints

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

• In the initial model, we do not consider configurations
  < Mo q q q q | M > because they are simplified into < Mo $ | M >

• Impose an irreducibility restriction on the left side of <_|_>

• At the logical level, a configuration < Mo q q X | M > cannot be
  instantiated to \{X \rightarrow q q\}
Outline

6 Narrowing with irreducibility constraints
   Contextual Symbolic Reachability
   Asymmetric Unification
A contextual rewrite theory is

\( \mathcal{R} = (\Sigma, E, R, T, \phi) \), with:

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

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

3. \( \phi \) is a function mapping each \( f \in \Sigma \) to a set of its arguments, i.e., \( \phi(f) \subseteq \{1, \ldots, \text{ar}(f)\} \). Irreducibility requirements

Intuitively, \( T \) specifies a concurrent system, whose states are elements of the initial algebra \( T_{\Sigma/(E \cup R)} \) specified by \((\Sigma, E \cup R)\), and whose concurrent transitions are specified by the rules \( T \).

For each term in \( T_{\Sigma/(E \cup R)} \) and each subterm under an irreducibility requirement, the subterm is in normal form w.r.t. \( R \) modulo \( E \).
Symbolic reachability analysis in contextual rewrite theories

- Given \((\Sigma, E, R, T, \phi)\) as a concurrent system, a symbolic reachability problem is

\[
(\exists X) \ t \longrightarrow^* \ t'
\]

- From what instances of \(t\) there is a transition sequence modulo \(E \uplus R\) to a instance of \(t'\).

- The rewriting sequence must satisfy the irreducibility requirements.

- Irreducibility constraints have to be included during the symbolic reachability analysis.

- Variants of symbolic states have to be generated before adding the irreducibility constraints.
Contextual Narrowing modulo (I)

Contextual Narrowing is

Given \((\Sigma, E, R, T, \phi)\), \(\langle t, \Pi \rangle \rightsquigarrow_{\sigma,T,(E\cup R)} \langle t', \sigma(\Pi) \rangle\) if there is

- a non-variable position \(p \in \text{Pos}(t)\);
- a rule \(l \rightarrow r\) in \(T\);
- a unifier \(\sigma\) (\(R\)-normalized and modulo \(E\)) such that \(\sigma(t|_p) = (E\cup R) \sigma(l)\), and \(\sigma(\Pi)\) still \(R,E\)-irreducible
- \(t' = \sigma(t[r]_p)\).

New unification paradigm is necessary here: asymmetric unification

\(\phi, R, E\)-variants

\(\langle t, \Pi \rangle \rightarrow_{R,E}^\theta \langle w, \overline{\Pi} \rangle\) denotes \((w, \theta) \in \llbracket \langle t, \Pi \rangle \rrbracket_{R,E}^\phi\) and \(\overline{\Pi} = \theta(\Pi) \cup \{w\}\).
Contextual Narrowing modulo (II)

Contextual Symbolic Reachability is

Given \((\Sigma, E, R, T, \phi)\), a reachability goal \(t \rightarrow^* t'\), and a solution \(\sigma\), there is a sequence

\[
\langle t, \Pi_0 \rangle \rightarrow_{R,E}^{\theta_1} \langle w_1, \Pi_1 \rangle \sim_{T,R,E,\phi}^{\theta'_1} \langle u_1, \overline{\Pi_1} \rangle \quad \Pi_1 = \Pi_0 \cup \{w_1\}
\]
\[
\rightarrow_{R,E}^{\theta_2} \langle w_2, \Pi_2 \rangle \sim_{T,R,E,\phi}^{\theta'_2} \langle u_2, \overline{\Pi_2} \rangle \quad \Pi_2 = \Pi_1 \cup \{w_2\}
\]
\[
\vdots
\]
\[
\rightarrow_{R,E}^{\theta_n} \langle w_n, \Pi_n \rangle \sim_{T,R,E,\phi}^{\theta'_n} \langle u_n, \overline{\Pi_n} \rangle \quad \Pi_n = \Pi_{n-1} \cup \{w_n\}
\]
\[
\rightarrow_{R,E}^{\theta_{n+1}} \langle w_{n+1}, \Pi_{n+1} \rangle \sim_{T,R,E,\phi}^{\theta'_{n+1}} \langle t'', \overline{\Pi_{n+1}} \rangle \quad \Pi_{n+1} = \Pi_n \cup \{w_{n+1}\}
\]

\(t''\) more general than \(t'\), and \(\theta_1 \theta'_1 \theta_2 \theta'_2 \cdots \theta_{n+1} \theta'_{n+1}\) more general than the solution \(\sigma\).
Contextual Symbolic Reachability

(search [,4] in IRR-NARROWING-VENDING-MACHINE :
   < Mo:Money | M:Marking > ~>* < empty | cap cookie > .)
Solution 1
Mo:Money --> $ q q q, M:Marking --> empty

(search [,4] in IRR-NARROWING-VENDING-MACHINE :
   < q q Mo:Money | M:Marking > ~>* < Mo’:Money | cap cookie > .)
Solution 1
Mo:Money --> $ $, M:Marking --> empty, Mo’:Money --> q q q
Outline

6 Narrowing with irreducibility constraints
   Contextual Symbolic Reachability
   Asymmetric Unification
Asymmetric Unification

- New paradigm interesting by itself with many implications in rewriting modulo an equational theory

Definition

Given an order-sorted equational theory \((\Sigma, E \cup Ax)\), an asymmetric \(E \cup Ax\)-unification problem is

\[ t = \downarrow t' \]

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

\[ \sigma(t) \downarrow_{E, Ax} = Ax \sigma(t') \]
Asymmetric Unification

\[
\text{(asymm variant unify in NARROWING-VENDING-MACHINE :}
\]
\[
< q q X:\text{Marking} > =? < $ Y:\text{Marking} > .)
\]

Solution 1
X:Marking \rightarrow q q Y:Marking

Solution 2
X:Marking \rightarrow $ #12:\text{Marking} ; Y:Marking \rightarrow q q #12:\text{Marking}

\[
\text{(asymm variant unify in NARROWING-VENDING-MACHINE :}
\]
\[
< $ Y:\text{Marking} > =? < q q X:\text{Marking} > .)
\]

Solution 1
X:Marking \rightarrow $ #12:\text{Marking} ; Y:Marking \rightarrow q q #12:\text{Marking}
Experiments

Table 1. Experiments with standard reachability analysis using regular XOR unification algorithm vs contextual reachability analysis using asymmetric XOR unification algorithm. A pair \(n/t\) means: \(n\) = number of states, and \(t\) = time in seconds.

<table>
<thead>
<tr>
<th>states/seconds</th>
<th>1 step</th>
<th>2 steps</th>
<th>3 steps</th>
<th>4 steps</th>
<th>5 steps</th>
<th>Finite Analysis?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP - Standard</td>
<td>2/0.08</td>
<td>5/0.16</td>
<td>13/0.86</td>
<td>49/3.09</td>
<td>267/17.41</td>
<td>No, timeout with 6 steps</td>
</tr>
<tr>
<td>RP - Contextual</td>
<td>1/0.03</td>
<td>45/1.08</td>
<td>114/2.26</td>
<td>1175/37.25</td>
<td>13906/4144.30</td>
<td>Yes, at step 10</td>
</tr>
<tr>
<td>WEPP - Standard</td>
<td>5/0.09</td>
<td>9/0.42</td>
<td>26/1.27</td>
<td>106/5.80</td>
<td>503/34.76</td>
<td>No, timeout with 7 steps</td>
</tr>
<tr>
<td>WEPP - Contextual</td>
<td>4/0.05</td>
<td>9/0.12</td>
<td>26/0.64</td>
<td>257/144.65</td>
<td>2454/612.08</td>
<td>Yes at step 5</td>
</tr>
<tr>
<td>TMN - Standard</td>
<td>5/0.11</td>
<td>15/0.55</td>
<td>99/3.82</td>
<td>469/25.68</td>
<td>timeout</td>
<td>No, timeout with 7 steps</td>
</tr>
<tr>
<td>TMN - Contextual</td>
<td>4/0.06</td>
<td>24/0.53</td>
<td>174/3.63</td>
<td>1079/170.29</td>
<td>9737/1372.55</td>
<td>Yes, at step 21</td>
</tr>
</tbody>
</table>

Table 2. Experiments for contextual reachability analysis using asymmetric XOR unification algorithm with and without optimizations.

<table>
<thead>
<tr>
<th>states/seconds</th>
<th>1 step</th>
<th>2 steps</th>
<th>3 steps</th>
<th>4 steps</th>
<th>5 steps</th>
<th>Finite Analysis?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP - w/o Opt.</td>
<td>1/0.03</td>
<td>45/1.08</td>
<td>114/2.26</td>
<td>1175/37.25</td>
<td>13906/4144.30</td>
<td>No, timeout with 6 steps</td>
</tr>
<tr>
<td>RP - with Opt.</td>
<td>4/0.59</td>
<td>7/0.59</td>
<td>7/1.92</td>
<td>7/1.89</td>
<td>7/3.02</td>
<td>Yes, at step 10</td>
</tr>
<tr>
<td>WEPP - w/o Opt.</td>
<td>4/0.05</td>
<td>9/0.12</td>
<td>26/0.64</td>
<td>257/144.65</td>
<td>2454/612.08</td>
<td>No, timeout with 7 steps</td>
</tr>
<tr>
<td>WEPP - with Opt.</td>
<td>2/0.36</td>
<td>2/0.20</td>
<td>1/0.80</td>
<td>2/1.42</td>
<td>1/0.03</td>
<td>Yes at step 5</td>
</tr>
<tr>
<td>TMN - w/o Opt.</td>
<td>4/0.06</td>
<td>24/0.53</td>
<td>174/3.63</td>
<td>1079/170.29</td>
<td>9737/1372.55</td>
<td>No, timeout with 7 steps</td>
</tr>
<tr>
<td>TMN - with Opt.</td>
<td>3/0.42</td>
<td>6/9.85</td>
<td>9/1.78</td>
<td>9/4.43</td>
<td>8/3.20</td>
<td>Yes, at step 21</td>
</tr>
</tbody>
</table>
Thanks!

Questions?